Software based Method to Specify the Extreme Learning Machine Network Architecture Targeting Hardware Platforms

Alaa M. Abdul-Hadi
Department of Computer Engineering
University of Baghdad

Abdullah M. Zyarah
Department of Electrical Engineering
University of Baghdad

Haider M. Abdul-Hadi
Department of Computer Science
University of Baghdad

ABSTRACT

Extreme learning machine (ELM) is a biologically inspired feed-forward machine learning algorithm that offers a significant training speed. Typically, ELM is used in classification applications, where achieving highly accurate results depend on raising the number of ELM hidden layer neurons, which are randomly weighted independently of the training data and the environment. To this end, determining the rational number of hidden layer neurons in the extreme learning machine (ELM) is an approach that can be adapted to maintain the balance between the classification accuracy and the overall physical network resources. This paper proposes a software based method that uses gradient descent algorithm to determine the rational number of hidden neurons to realize an application specific ELM network in hardware. The proposed method was validated with MNIST standard database of hand-written digits and human faces database (LFW). Classification accuracy of 93.4% has been achieved using MNIST and 90.86% for LFW database.

General Terms

Neural Networks, Classification Applications

Keywords

Extreme Learning Machine, Gradient Descent, Random feature mapping

1. INTRODUCTION

Inspired by the sophisticated capabilities of the biological human brain, the extreme learning machine is introduced by Huang el al [1,2,3], as a machine learning algorithm that overcomes the main challenges faced in other machine learning techniques, such as low learning speed and human intervention during the learning process. Unlike other machine learning algorithms, extreme learning machine offers a significant training speed. This is achieved by confining the neurons weights tuning to the output layer only and leaving the hidden layer(s) neurons weights un-tuned after initializing them randomly and independently on the training data and environment. This is based on the conjecture that there are neurons in the live brain randomly parametrized independently of the environment and this has been evidently proved in [4]. Having such network structure reduces ELM complexity and makes it more suitable for hardware mapping especially in classification applications.

Fundamentally, the ELM neural network consists of three layers: Input, hidden, and output layer. The input layer is utilized to introduce the input data, training or testing examples, to the network. These data are transferred to *N*-dimensional_space in the next layer, which is also called (hidden layer or feature mapping layer [5]), where the input features are represented in a more meaningful way [6]. The hidden layer output is relayed to the last layer in the network (output layer), where it gets evaluated. Determining the number of neural units in each layer depends on the task being performed by that layer. The input layer units are set depending on the number of features represented by input examples, while the number of classes that need to be classified determine the number of output layer units. Selecting the number of hidden layer units depends on data variance. The more variance in input data, the larger network is required to improve performance. Typically, the ELM algorithm comes either with or without a kernel. In the kernel model version, which is unlike the basic version (without a kernel), there is no need to set the number of hidden layer neurons. Also, it is hardware dependent, i.e., when the database is large, a computer with a massive memory unit is required to run the algorithm, as will be shown in the experimental results section. For the above reasons, the basic version is recommended to be used when it comes to the normal computer users. The main issue in the basic version, is that the number of hidden layer neurons is set manually, and it is computationally extensive especially when the training data has a large number of features.

The main contribution of this work is to develop a software based method that determines the network architecture parameters, such as the number of hidden neurons and the activation function in a way that keeps the balance between the accuracy and the size of the network. The proposed method, which is based on the classification accuracy curve, exploits the logarithmic rise of curve to precisely predict the future network performance. This can be used to select the network architecture parameters that minimize the used hardware resources and produces high classification accuracy.

This paper is organized as follows: section 2 reviews the ELM algorithm, section 3 presents the proposed approach to determine rational number of hidden layer neurons in ELM. The validation database and the obtained results are discussed in section 4, while section 5 concludes the paper.

2. REVIEW OF ELM

The ELM is a promising learning algorithm that can be used as an effective technique in classification applications. It is resulted from combining the support vector machine (SVM) and the feed-forward neural networks. This section reviews two types of ELM which are categorized based on the structure of the network.

2.1 Single Layer ELM

The single layer ELM has a single hidden layer and it is also known as a single layer feed-forward network (SLFN). The general architecture of this network is composed of an input layer where the data is fetched to the network, a feature mapping layer that effectively extracts the most important and relevant features out of the presented data, and an output layer where the output of the network gets evaluated. Unlike other neural networks, the weights of the ELM hidden layer neurons need not to be tuned during the learning stage. The training process in the ELM is confined to the output layer weights. The training starts by randomly initializing the hidden layer weights independently of the training data. The feed-forward of the network is then computed by finding the activation function output of the hidden units and the output units as well. Tuning the output layer weights is the step followed to achieve the generalization feature of the network.

