An overview on weight initialization methods for feedforward neural networks

Celso A. R. de Sousa
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo, São Carlos, Brazil
email: sousa@icmc.usp.br

***Abstract*—Feedforward neural networks are neural networks with (possibly) multiple layers of neurons such that each layer is fully connected to the next one. They have been widely studied in the past partially due to their universal approximation capabilities and empirical effectiveness on a variety of application domains for both regression and classification tasks. In this paper, we provide an overview on feedforward neural networks, focusing on weight initialization methods.**

I. Introduction

A *feedforward neural network* (FNN) consists of a neural network with (possibly) multiple layers of neurons such that each layer is fully connected to the next one. FNNs learn by adjusting their connection weights in order to iteratively decrease an error measure associated to its outputs for a given labeled training sample. FNNs are useful connectionist models due to their universal approximation capabilities [1]–[9]; theoretically speaking, FNNs are capable to approximate any function with any degree of accuracy. Through a comprehensive experimental analysis, *Caruana & Niculescu-Mizil* [10] concluded the following:

> "*Of the earlier learning methods, feed-forward neural nets have the best performance and are competitive with some of the newer methods, particularly if models will not be calibrated after training.*"

FNNs have been effectively applied in many learning problems, such as feature selection [11]–[14], function approximation [15], and multi-label learning [16]. Most studies on FNNs focus on nonparametric regression estimators, which are *consistent* for all regression problems. A consistent learning algorithm is said to be unbiased because it is not a priori dedicated to a particular solution or a class of solutions, achieving, in theory, *optimal* performance whatever the task [17]. Unfortunately, these algorithms may have high variance to achieve this goal on some application domains [18].

White [19] used statistical theory for the *method of sieves* [20] to prove that multilayer FNNs can be used to obtain a consistent learning procedure under fairly general conditions. The method of sieves is a general approach to nonparametric estimation in which an object of interest lying in some (possibly infinite dimensional) space Θ is approximated using a sequence of parametric models in which the dimension of the parameter space grows along with the sample size.

Consider that $\mathbb{E}[y|x]$ is the conditional expectation of y given x. Given a training sample $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n} \subset \mathcal{X} \times \mathcal{Y}$, in which $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}$, and a function $f : \mathcal{X} \mapsto \mathbb{R}$, it is shown in [18] that

$$\mathbb{E}\left[(y_i - f(\boldsymbol{x}_i))^2 \middle| \boldsymbol{x}_i\right] \geq \mathbb{E}\left[(y_i - \mathbb{E}[y_i|\boldsymbol{x}_i])^2 \middle| \boldsymbol{x}_i\right], \forall i \quad (1)$$

Eq. (1) says that the regression $\mathbb{E}[y|\boldsymbol{x}]$ is the best predictor of y given $\boldsymbol{x}$ in the mean-square-error sense [18]. The mathematical theories behind FNNs can be traced back to the so-called Kolmogorov's superposition theorem [21]–[22], which proves that an arbitrary multivariate continuous function $f : [0,1]^n \mapsto \mathbb{R}$ can be represented as

$$f(\boldsymbol{x}) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{q,p}(\boldsymbol{x}_p) \right) \quad (2)$$

where $\Phi_q : \mathbb{R} \mapsto \mathbb{R}$ and $\psi_{q,p} : \mathbb{R} \mapsto \mathbb{R}$ are continuous functions. Although Kolmogorov has proved the existence of a function represented by Eq. (2), it is impossible to use this result in an algorithm for numerical calculations. Kurkova [23] partly eliminated these difficulties by substituting the exact representation in Eq. (2) with an approximation of the function f. This approach also enabled an estimation of the number of hidden neurons in a single-hidden-layer FNN as a function of the desired accuracy ϵ.

FNNs are usually referred as "black boxes", i.e., determining why an FNN makes a particular decision can be a hard task. To the best of our knowledge, this is still an open problem. In order to enhance the interpretability of FNNs, *Chung et al.* [24] proposed a novel universal approximator model based on FNNs. Such a model is proven to be functionally equivalent to a fuzzy inference system based on syllogistic fuzzy reasoning, being effective on providing an FNN model through a set of comprehensive fuzzy rules.

Although FNNs may not be interpretable by humans even if we use a small network, they may be free of the "*curse of dimensionality*" [3] since the number of neurons of an FNN needed for approximating a target function depends only upon the basic geometrical shape of the target function, not on the dimension of the input space. This is clearly an advantage over conventional linear regression methods, which may drastically suffer from this problem.

When the cost function is chosen to satisfy a certain mild condition [17], the optimal weights of an FNN given a training sample have the fundamental information theoretic interpretation that they minimize the expected *Kullback-Leibler* (KL)[1] divergence with respect to the network architecture, the training sample, and the cost function applied. With this interpretation, learning in FNNs can be posed as a *quasi-maximum likelihood*[2] statistical estimation procedure. A relevant discussion on this topic for connectionist models can be found in [25].

A. Motivation

Many researchers have focused on providing novel learning algorithms for training FNNs while many related research topics on FNNs have taken little attention, such as *weight initialization*, *knowledge extraction*, *geometrical interpretation*, and *network sparseness*. For instance, the weight initialization process was not effectively studied in the past, partially due to the technical difficulties on providing novel algorithms for this research field.

Since the weight initialization process usually has a high influence in the classification performance of an FNN, such a process is of great interest to applied researchers. However, we have no tools to recommend good initial weights for a given FNN with respect to the training sample and the network architecture.

Specifically, the motivations of this paper are summarized as follows:

- the weight initialization process usually have a higher impact on the classification performance than the learning algorithm used with respect to the network architecture and the training sample[3];
- providing a deep review of existing methods with discussions on open problems and possible insights on how to overcome them is of great interest for future research on FNNs.

B. Contributions

The contributions of this paper can be summarized as follows:

- we provide a deep review on weight initialization methods for FNNs in *shallow*[4] architectures;
- we provide a unified notation for the methods under review;
- we discuss strengths and weaknesses of the methods under review and discuss open problems on FNNs concerning the weight initialization process;
- we provide insights for future research on how to overcome some of the open problems on FNNs based on the study of existing methods.

C. Outline

The remainder of this paper is organized as follows. Section II describes the notation used in the paper. Section III revises weight initialization methods. Section IV discusses open problems. Finally, Section V concludes the paper.

II. Notation

Our notation is based on the notation and theoretical results in [26]. Let $\mathbb{N}_p := \{i \in \mathbb{N}^* | 1 \leq i \leq p\}$ with $p \in \mathbb{N}^*$. Consider $\mathbf{A}_{i,\cdot} \in \mathbb{R}^{1\times n}$ and $\mathbf{A}_{\cdot,j} \in \mathbb{R}^m$ the i-th row and j-th column vectors, respectively, of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$. Let $\mathbf{1}_n$ and $\mathbf{0}_n$ be the n-dimensional 1-entry and 0-entry vectors, respectively. Let $\mathbf{I}_n$ be the $n \times n$ identity matrix. The symbol $\odot$ represents the Hadamard (or element-wise) product of matrices or vectors. Given two matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{m\times n}$, we can compute the matrix $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$ as $\mathbf{C}_{ij} = \mathbf{A}_{ij}\mathbf{B}_{ij}$, $\forall i \in \mathbb{N}_m$, $\forall j \in \mathbb{N}_n$. We define the inner product of vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ by $\langle \boldsymbol{u}, \boldsymbol{v}\rangle_n := \boldsymbol{u}^\top \boldsymbol{v} = \boldsymbol{v}^\top \boldsymbol{u} = \sum_{i=1}^n \boldsymbol{u}_i \boldsymbol{v}_i$ where the superscript $^\top$ denotes transposition of matrices or vectors.

