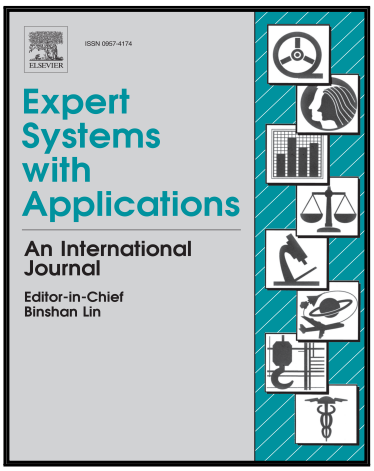



# Restricted Boltzmann machine to determine the input weights for extreme learning machines

Andre G.C. Pacheco, Renato A. Krohling, Carlos A.S. da Silva

PII: S0957-4174(17)30810-2
DOI: 10.1016/j.eswa.2017.11.054
Reference: ESWA 11697

To appear in: *Expert Systems With Applications*

Received date: 16 August 2017
Revised date: 7 November 2017
Accepted date: 28 November 2017

Please cite this article as: Andre G.C. Pacheco, Renato A. Krohling, Carlos A.S. da Silva, Restricted Boltzmann machine to determine the input weights for extreme learning machines, *Expert Systems With Applications* (2017), doi: 10.1016/j.eswa.2017.11.054

**Highlights**

- The extreme learning machine's weights initialization problem is tackled
- The restricted Boltzmann machine is used to determine the ELM input weights
- The proposed approach is applied to standard classification benchmarks
- The results present promising performance compared with other well-known approaches

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

Andre G. C. Pacheco[a], Renato A. Krohling[a,b], Carlos A. S. da Silva[a]

[a] *Graduate Program in Computer Science, PPGI, UFES - Federal University of Espírito Santo, Av. Fernando Ferrari 514, Vitória CEP: 29060-270, Brazil*
[b] *Production Engineering Department, UFES - Federal University of Espírito Santo, Av. Fernando Ferrari 514, Vitória CEP: 29060-270, Brazil*

**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 may affect the algorithm 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 to the well-known ELM-AE and to the ELM-RO, a state of the art algorithm to select the input weights for the ELM. The experimental results show that the RBM-ELM achieves a better performance than the ELM and outperforms the ELM-AE and ELM-RO.

*Keywords:* Neural networks, Extreme learning machine, Restricted Boltzmann machine, 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).

*Email addresses:* pacheco.comp@gmail.com (Andre G. C. Pacheco), krohling.renato@gmail.com (Renato A. Krohling), carlosalexandress@gmail.com (Carlos A. S. da Silva)

The ELM learning algorithm assigns the weights from the input layer to hidden layer ($\mathbf{W}$) randomly. Next, 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; Taormina and Chau, 2015; 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 input 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 kernel ELM (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 may 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 principal component analysis ELM (PCA-ELM), a deterministic algorithm proposed by Castaño et al. (2013), which is used to initiate the hidden neurons based on principal component analysis (PCA) and the parallel layers ELM (PL-ELM), an architecture developed by Henríquez and Ruz (2017), which is based on two parallel non-linear layers 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 random orthogonal (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 calculate the input 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 and Salakhutdinov, 2006). Therefore, the architecture presented by Tissera and McDonnell (2016), a stack of autoencoders to extract features from the data and set it as the ELM hidden nodes, can be done by using RBMs. In fact, 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 (Chen et al., 2017)[1]. 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 array, $\mathbf{y}$ is the output layer

[1]During the revision time, this conference paper was published with a similar idea.

array, $\mathbf{W}$ is the weights matrix of the input layer, $\boldsymbol{\beta}$ is the weights matrix of the hidden layer and $\mathbf{b}$ is the bias array 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.

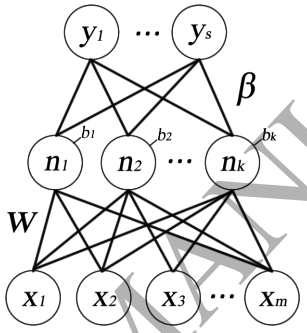



Figure 1: The architecture of a single-hidden layer feedforward neural network

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:

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

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of $\mathbf{H}$ (Serre, 2002; Huang et al., 2006). 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 minima as the gradient descent-based learning methods (Huang et al., 2006). A pseudocode of the ELM is presented in Algorithm 1.

