A FAST DEEP STACKED NETWORK USING EXTREME LEARNING MACHINE

Bruno Légora Souza da Silva*, Fernando Kentaro Inaba*, Patrick Marques Ciarelli*

*Laboratório de Computadores e Sistemas Neurais (LabCISNE)
Universidade Federal do Espírito Santo, 29075-910
Vitória, ES, Brasil

Emails: bruno.l.silva@aluno.ufes.br, kentaro@ele.ufes.br, patrick.ciareli@ufes.br

Abstract— Access to large amounts of data is becoming more common, as well as the use of methods based on "deep" learning to obtain better results. However, using those techniques can lead to long training times. To deal with this problem, the Deep Stacked Network (DSN) was proposed, where several small modules are stacked to increase the model efficiency. However, this architecture suffers from some problems that increase its training time and the amount of memory required to store it. To deal with some of these problems, we propose in this paper a fast algorithm to train a DSN using Extreme Learning Machine (ELM). Experiments performed on many classification datasets showed that the proposed method achieves similar accuracy when compared with other techniques, with the advantage of training the network in less time and storing fewer parameters.

Keywords— Extreme Learning Machine, Deep Stacked Networks, Big Data, Classification

Resumo— O acesso a grandes quantidades de dados está mais comum, assim como o uso de métodos de aprendizado profundo em busca de obter análises cada vez melhores. Entretanto, o uso de tais técnicas pode acarretar em longos tempos de treinamento. Para lidar com este problema, uma arquitetura chamada *Deep Stacked Network* (DSN) foi proposta, onde vários módulos pequenos são empilhados para melhorar sua eficiência. Porém, esta arquitetura sofre com alguns problemas que aumentam seu tempo de treino e a quantidade de memória requerida para armazená-lo. Para lidar com estes problemas, neste artigo é proposto um algoritmo para treinar uma DSN baseado em Extreme Learning Machine (ELM). Experimentos em várias bases de dados de problemas de classificação mostraram que o método proposto atinge métricas similares quando comparadas a outras técnicas, com a vantagem de reduzir o tempo de treinamento e o número de parâmetros armazenados.

Palavras-chave— Extreme Learning Machine, Deep Stacked Network, Big Data, Classificação

1 Introduction

In recent years, large amounts of data are produced each year by a great variety of applications, and it is becoming easier to access them (Kaisler et al., 2013). To process all these data, the use of "deep learning" techniques are becoming common, where many layers (stages) of information processing in hierarchical architectures are used to extract complex structures and build internal representations of a dataset (Deng and Yu, 2013).

However, these "deep" architectures have many layers that must be trained at the same time, which implies that their training has a high computational and memory cost. Furthermore, the large volume of available data makes the training process very difficult (scalability problem) (Deng and Yu, 2011).

To overcome these problems, Deng and Yu (2011) proposed the Deep Stacking Network (DSN), where smaller modules are trained and "stacked" on top of others. The concept of DSN does not specify the training algorithm and type of each module, and thus fast algorithms, such as Extreme Learning Machine (ELM) can be used.

ELM (Huang et al., 2004; Huang et al., 2006; Huang et al., 2012) is an algorithm to train Single-Layer Feedforward Networks (SLFNs). Recently, ELM gained significant attention by researchers due to its low computational cost and good generalization performance (Huang et al., 2012).

The main idea of ELM is to randomly generate part of the SLFN parameters and find the remaining, using a closed-form solution (Huang et al., 2004). However, the number of hidden neurons of the SLFN needs to be well-defined to avoid problems such as underfitting or overfitting. The effects of this problem can be reduced by using regularization, such as proposed by Deng et al. (2009), but it still can suffer from the limitation of memory and intense computational cost of large matrices inversion (Inaba et al., 2018).

Zhou et al. (2014) proposed the Stacked Extreme Learning Machine (S-ELM) which applies the concept of "stacking" in the context of ELM. This algorithm stacks modules trained with ELM and uses Principal Component Analysis (PCA) to reduce dimensionality and "send" relevant information to the next modules (Zhou et al., 2014).

Compared with a common "deep learning" model, S-ELM or DSNs trained with ELM have a low computational cost. However, they still need to store a considerable amount of parameters, which can be a problem if they are used in an environment with restricted resources.

In this context, our main contribution is a fast algorithm to train a DSN using modules trained with ELM. We propose to retain part of the calculations of one module and using it in the following. As a result, the proposed algorithm trains networks with reduced training time, memory requirement, and model size (number of stored pa-

rameters), while keeping similar results when compared to other methods trained using ELM.

The remainder of this paper is organized as follows. In Section 2, some brief literature review is presented, where shallow and deep models are introduced. In Section 3, the proposed algorithm is described. The experiments to evaluate the proposed method and the respective results are discussed in Section 4. Finally, the conclusion and future paths are presented in Section 5.

2 Literature Review

According to Deng and Yu (2013), until recently, most of the works related to machine learning and pattern recognition exploited architectures considered "shallow", where a single layer of non-linear transformation is used. Some examples of "shallow" architectures are SLFN, Support Vector Machine and Hidden Markov Models.

These methods transform the input data into a feature space, which may be unobservable. Despite the good results in simple or well-behaved problems, these methods suffer in more difficult tasks (e.g. speech, sound and image processing) since they cannot extract complex structures and features from the data (Deng and Yu, 2013).