Consider a training sample $\mathbb{S}_{\mathcal{X}\mathcal{Y}} := \{\boldsymbol{x}_i, \boldsymbol{y}_i\} \subset \mathcal{X} \times \mathcal{Y}$ in which $\mathcal{X} := \{\boldsymbol{x}_i\}_{i=1}^n \subset \mathbb{R}^d$ is the set of training examples with the corresponding outputs[5] $\mathcal{Y} := \{\boldsymbol{y}_i\}_{i=1}^n \subset \mathbb{R}^c$. Let $\mathcal{Q}(\cdot)$ be a cost function to be minimized with respect to $\mathbb{S}_{\mathcal{X}\mathcal{Y}}$ and the network weights. Therefore, an FNN is trained to minimize $\mathcal{Q}(\cdot)$ with respect to $\mathbb{S}_{\mathcal{X}\mathcal{Y}}$ and its *network architecture* in order to provide a function $f : \mathbb{R}^d \mapsto \mathbb{R}^c$. Mathematically, training an FNN is equivalent to find a set of values for the network weights with respect to $\mathbb{S}_{\mathcal{X}\mathcal{Y}}$ that minimizes $\mathcal{Q}(\cdot)$, usually not *convex*[6] in the network weights, with satisfactory approximation.

Network architecture refers to the way neurons are interconnected in the neural network. FNNs have a feedforward architecture, in which there exists $\mathcal{L}$ layers of neurons and each neuron on a given layer is connected to all neurons in the next layer. In addition, there exist connections between neurons only on consecutive layers. Also, a *bias* term is included in the input of each neuron to improve the generalization capability and convergence of FNNs.

Let $\zeta^{(l)} \in \mathbb{N}^*$, $l \in \mathbb{N}_{\mathcal{L}} \cup \{0\}$, be the number of neurons in the l-th layer. By definition, $\zeta^{(0)} = d$ and $\zeta^{(\mathcal{L})} = c$. For simplicity, we consider that the 0-th layer is associated to the input example $\boldsymbol{x} \in \mathbb{R}^d$. Mathematically, the values $\left\{\zeta^{(l)}\right\}_{l=1}^{\mathcal{L}}$ represent the network architecture of a given FNN.

[1]Given two discrete probability distributions P and Q, the KL divergence of Q from P is defined by $D_{KL}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$. If P and Q are distributions of a continuous random variable X, the KL divergence is given by $D_{KL}(P\|Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \mathrm{d}x$.

[2]Roughly speaking, a quasi-maximum (or pseudo) likelihood estimate is an estimate of a parameter Θ in a statistical model that is formed by maximizing a function that is related to the logarithm of the likelihood function $\mathcal{L}(\Theta|x)$ given outcomes x.

[3]This statement is based on our experimental knowledge on FNNs. However, there is a lack of comprehensive experimental analyses in the literature that show the impact of weight initialization in the classification performance of FNNs on benchmark data sets.

[4]A shallow architecture refers to FNNs with only one hidden layer. We can also write single-hidden-layer FNN in this case.

[5]In this paper, we focus our attention on supervised learning for both regression and classification tasks. Therefore, there always exist an expected output for a given input example.

[6]A set X in a vector space over the real numbers is said to be convex if, $\forall x, y \in X, \forall t \in [0,1]$, the point $z = tx + (1-t)y$ also belongs to X. Given $x_1, x_2 \in X$ such that X is a convex set, a function $f : X \mapsto \mathbb{R}$ is called convex if, $\forall t \in [0,1]$, $f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$. Many machine learning algorithms lead with convex objective functions [27]–[30], as opposed to many research fields on neural networks.

Let $\mathbf{W}^{(l)} \in \mathbb{R}^{\left(\zeta^{(l-1)}+1\right)\times\zeta^{(l)}}$, $l \in \mathbb{N}_{\mathcal{L}}$, be the weighted matrix that encodes the weights between neurons of the layers $l-1$ and l. Specifically, $\mathbf{W}^{(l)}_{i,j}$, $\forall i \in \mathbb{N}_{\zeta^{(l-1)}}$, $\forall j \in \mathbb{N}_{\zeta^{(l)}}$, is the weight between neurons i and j of the $(l-1)$-th and l-th layers, respectively. In addition, $\mathbf{W}^{(l)}_{0,j}$, $\forall j \in \mathbb{N}_{\zeta^{(l)}}$, represents the bias associated to the j-th neuron of the l-th layer. Therefore, $\mathbf{W}^{(l)}_{0,\cdot} \in \mathbb{R}^{1\times\zeta^{(l)}}$ is the bias vector associated to the l-th layer. Consider $\mathbf{W}^{(l)}_{a:b}$, $b \geq a$, the submatrix of $\mathbf{W}^{(l)}$ containing the row vectors of $\mathbf{W}^{(l)}$ with indices between a and b. Mathematically, $\mathbf{W}^{(l)}_{a:b} := \left[\left(\mathbf{W}^{(l)}_{a,\cdot}\right)^\top, \cdots, \left(\mathbf{W}^{(l)}_{b,\cdot}\right)^\top\right]^\top$.

Let $\boldsymbol{\eta}^{(l)} \in \left(\mathbb{R}^*_+\right)^{\left(\zeta^{(l-1)}+1\right)\times\zeta^{(l)}}$, $l \in \mathbb{N}_{\mathcal{L}}$, be the learning rates of the network connections between the layers $l-1$ and l. Specifically, every $\boldsymbol{\eta}^{(l)}_{i,j}$ is associated to the network weight $\mathbf{W}^{(l)}_{i,j}$. One can use a "global" learning rate ρ if necessary. This means that $\boldsymbol{\eta}^{(l)}_{i,j} = \rho$, $\forall l \in \mathcal{L}$, $\forall i \in \mathbb{N}_{\zeta^{(l-1)}+1}$, $\forall j \in \mathbb{N}_{\zeta^{(l)}}$.

Let $\boldsymbol{\Psi}^{(l)}_{\boldsymbol{x}} \in \mathbb{R}^{\zeta^{(l-1)}+1}$, $\forall l \in \mathbb{N}_{\mathcal{L}}$, be the inputs for the l-th layer given an input example $\boldsymbol{x}$. By definition, $\boldsymbol{\Psi}^{(1)}_{\boldsymbol{x}} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}$. We can recursively compute the inputs of a given layer l as [26]:

$$\begin{aligned}\boldsymbol{\Psi}^{(l)}_{\boldsymbol{x}} &= \begin{bmatrix} \mathbf{0}^\top_{\zeta^{(l-1)}} \\ \mathbf{I}_{\zeta^{(l-1)}} \end{bmatrix} \xi^{(l-1)}\left(\left(\mathbf{W}^{(l-1)}\right)^\top \boldsymbol{\Psi}^{(l-1)}_{\boldsymbol{x}}\right) \\ &+ \begin{bmatrix} 1 \\ \mathbf{0}_{\zeta^{(l-1)}} \end{bmatrix}\end{aligned}$$

where $\xi^{(l)} : \mathbb{R}^{\zeta^{(l)}} \mapsto \mathbb{R}^{\zeta^{(l)}}$ is the activation function in the l-th layer. For simplicity, assume that the activation function is the same for all layers. Typically, $\xi^{(l)}$ is the *logistic* or the *hyperbolic tangent* function.