**Algorithm 1:** Training an SLFN by the ELM procedure

**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 $\mathbf{b}$;
**if** *$\boldsymbol{W}$ and $\boldsymbol{b}$ are empty* **then**
  Initialize $\mathbf{W}$ and $\mathbf{b}$ according to $U(-1, 1)$;
**end**
Compute the feature map $\mathbf{H}$ from equation 2;
Compute $\boldsymbol{\beta}$ from equation 3;
**Return**: $\mathbf{W}$, $\mathbf{H}$ and $\boldsymbol{\beta}$;

### *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 $\mathbf{W}_r$ and each layer has its own bias, $\mathbf{a}$ and $\mathbf{c}$. During the training phase, the RBM learns the probability distribution over the input data training through unsupervised learning. Therefore, this network has been applied in different tasks, such as feature extraction, pattern recognition, dimensionality reduction, data classification etc (Hinton, 2010).

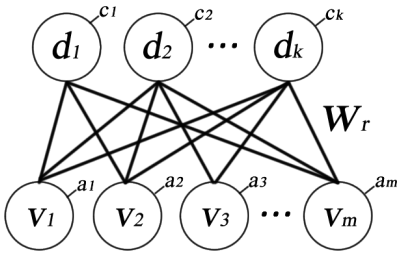



Figure 2: The architecture of a RBM with $m$ visible nodes and $k$ hidden nodes

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 (**v**,**d**) has an associated energy value defined by:

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

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

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

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 networks (Hinton, 2010).

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

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

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

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

where $G$ is the normal distribution with mean $v$ and standard deviation $\sigma^2$.

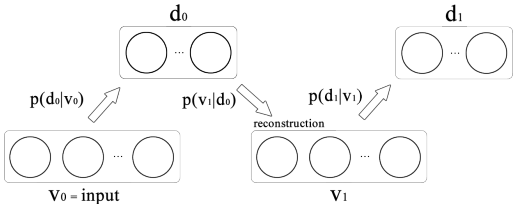



Figure 3: The Gibbs sampling procedure on CD algorithm

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:

$$\mathbf{W}_r^{t+1} = \mathbf{W}_r^t + \Delta\mathbf{W}_r^t \rightarrow \Delta\mathbf{W}_r^t = \eta(\mathbf{v}_0\mathbf{d}_0^T - \mathbf{v}_1\mathbf{d}_1^T) - \rho\mathbf{W}_r^t + \alpha\Delta\mathbf{W}_r^{t-1} \tag{8}$$

$$\mathbf{a}^{t+1} = \mathbf{a}^t + \Delta\mathbf{a}^t \rightarrow \Delta\mathbf{a}^t = \eta(\mathbf{v}_0 - \mathbf{v}_1) + \alpha\Delta\mathbf{a}^{t-1} \tag{9}$$

$$\mathbf{c}^{t+1} = \mathbf{c}^t + \Delta\mathbf{c}^t \rightarrow \Delta\mathbf{c}^t = \eta(\mathbf{d}_0 - \mathbf{d}_1) + \alpha\Delta\mathbf{b}^{t-1} \tag{10}$$

where $\eta$, $\rho$ and $\alpha$ 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

**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 $\eta$, $\rho$ and $\alpha$;
    The batch size $bs$;
    The maximum number of iterations $it$
  Sample $\boldsymbol{\theta}$ from a normal distribution with $\mu = 0.1$ and $\sigma = 1$;
  Split $\mathbf{X}$ in batches $\mathbf{X}_b$ with $bs$ samples;
  Initialize $i = j = 0$;
  **while** $j < it$ *or* $\boldsymbol{\theta}$ *converged* **do**
    **for** *each batch* $\mathbf{X}_b$ **do**
      **while** $i < bs$ **do**
        $\mathbf{v}_0 = \mathbf{X}_b^i$ Sample $\mathbf{d}_0$ from equation 6;
        Sample $\mathbf{v}_1$ from equation 7;
        Sample $\mathbf{d}_1$ from equation 6;
        Update $\boldsymbol{\theta}$ from equations 8, 9 and 10;
        $i = i + 1$
      **end**
    **end**
    $j = j + 1$
  **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.

