Restricted Boltzmann machine to determine the input weights for extreme learning machines

Andre G. C. Pacheco
Graduate Program in Computer Science, PPGI
Federal University of Espírito Santo - UFES
Vitória, ES - Brazil
`pacheco.comp@gmail.com`

Renato A. Krohling
Production Engineering Department and Graduate Program in Computer Science, PPGI
Federal University of Espírito Santo - UFES
Vitória, ES - Brazil
`krohling.renato@gmail.com`

Carlos A. S. da Silva
Graduate Program in Computer Science, PPGI
Federal University of Espírito Santo - UFES
Vitória, ES - Brazil
`carlosalexandress@gmail.com`

Abstract

The Extreme Learning Machine (ELM) is a single-hidden layer feedforward neural network (SLFN) learning algorithm that can learn effectively and quickly. The ELM training phase assigns the input weights and bias randomly and do not change them in the whole process. Although the network works well, the random weights in the input layer can make the algorithm less effective and impact on its performance. Therefore, we propose a new approach to determine the input weights and bias for the ELM using the restricted Boltzmann machine (RBM), which we call RBM-ELM. We compare our new approach with a well-known approach to improve the ELM and a state of the art algorithm to select the weights for the ELM. The results show that the RBM-ELM outperforms both methodologies and achieve a better performance than the ELM.

Keywords: Extreme learning machine; Restricted Boltzmann machine; Neural networks; Weights initialization

1 Introduction

The extreme learning machine (ELM) is an approach for training single-hidden layer feedforward neural network (SLFN) proposed by Huang et al. (2004) and Huang et al. (2006). Its use has become popular due to its fast and analytically training phase. Comparing it with backpropagation Rumelhart et al. (1986), the most common algorithm used to train neural networks, its learning stage can be thousand of times faster and it can achieve a better generalization Huang et al. (2006).

The ELM learning algorithm assigns the weights from the input layer to hidden layer ($\mathbf{W}$) randomly. Then, it computes the weights from hidden layer to output layer ($\boldsymbol{\beta}$) analytically using the least-squares method. The ELM has been used to solve many different problems, such as classification data

Huang et al. (2010); Xu and Wang (2017), time series forecasting Butcher et al. (2013), regression problem Huang et al. (2012); Shihabudheen and Pillai (2017), among others.

Although Huang et al. (2004) proved the universal approximation capability of SLFNs with unchanged random weights throughout the whole training phase, this issue has attracted the efforts of many researchers. As shown in Wang et al. (2011), the randomness of the input weights can make the algorithm less effective, depending on the assigned values for $\mathbf{W}$. Moreover, this point also influences the algorithm performance, i.e., the ELM output is not quite stable Wang et al. (2015). Due to this issue, many approaches have been proposed aiming to improve the generalization and performance of the ELM. Some of these approaches avoid assigning the input weights by providing a different way to compute the values of the hidden neurons, i.e., the feature map. One standard approach is the K-ELM, presented by Huang et al. (2012). It is a deterministic methodology that uses a suitable kernel function specified by the user to compute the hidden neurons. The K-ELM can achieve good performance, however, it does not work well when the database grows, because it demands a large computational time and memory consuming. We can highlight other methodologies, such as the PCA-ELM, a deterministic algorithm proposed by Castaño et al. (2013), which is used to initiate any hidden neurons based on principal component analysis (PCA) and the PL-ELM, an architecture developed by Henríquez and Ruz (2017), which is based on a non-linear layer in parallel by another non-linear layer and with entries of independent weights. Deep learning techniques have also been used to improve the ELM performance. Kasun et al. (2013) presented the ELM autoencoder (ELM-AE) and Sun et al. (2017) expanded it developing the generalized ELM autoencoder (GELM-AE). Inspired by ELM-AE, Tissera and McDonnell (2016) proposed a deep neural network (DBN) using ELMs as a stack of supervised autoencoders. All these approaches are used with the same goal of the K-ELM and PCA-ELM, i.e., to extract the feature map of the first layer and to avoid the random weights assignment.

Other approaches focus on finding a way to provide a better initialization of the input weights, and consequently improving the ELM performance. Han et al. (2013) introduced an optimization of the input weights via particle swarm optimization (PSO). Cervellera and Macciò (2016) presented an algorithm that replaces the random initialization by a deterministic one using low-discrepancy sequences (LDS). Recently, Wang and Liu (2017) proposed the ELM-RO, a state of the art approach that is mainly based on the Gram-Schmidt orthogonalization of the input weights, which achieves a better performance than the standard ELM and other algorithms that compute the weights for the ELM.

In this work, our main contribution is to present a new approach to determine the input weights for the ELM using the restricted Boltzmann machine (RBM) Smolensky (1986); Hinton (2002), which we call RBM-ELM. Basically, the RBM is an energy-based system that can learn the probability distribution of a database by unsupervised learning. As autoencoders, RBMs are widely used to compose DBNs Hinton et al. (2006). Therefore, the architecture presented by Tissera and McDonnell (2016), the stack of autoencoders to extract features from the data and set it as the ELM hidden nodes, can be done by using RBMs. Indeed, it was introduced by Cao et al. (2014). Nonetheless, we need to make clear this is not our intention in this work. We just use one RBM to determine good values for the ELM input weights and then proceed with the standard ELM training. As we present throughout this paper, we need just a few RBM training epochs to set the ELM input weights. In the experiments section, we show that our approach has a relatively fast learning phase and achieves a good performance. Furthermore, we compare the RBM-ELM with the well-known ELM-AE and with the state of the art ELM-RO. Experimental results show that the RBM-ELM can outperform them for the benchmarks used in this paper. The remainder of the paper is organized as follows. In section 2, we present a brief background on ELM and RBM. Section 3 describes our new approach. In section 4, present the experimental results. Lastly, in section 5 we draw our final conclusions.