Let $\boldsymbol{\psi}^{(l)}_{\boldsymbol{x}} \in \mathbb{R}^{\zeta^{(l-1)}+1}$, $l \in \mathbb{N}_{\mathcal{L}} \cup \{0\}$, be the outputs for the l-th layer given an input example $\boldsymbol{x}$. By definition, $\boldsymbol{\psi}^{(0)}_{\boldsymbol{x}} = \boldsymbol{x}$. Note that $\boldsymbol{\Psi}^{(l)}_{\boldsymbol{x}} = \begin{bmatrix} 1 \\ \boldsymbol{\psi}^{(l-1)}_{\boldsymbol{x}} \end{bmatrix}$, $\forall l \in \mathbb{N}_{\mathcal{L}}$. Therefore [26]:

$$\boldsymbol{\psi}^{(l)}_{\boldsymbol{x}} = \xi^{(l)}\left(\left(\mathbf{W}^{(l)}\right)^\top \begin{bmatrix} 1 \\ \boldsymbol{\psi}^{(l-1)}_{\boldsymbol{x}} \end{bmatrix}\right)$$

Let $\boldsymbol{\pi}^{(l)}_{\boldsymbol{x}} \in \mathbb{R}^{\zeta^{(l)}}$, $l \in \mathbb{N}_{\mathcal{L}}$, be the *induced local field* of the l-th layer given an input example $\boldsymbol{x}$. By definition:

$$\left(\boldsymbol{\pi}^{(l)}_{\boldsymbol{x}}\right)_i = \sum_{j=0}^{\zeta^{(l-1)}} \mathbf{W}^{(l)}_{j,i}\left(\boldsymbol{\Psi}^{(l)}_{\boldsymbol{x}}\right)_j = \left\langle \mathbf{W}^{(l)}_{\cdot,i}, \boldsymbol{\Psi}^{(l)}_{\boldsymbol{x}}\right\rangle_{\zeta^{(l-1)}+1}$$

Therefore, we have:

$$\boldsymbol{\pi}^{(l)}_{\boldsymbol{x}} = \begin{bmatrix} \left\langle \mathbf{W}^{(l)}_{\cdot,1}, \boldsymbol{\Psi}^{(l)}_{\boldsymbol{x}}\right\rangle_{\zeta^{(l-1)}+1} \\ \vdots \\ \left\langle \mathbf{W}^{(l)}_{\cdot,\zeta^{(l)}}, \boldsymbol{\Psi}^{(l)}_{\boldsymbol{x}}\right\rangle_{\zeta^{(l-1)}+1} \end{bmatrix}$$

Let $f_{\mathbb{W}}(\boldsymbol{x})$ be the output of the FNN for an input example $\boldsymbol{x}$ and network weights $\left\{\mathbf{W}^{(l)}\right\}_{l=1}^{\mathcal{L}}$. Therefore, we can write:

$$\begin{aligned} f_{\mathbb{W}}(\boldsymbol{x}) &= \xi^{(\mathcal{L})}\left(\left(\mathbf{W}^{(\mathcal{L})}\right)^\top \boldsymbol{\Psi}^{(\mathcal{L})}_{\boldsymbol{x}}\right) \\ &= \xi^{(\mathcal{L})}\left(\left(\mathbf{W}^{(\mathcal{L})}\right)^\top \begin{bmatrix} 1 \\ \boldsymbol{\psi}^{(\mathcal{L}-1)}_{\boldsymbol{x}} \end{bmatrix}\right)\end{aligned}$$

Let $\boldsymbol{\delta}^{(l)} \in \mathbb{R}^{\zeta^{(l)}}$, $l \in \mathbb{N}_{\mathcal{L}}$, be the *local gradients* of the l-th layer. Let $f^{(\kappa)}_{\text{log}}(x)$ and $f^{(\kappa,\omega)}_{\text{tanh}}(x)$ be the *logistic* and *hyperbolic tangent* functions, respectively. Then, we have:

$$f^{(\kappa)}_{\text{log}}(x) = \frac{1}{1+\exp(-\kappa x)}, \quad f^{(\kappa,\omega)}_{\text{tanh}}(x) = \omega\tanh(\kappa x)$$

Assume that $\mathcal{Q}(\cdot)$ is a quadratic cost function and we show to the network a sample $(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{X}\times\mathcal{Y}$. Using the logistic function as activation function, the local gradients for the output and hidden layers, $\boldsymbol{\delta}^{(\mathcal{L})}_{\boldsymbol{x}}$ and $\boldsymbol{\delta}^{(l)}_{\boldsymbol{x}}$, respectively, can be computed as:

$$\begin{aligned}\boldsymbol{\delta}^{(\mathcal{L})}_{\boldsymbol{x}} &= \kappa\left(\boldsymbol{y} - f_{\mathbb{W}}(\boldsymbol{x})\right) \odot f_{\mathbb{W}}(\boldsymbol{x}) \odot \left(\mathbf{1}_c - f_{\mathbb{W}}(\boldsymbol{x})\right) \\ \boldsymbol{\delta}^{(l)}_{\boldsymbol{x}} &= \kappa\boldsymbol{\psi}^{(l)}_{\boldsymbol{x}} \odot \left(\mathbf{1}_{\zeta^{(l)}} - \boldsymbol{\psi}^{(l)}_{\boldsymbol{x}}\right) \odot \boldsymbol{\varphi}^{(l)}_{\boldsymbol{x}}\end{aligned}$$

where $\kappa, \omega \in \mathbb{R}^*_+$ are free parameters. Also, $\boldsymbol{\varphi}^{(l)}_{\boldsymbol{x}} \in \mathbb{R}^{\zeta^{(l)}}$ is defined by:

$$\boldsymbol{\varphi}^{(l)}_{\boldsymbol{x}} = \begin{bmatrix} \left\langle \mathbf{W}^{(l+1)}_{1,\cdot}, \boldsymbol{\delta}^{(l+1)}_{\boldsymbol{x}}\right\rangle_{\zeta^{(l+1)}} \\ \vdots \\ \left\langle \mathbf{W}^{(l+1)}_{\zeta^{(l)},\cdot}, \boldsymbol{\delta}^{(l+1)}_{\boldsymbol{x}}\right\rangle_{\zeta^{(l+1)}} \end{bmatrix}$$

$\forall l \in \mathbb{N}_{\zeta^{(l)}-1}$. When we apply the hyperbolic tangent function as activation function, the local gradients for the output and hidden layers can be computed as:

$$\begin{aligned}\boldsymbol{\delta}^{(\mathcal{L})}_{\boldsymbol{x}} &= \frac{\kappa}{\omega}\left(\boldsymbol{y} - f_{\mathbb{W}}(\boldsymbol{x})\right) \odot \left(\omega^2\mathbf{1}_c - f_{\mathbb{W}}(\boldsymbol{x}) \odot f_{\mathbb{W}}(\boldsymbol{x})\right) \\ \boldsymbol{\delta}^{(l)}_{\boldsymbol{x}} &= \frac{\kappa}{\omega}\left(\omega^2\mathbf{1}_{\zeta^{(l)}} - \boldsymbol{\psi}^{(l)}_{\boldsymbol{x}} \odot \boldsymbol{\psi}^{(l)}_{\boldsymbol{x}}\right) \odot \boldsymbol{\varphi}^{(l)}_{\boldsymbol{x}}\end{aligned}$$

Finally, the weight updates are given by:

$$\begin{aligned}\mathbf{W}^{(l,t)} &= (1+\alpha)\mathbf{W}^{(l,t)} + \boldsymbol{\eta}^{(l)} \odot \left(\boldsymbol{\Psi}^{(l)}\left(\boldsymbol{\delta}^{(l)}\right)^\top\right) \\ &- \alpha\mathbf{W}^{(l,t-1)}\end{aligned}$$

where $\mathbf{W}^{(l,t)}$ represents the matrix $\mathbf{W}^{(l)}$ in the t-th iteration and $\alpha \in [0,1]$ is the momentum coefficient.