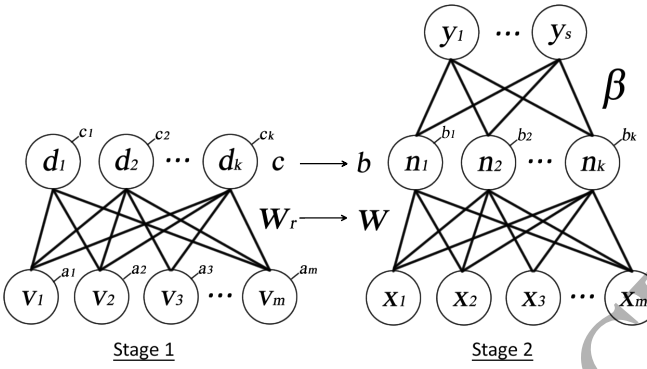



Figure 4: An illustration of the RBM-ELM approach

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 $\{\mathbf{W}, \mathbf{W}_r\}$ and $\{\mathbf{b}, \mathbf{c}\}$, we need to set the same number of hidden neurons for both algorithms. A pseudocode of this approach is described in Algorithm 3.

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 of the training data, when we compute the ELM weights and

**Algorithm 3:** The RBM-ELM procedure

**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 $\eta$, $\rho$ and $\alpha$;

From Algorithm 2 call rbm_training $(\mathbf{X}_t, k, \eta, \rho, \alpha)$ to get $\mathbf{W}_r$ and $\mathbf{c}$;

From Algorithm 1 call elm_training $(\mathbf{X}_t, \mathbf{Y}_t, k, \mathbf{W}_r, \mathbf{c})$ to get $\mathbf{H}$ and $\boldsymbol{\beta}$;

**Return**: $\mathbf{W}$, $\mathbf{H}$ and $\boldsymbol{\beta}$;

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.

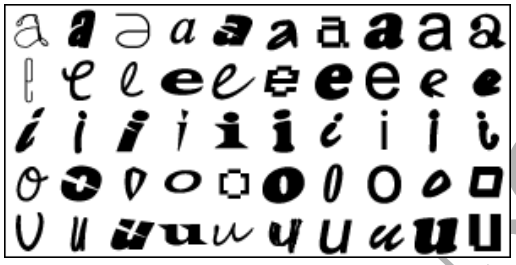


Figure 5: Some samples from the vowels database

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.

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.

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 $\eta$, $\rho$, $\alpha$ and $bs$, we use the values in the interval suggested by Hinton (2010), presented in section 2.2.

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

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

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 $\boldsymbol{\theta}$ 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.

| **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 | $90.708 \pm 1.518$ | $1.421 \pm 0.115$ |
| 30 | $90.442 \pm 1.144$ | $3.617 \pm 0.120$ |
| **50** | $\mathbf{92.769 \pm 1.078}$ | $\mathbf{6.008 \pm 0.099}$ |
| 100 | $90.724 \pm 1.303$ | $11.667 \pm 0.096$ |
| 300 | $90.611 \pm 1.420$ | $33.521 \pm 0.311$ |
| 500 | $89.067 \pm 1.441$ | $54.901 \pm 1.336$ |

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

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. Nonetheless, It is worth to note that we chose a setup that prioritizes the performance accuracy rather the computational time. If the need is the opposite, one could reduce the number of iterations and gets a lower computational time.

As we do not need to fine tune the RBM weights, a good strategy that can be adopted to reduce the computational time is to increase the RBM's learning rate and to decrease the number of iterations. Thus, the gradient step is bigger and weights can be taken nearly to the global minimum in a faster way. As such, keeping the same setup described in Table 2, we just increased the learning rate ($\eta$) to 0.1 and 0.05. Nonetheless, these values are too high and the model did not work with them. Next, we use $\eta = 0.005/0.01$ and the results for these values are described in Table 3. As we can note, indeed the computational time decreased. However, the model becomes more unstable and the standard deviation has increased. It happens because the higher the gradient step, the higher the probability of the algorithm gets stuck in a local minimum. However, if the computational time is a limiting factor, this strategy can be used to reduce it.

| **# of iterations (*it*)** | $\eta = 0.005$ | | $\eta = 0.01$ | |
|---|---|---|---|---|
| | **Accuracy (%)** | **Time (sec)** | **Accuracy (%)** | **Time (sec)** |
| 10 | $92.081 \pm 1.197$ | 1.377 | $92.611 \pm 1.694$ | 1.180 |
| 20 | $92.35 \pm 1.367$ | 2.318 | $92.512 \pm 1.489$ | 2.153 |
| 30 | $91.505 \pm 1.472$ | 3.734 | $92.499 \pm 1.381$ | 3.299 |
| 40 | $91.605 \pm 1.532$ | 4.778 | $91.987 \pm 1.22$ | 4.125 |

Table 3: The RBM-ELM performance when increasing the learning rate and decreasing the number of iterations

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.