2 Background

2.1 Extreme learning machine

The extreme learning machine approach was developed specifically to handle with SLFN architecture. We depict in Figure 1 an SLFN, where $\mathbf{x}$ is the input data, $\mathbf{y}$ is the output layer, $\mathbf{W}$ is the weights

matrix of the input layer, $\boldsymbol{\beta}$ is the weights matrix of the hidden layer and $\mathbf{b}$ is the bias of the input layer. We represent all these notation as follows:

$$\mathbf{x} = [x_1, \cdots, x_m, 1] \quad \mathbf{W} = \begin{bmatrix} w_{11} & \cdots & w_{1k} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mk} \\ b_1 & \cdots & b_k \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_{11} & \cdots & \beta_{1s} \\ \vdots & \ddots & \vdots \\ \beta_{k1} & \cdots & \beta_{ks} \end{bmatrix} \quad \mathbf{y} = [y_1, \cdots, y_s] \tag{1}$$

where m, s and k are the number of input, output and hidden neurons, respectively. Observe that the weights and bias are put together in the same matrix to ease the computation.
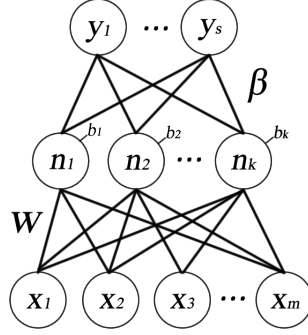

Figure 1: The architecture of a single-hidden layer feedforward neural network

In the ELM algorithm, the matrix $\mathbf{W}$ is initialized randomly by sampling all the weight values from a continuous distribution, normally the uniform distribution in the interval [-1,1], and it does not change during the whole learning phase. Huang et al. (2006) proved that we can obtain the matrix $\boldsymbol{\beta}$ performing an analytical process maintaining the universal approximation capability for the SLFNs. After sampling the weights we compute the hidden neurons, that is, the feature map ($\mathbf{H}$), as follows:

$$\mathbf{h}^i = [x_1^i, \cdots, x_m^i, 1] \times \begin{bmatrix} w_{11} & \cdots & w_{1k} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mk} \\ b_1 & \cdots & b_k \end{bmatrix} \Rightarrow \mathbf{H} = \begin{bmatrix} f(\mathbf{h}^1) \\ f(\mathbf{h}^2) \\ \vdots \\ f(\mathbf{h}^N) \end{bmatrix}_{N \times k} \tag{2}$$

where $f(.)$ is the activation function, e.g., a logistic function, $i = (1, \ldots, N)$ and N is the number of samples in the training set. Next, we compute $\boldsymbol{\beta}$ by solving the linear system through a simple generalized inverse operation, as described by:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{Y} \to \boldsymbol{\beta} = \mathbf{H}^\dagger \mathbf{Y} \tag{3}$$

where $\mathbf{H}^\dagger$ is the Moore-Penrose generalized inverse of $\mathbf{H}$ (Serre, 2002). The Moore-Penrose based solution is one of the least-square solutions for a general linear system. It can achieve: the smallest training error, the smallest norm of the weights and, as consequence, a good generalization performance. Moreover, it does not get stuck in local as the gradient descent-based learning methods (Huang et al., 2006). A pseudocode of the ELM is presented in Algorithm 1.

2.2 Restricted Boltzmann machine

The restricted Boltzmann machine (RBM) is a stochastic network composed of a visible layer ($\mathbf{v}$) and a hidden layer ($\mathbf{d}$). As illustrated in Figure 2, there is no connection within a layer, $\mathbf{v}$ and $\mathbf{d}$ have symmetric connectivity $\mathbf{W}$ and each layer has its own bias, $\mathbf{a}$ and $\mathbf{b}$. During the training phase, the RBM learns the probability distribution over the input data training through unsupervised

---
Algorithm 1: Training an SLFN by the ELM procedure
---
1 **Function** elm_training $(\mathbf{X}, \mathbf{Y}, k, \mathbf{W}, \mathbf{b})$:
   Input : A training set $\{\mathbf{X}, \mathbf{Y}\}$;
            The number of hidden neurons k;
            The input weights $\mathbf{W}$ and the bias $\mathbf{b}$;
2 **if** *W and b are empty* **then**
3 | Sampled them by a uniform distribution in the interval $[-1, 1]$;
4 **end**
5 Compute the feature map $\mathbf{H}$ from equation 2;
6 Compute β from equation 3;
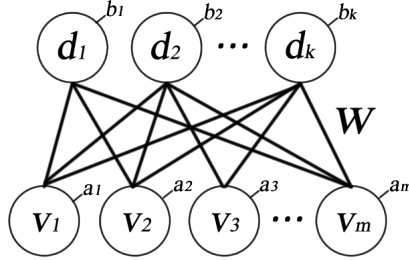   **Return** : $\mathbf{W}$, $\mathbf{H}$ and β;

Figure 2: The architecture of a RBM with m visible nodes and k hidden nodes

learning. Therefore, this network has been applied in different tasks, such as feature extraction, pattern recognition, dimensionality reduction, data classification etc (Hinton, 2010).

Originally, the RBM was proposed for binary input data. However, Hinton and Salakhutdinov (2006) extended it for continuous input data. In this paper we use the continuous approach.

In the RBM model, each configuration $(\mathbf{v}, \mathbf{d})$ has an associated energy value defined by:

$$E(\mathbf{v}, \mathbf{d}; \boldsymbol{\theta}) = -\sum_{i=1}^{m} \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_{j=1}^{k} b_j d_j - \sum_{i,j=1}^{m,k} \frac{v_i}{\sigma^2} d_j w_{ij} \qquad (4)$$

where $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{a}, \mathbf{b})$. The joint probability of $(\mathbf{v}, \mathbf{d})$ is computed as follows:

$$p(\mathbf{v}, \mathbf{d}; \boldsymbol{\theta}) = \frac{e^{-E(\mathbf{v}, \mathbf{d}; \boldsymbol{\theta})}}{\sum_{\mathbf{v}, \mathbf{d}} e^{-E(\mathbf{v}, \mathbf{d}; \boldsymbol{\theta})}} \qquad (5)$$

In general, the goal of the RBM learning algorithm is estimating $\boldsymbol{\theta}$ that decreases the energy function (Hinton, 2010). Hinton (2002) proposed an efficient algorithm for training RBMs, known as contrastive divergence (CD). The CD is an unsupervised algorithm that uses an iterative process known as Gibbs sampling. The main idea of this algorithm is initializing the visible layer with training data and then perform the Gibbs sampling. The CD is an easy and fast learning algorithm. Therefore, the most important use of the RBM is as learning modules that are composed to form deep belief nets (Hinton, 2010).