To extract more complex features from the data, a "deep" model is preferred, where several non-linear layers are used. Historically, the concept of "deep" learning originated in neural network research, so the term "deep learning" usually refers to methods similar to Multi-Layer Perceptron with many layers, also called Deep Neural Networks (DNNs) (Deng and Yu, 2013).

To train an SLFN, which is a "shallow" model, we can use algorithms based on gradient descent (GD) or methods based on the ELM (Huang et al., 2006) algorithm, which is usually much faster and can obtain similar generalization performance, when compared to GD-based methods.

However, it is not possible to use the original ELM (Huang et al., 2006) algorithm to train a neural network with many layers ("deep" model), therefore, usually, GD-based methods are used. Nevertheless, since the model has many layers, this training process usually takes much more time than a "shallow" model, but better results are usually obtained (Deng and Yu, 2013).

Another problem with these methods is their training time and the memory requirement, especially when these models are trained using large datasets, which is common for most recent applications (Bottou and LeCun, 2004). The effects of this problem in the training stage can be reduced if the Stochastic Gradient Descent (SGD) method is used (Deng and Yu, 2013), which takes subsets of the training data in each one of its iterations.

"Deep learning" models have been successfully applied to many applications, such as speech recognition (Deng and Yu, 2011), semantic segmentation (Long et al., 2015) and text localization in real-world images (Zhu and Zanibbi, 2016) due to its ability of extracting more complex structures and features from the used dataset.

However, the high computational cost and the great memory usage can discourage researchers from using this type of model. Thus, training a "deep" architecture using algorithms with reduced computational time and memory requirement, such as the ELM, can be preferable.

In the following sections, we consider that a classifier is trained with a generic classification dataset, composed of N pairs (samples) $(\mathbf{x}_i, \mathbf{t}_i)$, $i = 1, \ldots, N$, where $\mathbf{x}_j$ is a vector with d features and $\mathbf{t}_i$ is a binary vector with m dimensions, where m is the number of targets (classes). All $\mathbf{x}_i$ and $\mathbf{t}_i$ row vectors can be used to construct the $\mathbf{X} = [\mathbf{x}_1^\mathsf{T}, \ldots, \mathbf{x}_N^\mathsf{T}]^\mathsf{T}$ and $\mathbf{T} = [\mathbf{t}_1^\mathsf{T}, \ldots, \mathbf{t}_N^\mathsf{T}]^\mathsf{T}$ matrices, respectively, where $(\cdot)^\mathsf{T}$ indicates transposition.

2.1 Extreme Learning Machine

ELM is a fast learning algorithm used to train SLFNs, where it is not necessary to tune all the network weights: some of them are randomly generated and the remaining are calculated using a closed-form solution.

The output function of an SLFN with L hidden neurons and one output node ($m = 1$, the matrix $\mathbf{T}$ changes into a vector $\mathbf{t}$) is given by

$$f(\mathbf{x}) = \sum_{i=1}^{L} \beta_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta}, \tag{1}$$

where $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_L]^\mathsf{T}$ is the weight vector between the hidden layer and the output node, $\mathbf{h}(\mathbf{x}_i) = [h(\mathbf{x}_i \cdot \mathbf{w}_1 + \nu_1), \ldots, h(\mathbf{x}_i \cdot \mathbf{w}_L + \nu_L)]$ is the hidden layer output with respect to the input sample $\mathbf{x}_i$.

The original ELM goal (Huang et al., 2006) is to obtain the solution ($\boldsymbol{\beta}_{\boldsymbol{opt}}$) which minimizes the ℓ_2 norm of the training error, i.e.,

$$\boldsymbol{\beta}_{\boldsymbol{opt}} = \arg\min_{\boldsymbol{\beta}} ||\boldsymbol{H}\boldsymbol{\beta} - \mathbf{t}||_2^2, \tag{2}$$

where $\mathbf{t}$ is the vector of target values and $\mathbf{H}$ is the hidden layer output matrix given by

$$\mathbf{H} = \begin{bmatrix} h(\mathbf{x}_1 \cdot \mathbf{w}_1 + \nu_1) & \ldots & h(\mathbf{x}_1 \cdot \mathbf{w}_L + \nu_L) \\ \vdots & \ddots & \vdots \\ h(\mathbf{x}_N \cdot \mathbf{w}_1 + \nu_1) & \ldots & h(\mathbf{x}_N \cdot \mathbf{w}_L + \nu_L) \end{bmatrix} \tag{3}$$

where the weights $\mathbf{w}_j$ (between the input and the j-th hidden neuron) and bias ν_j are randomly generated according to any continuous probability distribution, and $h(\cdot)$ can be any nonlinear piecewise continuous function satisfying the ELM theorem (Huang et al., 2006). Each row i of $\mathbf{H}$ corresponds to the vector $\mathbf{h}(\mathbf{x}_i)$.

The optimization problem presented in Eq. (2) has a closed-form solution, which is given by

$$\boldsymbol{\beta_{opt}} = \mathbf{H}^{\dagger}\mathbf{t}, \tag{4}$$

where $\mathbf{H}^{\dagger} = (\mathbf{H}^{\mathsf{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathsf{T}}$ is the Moore-Penrose generalized inverse of the matrix $\mathbf{H}$. Finally, the output function of an SLFN trained using ELM is given by

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta_{opt}} = \mathbf{h}(\mathbf{x})\mathbf{H}^{\dagger}\mathbf{t}, \tag{5}$$

where $\mathbf{x}$ corresponds to the input sample.