To conclude, we present the graphics differences between the input weights of each model. From a trained ELM and RBM-ELM, we selected randomly three hidden neurons. Next, we got the connection weights between them to all input neurons. As we have 900 input neurons, we also have

900 connection weights. Thus, we reshape these weights into a $30 \times 30$ matrix and their plots are illustrated in Figure 6. 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.

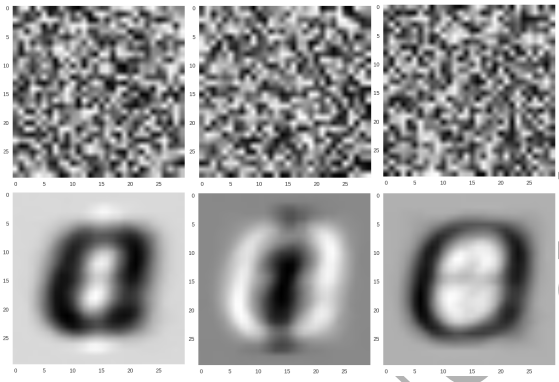


Figure 6: The plot of the input weights for the ELM (above) and RBM-ELM (below).

### *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 University of California Irvine (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. We decided to compare our algorithm with these approaches because of the following reasons: first, the ELM-RO (Wang and Liu, 2017) is a state-of-art approach that improves the ELM with a straightforward idea; second, the ELM-AE (Kasun et al., 2013) is an approach that uses a similar unsupervised learning algorithm to improve the ELM. So, it would be a suitable comparison. In Appendix A we describe the ELM-AE and present a brief comparison to the RBM-ELM.

All the databases used in this experiment is described in Table 4. 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).

| Database | # of samples | # of features | # of labels | Cardinality | Permutation |
|---|---|---|---|---|---|
| *Credit Australia* | 690 | 14 | 2 | 9660 | Yes |
| *Diabetic* | 1151 | 19 | 2 | 21869 | Yes |
| *DNA* | 3186 | 180 | 3 | 573480 | No |
| *Gisette* | 7000 | 5000 | 3 | 35000000 | No |
| *Isolet* | 7797 | 617 | 26 | 4810749 | No |
| *Madelon* | 2600 | 500 | 2 | 1300000 | Yes |
| *MNIST* | 70000 | 784 | 10 | 54880000 | No |
| *Spam* | 4601 | 57 | 2 | 262257 | Yes |
| *Urban land cover* | 675 | 147 | 9 | 99225 | Yes |

Table 4: The classifications databases used in this experiment

In Table 5 is described the RBM-ELM parameters setup for each database presented in Table 4. 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 5. In Table 6 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.

| Database | RBM-ELM parameters |
|---|---|
| **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 $bs = 50$ |
| **MNIST** | $k = 700$, $it = 25$, $\eta = 0.01$, $\rho = 0.001$, $\alpha = 0.5/0.9$ and $bs = 100$ |
| **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 $bs = 20$ |

Table 5: The RBM-ELM parameters setup 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, for these two databases, all four algorithms got the same performance.