In order to perform the CD algorithm, we need to compute $p(\mathbf{d}|\mathbf{v})$ as follows:

$$p(d_j = 1 | \mathbf{v}; \boldsymbol{\theta}) = \phi(b_j + \sum_{i=1}^{m} v_i w_{ij}) \qquad (6)$$

4

where $\phi(x) = \frac{1}{1+e^{-x}}$, the logistic function. Then, we compute $p(\mathbf{v}|\mathbf{d})$ by:

$$p(v_i = v|\mathbf{d}; \boldsymbol{\theta}) = N(v|a_i + \sum_{j=1}^{k} d_j w_{ij}, \sigma^2) \tag{7}$$

where N is the normal distribution with mean v and standard deviation σ^2.
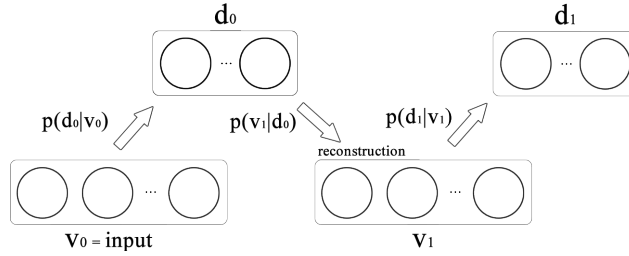

Figure 3: The Gibbs sampling procedure on CD algorithm

The Gibbs sampling procedure is illustrated in Figure 3. As we can note, first we initialize the visible layers with the training data. Next, we estimate $\mathbf{d}_0$ using the equation 6. From $\mathbf{d}_0$ we estimate $\mathbf{v}_1$ by equation 7. This step is called reconstruction. Finally, from $\mathbf{v}_1$ we estimate $\mathbf{d}_1$ using equation 6 again. This whole procedure can be done z times, however, with just one iteration the algorithm works quite well (Hinton, 2010). After proceeding with the Gibbs sampling, the CD update rules for $\boldsymbol{\theta}$ are described by the following equations:

$$\mathbf{W}^{\mathbf{t+1}} = \mathbf{W}^{\mathbf{t}} + \Delta\mathbf{W}^{\mathbf{t}} \rightarrow \Delta\mathbf{W}^{\mathbf{t}} = \eta(\mathbf{v_0}\mathbf{d_0^T} - \mathbf{v_1}\mathbf{d_1^T}) - \rho\mathbf{W}^{\mathbf{t}} + \alpha\Delta\mathbf{W}^{\mathbf{t-1}} \tag{8}$$

$$\mathbf{a}^{\mathbf{t+1}} = \mathbf{a}^{\mathbf{t}} + \Delta\mathbf{a}^{\mathbf{t}} \rightarrow \Delta\mathbf{a}^{\mathbf{t}} = \eta(\mathbf{v_0} - \mathbf{v_1}) + \alpha\Delta\mathbf{a}^{\mathbf{t-1}} \tag{9}$$

$$\mathbf{b}^{\mathbf{t+1}} = \mathbf{b}^{\mathbf{t}} + \Delta\mathbf{b}^{\mathbf{t}} \rightarrow \Delta\mathbf{b}^{\mathbf{t}} = \eta(\mathbf{d_0} - \mathbf{d_1}) + \alpha\Delta\mathbf{b}^{\mathbf{t-1}} \tag{10}$$

where η, ρ and α are known as learning rate, weight decay and momentum, respectively. Hinton (2010) suggests $\eta = 0.01$, $\rho = [0.01, 0.0001]$ and $\alpha = 0.5$ for the first five iterations and $\alpha = 0.9$ otherwise. Usually, the terms of $\boldsymbol{\theta}$ are initialized randomly. Further, Hinton (2010) also suggests to

divide the training set into small mini-batches of 10 to 100 cases. A pseudocode of the constrastive divergence is presented in Algorithm 2.

Algorithm 2: The contrastive divergence procedure

1 **Function** rbm_training $(\mathbf{X}, k, \eta, \rho, \alpha, bs, it)$:
   Input : The training data $\mathbf{X}$ with N samples;
          The number of hidden neurons k;
          The parameters η, ρ and α;
          The batch size bs;
          The maximum number of iterations it
2 Sample $\boldsymbol{\theta}$ from a normal distribution with $\mu = 0.1$ and $\sigma = 1$;
3 Split $\mathbf{X}$ in batches $\mathbf{X_b}$ with bs samples;
4 Initialize $i = j = 0$;
5 **while** $j < it$ *or* $\boldsymbol{\theta}$ *converged* **do**
6   **for** *each batch* $\mathbf{X_b}$ **do**
7   **while** $i < bs$ **do**
8          $\mathbf{v}_0 = \mathbf{X_b}_i$ Sample $\mathbf{d}_0$ from equation 6;
9   Sample $\mathbf{v}_1$ from equation 7;
10   Sample $\mathbf{d}_1$ from equation 6;
11   Update $\boldsymbol{\theta}$ from equations 8, 9 and 10;
12         $i = i + 1$
13   **end**
14   **end**
15    $j = j + 1$
16 **end**
   Return : $\boldsymbol{\theta}$;

---

3   A new approach to determine weights for ELMs

In this section, we introduce the new approach to determine weights for the extreme learning machine using the restricted Boltzmann machine, which we call RBM-ELM. Our focus is on the input weights $\mathbf{W}$. Since the ELM assigns $\mathbf{W}$ at random and it is used to compute $\mathbf{H}$ and $\boldsymbol{\beta}$, there inevitably exists a set of nonoptimal input weights and hidden biases values, which may influence on the ELM performance (Han et al., 2013). Thus, the RBM-ELM main idea is to replace the ELM input weights and bias by the RBM visible weights and hidden bias, as shown in Figure 4.
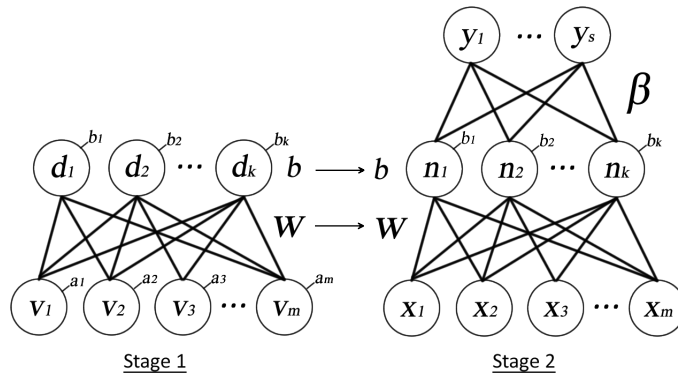

Figure 4: An illustration of the RBM-ELM approach

As we can note in Figure 4, the RBM-ELM approach has two stages. In the first stage, we compute the input weights and bias for the ELM through the RBM training. In the second stage, the results of the first stage are used to set $\{\mathbf{W}, \mathbf{b}\}$ and then to proceed with the standard ELM training. To better describe these two stages, consider a training set composed by $\{\mathbf{X}_t, \mathbf{Y}_t\}$, where $\mathbf{X}_t$ and $\mathbf{Y}_t$ are the input and output training data, respectively. First, the RBM is trained with $\mathbf{X}_t$. All the knowledge obtained by this network is stored on its weights and bias. So, after the RBM training, we set the

ELM input weights and bias with the same values of the RBM weights connections and hidden bias, respectively. Next, we carry out the ELM training using $\mathbf{X}_t$, $\mathbf{Y}_t$ and the computed $\{\mathbf{W}, \mathbf{b}\}$ to compute $\mathbf{H}$ and $\boldsymbol{\beta}$. Since we use the same input training data to feed the visible and input layer in the RBM and ELM, respectively, $\mathbf{v}$ and $\mathbf{x}$ have the same shape. Thus, to guarantee the same shape for $\mathbf{W}$ and $\mathbf{b}$ on both networks, we need to set the same number of hidden neurons for both algorithms. A pseudocode of this approach is described in Algorithm 3.