One problem of the original ELM algorithm is that it tends to overfit if the number of hidden neurons is not properly chosen. To avoid this problem, Regularized ELM (R-ELM) was proposed by Deng et al. (2009). R-ELM adds a norm of the output weights in the objective function since it is a known practice to achieve better generalization (Bartlett, 1998).

The objective function of R-ELM is composed of a sum of the weights model ℓ_2 norm and the residual error ℓ_2 norm with a regularization parameter C. The R-ELM algorithm finds the solution $\boldsymbol{\beta_{opt}}$ that minimizes its objective function:

$$\boldsymbol{\beta_{opt}} = \arg\min_{\boldsymbol{\beta}} C||\boldsymbol{H\beta} - \mathbf{t}||_2^2 + ||\boldsymbol{\beta}||_2^2. \tag{6}$$

According to Deng et al. (2009), this problem also has a closed-form solution, given by

$$\boldsymbol{\beta_{opt}} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^{\mathsf{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathsf{T}}\mathbf{t}. \tag{7}$$

These analyses can be extended to multiple output problems, where $\boldsymbol{\beta}$ and $\mathbf{t}$ change into matrices $\mathbf{B}$ and $\mathbf{T}$, respectively, where each column of $\mathbf{B}$ can be calculated using Eqs. (4) or (7) and the correspondent column of $\mathbf{T}$.

2.2 Deep Stacked Networks

While DNNs have shown good results in many tasks, its training stage has proven to be computationally intense (Deng and Yu, 2013), and usually, multiple CPUs and/or GPUs are used to speed up this stage (Zhou et al., 2014).

One of the main reasons of this expensive training of a DNN is that it usually has many layers, which means that the network has many parameters that need to be tuned at the same time using an algorithm like SGD. This problem scales with the usual large amount of data that is used to train these networks.

To deal with this scalability problem, Deng and Yu (2011) proposed a new deep learning architecture, called Deep Stacking Network (DSN). The central idea of this method is the concept of stacking (Wolpert, 1992), where simple modules of functions or classifiers are "stacked" on top of each other in order to learn complex representations.

A DSN, as proposed by Deng and Yu (2011), includes a number of modules S (a hyperparameter). Each module can be an SLFN, with two sets of trainable weights (and one set of bias) and non-linear hidden units. In each module k, the input is mapped by a weight matrix $\mathbf{W}_k$ followed by a non-linear function (e.g. sigmoidal). These outputs of the hidden layer can be mapped to linear units by a second matrix, $\mathbf{U}_k$. To the best of our knowledge, ELM was not used to train the modules of a DSN, at least in its typical approach, although it was suggested by Deng and Yu (2011). In this case, $\mathbf{W}_k$ is randomly generated and each column of $\mathbf{U}_k$ is calculated as $\boldsymbol{\beta_{opt}}$ in Eq. (4) or Eq. (7). An example of a DSN architecture is shown in Fig. 1.

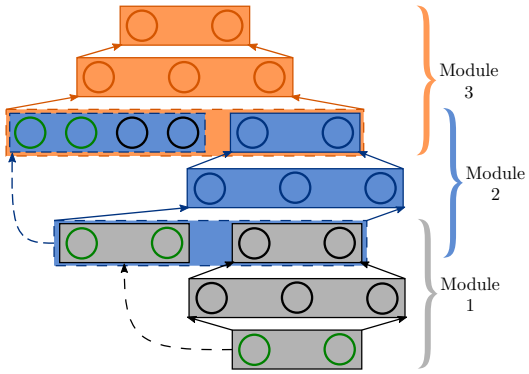


Figure 1: A typical DSN architecture. Rectangles with the same color represent layers of the same module, and green circles the original input data.

The lowest (first) module of a DSN is trained in the same way a typical SLFN is, where the raw input data $\mathbf{X}_1 = \mathbf{X}$ is mapped to an estimated output $\hat{\mathbf{T}}_1$. The remaining modules inputs are "constructed" by concatenating the former module input with its estimated output. This means that the second module input is: $\mathbf{X}_2 = [\mathbf{X}_1,\ \hat{\mathbf{T}}_1]$. This can be extended to any module (except the first):

$$\mathbf{X}_k = [\mathbf{X}_{k-1},\ \hat{\mathbf{T}}_{k-1}] \tag{8}$$

Using these "constructed" inputs, the modules can be trained serially. By stacking them, it is possible to achieve results similar to deep networks with reduced training time (Deng and Yu, 2011).

2.3 Stacked Extreme Learning Machines

As presented in Section 2.1, the ELM algorithm is used to train an SLFN model with very fast learning speed, good generalization ability and ease of implementation (Zhou et al., 2014). However, as pointed by Zhou et al. (2014), the accuracy obtained in some datasets (such as MNIST, which has 60000 samples with 768 dimensions) was limited by hardware, since it could be improved by adding more neurons, but their computer could not handle.

Following the "stacking" concept, the S-ELM algorithm was proposed by Zhou et al. (2014). Note that, although S-ELM has a similar architecture of a DSN, an important difference exists: instead of using a module output to construct the following module input, it uses a "projection" of its hidden layer with reduced dimension (using PCA) as part of the hidden layer of the next module.