III. Weight initialization

For the weight initialization process, we want to create a method that can provide an effective answer to the following question:

> *Given a labeled training sample $\mathbb{S}_{\mathcal{X}\mathcal{Y}}$ and a predefined network architecture, how can we provide a good ("near optimal") initialization for the network weights with respect to a given regression or classification problem?*

Providing a method for this task is usually very hard due to the (possibly) large number of connection weights. Mathematically, we have to generate a mapping $f_{WI}\left(\mathbb{S}_{\mathcal{X}\mathcal{Y}}, \{\zeta^{(l)}\}_{l=1}^{\mathcal{L}}\right)$ in order to initialize the matrices $\{\mathbf{W}^{(l)}\}_{l=1}^{\mathcal{L}}$ such that $\mathcal{Q}(\cdot)$ gives a *small enough* value[7].

A. Strenghts and weaknesses

The weight initialization process has the following strenghts:

- we can improve the classification performance of an FNN using near optimal initialization of the connection weights;
- the convergence of gradient-based methods on well initialized networks is usually much faster than on randomly initialized ones.

The weight initialization process has the following weaknesses:

- there is a lack of statistical models in the literature that could indicate when a given weight initialization method could give satisfactory results for a given training sample and network architecture;
- many of the existing methods are computationally expensive in the number of connection weights, being unfeasible for large networks.

B. Algorithms

In this section, we revise the weight initialization methods proposed in [31]–[34]. These methods were chosen due to their good mathematical background and empirical effectiveness. Other relevant methods can be found in [35]–[50].

1) Statistically Controlled Activation Weight Initialization (SCAWI) [31]: The SCAWI method shows that there exists an amplitude window in the maximum magnitude of the initial weights. Small values of the previous quantity lead to poor local minima, while large ones may lead neurons to a *saturated state*. This, with high probability, prevents the network from reaching the desired solution. We begin the description of the SCAWI method with Definition 1, which defines the *saturated* and *paralyzed* states.

Definition 1 (Saturated and paralyzed states): Given a sample $(\boldsymbol{x},\boldsymbol{y}) \in \mathbb{S}_{\mathcal{X}\mathcal{Y}}$, the neuron j of layer l is in a saturated state when $\left(\psi_{\boldsymbol{x}}^{(l)}\right)_j \geq \overline{\mathcal{S}}_{\mathcal{S}}$ where $\overline{\mathcal{S}}_{\mathcal{S}} \in \mathbb{R}_+^*$ is a predefined value for neuron saturation. The output neuron j is in a paralyzed state when it is saturated (i.e., $[f_{\mathbb{W}}(\boldsymbol{x})]_j \geq \overline{\mathcal{S}}_{\mathcal{S}}$) and $\left|[f_{\mathbb{W}}(\boldsymbol{x})]_j - \boldsymbol{y}_j\right| \geq \overline{\mathcal{S}}_{\mathcal{P}}$ such that $\overline{\mathcal{S}}_{\mathcal{P}} \in \mathbb{R}_+^*$ is a predefined value for the paralyzed state.

$\overline{\mathcal{S}}_{\mathcal{P}}$ and $\overline{\mathcal{S}}_{\mathcal{S}}$ are free parameters in the statistical model. The authors in [31] suggested to set $\overline{\mathcal{S}}_{\mathcal{S}} = \overline{\mathcal{S}}_{\mathcal{P}} = 0.9$, which achieved satisfactory results. The SCAWI method was specifically designed for FNNs with a shallow architecture ($\mathcal{L} = 2$) with $\xi^{(1)}(x) = \xi^{(2)}(x) = \tanh(x)$. The main idea of such a method is to randomly generate the weights $\mathbf{W}_{i,j}^{(l)}$ with some statistical guarantees. For such purpose, we take into account Definition 2.

Definition 2 (Paralyzed neuron percentage (PNP)): PNP is defined by PNP $:= \Gamma_{\mathcal{X},\mathbb{W}}/(nc)$ where $\Gamma_{\mathcal{X},\mathbb{W}}$ is the number of paralyzed neurons with respect to $\mathcal{X}$ with predefined network weights $\mathbb{W}$.

The mathematical formulation used for weight initialization is given by[8]:

$$\mathbf{W}_{i,j}^{(l)} = \frac{\varpi \alpha_0}{\theta_l}\nu_{i,j}, \qquad \theta_l = \sqrt{\sum_{i=1}^{\zeta^{(l-1)}+1} \mathbb{E}\left[\left(\left(\psi_{\boldsymbol{x}}^{(l)}\right)_i\right)^2\right]} \tag{3}$$

where $\varpi \in \mathbb{R}_+^*$ is a scale factor to be computed, $\alpha_0 \in \mathbb{R}_+^*$ is the value of $\left(\boldsymbol{\pi}_{\boldsymbol{x}}^{(l)}\right)_j$ such that $\left(\psi_{\boldsymbol{x}}^{(l)}\right)_j = \overline{\mathcal{S}}_{\mathcal{S}}$, $\nu_{i,j}$ is a random number uniformly distributed in the interval $[-1,+1]$, and θ_l is a normalization factor. Since $\overline{\mathcal{S}}_{\mathcal{S}} = 0.9$, we have $\alpha_0 = \tanh^{-1}(0.9) \approx 1.47$. With these definitions, the weights $\mathbf{W}_{i,j}^{(l)}$ can be treated as *independent random variables*[9] such that:

$$\mathbb{E}\left[\mathbf{W}_{i,j}^{(l)}\right] = 0, \quad \mathbb{E}\left[\left(\mathbf{W}_{i,j}^{(l)}\right)^2\right] = \frac{1}{3}\frac{\varpi^2\alpha_0^2}{\theta_l} \tag{4}$$

In order to provide a weight initialization method that guarantees that Eq. (4) holds, the authors in [31] elaborated Hypothesis 1. Such a hypothesis describes the statistical assumption on the external inputs of the network and the outputs of each layer.

Hypothesis 1: The external inputs of the network, $\mathcal{I}_{i,j}^{(l)}$, are considered equivalent to random variables characterized by their mean values, $\mathbb{E}\left[\mathcal{I}_{i,j}^{(l)}\right]$, and by their variance, $\sigma^2_{\mathcal{I}_{i,j}^{(l)}}$. Mathematically,

[7]There is no clear definition of a small enough value. Actually, a very small value of $\mathcal{Q}(\cdot)$ could lead to *overfitting* on the test sample, which is undesired for real applications. It is useful to evaluate the performance of a weight initialization method on a validation sample, instead of on the training sample.