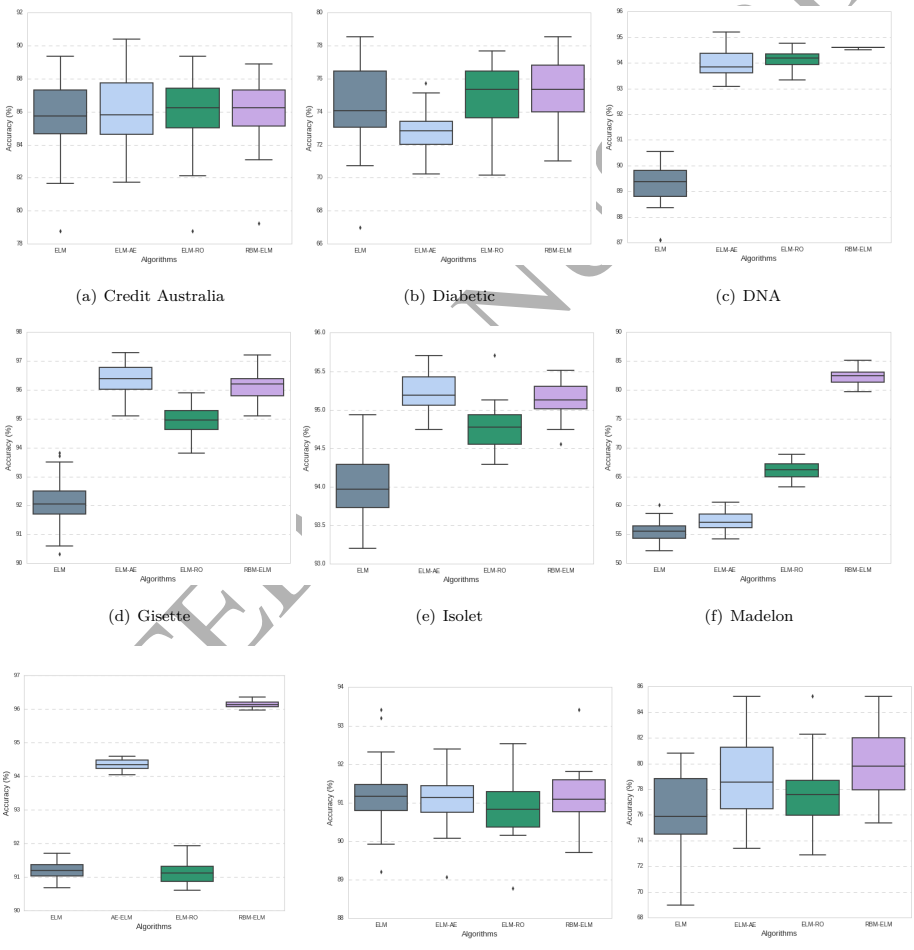
| Database | ELM | | ELM-AE | | ELM-RO | | RBM-ELM | |
|---|---|---|---|---|---|---|---|---|
| | *Accuracy (%)* | *Time (sec)* | *Accuracy (%)* | *Time (sec)* | *Accuracy (%)* | *Time (sec)* | *Accuracy (%)* | *Time (sec)* |
| **Credit Australia** | $85.732 \pm 2.292$ | 0.004 | $86.025 \pm 2.237$ | 0.078 | $85.829 \pm 2.280$ | 0.005 | $86.070 \pm 1.960$ | 0.380 |
| **Diabetic** | $74.415 \pm 2.562$ | 0.010 | $72.726 \pm 1.280$ | 0.016 | $\mathbf{75.033 \pm 1.866}$ | 0.011 | $\mathbf{75.323 \pm 1.996}$ | 0.442 |
| **DNA** | $89.232 \pm 0.827$ | 0.146 | $93.931 \pm 0.549$ | 0.225 | $94.139 \pm 0.364$ | 0.152 | $\mathbf{94.592 \pm 0.028}$ | 3.279 |
| **Gisette** | $92.130 \pm 0.809$ | 2.922 | $\mathbf{96.383 \pm 0.479}$ | 11.804 | $94.953 \pm 0.519$ | 38.587 | $\mathbf{96.116 \pm 0.459}$ | 160.23 |
| **Isolet** | $94.032 \pm 0.385$ | 3.738 | $\mathbf{95.244 \pm 0.233}$ | 6.155 | $94.746 \pm 0.298$ | 3.831 | $\mathbf{95.135 \pm 0.218}$ | 23.342 |
| **Madelon** | $55.393 \pm 1.732$ | 0.129 | $57.234 \pm 1.429$ | 0.253 | $66.145 \pm 1.421$ | 0.955 | $\mathbf{82.286 \pm 1.139}$ | 9.706 |
| **MNIST** | $91.191 \pm 0.251$ | 15.514 | $94.352 \pm 0.151$ | 36.642 | $91.454 \pm 0.341$ | 31.492 | $\mathbf{96.155 \pm 0.091}$ | 101.941 |
| **Spam** | $91.178 \pm 0.899$ | 0.096 | $91.066 \pm 0.684$ | 0.175 | $90.826 \pm 0.692$ | 0.099 | $91.137 \pm 0.696$ | 1.715 |
| **Urban land cover** | $76.288 \pm 2.860$ | 0.044 | $\mathbf{78.998 \pm 3.080}$ | 0.119 | $77.602 \pm 2.523$ | 0.045 | $\mathbf{80.098 \pm 2.589}$ | 2.161 |
| **Overall mean/Total time** | $83.288 \pm 1.402$ | 22.663 | $85.107 \pm 1.125$ | 55.467 | $85.636 \pm 1.145$ | 75.177 | $\mathbf{88.545 \pm 1.019}$ | 303.196 |