The first module of S-ELM is trained using the R-ELM algorithm, where the weight matrix $\mathbf{B}_1$ between hidden and output layers is calculated by extending Eq. (7). PCA is used in $\mathbf{B}_1^\mathsf{T}$, where the eigenvalues and eigenvectors of its covariance matrix are calculated. To reduce the hidden layer dimension, L' ($L' < L$) eigenvectors, with respect to the L' largest eigenvalues, are used to compose a matrix $\tilde{\mathbf{V}}_1$. Then, the hidden layer output $\mathbf{H}_1$ is "projected" using the calculated $\tilde{\mathbf{V}}_1$:

$$\mathbf{H}_1^{'} = \mathbf{H}_1 \tilde{\mathbf{V}}_1. \tag{9}$$

Therefore, the hidden layer output of the following module is constructed using

$$\mathbf{H}_i = [\mathbf{H}_{i-1}^{'},\ \mathbf{H}_{new}] \tag{10}$$

where $\mathbf{H}_{new}$ is a matrix of new generated random nodes. The same process is repeated until S modules are trained and stacked, where the dimensionality reduction is used in each module, with exception of the network last module, since it needs to return an estimated output. An example of the S-ELM architecture is shown in Fig. 2.

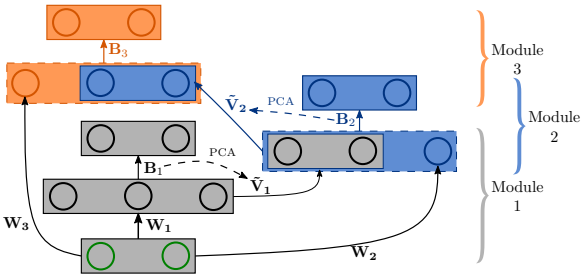


Figure 2: An example of the S-ELM architecture. Rectangles with the same color represent layers of the same module, and green circles represent the original input data. Note that it is not necessary to store the $\mathbf{B}$ matrix of all modules. Every arrow means a matrix multiplication, while a dashed arrow means a PCA dimensionality reduction.

A variation of this S-ELM method, called Autoencoder Stacked Extreme Learning Machine (AE-S-ELM) is also proposed by Zhou et al. (2014). The main difference of this method, when compared to the S-ELM, is that instead of using randomly generated weights in each module, the AE-S-ELM method constructs an "auxiliary" autoencoder SLFN with random weights, and find its matrix $\mathbf{B}_{aux}$ using $\mathbf{X}$ as both input and output of this network. $\mathbf{B}_{aux}^\mathsf{T}$ is used as weights between the input and hidden layers of the AE-S-ELM module. This procedure is used every time the algorithm needs to generate random nodes.

It was observed by Zhou et al. (2014) that both techniques showed good results in known datasets when compared to other methods. S-ELM and AE-S-ELM obtained similar results, but usually the latter needed smaller architectures (fewer neurons in the hidden layer of each module and/or less number of modules) to achieve similar accuracy to the former method. Although sometimes the training time of these techniques was greater than the time of a large SLFN, less memory was used, which allows the use of these techniques in systems with limited resources.

3 The Proposed Method: FDSN-ELM

3.1 Problems in training stacked networks using ELM

Although it has been shown that methods such as DSN (Deng and Yu, 2011; Deng and Yu, 2013) and S-ELM (Zhou et al., 2014), based on the stacking concept (Wolpert, 1992), were capable of improving the results (such as the accuracy metric when new modules are added to the network) in public datasets, they still have some drawbacks.

The DSN method is composed of stacking some kind of classifier, and each module has a parameter set that needs to be stored. Using SLFN in each module, it is needed to store the weights between the hidden and output layers, the biases and the weights between the input and hidden layers, whose size grows along the modules, as explained in Section 2.2. This implies that the training time and memory usage is increased along the modules, which might be prohibited sometimes.

The S-ELM method (and its variant AE-S-ELM) also stacks modules of SLFN with L hidden neurons. However, it does not have the growing size problem of DSN. Since it does not predict an output in each module, a reduced projection of the matrix $\mathbf{H}_i$ ($\mathbf{H}_i^{'}$) is used in each module i, except in the last one, and, therefore, there is no need to store the $\mathbf{B}_i \in \mathcal{R}^{L \times m}$ matrices, only in the last module. Nevertheless, it needs to store the matrix $\tilde{\mathbf{V}}_i \in \mathcal{R}^{L \times L^{'}}$ of each module i, where $L^{'}$ is the number of remaining nodes after the PCA reduction, which usually is larger than m.

Even with the need of storing a usually larger matrix ($\tilde{\mathbf{V}}_i$ instead of $\mathbf{B}_i$) in each module, the S-ELM method still tends to need less memory when compared with a DSN composed of SLFN modules trained with the R-ELM algorithm (DSN-ELM). However, it still needs to reduce dimension with PCA in every module.

Another limitation of S-ELM method is that it cannot be used when the dataset has one dimensional target ($m = 1$) (specially when it is a

regression problem, considering that a binary classification can be modeled using two output nodes), since the PCA is calculated using $\mathbf{B}^{\mathsf{T}} \in \mathcal{R}^{1\times L}$ of every module, which is not possible.

3.2 FDSN-ELM

The algorithm proposed in this paper, called Fast Deep Stacked Network (FDSN-ELM), is a fast method to train a DSN with SLFN modules trained using the R-ELM algorithm. Our method deals with two of the main problems of the DSN-ELM model: the quantity of memory it needs to be stored and the time needed to train it. The architecture used is the same shown in Figure 1.