[8]The generalized formulation in [31] takes into account an arbitrary number of external inputs for each neuron. However, due to its ineffectiveness in practice, we consider that only the *bias* of the neurons are the external inputs available, which is the most commonly used setting. Therefore, each neuron has only one external input with predefined value 1.

[9]Although the weights $\mathbf{W}_{i,j}^{(l)}$ are randomly generated, we have control on the expected value of these random variables, i.e., $\mathbb{E}\left[\mathbf{W}_{i,j}^{(l)}\right]$ and $\mathbb{E}\left[\left(\mathbf{W}_{i,j}^{(l)}\right)^2\right]$ are known a priori.

$$\mathbb{E}\left[\left(\mathcal{I}_{i,j}^{(l)}\right)^2\right] = \sigma^2_{\mathcal{I}_{i,j}^{(l)}} + \mathbb{E}^2\left[\mathcal{I}_{i,j}^{(l)}\right] \tag{5}$$

Moreover, the outputs of the neurons can be considered as a random variable characterized by[10]*:*

$$\mathbb{E}\left[\left(\boldsymbol{\psi}_{\boldsymbol{x}}^{(l)}\right)_j\right] = \mathbb{E}\left[\tanh^2\left(\left(\boldsymbol{\pi}_{\boldsymbol{x}}^{(l)}\right)_j\right)\right] \tag{6}$$

$\forall j \in \mathbb{R}^{\zeta^{(l)}}$.

Assume that the number of inputs of each neuron, given by $\zeta^{(l-1)} + 1$, is high enough to allow the use of the *central limit theorem*[11]. This implies that the activation of a neuron, $\left(\boldsymbol{\pi}_{\boldsymbol{x}}^{(l)}\right)_j$, has, with a good approximation, a Gaussian probability distribution. From this approximation and using Hypothesis 1, we obtain:

$$\mathbb{E}\left[\left(\boldsymbol{\pi}_{\boldsymbol{x}}^{(l)}\right)_j\right] = 0, \qquad \sigma^2_{\left(\boldsymbol{\pi}_{\boldsymbol{x}}^{(l)}\right)_j} = \sigma_{\boldsymbol{\pi}} = \frac{\varpi \alpha_0}{\sqrt{3}}$$

Therefore, the standard deviation of the activation is the same for *every* neuron, which is an interesting statistical property. That is why the normalization factor θ_l is used in Eq. (3). From Definition 2 and standard statistical analysis in [31], we obtain:

$$\text{PNP} = \left(1 - \frac{1}{2^c}\right)\left(1 - 2\text{erf}\left(\frac{\sqrt{3}}{\varpi}\right)\right)$$

such that

$$\text{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-t^2} \mathrm{d}t$$

Therefore, we have:

$$\varpi = \frac{\sqrt{3}}{\text{erf}^{-1}\left(\frac{1}{2} - \frac{\left(1 - \frac{1}{2^c}\right)^{-1}}{2}\text{PNP}\right)\text{PNP}}$$

From Definition 2, we can easily compute PNP in practice. Therefore, the value of ϖ is obtained in a deterministic way. Assuming that the bias is the unique external input of each neuron (i.e., $\mathbb{E}\left[\mathcal{I}_{i,j}^{(l)}\right] = \mathbb{E}\left[\left(\mathcal{I}_{i,j}^{(l)}\right)^2\right] = 1$) and the previous results, we obtain the following weight initialization scheme:

$$\begin{aligned} \mathbf{W}_{i,j}^{(1)} &= \frac{1.3}{\sqrt{2}} \nu_{i,j} \\ \mathbf{W}_{i,j}^{(2)} &= \frac{1.3}{\sqrt{1 + 0.3\zeta^{(1)}}} \nu_{i,j} \end{aligned} \tag{7}$$

The SCAWI method is summarized in Alg. 1.

Algorithm 1 The SCAWI method

```
Input: the number of hidden neurons $\zeta^{(1)}$
    1) choose uniformly distributed random numbers $\nu_{i,j}$ in $[-1, 1]$ and use Eq. (7) to initialize $\mathbf{W}_{i,j}^{(1)}$ and $\mathbf{W}_{i,j}^{(2)}$
Output: $\mathbf{W}_{i,j}^{(1)}$ and $\mathbf{W}_{i,j}^{(2)}$
```

2) Weight Initialization for Parametric Estimation (WIPE) [32]: The WIPE method is a weight initialization approach for single-hidden-layer FNNs that applies the *singular value decomposition*[12] (SVD) on the input data matrix considering the Taylor expansion of the mapping problem and the nonlinearity of sigmoids. Specifically, the WIPE method takes into account Assumption 1.

Assumption 1: Given a vector $\boldsymbol{x}$, $\xi^{(1)}(\boldsymbol{x}) = \xi^{(2)}(\boldsymbol{x}) \approx \boldsymbol{x}$. *Moreover,* $\mathbf{W}_{0,\cdot}^{(1)} = \mathbf{0}_{\zeta^{(1)}}^\top$.

Assumption 1 states that the nonlinearities $\xi^{(1)}(\cdot)$ and $\xi^{(2)}(\cdot)$ approach the identity function and the biases for the hidden neurons are zero. Let $\mathbf{X} := [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be the data matrix and $\mathbf{Y} := [\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n] \in \mathbb{R}^{c \times n}$ be the desired output matrix for the training sample $\mathbb{S}_{\mathcal{X}\mathcal{Y}}$. Let us define $\mathbf{W}_2 = \left(\mathbf{W}_{1:\zeta^{(1)}+1}^{(2)}\right)^\top$, $\mathbf{W}_1 = \left(\mathbf{W}_{1:\zeta^{(0)}+1}^{(1)}\right)^\top$, $\boldsymbol{b}_1 = \left(\mathbf{W}_{0,\cdot}^{(1)}\right)^\top$, and $\boldsymbol{b}_2 = \left(\mathbf{W}_{0,\cdot}^{(2)}\right)^\top$. Assuming that the network output is equivalent to $\mathbf{Y}$ on the training sample, we can write:

$$\mathbf{Y} = \xi^{(2)}\left(\mathbf{W}_2 \xi^{(1)}\left(\mathbf{W}_1 \mathbf{X} + \boldsymbol{b}_1 \mathbf{1}_{\zeta^{(1)}}^\top\right) + \boldsymbol{b}_2 \mathbf{1}_{\zeta^{(2)}}^\top\right)$$

Note that $\zeta^{(2)} = c$ for single-hidden-layer FNNs. From Assumption 1, we have:

$$\mathbf{Y} = \mathbf{W}_2 \mathbf{W}_1 \mathbf{X} + \boldsymbol{b}_2 \mathbf{1}_c^\top$$

Therefore:

$$\mathbf{W}_2 \mathbf{W}_1 = \left(\mathbf{Y} - \boldsymbol{b}_2 \mathbf{1}_c^\top\right) \mathbf{X}^\dagger$$

where $\mathbf{X}^\dagger$ is the *Moore-Penrose pseudo-inverse*[13] of $\mathbf{X}$. Making an SVD of $\mathbf{X}$, we have $\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$, such that $\mathbf{U} \in \mathbb{R}^{d \times d}$, $\mathbf{V} \in \mathbb{R}^{n \times n}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times n}$. Note that:

[10]The statistical model proposed in [31] was specifically designed for hyperbolic tangent functions with $\omega = \kappa = 1$. However, one might probably generalize such a model for other activation functions as well.