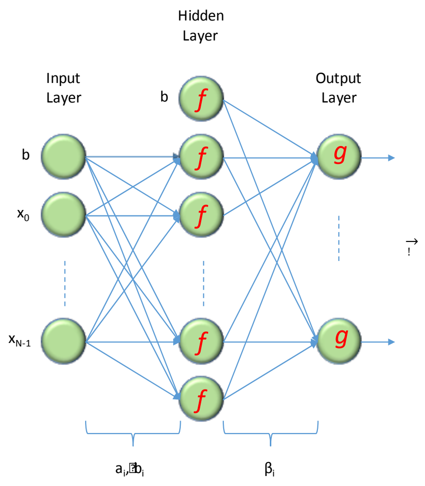

Fig. 1. Single layer feed-forward network. It is composed of three layers: input layer (has no activation function), feature mapping layer (hidden layer), and output layer.

The training process begins by presenting the training examples as a pair composes a training example x_i and class label t_i, (x_i, t_i), where $x_i \in R^n$, $t_i \in R^m$, and ($i=1, 2, ..., NH$). Assuming that the number of hidden units NH, and g_i, f_i are hidden and output layer activation functions, respectively. From Figure 1, the feed-forward output of the network can be computed using Equation 1.

$$t_i=g_i(\sum_{i=1}^{NH} ß_i\, f_i\,(X,b)) \qquad (1)$$

One approach to compute the optimal network parameters is based on using the normal equation as stated in [6]. The output weight is computed using Equation 2, where $ß$ denotes the output layer weight, H refers to the hidden layer output, and T is output layer vector.

$$ß = H^{-1} * T \qquad (2)$$

As in Equation 2, the output layer weight vector is resulted from multiplying the Moore-Penrose generalized inverse of the hidden layer output matrix H by the final network output vector T. Efficiently, the inverse of the matrix H can be found using the orthogonal projection method where $H^{-1} = (H^T H)^{-1} H^T$ [7]. As a result, the output weight can be computed as in equation Equation 3.

$$ß = (H^T H)^{-1} H^T T \qquad (3)$$

This works perfectly when there are a small number (can not be guaranteed) of hidden neurons. Finding the inverse of this term, $(H^T\ H)^{-1}$ is a computationally extensive, since this matrix is going to be very large in size as the number of hidden neurons grows up and the number of classes needed to be classified increases. In terms of software, a workstation with a massive memory is required to set the ELM network parameters as being used in [7]. When it comes to hardware, finding the inverse of a matrix requires a large number of resources as in [8, 9]. An alternative to compute the parameters of the ELM machine without finding the matrix inversion is using gradient descent. Although the gradient descent is an iterative based algorithm, in the sense that it takes a decent amount of time to set the network parameters, it is efficient in terms of hardware. Therefore, it can be employed to specify the general network architecture such as the number of hidden neurons and layers prior to implement the network in hardware.

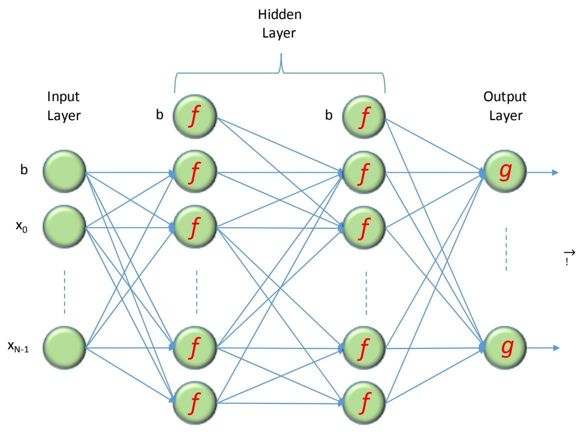

Fig. 2. Multi layer feed-forward network. It is structured from a single input layer, two hiddens and single output layers.

2.2 Multi-Layer ELM

The ELM architecture can be further extended to form a more complex architecture that may improve the general network performance. This is accomplished by raising the number of hidden layers while fixing hidden neurons count in each layer. Similar to the SLFN, the weights of hidden layer neurons are not tuned during the learning process and the output of each hidden layer is computed analytically as in Equation 4 [6]. Determining the number of hidden layers that may improve the network performance is still hard to guess, as there is no particular mathematical model or method can be employed for this purpose. To this end, exploiting the iterative increment in the network accuracy offered by the gradient descent algorithm can be used to guess the number of hidden layers which offer best network performance.

$$H^n = f\,[(ß^n)^T\, H^{n-1}] \qquad (4)$$

3. ELM AND OPTIMIZATION METHOD

3.1 Gradient Descent ELM Algorithm

In this paper, it is suggested to use gradient descent based ELM algorithm to specify several network architecture parameters such as the number of hidden units, layers, and activation function that shape the general architecture of the ELM application specific algorithm targeting hardware platforms. The reason behind using the gradient descent instead of the normal equation, is attributed to that the gradient descent work iteratively unlike the normal equation which works analytically. This facilitates observing the accuracy of training and testing classification results which increase in each iteration logarithmically. The preliminary results of the classification accuracy curve can be exploited to predict the future network performance and to determine the network architecture parameters that gives high accuracy and minimizes the used physical resources.