In FDSN-ELM, the first module is trained using the same method employed to train a SLFN with R-ELM algorithm: a set of random weights $\mathbf{W}_1$ and biases $\mathbf{N}_1$ are generated, the matrix $\mathbf{H}_1$ is calculated as in Eq. (3), which can be rewritten as

$$\mathbf{H}_1 = \mathbf{G}(\boldsymbol{\eta}_1), \tag{11}$$

where $\mathbf{G}$ is an operator that applies the activation function $h(\cdot)$ in every element of the matrix $\boldsymbol{\eta}_1$, which is defined as

$$\boldsymbol{\eta}_1 = \mathbf{X}_1\mathbf{W}_1 + \mathbf{N}_1, \tag{12}$$

where $\mathbf{X}_1$ is the input matrix of the first module (samples of the dataset), and each row of the matrix $\mathbf{N}_1$ is the $\boldsymbol{\nu}_1$ vector. Finally, $\mathbf{B}_1$ is calculated using $\mathbf{H}_1$ by extending Eq. (7). Therefore, the first module output is given by:

$$\hat{\mathbf{T}}_1 = \mathbf{H}_1\mathbf{B}_1. \tag{13}$$

The input of the second module is constructed by concatenating the matrix $\mathbf{X}_1$ with $\hat{\mathbf{T}}_1$:

$$\mathbf{X}_2 = [\mathbf{X}_1,\ \hat{\mathbf{T}}_1], \tag{14}$$

which will be used to calculate the hidden layer output of this module. This can be written using the notation of Eqs. (11) and (12):

$$\mathbf{H}_2 = \mathbf{G}(\boldsymbol{\eta}_2), \tag{15}$$

where $\boldsymbol{\eta}_2$ is given by:

$$\boldsymbol{\eta}_2 = \mathbf{X}_2\mathbf{W}_2 + \mathbf{N}_2. \tag{16}$$

Eq. (16) can be rewritten as:

$$\boldsymbol{\eta}_2 = [\mathbf{X}_1,\ \hat{\mathbf{T}}_1]\begin{bmatrix}\mathbf{W}_2^{(1)}\\ \mathbf{W}_2^{(2)}\end{bmatrix} + \mathbf{N}_2, \tag{17}$$

where $\mathbf{W}_2^{(1)}$ and $\mathbf{W}_2^{(2)}$ are submatrices of $\mathbf{W}_2$. Then, we can write:

$$\boldsymbol{\eta}_2 = \mathbf{X}_1\mathbf{W}_2^{(1)} + \hat{\mathbf{T}}_1\mathbf{W}_2^{(2)} + \mathbf{N}_2. \tag{18}$$

We can observe in Eq. (18) that the calculation of $\boldsymbol{\eta}_i$ is similar to $\boldsymbol{\eta}_{i-1}$ in Eq. (12), but with a new term $\hat{\mathbf{T}}_{i-1}\mathbf{W}_i^{(i)}$. As pointed in Section 2.2, this increases the time to train each module (and the number of model parameters), since it needs to deal with matrices of bigger dimensions when training a module that uses R-ELM.

To speed up the training process and reduce the memory requirements, instead of using new values for $\mathbf{W}_2^{(1)}$ and $\mathbf{N}_2$, FDSN-ELM considers the following identities (in the second module):

$$\mathbf{W}_2^{(1)} = \mathbf{W}_1, \tag{19}$$

and

$$\mathbf{N}_2 = \mathbf{N}_1. \tag{20}$$

Then, Eq. (18) can be rewritten as:

$$\boldsymbol{\eta}_2 = \mathbf{X}_1\mathbf{W}_1 + \hat{\mathbf{T}}_1\mathbf{W}_2^{(2)} + \mathbf{N}_1, \tag{21}$$

and by replacing Eq. (12) in Eq. (21), we obtain:

$$\boldsymbol{\eta}_2 = \boldsymbol{\eta}_1 + \hat{\mathbf{T}}_1\mathbf{W}_2^{(2)}, \tag{22}$$

where $\mathbf{W}_2^{(2)}$ is a randomly generated matrix.

Then, by using $\mathbf{H}_2$, $\mathbf{B}_2$ is calculated by extending Eq. (7). The second module output is:

$$\hat{\mathbf{T}}_2 = \mathbf{H}_2\mathbf{B}_2. \tag{23}$$

The method to train the second module described above can be extended to train all SLFNs modules of a DSN network.

There are two main advantages of assuming Eqs. (19) and (20). The first is that some computational operations of the i-th module training stage are done in the former one (the matrix $\boldsymbol{\eta}_i$ is calculated using $\boldsymbol{\eta}_{i-1}$), implying that their training time is reduced, when compared to a common SLFN. The second is that each module i needs to store only the matrix $\mathbf{W}_i^{(i)}$ since it is the only new matrix needed to calculate the new hidden layer output, which implies that each module needs to store a reduced number of parameters.

3.3 Size of a FDSN-ELM network

We can compare the model size (number of parameters or bytes it needs to be stored) of FDSN-ELM and similar techniques. An SLFN needs to store $\mathbf{W}$ and $\mathbf{B}$ matrices, of dimensions $d \times L$ and $L \times m$, respectively, and the vector $\boldsymbol{\nu}$, with L elements. Then, the total memory needed to store the SLFN model is: $Z_{slfn} = L(d + m + 1)$.