[11]Let $\{X_1, \cdots, X_n\}$ be a sequence of independent and identically distributed random variables drawn from a distribution described by (μ, σ^2). Let $S_n := \frac{1}{n}\sum_{i=1}^n X_i$. By the law of large numbers, S_n converges in probability and almost surely to μ as $n \to +\infty$, i.e., $\lim_{n\to\infty} S_n = \mu$. The classical central limit theorem describes the size and the distributional form of the stochastic fluctuations around the deterministic number μ during this convergence.

[12]Given a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, the SVD of $\mathbf{M}$ is given by $\mathbf{M} := \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$, in which $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are *orthogonal* matrices (mathematically, $\mathbf{U}\mathbf{U}^\top = \mathbf{U}^\top\mathbf{U} = \mathbf{I}_m$ and $\mathbf{V}\mathbf{V}^\top = \mathbf{V}^\top\mathbf{V} = \mathbf{I}_n$). $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix with non-negative diagonal entries.

[13]In this case, $\mathbf{X}^\dagger = \mathbf{X}^\top\left(\mathbf{X}\mathbf{X}^\top\right)^{-1}$.

$$\begin{aligned}\mathbf{X}^\dagger &= \mathbf{X}^\top\left(\mathbf{X}\mathbf{X}^\top\right)^{-1}\\ &= \left(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top\right)^\top\left(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top\left(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top\right)^\top\right)^{-1}\\ &= \mathbf{V}\boldsymbol{\Sigma}^\top\mathbf{U}^\top\left(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top\mathbf{V}\boldsymbol{\Sigma}^\top\mathbf{U}^\top\right)^{-1}\\ &= \mathbf{V}\boldsymbol{\Sigma}^\top\mathbf{U}^\top\mathbf{U}\left(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^\top\right)^{-1}\mathbf{U}^\top = \mathbf{V}\boldsymbol{\Sigma}^\dagger\mathbf{U}^\top\end{aligned}$$

Retaining the first $\zeta^{(1)}$ columns of $\mathbf{U}$ and $\mathbf{V}$, and the first $\zeta^{(1)}$ rows and columns of $\boldsymbol{\Sigma}$, we obtain the matrices $\mathbf{U}_{\zeta^{(1)}} \in \mathbb{R}^{d\times\zeta^{(1)}}$, $\mathbf{V}_{\zeta^{(1)}} \in \mathbb{R}^{n\times\zeta^{(1)}}$, and $\boldsymbol{\Sigma}_{\zeta^{(1)}} \in \mathbb{R}^{\zeta^{(1)}\times\zeta^{(1)}}$. Then, $\mathbf{X}^\dagger \approx \mathbf{V}_{\zeta^{(1)}}\boldsymbol{\Sigma}^\dagger_{\zeta^{(1)}}\mathbf{U}^\top_{\zeta^{(1)}} = \mathbf{V}_{\zeta^{(1)}}\boldsymbol{\Sigma}^{-1}_{\zeta^{(1)}}\mathbf{U}^\top_{\zeta^{(1)}}$. Therefore, we have:

$$\mathbf{W}_2\mathbf{W}_1 = \left(\mathbf{Y} - \boldsymbol{b}_2\mathbf{1}_c^\top\right)\mathbf{V}_{\zeta^{(1)}}\boldsymbol{\Sigma}^{-1}_{\zeta^{(1)}}\mathbf{U}^\top_{\zeta^{(1)}}$$

A simple solution for this problem is given by:

$$\begin{aligned}\mathbf{W}_1 &= \mathbf{T}\boldsymbol{\Sigma}^{-1}_{\zeta^{(1)}}\mathbf{U}^\top_{\zeta^{(1)}}\\ \mathbf{W}_2 &= \left(\mathbf{Y} - \boldsymbol{b}_2\mathbf{1}_c^\top\right)\mathbf{V}_{\zeta^{(1)}}\mathbf{T}^{-1}\end{aligned} \tag{8}$$

where $\mathbf{T} \in \mathbb{R}^{\zeta^{(1)}\times\zeta^{(1)}}$ is an arbitrary non-singular, diagonal matrix. The WIPE method is summarized in Alg. 2.

```
Algorithm 2 The WIPE method
Input: the number of hidden neurons $\zeta^{(1)}$; the data matrix $\mathbf{X} \in \mathbb{R}^{d\times n}$; the desired output matrix $\mathbf{Y} \in \mathbb{R}^{c\times n}$; a non-singular, diagonal matrix $\mathbf{T} \in \mathbb{R}^{\zeta^{(1)}\times\zeta^{(1)}}$
    1) $\mathbf{W}^{(1)}_{0,\cdot} \leftarrow \mathbf{0}^\top_{\zeta^{(1)}}$
    2) generate random values for the bias vector $\boldsymbol{b}_2$
    3) compute $\mathbf{W}_1$ and $\mathbf{W}_2$ as in Eq. (8)
    4) $\mathbf{W}^{(2)}_{1:\zeta^{(1)}+1} \leftarrow \mathbf{W}_2^\top$, $\mathbf{W}^{(1)}_{1:\zeta^{(0)}+1} \leftarrow \mathbf{W}_1^\top$,
       $\mathbf{W}^{(2)}_{0,\cdot} \leftarrow \boldsymbol{b}_2^\top$
Output: $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$
```

3) Clustering-based Weight Initialization (CWI) [34]: The CWI method was developed for FNNs with shallow architectures and is subdivided as follows:

- **clustering procedure**. The clustering procedure generates a number of cluster sets that represent the training examples;
- **network construction**. The network construction algorithm will use the cluster sets previously generated to construct initial neural networks as a function of the *network complexity*[14];
- **network optimization**. The network optimization stage will select the best neural network with respect to the network complexity.

[14]Network complexity is a measure that takes into account the number of parameters comprised in the network. Larger networks have higher network complexities.

Since *clustering per class* usually provides better results than *global clustering* [34], we revise the CWI method based on the *clustering per class* approach. Consider $\mathcal{J} \in \mathbb{N}^*$ the total number of cluster sets and $\mathcal{J}_k \in \mathbb{N}^*$ the number of cluster sets of the k-th class. By definition, $\mathcal{J} = \sum_{k=1}^{c}\mathcal{J}_k$. Let $\boldsymbol{\mu}_i \in \mathbb{R}^d$ and $\boldsymbol{\sigma}_i \in \mathbb{R}^d$ be, respectively, the cluster center and standard deviation of the i-th cluster. Therefore, the i-th cluster is represented by $(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i)$.