Table 6: Algorithms performance for each database. In bold, the best algorithm(s) according to the statistical test

However, for the remaining databases, the ELM was improved by at least one of the other 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 performance, alone or followed by another method for all database. As consequence, the RBM-ELM obtained the highest overall accuracy. However, it is also the most time-consuming 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 performance accuracy 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 *MNIST*, *DNA*, and *madelon*, the RBM-ELM is significantly different when compared to all others.
- For the *gisette*, *isolet* and *urban land cover*, there is no significant difference between the




(a) Credit Australia (b) Diabetic (c) DNA

(d) Gisette (e) Isolet (f) Madelon

(g) MNIST (h) Spam (i) Urban land cover

Figure 7: Boxplots for the algorithms performance for each database

ELM-RO 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 performance accuracy described in Table 6, we conclude that for *MNIST*, *DNA*, and *madelon*, the RBM-ELM is the best algorithm; For the *gisette*, *isolet* and *urban land cover*, 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. Results discussion*

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. In Figure 8 is presented the plot of all algorithms' computational time for each database. The databases are sorted according to their cardinality described in Table 4. Observing the plot, we can note the computational time increasing more for the RBM-ELM than the ELM. Indeed, it is a drawback, however, for the *MNIST* and *Gisette*, the databases with the highest cardinalities in our experiment, the RBM-ELM spent on average 101.94 and 160.23 seconds to improve almost 4% and 5%, respectively, when compared to ELM. We consider it acceptable, since we have a good improvement on the final performance and the time is manageable. In addition, it is important to note that we prioritize the performance accuracy in this experiment. We can reduce the computational time by applying the strategies described in section 4.1.

Still analyzing the plot in Figure 8 and still considering the MNIST and Gisette datasets, we can notice a different behavior between the ELM and RBM-ELM. For the ELM, the computational time increase from Gisette to MNIST. On the other hand, the opposite occurs from the RBM-ELM. As we can observe in Table 4, the *MNIST* has 70000 samples and 784 features, whereas the *Gisette* has 7000 samples and 5000 features. Therefore, the difference in the computational time happens because, for the ELM, the number of samples affects directly in the size of the matrix $\mathbf{H}$, whereas for the RBM-ELM, the number of features influences the Gibbs sampling process.

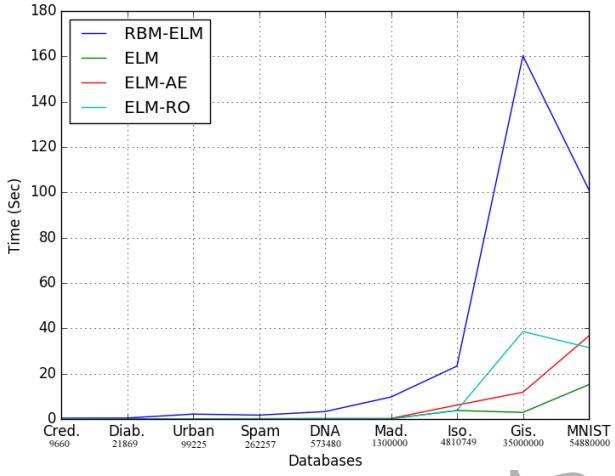



Figure 8: The algorithm's time-consuming according to the dataset's cardinality

Another issue about the RBM-ELM is the RBM parameters setup. 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 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 *MNIST* databases, respectively. This did 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 setup. Next, we carried out an experiment with standard benchmarks and compare the RBM-ELM performance with standard ELM, ELM autoencoder (ELM-AE) and the random orthogonal ELM (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 is the most 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 and adjusting it to tackle the time series prediction problem

**Acknowledgements:** A. Pacheco and C. da Silva would like to thank the financial support of the Brazilian agency CAPES and R. Krohling thanks the financial support of the Brazilian agency CNPq under grant nr. 309161/2015-0 and the local Agency of the state of Espirito Santo FAPES under grant nr. 039/2016.