An S-ELM (and AE-S-ELM) network stores in each module a $\tilde{\mathbf{V}}_i$ matrix, of dimension $L \times L^{'}$, instead of $\mathbf{B}_i$ (except in the last module), $\mathbf{W}_{new}$ and $\boldsymbol{\nu}_{new}$ corresponding to the $L - L^{'}$ "missing" nodes. It also stores $\mathbf{W}_1$ (and $\boldsymbol{\nu}_1$) and $\mathbf{B}_S$ of the first and last modules, respectively. Hence, the memory needed to store a S-ELM model is: $Z_{selm} = (d + m + 1)\, L + (S-1)(LL^{'} + Ld - L^{'}d + L - L^{'})$.

A DSN-ELM consists of an SLFN in each module, with d increasing along the modules, following the rule $d_i = d+mi$. Then, the total memory needed to store this model is:

$$Z_{dsn-elm} = L\,(d+m+1) + L\left(\sum_{i=1}^{S-1}(1+m+(d+mi))\right).$$

A network trained using the proposed algorithm, in the first module, needs to store W_1, of size $d \times L$, $\boldsymbol{\nu}$ with L elements and $\mathbf{B}_1$, of size $L \times m$, and in the remaining modules, $W_i^{(2)}$ and $\mathbf{B}_i$, of sizes $m \times L$ and $L \times m$, respectively. Then, the total memory needed to store this model is: $Z_{fdsn-elm} = (d+m+1)L + 2(S-1)mL$.

Comparing the number of parameters in each model using the stacking principle, and considering that usually $m < d$, we can observe that a model trained with FDSN-ELM needs less memory than the other considered techniques.

4 Experiments and Discussions

In this section, experiments were carried out to test our proposed algorithm, comparing it with similar techniques. The experiments were conducted using MATLAB on a computer with Intel Core i5-2500, 8GB of RAM and Ubuntu 16.04.

4.1 Datasets

The datasets used in the experiments are shown in Table 1. If a dataset does not give a test set, its samples are randomly divided into train and test sets with roughly the same number of samples.

Table 1: Datasets used in the experiment. The columns indicate the number of training and testing samples, features (d) and classes (m).

Dataset	#Tr. Data	#Te. Data	#Feat.	#Clas.
IRIS	75	75	4	3
MNIST	60000	10000	784	10
SatImage	4435	2000	36	6
Segment	1155	1155	19	7
USPS	7291	2007	256	10
Vowel	528	462	10	11
Wine	89	89	13	3

The features of these datasets were all normalized between the interval $[-1, 1]$, using the minimum and maximum values found in the training subset. Testing samples were also normalized using these values. For every sample $\mathbf{x_i}$ of the k-th class, its correspondent target $\mathbf{t}_i$ is a vector with m dimensions, where all elements equal to zero, except for the k-th element, which is equal to 1.

4.2 Experiment

In this experiment, we compare the algorithms presented in Table 2[1]. The purpose of this experiment is to evaluate if the accuracy metric can be improved by stacking modules. We compare the values obtained using SLFNs with $L = 100$ and $L = 3000$ hidden nodes (this number was chosen due to memory limitations on the MNIST dataset). In this experiment, all "stacking" algorithms were tested using $S = 30$ modules with $L = 100$ neurons each, and in the S-ELM and AE-S-ELM algorithms, the hidden layer dimension was reduced to $L^{'} = 50$ neurons using PCA.

Table 2: Algorithms tested in this experiment.

Algor.	Params			Algor.	Params		
	L	S	L'		L	S	L'
RELM	100	-	-	Proposed	100	30	-
RELM	3000	-	-	SELM	100	30	50
DSNELM	100	30	-	AESELM	100	30	50

In this experiment, we are concerned if the "stacking" techniques are capable of improving the results obtained with a small SLFN, if the result is comparable to a larger SLFN and how much time and memory it costs to achieve these results. Thus, we give the same regularization parameter C to all algorithms, which is defined by doing a 5-fold cross-validation on the training set using the SLFN with 100 neurons. We test the values $\left[2^{-10}, 2^{-9}, \cdots, 2^{9}, 2^{10}\right]$ and the one which returns the best accuracy is chosen. The sigmoid activation function was used in all techniques.

The tests were repeated 100 times and its mean and standard deviation are presented in Tables 3, 4 and 5 for testing accuracy, training time and memory used, respectively, where the best results (excluding RELM-100) were highlighted. Table 5 also presents the regularization parameter found in the 5-fold method for each dataset.

The results showed that, usually, training stacked models is faster than a large SLFN model, achieving similar accuracies. The proposed technique, FDSN-ELM, was the fastest and required the smallest amount of memory among the stacking models (in many cases the amount of memory was more than 3 times smaller). For most datasets, it was also faster and consumed less memory than R-ELM-3000, just RELM-100 achieved better results in these parameters. Analyzing only accuracies, the proposed technique achieved similar results (and overcame RELM-100) in almost all datasets, except in MNIST and USPS, where both DSN-ELM and FDSN-ELM showed inferior results. Since they use the same architecture and both datasets samples are (almost) binary images of digits, the results could imply that this type of architecture does not deal appropriately with (almost) binary inputs.

Figures 3a and 3b shows the effect of inserting new modules on the tested algorithms in Segment Dataset. In this particular set, the insertion of new modules improved the accuracy of all methods, showing that stacking modules is a

[1]Implementations of these methods are available in https://github.com/labcisne/elmtoolbox

Table 3: Test accuracy obtained using the methods of Table 2 in Table 1 datasets.