In order to generate cluster sets from the training examples, we can use, for instance, *k-means clustering* or *learning vector quantization* (LVQ)[15]. By updating the cluster centers $\{\boldsymbol{\mu}_i\}_{i=1}^{\mathcal{J}}$, these LVQ-based algorithms tend to move the decision boundaries toward the Bayesian ones. To compute the appropriate hyperplanes, we select one particular cluster configuration emerging from the k-means clustering. Once the selection is performed, the cluster centers are eventually modified by means of a limited number of LVQ iterations. The midplanes of the cluster center pairs $\left(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j\right)$ are then identified as the hyperplanes representing the candidate hidden units. The equation of the midplane of $\left(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j\right)$ is given by:

$$\alpha\left\{\sum_{m=1}^{d}\left(\boldsymbol{\mu}_{i,m} - \boldsymbol{\mu}_{j,m}\right)\boldsymbol{x}_m - \frac{1}{2}\left(\|\boldsymbol{\mu}_i\|_2^2 - \|\boldsymbol{\mu}_j\|_2^2\right)\right\} = 0$$

where $\alpha \in \mathbb{R}_+^*$ is a scaling factor. The value of α is determined in such a way the node outputs become ϵ and $1-\epsilon$ if $\boldsymbol{x} = \boldsymbol{\mu}_i$ and $\boldsymbol{x} = \boldsymbol{\mu}_j$, respectively. This is true if:

$$\alpha = \frac{2\ln(\epsilon/(1-\epsilon))}{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|_2^2}$$

where $\epsilon \in [0,1]$ is a free parameter. $\mathcal{J}$ cluster centers yield $\mathcal{J}(\mathcal{J}-1)/2$ hyperplanes. However, many of these hyperplanes do not contribute to the *Voronoi diagram*[16]. For instance, the midplanes between cluster centers of the same class need not be taken into account, so that the number of hyperplanes to be generated is reduced to:

$$n_{\mathcal{H}} = \frac{1}{2}\left[\mathcal{J}(\mathcal{J}-1) - \sum_{k=1}^{c}\mathcal{J}_k(\mathcal{J}_k - 1)\right]$$

To eliminate more irrelevant hyperplanes, some heuristic is required. Suppose that $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$ are two cluster centers assigned to different classes, and that $\mathcal{N}_i$ and $\mathcal{N}_j$ examples of the evaluation sample are vector quantized to these centers. The midplane of the centers is retained only if there are enough examples $\mathcal{N}_{i,j}$ in cluster i having $\boldsymbol{\mu}_j$ as their second nearest cluster center, or if there are enough examples $\mathcal{N}_{j,i}$ in cluster j having $\boldsymbol{\mu}_i$ as their second cluster center, i.e., a hyperplane is generated if and only if:

[15]One of the widely used LVQ algorithms is the *Self Organizing Map* (SOM) [51]. Although there is no guarantee that LVQ-based approaches are optimal with respect to the classification task at hand and the network architecture, this approach usually gives satisfactory results.

[16]Given a set of 2-dimensional points and a subset of the plane that contains these points, the Voronoi diagram is defined as a partitioning of that subset into regions, such that for each point there is a corresponding region consisting of all points closer to that point than to any other.

$$\max\left(\frac{\mathcal{N}_{i,j}}{\mathcal{N}_i}, \frac{\mathcal{N}_{j,i}}{\mathcal{N}_j}\right) > \gamma$$

where $\gamma \in \mathbb{R}_+^*$ is a free parameter. Any combination comprising 1 to $n_{\mathcal{H}}$ of these hyperplanes can be considered by the network optimization algorithm as representing the squashing hidden layer part of an initial single-hidden-layer FNN.

The next step is to create *concentric hidden unit* candidates. Concentric hidden units are units whose input-output relation if of the form $y_i = \xi\left(\|\boldsymbol{x} - \boldsymbol{\mu}_i\|_{\boldsymbol{\Sigma}_i}\right)$ where $\|\cdot\|_{\boldsymbol{\Sigma}_i}$ is the norm with respect to a square matrix $\boldsymbol{\Sigma}_i$. For instance, we may use the Mahalanobis norm, which is defined as $\|\boldsymbol{x}\|_{\boldsymbol{\Sigma}_i} := \boldsymbol{x}^\top \boldsymbol{\Sigma}_i \boldsymbol{x}$ with $\boldsymbol{\Sigma}_i$ being the covariance matrix of the input vectors. In [34], the matrix $\boldsymbol{\Sigma}_i$ is assumed to be diagonal whose diagonal entries are the standard deviations, $\boldsymbol{\sigma}_i$, of the custer representation $(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i)$. Therefore, the input-output relation becomes:

$$y_i = \xi\left(\|\boldsymbol{x} - \boldsymbol{\mu}_i\|_{\boldsymbol{\sigma}_i}\right) = \xi\left(\sqrt{\sum_{m=1}^{d}\left(\frac{\boldsymbol{x}_m - \boldsymbol{\mu}_{i,m}}{\boldsymbol{\sigma}_{i,m}}\right)^2}\right)$$

For the network optimization phase, [34] provided a measure of complexity of an FNN given its network architecture and the training sample. Moreover, the authors developed a search algorithm that automatically determines, as a function of the network complexity, the single-hidden-layer FNN yielding the best classification performance on the training examples. Due to space limitations, we ommit additional comments on the CWI method.

C. *Discussions*

Based on our review, we observed the following:

- although the mathematical background of the SCAWI method may be hard to understand, we can easily implement such a method using Eq. (7). Moreover, SCAWI runs very fast;
- the WIPE method is mathematically simple. However, such a method has a strong assumption on the linearity of the activation function and has an SVD on the data matrix. Therefore, WIPE may be unfeasible for large data sets;
- the CWI method provides an interesting idea of weight initialization based on network optimization with respect to network complexity. However, such a method may be hard to use in practice and be unfeasible for large data sets.

IV. Open questions

Revising weight initialization methods for FNNs, we found open questions for future research. Due to space limitations, we discuss in this paper just some of these questions.

Question 1: How can we generalize the statistical model of SCAWI [31] to other activation functions?

The SCAWI method was specifically designed for hyperbolic tangent activation functions with $\kappa = \omega = 1$. Mathematically, $\xi(x) = \tanh(x)$. In order to generalize the SCAWI method for all hyperbolic tangent functions, we have to change Eq. (6) by:

$$\mathbb{E}\left[\left(\boldsymbol{\psi}_{\boldsymbol{x}}^{(l)}\right)_j\right] = \mathbb{E}\left[\omega^2 \tanh^2\left(\kappa\left(\boldsymbol{\pi}_{\boldsymbol{x}}^{(l)}\right)_j\right)\right]$$

Although the main idea is simple, solving this problem may be a hard task. Finding generalized solutions for the SCAWI method to other activation functions could generate better weight initialization methods for FNNs, being of great interest to practitioners.

Question 2: How can we provide a weight initialization method when there are missing labels in the training sample? In other words, how can we provide such a method for semi-supervised learning?

The field of semi-supervised learning (SSL) has attracted much attention in the last few years [28], [52]. Although there exist neural network models for SSL, the weight initialization process does not take into account the unlabeled examples. Providing semi-supervised weight initialization methods could increase the performance of neural networks on SSL.

V. Conclusion

In this paper, we provided an overview on widely used weight initialization methods for FNNs. Specifically, we revised the SCAWI, WIPE, and CWI methods, discussing their strenghts and weaknesses. Moreover, we discussed open questions on weight initialization methods for FNNs.

References

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[2] T. P. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, 1995.

[3] A. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, pp. 930–945, 1993.

[4] T. P. Chen and H. Chen, "Approximations of continuous functionals by neural networks with application to dynamic systems," *IEEE Transactions on Neural Networks*, vol. 4, no. 6, pp. 910–918, 1993.

[5] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183–192, 1989.

[6] M. Stinchcombe and H. White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," in *Proceedings of the International Joint Conference on Neural Networks*, 1989, pp. 613–617 vol.1.

[7] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.

[8] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993.

[9] T. P. Chen, H. Chen, and R. W. Liu, "Approximation capability in c$\left(\overline{R}^n\right)$ by multilayer feedforward networks and related problems," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 25–30, 1995.