Algorithm 3: The RBM-ELM procedure

---
1 **Function** rbm_elm_training $(\{\mathbf{X}_t, \mathbf{Y}_t\}, k, \eta, \rho, \alpha)$:
   Input : The training set $\{\mathbf{X}_t, \mathbf{Y}_t\}$ with N samples;
         The number of neurons k;
         The parameters η, ρ and α;
2 From Algorithm 2 call rbm_training $(\mathbf{X}_t, k, \eta, \rho, \alpha)$ to get $\mathbf{W}$ and $\mathbf{b}$;
3 From Algorithm 1 call elm_training $(\mathbf{X}_t, \mathbf{Y}_t, k, \mathbf{W}, \mathbf{b})$ to get $\mathbf{H}$ and $\boldsymbol{\beta}$;
   Return : $\mathbf{W}$, $\mathbf{H}$ and $\boldsymbol{\beta}$;

As mentioned earlier, a stack of RBMs has been used to improve the ELM performance (Cao et al., 2014). Nonetheless, its goal is to find the feature map for the input layer, i.e., setting the matrix $\mathbf{H}$ on the ELM. As a drawback, this method may take a long computational time to achieve a good feature map, which removes from the ELM one of its great advantages: the fast training phase. Our approach aims to take advantage of the RBM generalization capability. Since the RBM models the probability distribution over the data training, when we compute the ELM weights and bias from an RBM, we also transfer the knowledge obtained by the RBM training phase to the ELM. Consequently, we improve the feature map extracted by the input layer in the ELM.

As we can note in Algorithm 2, the RBM training is carried out until the weights convergence or a specific number of epochs. Comparing with the ELM training phase, the RBM training may take a considerable computational time if we wait for the convergence or set a high value for the number of epochs. However, the reconstruction error on the entire training set falls rapidly and consistently at the start of learning and then more slowly (Hinton, 2010). So, the weights converge nearly to their final values after only a few epochs, but the further fine tuning takes much longer (Yosinski and Lipson, 2012). As our intention is to determine the input weights and bias for the ELM, the proposed approach works well performing just a few epochs of the RBM training. So, the RBM training phase does not affect too much the time-consuming of the whole algorithm. In the next section, we show the effect of the number of epochs on the algorithm performance.

4 Experimental results

In this section, we carry out two experiments for classification problems. First, we present a thorough example to better describe the RBM-ELM. Next, we compare our approach with the standard ELM, the state of the art ELM-RO, and the ELM-AE, using several well-known benchmarks. All procedures were implemented in Python and Tensorflow, and performed on an *intel core i7-6* CPU @ 2.50 GHz PC with 8 GB of RAM and a Nvidia Geforce 940M. The code developed is available upon request.

4.1 Illustrative example

In this illustrative example, we investigate the RBM-ELM configurations in order to improve the algorithm performance. We developed a vowels database, which contains 1380 samples with 276 examples for each vowel. In Figure 5 is shown some samples from the database, where each vowel is represented by a image with 30×30 pixels. Thus, the vowels database has 900 input features and five classification labels.

The vowels database was shuffled and split to 70% for training and 30% for tests. The ELM and RBM-ELM were run 30 times to compute statistics and we use the mean and standard deviation of the classification accuracy as performance metric. In general, our goal in this section is to present a discussion about the RBM parameters, the number of epochs used to train a RBM, the running time of the whole approach, the number of hidden neurons, and compare the performance between the ELM and RBM-ELM.
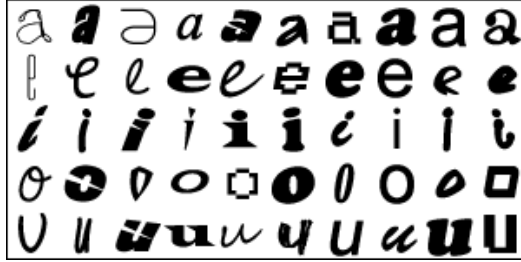
Figure 5: Some samples from the vowels database

First of all, we need to choose the number of hidden neurons (k). We perform the ELM and choose k empirically based on the best performance of this algorithm. To be fair, we decided to use the same value of k to compare both methodologies. In table 1 is described the ELM performance varying the number of hidden neurons. As we can note, the best performance is achieved when $k = 400$. Thus, we use this number of hidden neurons to perform the RBM-ELM as well.

Table 1: The ELM performance for vowels database varying the number of hidden neurons

# of neurons	Accuracy (%)	Time (sec)
100	79.677 ± 1.985	0.050 ± 0.006
200	85.789 ± 1.356	0.096 ± 0.010
300	86.771 ± 1.555	$0.166. \pm 0.016$
400	$\mathbf{87.975 \pm 1.500}$	$\mathbf{0.239 \pm 0.029}$
500	86.972 ± 1.721	0.329 ± 0.027
600	85.668 ± 1.793	0.436 ± 0.026

As described in section 3, in order to apply the RBM-ELM, first we need to perform the stage 1 of the algorithm, that is, to train the RBM and compute the input weights and bias for the ELM. As we can note in Algorithm 3, the RBM has seven parameters, where $\mathbf{X}$ is the input data and k is the number of hidden neurons, which is the same value that we found for ELM, $k = 400$. For η, ρ, α and bs, we use the values in the interval suggested by Hinton (2010), presented in section 2.2. These values are described in Table 2 and their fine tune was achieved empirically by tests. Finally, we need to choose the number of maximum iterations it, which affect directly the time consuming of the algorithm. Indeed, we could wait for θ convergence, however it may take a long time and we do not need to do that. As described in section 3, we need a few iterations to put the weights nearly to their final values and these values are good enough to improve the ELM. After finishing the stage 1, we perform the stage 2 of the algorithm with the weights and bias computed from the stage 1. In table 2 is described the RBM-ELM performance varying it and computing the time consuming for each scenario.