	R-ELM-100	R-ELM-3000	DSN-ELM	Proposed	S-ELM	AE-S-ELM
IRIS	0.929 ± 0.012	0.923 ± 0.005	0.969 ± 0.006	$\mathbf{0.971 \pm 0.005}$	0.929 ± 0.006	0.912 ± 0.018
MNIST	0.751 ± 0.013	$\mathbf{0.954 \pm 0.001}$	0.766 ± 0.009	0.759 ± 0.012	0.926 ± 0.001	0.940 ± 0.001
SatImage	0.857 ± 0.004	0.867 ± 0.005	0.893 ± 0.003	0.887 ± 0.005	$\mathbf{0.894 \pm 0.003}$	$\mathbf{0.894 \pm 0.003}$
Segment	0.933 ± 0.004	0.951 ± 0.003	$\mathbf{0.961 \pm 0.003}$	0.959 ± 0.001	0.953 ± 0.002	0.948 ± 0.002
USPS	0.862 ± 0.006	$\mathbf{0.945 \pm 0.002}$	0.888 ± 0.006	0.859 ± 0.007	0.937 ± 0.002	0.935 ± 0.003
Vowel	0.480 ± 0.030	0.418 ± 0.009	$\mathbf{0.536 \pm 0.035}$	0.509 ± 0.034	0.452 ± 0.012	0.500 ± 0.010
Wine	0.980 ± 0.007	0.984 ± 0.009	0.988 ± 0.008	$\mathbf{0.990 \pm 0.008}$	0.981 ± 0.009	0.978 ± 0.000

Table 4: Training time (in seconds) obtained using the methods of Table 2 in Table 1 datasets.

	R-ELM-100	R-ELM-3000	DSN-ELM	Proposed	S-ELM	AE-S-ELM
IRIS	0.009 ± 0.003	$\mathbf{0.076 \pm 0.011}$	0.088 ± 0.011	0.095 ± 0.022	0.121 ± 0.020	0.130 ± 0.017
MNIST	0.238 ± 0.012	23.759 ± 1.050	16.277 ± 0.416	$\mathbf{3.425 \pm 0.090}$	5.336 ± 0.518	10.957 ± 0.383
SatImage	0.020 ± 0.001	2.172 ± 0.103	0.546 ± 0.040	$\mathbf{0.327 \pm 0.063}$	0.336 ± 0.033	0.429 ± 0.033
Segment	0.013 ± 0.004	0.759 ± 0.041	0.226 ± 0.043	$\mathbf{0.153 \pm 0.026}$	0.198 ± 0.027	0.231 ± 0.023
USPS	0.029 ± 0.002	3.216 ± 0.095	1.416 ± 0.146	$\mathbf{0.529 \pm 0.077}$	0.579 ± 0.051	0.930 ± 0.070
Vowel	0.011 ± 0.003	0.314 ± 0.020	0.138 ± 0.011	$\mathbf{0.102 \pm 0.008}$	0.145 ± 0.027	0.167 ± 0.016
Wine	0.010 ± 0.003	$\mathbf{0.080 \pm 0.004}$	0.088 ± 0.008	$\mathbf{0.080 \pm 0.006}$	0.126 ± 0.011	0.136 ± 0.010

Table 5: Regularization parameter C and memory used (in Megabytes) by the methods of Table 2 in Table 1 datasets.

	Parameter C	R-ELM-100	R-ELM-3000	DSN-ELM	Proposed	S-ELM	AE-S-ELM
IRIS	2^5	0.006	0.183	1.181	**0.141**	1.236	1.236
MNIST	2^{-7}	0.607	18.196	21.517	**1.051**	10.621	10.621
SatImage	2^{10}	0.033	0.984	2.978	**0.300**	1.684	1.684
Segment	2^7	0.021	0.618	2.943	**0.333**	1.505	1.505
USPS	2^{-2}	0.204	6.111	9.432	**0.648**	4.377	4.377
Vowel	2^{10}	0.017	**0.504**	4.156	0.506	1.491	1.491
Wine	2^1	0.013	0.389	1.387	**0.148**	1.343	1.343

good approach when using limited computational resources. Figure 3b shows that DSN-ELM and FDSN-ELM, with $S = 30$, already reached a stage where its accuracy does not improve significantly when adding new modules, but there is still room for improvement in S-ELM and AE-S-ELM.

We also tested the trade-off between choosing the values of S and L using the DSN-ELM and FDSN-ELM algorithms. We tested networks with up to 100 modules with $L = [100, 200, \ldots, 1000]$ hidden neurons in each module. The accuracy of both techniques is shown in Figures 4a and 4b. These images show, as expected, that the accuracy in SatImage dataset can be increased by increasing both S and L. These values should be defined based on memory limitations and observing the impact of a new module on the metric.

5 Conclusions and Future Work

In this work was proposed a fast algorithm to train a DSN using ELM. FDSN-ELM uses the "stacking" principle, so that its modules (layers) can be trained individually, requiring less memory and training time than other methods of deep learning. The algorithm was compared with other techniques that uses the same principle, and obtained similar accuracies in most datasets, spending less time in the training stage. Furthermore, the number of parameters needed to be stored in the proposed algorithm architecture is much lower when compared to other techniques. Future work includes adapting the algorithm to deal with data in a sequential approach, turning possible to deal with samples that arrive over time.

Acknowledgment

The author would like to thank CAPES for the scholarship granted, No. 1444056.