[10] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the International Conference on Machine Learning*, 2006, pp. 161–168.

[11] E. Romero and J. M. Sopena, "Performing feature selection with multilayer perceptrons," *IEEE Transactions on Neural Networks*, vol. 19, no. 3, pp. 431–441, 2008.

[12] V. Sindhwani, S. Rakshit, D. Deodhare, D. Erdogmus, J. C. Principe, and P. Niyogi, "Feature selection in MLPs and SVMs based on maximum output information," *IEEE Transactions on Neural Networks*, vol. 15, no. 4, pp. 937–948, 2004.

[13] R. Setiono and H. Liu, "Neural-network feature selector," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 654–662, 1997.

[14] J.-B. Yang, K.-Q. Chen, and X.-P. Li, "Feature selection for MLP neural network: the use of random permutation of probabilistic outputs," *IEEE Transactions on Neural Networks*, vol. 20, no. 12, pp. 1911–1922, 2009.

[15] D. S. Chen and R. C. Jain, "A robust backpropagation learning algorithm for function approximation," *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 467–479, 1994.

[16] M.-L. Zhang and Z.-H. Zhou, "Multi-label neural networks with applications to functional genomics and text categorization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1338–1351, 2006.

[17] H. White, "Learning in artificial neural networks: a statistical perspective," *Neural Computation*, vol. 1, pp. 425–464, 1989.

[18] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.

[19] H. White, "Connectionists nonparametric regression: multilayer feedforward networks can learn arbitrary maps," *Neural Networks*, vol. 3, pp. 535–549, 1990.

[20] H. White and J. Wooldridge, *Some results on sieve estimation with dependent observations*. Cambridge University Press, 1989.

[21] A. Kolmogorov, "On the representation of continuos functions of many variables by superposition of continuos functions of one variable and addition," *Dokl. Nauk USSR*, vol. 114, pp. 953–956, 1957.

[22] R. Hecht-Nielsen, "Kolmogorov's mapping neural networks existence theorem," in *Proceedings of the International Joint Conference on Neural Networks*, 1987, pp. 11–14.

[23] V. Kurkova, "Kolmogorov's theorem and multilayer neural networks," *Neural Networks*, vol. 5, pp. 501–506, 1992.

[24] F.-L. Chung, S. Wang, Z. Deng, and D. Hu, "Catsmlp: Toward a robust and interpretable multilayer perceptron with sigmoid activation functions," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 36, no. 6, pp. 1319–1331, 2006.

[25] R. Golden, "A unified framework for connectionist systems," *Biological Cybernetics*, vol. 59, pp. 109–120, 1988.

[26] C. A. R. de Sousa, "Analysis of the backpropagation algorithm using linear algebra," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2012, pp. 7–14.

[27] C. A. R. de Sousa and G. E. A. P. A. Batista, "Robust multi-class graph transduction with higher order regularization," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 299–306.

[28] C. A. R. de Sousa, S. O. Rezende, and G. E. A. P. A. Batista, "Influence of graph construction on semi-supervised learning," in *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery on Databases (ECML/PKDD)*, 2013, pp. 160–175.

[29] C. A. R. de Sousa, V. M. A. Souza, and G. E. A. P. A. Batista, "Time series transductive classification on imbalanced data sets: an experimental study," in *Proceedings of the International Conference on Pattern Recognition (ICPR)*, 2014, pp. 3780–3785.

[30] C. A. R. de Sousa, "An overview on the gaussian fields and harmonic functions method for semi-supervised learning," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1439–1446.

[31] G. P. Drago and S. Ridella, "Statistically controlled activation weight initialization (SCAWI)," *IEEE Transactions on Neural Networks*, vol. 3, pp. 627–631, 1992.

[32] P. Costa and P. Larzabal, "Initialization of supervised training for parametric estimation," *Neural Processing Letters*, vol. 9, pp. 53–61, 1999.

[33] J. F. Shepansky, "Fast learning in artificial neural systems: multilayer perceptrons training using optimal estimation," in *Proceedings of the IEEE International Conference on Neural Networks*, 1988, pp. 465–472.

[34] N. Weymaere and J.-P. Martens, "On the initialization and optimization of multilayer perceptrons," *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 738–751, 1994.

[35] S. E. Fahlman, "An empirical study of learning speed in backpropagation networks," Carnegie Mellon University CMU-CS-88-162, Tech. Rep., 1988.

[36] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 1990, pp. 21–26.

[37] L. F. A. Wesseles and E. Barnard, "Avoiding false local minima by proper initialization of connections," *IEEE Transactions on Neural Networks*, vol. 5, pp. 899–905, 1992.

[38] E. G. W. Boers and H. Kuiper, "Biological metaphors and the design of modular artificial neural networks," Master's thesis, Leiden University Netherlands, 1992.

[39] Y. K. Kim and J. B. Ra, "Weight value intialization for improving training speed in back-propagation network," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 1991, pp. 2396–2401.

[40] J. Y. Yam and T. W. S. Chow, "Determining initial weights of feedforward neural networks based on least squares method," *Neural Processing Letters*, vol. 2, pp. 13–17, 1995.

[41] T. Denouex and R. Lengellé, "Initializing back propagation networks with prototypes," *Neural Networks*, vol. 6, pp. 351–363, 1993.

[42] J. Y. Yam and T. W. S. Chow, "A new method in determining the intial weights of feedforward neural networks," *Neurocomputing*, vol. 16, pp. 23–32, 1997.

[43] D. Erdogmus, O. Fontenla-Romero, J. C. Principe, A. Alonso-Betanzos, and E. Castillo, "Linear-least-squares initialization of multilayer perceptrons through backpropagation of the desired response," *IEEE Transactions on Neural Networks*, vol. 16, no. 2, pp. 325–337, 2005.

[44] J. Y. Yam and T. W. S. Chow, "A weight initialization method for improving training speed in feedforward neural network," *Neurocomputing*, vol. 30, pp. 219–232, 2000.

[45] J.-P. Martens, "A stochastically motivated random initialization of pattern classifying mlps," *Neural Processing Letters*, vol. 3, pp. 23–29, 1996.

[46] J. Y. Yam and T. W. S. Chow, "Feedforward networks training speed enhancement by optimal initialization of the synaptic coefficients," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 430–434, 2001.

[47] N. B. Karayiannis, "Accelerating the training of feedforward neural networks using generalized hebbian rules for initializing the internal representations," *IEEE Transactions on Neural Networks*, vol. 7, no. 2, pp. 419–426, 1996.

[48] G. Thimm and E. Fiesler, "High-order and multilayer perceptron initialization," *IEEE Transactions on Neural Networks*, vol. 8, pp. 349–359, 1997.

[49] J.-S. Pei, J. P. Wright, and A. W. Smyth, "Neural network initialization with prototypes - a case study in function approximation," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2005, pp. 1377–1382.

[50] S. P. Adam, D. A. Karras, G. D. Magoulas, and M. N. Vrahatis, "Solving the linear interval tolerance problem for weight initialization of neural networks," *Neural Networks*, vol. 54, pp. 17–37, 2014.

[51] T. Kohonen, "Self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[52] C. A. R. de Sousa, V. M. A. Souza, and G. E. A. P. A. Batista, "An experimental analysis on time series transductive classification on graphs," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 307–314.