Table 2: The RBM-ELM performance for vowels database varying the iteration number of the RBM training

RBM parameters: $k = 400$, $\eta = 0.001$, $\rho = 0.01$, $\alpha = 0.5/0.9$ and $bs = 100$		
# of iterations (*it*)	Accuracy (%)	Time (sec)
10	90.708 ± 1.518	1.421 ± 0.115
30	90.442 ± 1.144	3.617 ± 0.120
50	$\mathbf{92.769 \pm 1.078}$	$\mathbf{6.008 \pm 0.099}$
100	90.724 ± 1.303	11.667 ± 0.096
300	90.611 ± 1.420	33.521 ± 0.311
500	89.067 ± 1.441	54.901 ± 1.336

As we can note in Table 2, the RBM can improve the ELM performance even with a low number of iterations. For $it = 50$ the algorithm achieved the best performance. Nonetheless, for *it* less than 100, the approach may get good performance. However, the higher is the value of it, the higher is the running time. Comparing the RBM-ELM with $it = 50$ and the best ELM performance in Table 1, our approach improve the classification accuracy in almost 5% and decrease the standard deviation

around 0.5%. On the other hand, the standard ELM training is 25 times faster than RBM-ELM with 50 iterations. In fact, we need to make a trade-off between accuracy and computational time. In this case, the improved performance is very desired.

Huang et al. (2006) affirm that the ELM learning not only tends to reach the smallest training error but also the smallest norm of weights. Bartlett (1998) states that the smaller the norm of weights, the better generalization performance a feedforward network tends to have. So, we also compare the norm of the input weights obtained by the ELM and RBM-ELM for the vowels database. Based on the best performance of both ELM and RBM-ELM in Table 1 and 2, respectively, the norm for the ELM is 346.602 and for RBM-ELM is 2.839. Hence, according to Huang et al. (2006) and Bartlett (1998), for this database, the RBM-ELM tends to have a better generalization performance than the ELM.

In Figure 6 is illustrated the graphic difference between the ELM and RBM-ELM input weights. Since the ELM set the weights randomly, its plot looks like a noise. On the other hand, in the RBM-ELM the input weights show some low-level features, even using 50 iterations of the RBM training phase. These plots help to explain why RBM-ELM achieves a better performance than ELM in this database.
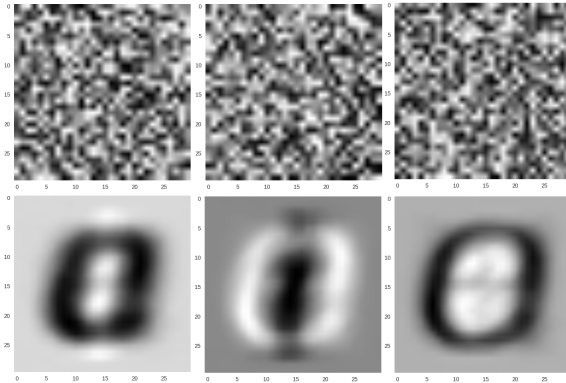

Figure 6: The plot of the input weights for the ELM (above) and RBM-ELM (below).

4.2 Standard benchmarks

In order to evaluate the performance of the RBM-ELM we carry out experiments with different well-known classification benchmarks from UCI repository (Lichman, 2013). We still compare the RBM-ELM performance with the standard ELM. In addition, we include two more algorithms, the ELM-RO and ELM-AE, both discussed in the introduction. All the databases used in this experiment is described in Table 3. In this table, each database with permutation equal to *yes* was shuffled and split to 70% for training and 30% for tests. For permutation equal to *no*, it means that this database has a test partition. The four algorithms were run 30 times to compute statistics. We used the mean and standard deviation of the classification accuracy as the performance metric. Further, we perform the non-parametric Friedman test followed by Wilcoxon test as a pos hoc to compare the algorithms performance (Derrac et al., 2011).

In Table 4 is described the RBM-ELM parameters configuration for each database presented in Table 3. These parameters were selected empirically using the same process detailed in section 4.1. For a fair comparison, the ELM-RBM, ELM-AE, and ELM-RO use the same number of hidden neurons of the ELM, which is described as k in Table 4. In Table 5 is reported the performance of all algorithms for each database used in this experiment. In this table is described the mean and standard deviation of the classification accuracy and the mean value of the running time for each approach. Moreover, in order to improve the results visualization, in Figure 7 is depicted the boxplots for all approaches also for each database.

According to the results presented in the tables and boxplots, we can note that the ELM-AE, ELM-RO and RBM-ELM have better performance than ELM for most of databases. Only for *credit Australia* and *spam* databases, these approaches were not able to improve the classification performance. In fact,

Table 3: The classifications databases used in this experiment

Database	# of samples	# of features	# of labels	Permutation
Alphabet	11960	900	26	Yes
Credit Australia	690	14	2	Yes
Diabetic	1151	19	2	Yes
DNA	3186	180	3	No
Gisette	7000	5000	3	No
Isolet	7797	617	26	No
Madelon	2600	500	2	Yes
Spam	4601	57	2	Yes
Urban land cover	675	147	9	Yes

Table 4: The RBM-ELM parameters configuration for each database

Database	RBM-ELM parameters
Alphabet	$k = 950$, $it = 25$, $\eta = 0.001$, $\rho = 0.001$, $\alpha = 0.5/0.9$ and $bs = 150$
Credit Australia	$k = 35$, $it = 25$, $\eta = 0.1$, $\rho = 0.1$, $\alpha = 0.5/0.9$ and $bs = 150$
Diabetic	$k = 50$, $it = 50$, $\eta = 0.01$, $\rho = 0.001$, $\alpha = 0.5/0.9$ and $bs = 100$
DNA	$k = 250$, $it = 50$, $\eta = 0.0001$, $\rho = 0.01$, $\alpha = 0.5/0.9$ and $bs = 250$
Gisette	$k = 850$, $it = 50$, $\eta = 0.001$, $\rho = 0.01$, $\alpha = 0.5/0.9$ and $bs = 250$
Isolet	$k = 1250$, $it = 25$, $\eta = 0.01$, $\rho = 0.01$, $\alpha = 0.5/0.9$ and $bs = 250$
Madelon	$k = 250$, $it = 50$, $\eta = 0.001$, $\rho = 0.001$, $\alpha = 0.5/0.9$ and $bs = 50$
Spam	$k = 150$, $it = 50$, $\eta = 0.001$, $\rho = 0.0001$, $\alpha = 0.5/0.9$ and $bs = 100$
Urban land cover	$k = 200$, $it = 25$, $\eta = 0.01$, $\rho = 0.01$, $\alpha = 0.5/0.9$ and $bs = 20$