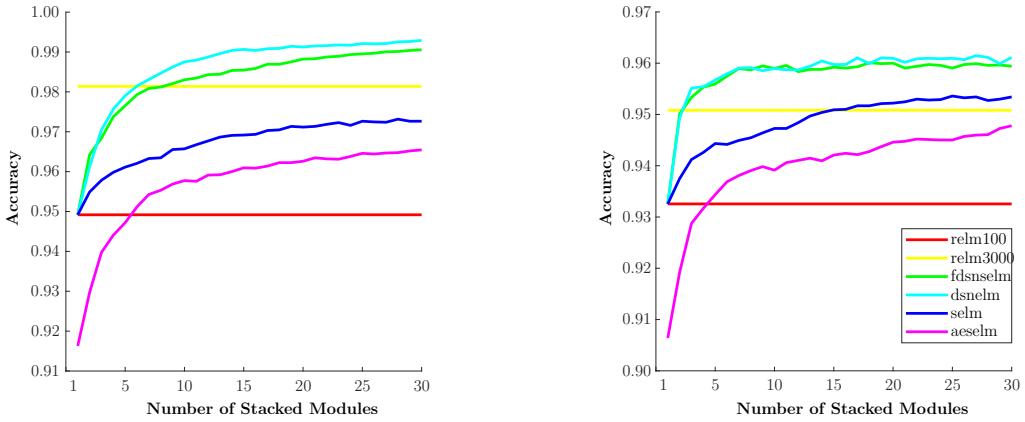
References

Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network, *IEEE transactions on Information Theory* **44**(2): 525–536.

Bottou, L. and LeCun, Y. (2004). Large scale online learning, *Advances in neural information processing systems*, pp. 217–224.

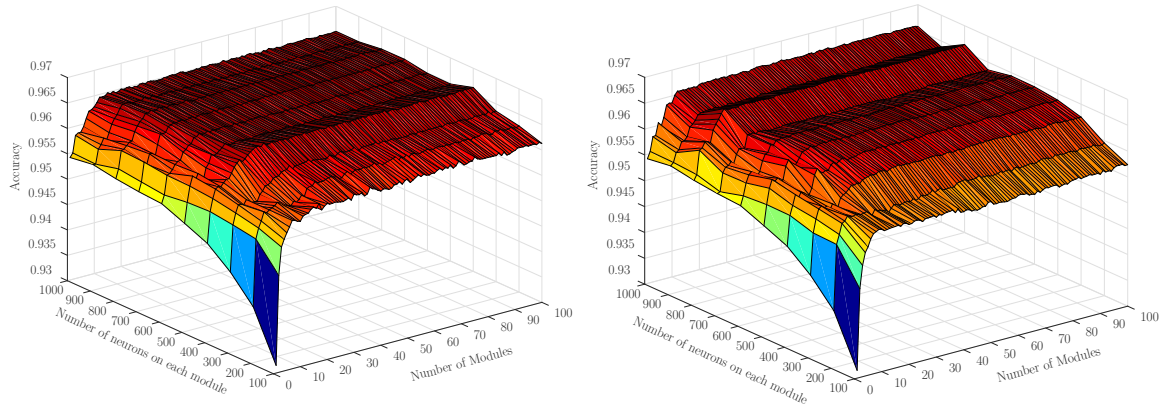
Deng, L. and Yu, D. (2011). Deep convex net: A scalable architecture for speech pattern classification, *Proceedings of the International Speech Communication Association* pp. 2285–2288.

Deng, L. and Yu, D. (2013). Deep Learning: Methods and Applications, *Foundations and Trends in Signal Processing* **7**(3-4): 197–387.



(a) Training Accuracy (b) Testing Accuracy

Figure 3: The accuracy (in Segment Dataset) when new modules are stacked in the tested algorithms.



(a) DSN-ELM (b) FDSN-ELM

Figure 4: The effect of choosing S and L when using DSN-ELM and FDSN-ELM in SatImage dataset.

Deng, W., Zheng, Q. and Chen, L. (2009). Regularized Extreme Learning Machine, *2009 IEEE Symposium on Computational Intelligence and Data Mining* (60825202): 389–395.

Huang, G.-B., Zhou, H., Ding, X. and Zhang, R. (2012). Extreme Learning Machine for Regression and Multiclass Classification, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **42**(2): 513–529.

Huang, G.-B., Zhu, Q.-Y. and Siew, C.-K. (2004). Extreme learning machine: a new learning scheme of feedforward neural networks, *2004 International Joint Conference on Neural Networks*, Vol. 2, IEEE, pp. 985–990.

Huang, G. B., Zhu, Q. Y. and Siew, C. K. (2006). Extreme learning machine: Theory and applications, *Neurocomputing* **70**(1-3): 489–501.

Inaba, F. K., Salles, E. O. T., Perron, S. and Caporossi, G. (2018). DGR-ELM - Distributed Generalized Regularized ELM for classification, *Neurocomputing* **275**: 1522–1530.

Kaisler, S., Armour, F., Espinosa, J. A. and Money, W. (2013). Big data: Issues and challenges moving forward, *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, IEEE, pp. 995–1004.

Long, J., Shelhamer, E. and Darrell, T. (2015). Fully convolutional networks for semantic segmentation, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440.

Wolpert, D. H. (1992). Stacked generalization, *Neural networks* **5**(2): 241–259.

Zhou, H., Huang, G.-B., Lin, Z., Wang, H. and Soh, Y. C. (2014). Stacked Extreme Learning Machines., *IEEE transactions on cybernetics* **45**(9): 1.

Zhu, S. and Zanibbi, R. (2016). A text detection system for natural scenes with convolutional feature learning and cascaded classification, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 625–632.