In the ELM that works based on the gradient descent, the weight is updated iteratively. This is performed after computing the network output in each iteration by propagating the input through the network as in Equation 1. Then, the output layer weights are updated for learning as in Equation 5 [10]. In this way, the weights converge to a value which yields high network accuracy.

$$\Delta\beta^{(p)}=\partial x_i^{(p)}\,(t^{(p)} - t^{`(p)}) \qquad (5)$$

where $t^{(p)}$ and $t^{`(p)}$ represent the expected and achieved outputs respectively; ∂ represents the learning rate; $x_i^{(p)}$ represents the input feature, and p refers to the number of training/testing example.

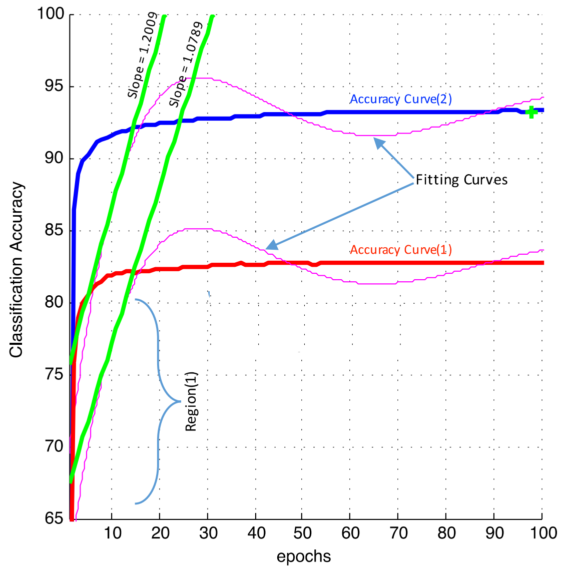

Fig. 3. Classification accuracy for a single hidden layer ELM with NH=220 (Blue) and NH=100 (Red). The classification accuracy curves are fitted using a 6th order polynomial curve that its slope at region (1) being determined by a tangent line touching the curve at the early stage of the training process.

3.2 Optimization Method

In order to set the final network architecture, the following algorithm is used:

Input: TestAccuracy $\in R^n$, polynomial equation order *rd*

Output: AccuracyCurveSlope $\in R^n$

```
1.  $e = 1:20$
2.  $f$ = PloyParameters($e$, TestAccuracy, $rd$)
3.  $y = (f.p1) * e^{rd} + (f.p2) * e^{rd-1} + (f.p3) * e^{rd-2} + \ldots$
4.  $\Delta y = \Delta y / \Delta e$
5.  $k = 5$
6.  tangentLine $= (e - e(k)) * \Delta y(k) + y(k)$
7.  AccuracyCurveSlope = Slope(tangentLine)
```

This algorithm facilitates the process of setting network architecture via finding the predicted network accuracy based on computed AccuracyCurveSlope vector (region (1)). Based on the slope value (shown in Figure 3) which indicates the high accuracy as it goes up, the number of neurons in the hidden layers is determined. Typically, the algorithm starts with a small number of hidden neurons that increases by a certain amount specified by the user in every iteration. After a particular set of iterations, the change in accuracy is evaluated to see whether a significant change in the accuracy has occurred. The same strategy is followed to set the rest of network parameters.

4. VERIFICATION DATASET AND THE EXPERIMENTAL RESULTS

4.1 Training and Testing Sets

To verify the validity of the proposed method in setting the ELM network architecture parameters prior to the hardware realization MNIST [11] standard database of hand-written digits and labeled faces in the wild (LFW) [12] are used to train and test the network

4.1.1 MNIST Standard Database

It is composed of 60,000 training and 10,000 testing examples. Each example is presented to the network as an image pixel vector that has 784 pixels. This can be reshaped to restore the original image which has 28x28 pixels as shown in Figure 4. Prior to fetching these images to the network, all image pixels are preprocessed to be ranged between [-1,1]. This is to optimize the network performance and achieve a faster convergence.