## Appendix A. Extreme learning machine autoencoder (ELM-AE)

An autoencoder is a neural network that is trained to attempt to copy its input ($\mathbf{x}$) to its output ($\hat{\mathbf{x}}$) (Goodfellow et al., 2016). As shown in Figure A.9, this network is composed by an encoder, which transforms the input data into a code space ($\mathbf{c}$), and a decoder, which recovers the original data from the generated code. The aim of the autoencoder training phase is to minimize the reconstruction error $||\mathbf{x} - \hat{\mathbf{x}}||$. Thus, it can be achieved by using a learning algorithm, such as the backpropagation or the extreme learning machine, in which the output is set as the input.

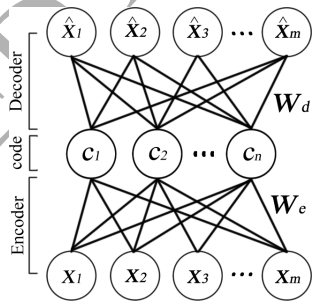



Figure A.9: The general structure of an autoencoder

In extreme learning machine autoencoder (ELM-AE) methodology, proposed by Kasun et al. (2013), the ELM training procedure is carried out twice. First, it is used to train an autoencoder to map its input into its output. Next, it is used to train a standard ELM using the transpose

matrix of the weights $\mathbf{W}_d$ obtained by the autoencoder network. In Figure A.10 is illustrated the ELM-AE.

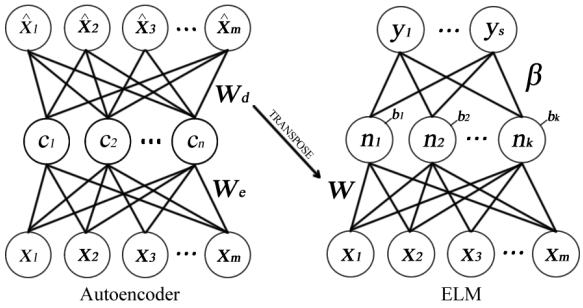



Figure A.10: An illustration of the ELM-AE approach

As we can note, the RBM-ELM and the ELM-AE share similar ideas to avoid the ELM random weights. In fact, the autoencoders and RBMs are quite similar. Both approaches are traditionally used for dimensionality reduction, feature learning, denoise, among others. In addition, both networks may work together for a general purpose (Hinton and Salakhutdinov, 2006). As the RBM needs to approximate the log-likelihood, sometimes it is harder to train. Nonetheless, the RBM explicitly estimates the probability distribution of the input data (Hinton, 2002), whereas the autoencoder does not. There is no clear winner between them, just general findings from particular applications (Deng et al., 2010; Vincent, 2011; Bengio et al., 2013). Therefore, choosing one approach rather other depends on the problem. In particular, according to the results presented in this paper, to initialize the ELM input weights, the RBM seems to be a better choice than the autoencoder.

## 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.

Bengio, Y., Yao, L., Alain, G., Vincent, P., 2013. Generalized denoising auto-encoders as generative models, in: Advances in Neural Information Processing Systems, pp. 899–907.

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, 141–150.

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.

Chen, L., Yang, L., Sun, C., Xi, H., 2017. A fast RBM-hidden-nodes based extreme learning machine, in: 29th Chinese Control and Decision Conference (CCDC), 2017, pp. 2121–2126.

Deng, L., Seltzer, M.L., Yu, D., Acero, A., Mohamed, A.r., Hinton, G., 2010. Binary coding of speech spectrograms using a deep auto-encode. INTERSPEECH , 1692–1695.

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.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Autoencoders, in: Deep Learning. MIT Press. chapter 14, pp. 499–523. http://www.deeplearningbook.org.

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., 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 315. University of Colorado at Boulder. CU Scholar.

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

Taormina, R., Chau, K.W., 2015. Data-driven input variable selection for rainfall–runoff modeling using binary-coded particle swarm optimization and extreme learning machines. Journal of hydrology 529, 1617–1632.

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

Vincent, P., 2011. A connection between score matching and denoising autoencoders. Neural computation 23, 1661–1674.

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.