for these two databases, all four algorithms got the same performance. However, for the remaining databases, the ELM was improved by at least one of the others methods. So, the ELM has the lowest overall accuracy of all methods. On the other hand, the ELM is still the fastest algorithm among all. Indeed, this is expected since the other approaches require more processing time to compute the input weights. When we look at the RBM-ELM performance, we can note that it achieves the best performanc, alone or followed by another method for all database. As consequence, the RBM-ELM obtained the highest overall accuracy. However, it also gets the highest total time among all algorithms.
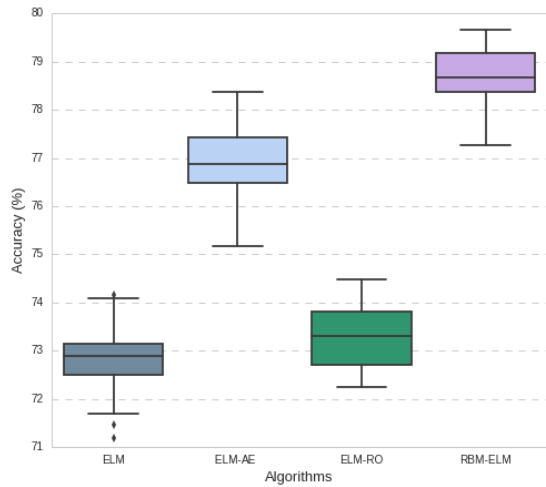
We perform the Friedman and Wilcoxon test to better evaluate the performance of the algorithms for each database. First, we perform the Friedman test, if it returns $p_{value} \leq 0.05$, it means that significant differences were found and we can proceed with a post-hoc procedure to characterize these differences. Next, we perform the Wilcoxon test for pairwise comparisons. If the Wilcoxon test returns $p_{value} \leq 0.01$, it means that there is a significant difference between the compared pair (Derrac et al., 2011). We applied these tests to all databases together and individually. For *credit Australia* and *spam*, the Friedman test returned $p = 0.841$ and $p = 0.180$, respectively. Thus, for these databases, there is no significant differences among the algorithms. Since the algorithms accuracy performance is too close, this is expected. For the rest of the databases, the values returned by the Friedman test is always less than 0.01, then we perform the Wilcoxon test for all of them. As we have too many pairwise comparisons, we decide to highlight the main test outcomes as follows:

- For the *alphabet*, *DNA*, and *madelon*, the RBM-ELM is significantly different when compared to all others.

- For the *gisette*, *isolet* and *urban cover land*, there is no significant difference between the ELM-AE and RBM-ELM. Moreover, they are significantly different when compared to all others.

- For the *diabetic*, there is no significant difference between the ELM-AE and RBM-ELM. In addition, this pair is significantly different when compared to all others.

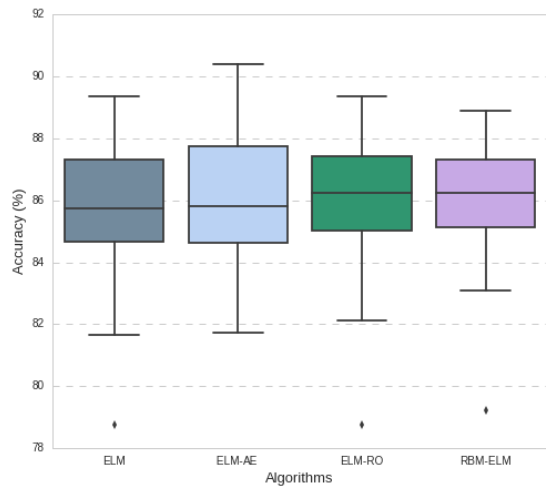- In the *overall*, the RBM-ELM is significantly different when compared to all others.

According to the statistical test and the accuracy performance described in Table 5, we conclude that for *alphabet*, *DNA*, and *madelon*, the RBM-ELM is the best algorithm; For the *gisette*, *isolet* and