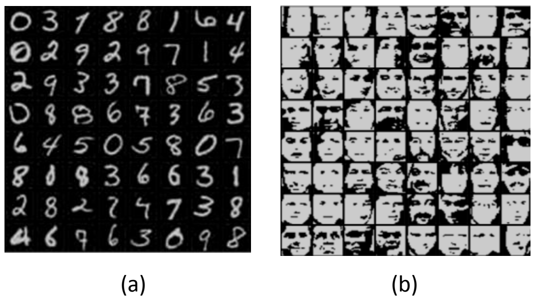

Fig. 4. A part of MNIST (a) and LFW images (b) arranged in an array. Each cell in the array represents a training or testing example.

4.1.2 LFW Database

Cross validation with the labeled faces in the wild (LFW), Figure 4, is performed to validate the network performance. The LFW is a database of face photographs designed to recognize the unconstrained faces. This database contains 13,000 images of human faces collected from the web. Prior to using this database for training and testing the ELM network, these images are preprocessed by cropping the images to have the face region only (remove the background). Then, as in MNIST database, all images' pixels are preprocessed to be ranged between [-1,1].

4.2 Experimental Results

4.2.1 Classification Accuracy

One of the main challenges in the ELM-kernel is that, it requires a computer with a massive memory unit to set the network parameters. This is the case when the proposed database is too large as MNIST. This has been proved by running the ELM-kernel MATLAB code provided in [13] on a FUJITSU laptop with Intel(R) Core(TM) i5-2450M 2.50GHz CPU, 4.0GB RAM, Windows 8.1 Enterprise using MATLAB R2015a which shows "Out of Memory" error, mentioning that a computer with at least 26GB of memory is needed to run this algorithm. This explains the reason behind using HP z820 workstation in [7] and here it comes the merit of the gradient descent based ELM.

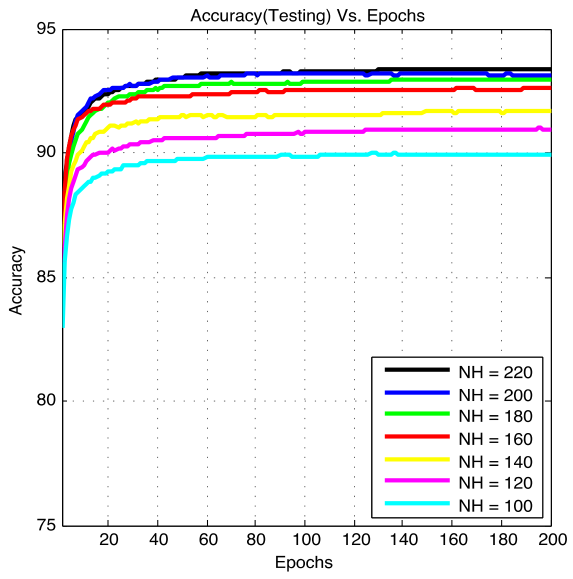

Fig. 5. MNIST classification accuracy of the gradient descent based ELM for a single hidden layer ELM with different number of hidden neurons.

Figure 5 demonstrates the classification accuracy of a single layer ELM with various numbers of hidden layer neurons. As can be seen, the classification accuracy is increasing logarithmically as the number of epochs raises. This is attributed to the fact that the network generalizes better as it is exposed to more training examples. Observing the classification curve reflects that there are two main regions close to be linear. The first region has a sharp rise in the classification accuracy in vertical horizon, whereas the second region changes are limited to horizontal horizon which tends to be unuseful compared to the first region, since the changes in accuracy tend to be negligible. By finding the fitting curve with a high order polynomial equation (4-6 orders) and the slope of its tangent line at early epochs as in Figure 3, the final accuracy of the network can be predicted.

Exploiting this curve shape in predicting the final accuracy of the network and eventually setting the network architecture parameters at early stage of hardware realization can save a reasonable amount of physical resources. The results illustrate that the classification accuracy goes up as the network expands in size in terms of hidden neurons and layers, but this is not always the case, especially when it comes to the number of hidden layers. Increasing the number of hidden layers may or may not improve the network classification accuracy and this depends on the feature extraction adapted technique as it has been already proved in [14].

4.2.2 Performance Evaluation

The proposed ELM is evaluated with respect to the ELM-Basic algorithm proposed in [13] and state-of-art support vector machine (SVM) for both MNIST and LFW dataset. It has been found that, although these algorithms are much faster than the proposed method, the classification accuracy tends to be very close (as shown in Figure 7), and the predicted hardware resources that are used, are much less relative to the original ELM since there are no extensive computations in the gradient descent based ELM.