Table 5: Algorithms performance for each database. In bold, the best algorithm(s) according to the statistical test
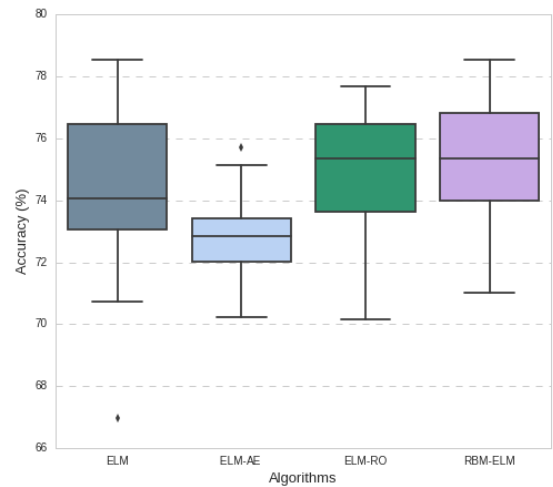
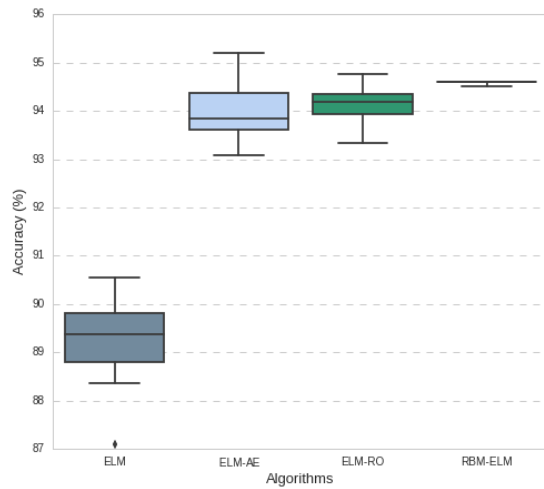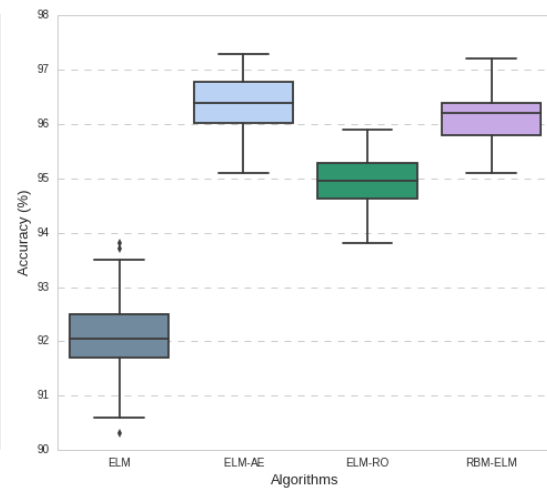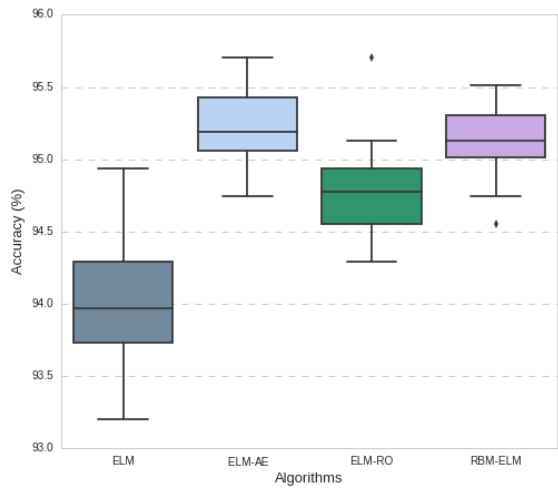| Database	ELM		ELM-AE		ELM-RO		RBM-ELM	
	Accuracy (%)	*Time (sec)*	*Accuracy (%)*	*Time (sec)*	*Accuracy (%)*	*Time (sec)*	*Accuracy (%)*	*Time (sec)*
Alphabet	72.784 ± 0.713	3.03	76.934 ± 0.777	12.22	73.285 ± 0.655	3.21	$\mathbf{78.677 \pm 0.611}$	38.74
Credit Australia	85.732 ± 2.292	0.004	86.025 ± 2.237	0.078	85.829 ± 2.280	0.005	86.070 ± 1.960	0.380
Diabetic	74.415 ± 2.562	0.010	72.726 ± 1.280	0.016	$\mathbf{75.033 \pm 1.866}$	0.011	$\mathbf{75.323 \pm 1.996}$	0.442
DNA	89.232 ± 0.827	0.146	93.931 ± 0.549	0.225	94.139 ± 0.364	0.152	$\mathbf{94.592 \pm 0.028}$	3.279
Gisette	92.130 ± 0.809	2.922	$\mathbf{96.383 \pm 0.479}$	11.804	94.953 ± 0.519	38.587	$\mathbf{96.116 \pm 0.459}$	160.23
Isolet	94.032 ± 0.385	3.738	$\mathbf{95.244 \pm 0.233}$	6.155	94.746 ± 0.298	3.831	$\mathbf{95.135 \pm 0.218}$	23.342
Madelon	55.393 ± 1.732	0.129	57.234 ± 1.429	0.253	66.145 ± 1.421	0.955	$\mathbf{82.286 \pm 1.139}$	9.706
Spam	91.178 ± 0.899	0.096	91.066 ± 0.684	0.175	90.826 ± 0.692	0.099	91.137 ± 0.696	1.715
Urban land cover	76.288 ± 2.860	0.044	$\mathbf{78.998 \pm 3.080}$	0.119	77.602 ± 2.523	0.045	$\mathbf{80.098 \pm 2.589}$	2.161
Overall mean/Total time	81.243 ± 1.453	10.119	83.171 ± 1.194	31.045	83.618 ± 1.179	46.895	$\mathbf{86.604 \pm 1.077}$	239.995

(a) Alphabet

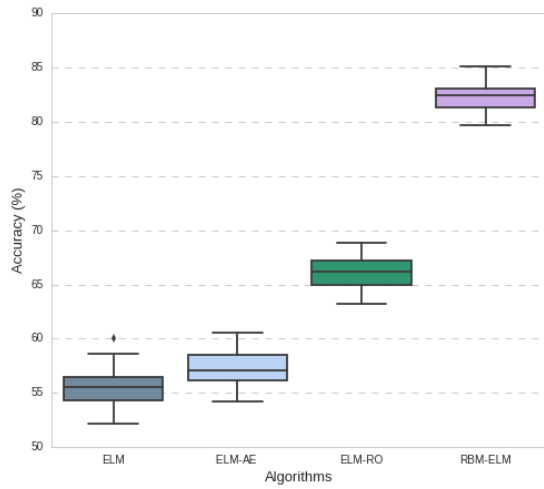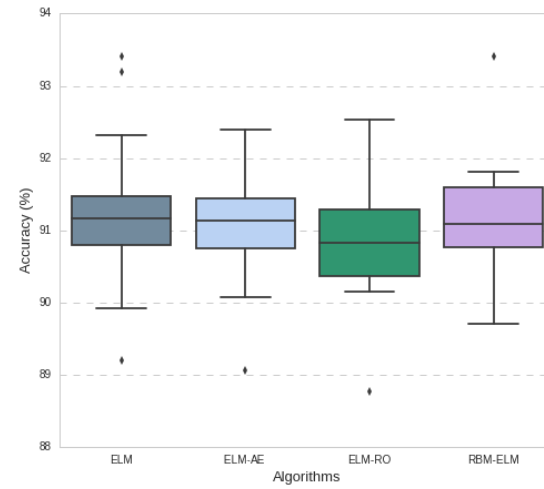(b) Credit Australia

(c) Diabetic
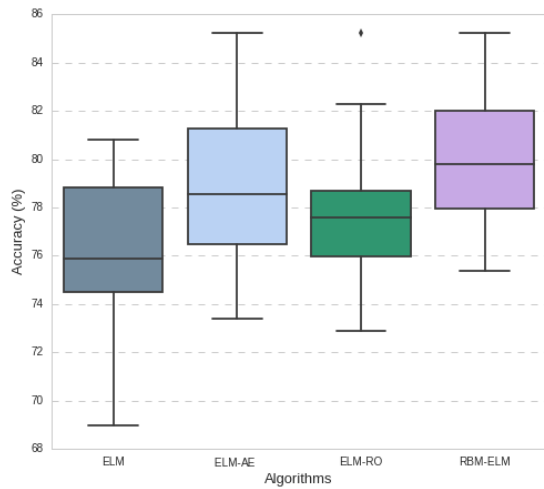
(d) DNA

(e) Gisette

(f) Isolet

(g) Madelon

(h) Spam

(i) Urban land cover

Figure 7: Boxplots for the algorithms performance for each database

urban cover land, both RBM-ELM and ELM-AE are the best approaches; And for the *diabetic*, the RBM-ELM and the ELM-RO are the best algorithms. Considering the overall result, for this group of benchmarks, our analysis indicates that the RBM-ELM is the best algorithm.

4.3 Experiments remarks

As we can see in sections 4.1 and 4.2, the RBM-ELM improved the standard ELM and has better performance than the ELM-AE and ELM-RO. Nonetheless, our approach presents the highest time-consuming. Indeed, it is a drawback, however, for the gisette database, the largest one in our experiment, the RBM-ELM spent on average 160.23 seconds to improve almost 4%, when compared to ELM. We consider it acceptable, since we have a good improvement on the final performance. Another issue about the RBM-ELM is the RBM parameters configuration. For some databases, we need to spend some time to find a good configuration. Unfortunately, to use the RBM we have to handle with this issue. Nonetheless, Hinton (2010) described a guide to setup it. Following this guide, we can reach good values for the RBM parameters in a faster way.

Although the statistical test points out that the RBM-ELM as the best algorithm for these databases, the ELM-AE and ELM-RO are still good approaches to improve the ELM. However, the RBM-ELM is more robust, since this approach is always in the group of the best algorithms for all databases. The ELM-AE and ELM-RO sometimes got bad performances such as in diabetic and alphabet databases, respectively. This does not occur with the proposed approach.

5 Conclusion

In this paper, we propose a new approach to determine the input weights for extreme learning machines (ELM) using the restricted Boltzmann machine (RBM), which we call RBM-ELM. In order to evaluate our new approach, we present an illustrative example detailing the RBM-ELM parameters configuration. Next, we carried out an experiment with standard benchmarks and compare the RBM-ELM performance with standard ELM, ELM autoencoder (ELM-AE) and a state of the art ELM random orthogonal (ELM-RO). The analysis of the results showed that the RBM-ELM was the best algorithm for the performed experiment and it was more stable than the other ones. On the other hand, our approach had the highest time-consuming among all algorithms. As we investigated, in this case it is worth to mention the trade-off between the improved accuracy and larger computational cost. In the future, we will work to improve the selection of the number of hidden neurons in the proposed approach.

References

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, 525–536.

Butcher, J., Verstraeten, D., Schrauwen, B., Day, C., Haycock, P., 2013. Reservoir computing and extreme learning machines for non-linear time-series data analysis. Neural networks 38, 76–89.

Cao, L.l., Huang, W.b., Sun, F.c., 2014. A deep and stable extreme learning approach for classification and regression. Proceedings of ELM-2014 Volume 1: Algorithms and Theories 3, 141.

Castaño, A., Fernández-Navarro, F., Hervás-Martínez, C., 2013. PCA-ELM: a robust and pruned extreme learning machine approach based on principal component analysis. Neural processing letters , 1–16.

Cervellera, C., Macciò, D., 2016. Low-discrepancy points for deterministic assignment of hidden weights in extreme learning machines. IEEE transactions on neural networks and learning systems 27, 891–896.

Derrac, J., García, S., Molina, D., Herrera, F., 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation 1, 3–18.

Han, F., Yao, H.F., Ling, Q.H., 2013. An improved evolutionary extreme learning machine based on particle swarm optimization. Neurocomputing 116, 87–93.

Henríquez, P.A., Ruz, G.A., 2017. Extreme learning machine with a deterministic assignment of hidden weights in two parallel layers. Neurocomputing 226, 109–116.

Hinton, G., 2010. A practical guide to training restricted Boltzmann machines. Momentum 9, 926.

Hinton, G.E., 2002. Training products of experts by minimizing contrastive divergence. Neural Computation 14, 1771–1800.

Hinton, G.E., Osindero, S., Teh, Y.W., 2006. A fast learning algorithm for deep belief nets. Neural Computation 18, 1527–1554.

Hinton, G.E., Salakhutdinov, R.R., 2006. Reducing the dimensionality of data with neural networks. Science 313, 504–507.

Huang, G.B., Ding, X., Zhou, H., 2010. Optimization method based extreme learning machine for classification. Neurocomputing 74, 155–163.

Huang, G.B., Zhou, H., Ding, X., Zhang, R., 2012. Extreme learning machine for regression and multiclass classification. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42, 513–529.

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

Huang, G.B., Zhu, Q.Y., Siew, C.K., 2006. Extreme learning machine: theory and applications. Neurocomputing 70, 489–501.

Kasun, L.L.C., Zhou, H., Huang, G.B., Vong, C.M., 2013. Representational learning with ELMs for big data. IEEE intelligent systems 28, 31–34.

Lichman, M., 2013. UCI machine learning repository. URL: http://archive.ics.uci.edu/ml.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning representations by back-propagating errors. Nature 323, 533–538.

Serre, D., 2002. Matrices: theory and applications. Springer, New York.

Shihabudheen, K., Pillai, G., 2017. Regularized extreme learning adaptive neuro-fuzzy algorithm for regression and classification. Knowledge-Based Systems 127, 100–113.

Smolensky, P., 1986. Information processing in dynamical systems: Foundations of harmony theory. Technical Report. University of Colorado at Boulder.

Sun, K., Zhang, J., Zhang, C., Hu, J., 2017. Generalized extreme learning machine autoencoder and a new deep neural network. Neurocomputing 230, 374–381.

Tissera, M.D., McDonnell, M.D., 2016. Deep extreme learning machines: supervised autoencoding architecture for classification. Neurocomputing 174, 42–49.

Wang, D., Wang, P., Ji, Y., 2015. An oscillation bound of the generalization performance of extreme learning machine and corresponding analysis. Neurocomputing 151, 883–890.

Wang, W., Liu, X., 2017. The selection of input weights of extreme learning machine: A sample structure preserving point of view. Neurocomputing . In press, https://doi.org/10.1016/j.neucom.2016.06.079.

Wang, Y., Cao, F., Yuan, Y., 2011. A study on effectiveness of extreme learning machine. Neurocomputing 74, 2483–2490.

Xu, S., Wang, J., 2017. Dynamic extreme learning machine for data stream classification. Neurocomputing 238, 433–449.

Yosinski, J., Lipson, H., 2012. Visually debugging restricted Boltzmann machine training with a 3d example, Representation Learning Workshop, 29th International Conference on Machine Learning, Edinburgh, Scotland, UK. p. 6.