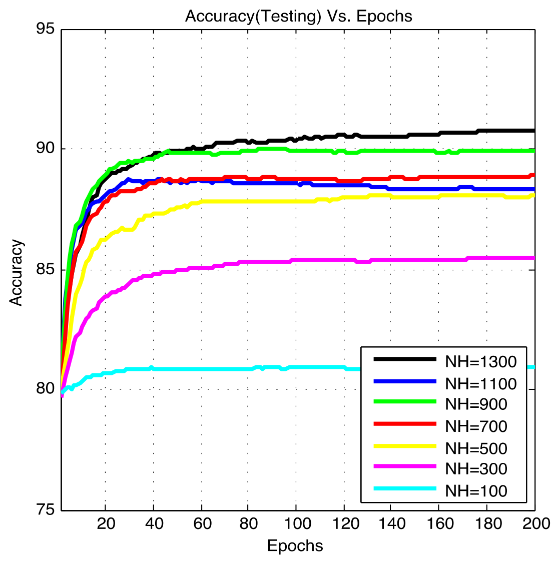

Fig. 6. LFW classification accuracy of the gradient descent based ELM for a single hidden layer ELM with different number of hidden neurons.

5. CONCLUSION

The main contribution of this paper is to propose a software based method that utilizes the gradient descent algorithm to choose the rational number of hidden neuron units and layers that improve the network performance and keep the hardware resources to a minimum. The suggested technique is verified with MNIST and LFW standard databases and the experimental results demonstrate the effectiveness of the proposed method. This work can be extended in the future to include optimization techniques such as the genetic algorithm and compare it with the gradient descent in terms of speed and performance.

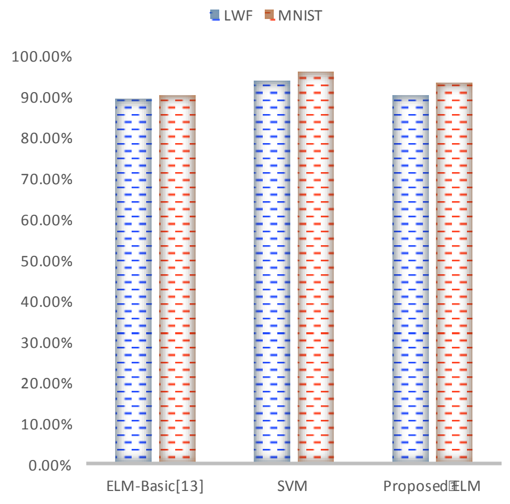

Fig. 7. Classification Accuracy for MNIST and LFW databases using the state-of-art SVM, ELM-Basic model, and the proposed version.

6. REFERENCES

[1] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," Neurocomputing, vol. 70, no. 1, pp. 489–501, 2006.

[2] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 42, no. 2, pp. 513–529, 2012.

[3] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on, vol. 2. IEEE, 2004, pp. 985–990.

[4] D. L. Sosulski, M. L. Bloom, T. Cutforth, R. Axel, and S. R. Datta, "Distinct representations of olfactory information in different cortical centres," Nature, vol. 472, no. 7342, pp. 213–216, 2011.

[5] G.-B. Huang, "An insight into extreme learning machines: random neurons, random features and kernels," Cognitive Computation, vol. 6, no. 3, pp. 376–390, 2014.

[6] R. Kumar Roul, A. Nanda, V. Patel, and S. Kumar Sahay, "Extreme learning machines in the field of text classification," in Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on. IEEE, 2015, pp. 1–7.

[7] J. Tang, C. Deng, G.-B. Huang, and J. Hou, "A fast learning algorithm for multi-layer extreme learning machine," in Image Processing (ICIP), 2014 IEEE International Conference on. IEEE, 2014, pp. 175–178.

[8] A. Irturk, S. Mirzaei, and R. Kastner, An efficient FPGA implementation of scalable matrix inversion core using QR decomposition. Department of Computer Science and Engineering, University of California, San Diego, 2009.

[9] G. A. Kumar, T. V. Subbareddy, B. M. Reddy, N. Raju, and V. Elamaran, "An approach to design a matrix inversion hardware module using fpga," in Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on.IEEE, 2014, pp. 87–90.

[10] J. Tapson, P. de Chazal, and A. van Schaik, "Explicit computation of input weights in extreme learning machines," in Proceedings of ELM-2014 Volume 1. Springer, 2015, pp. 41–49.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," Technical Report 07-49, University of Massachusetts, Amherst, Tech. Rep., 2007.

[13] Extreme learning machine. [Online]. Available: http://www.ntu.edu.sg/home/egbhuang/elmcodes.htm

[14] P.-Z. Zhang, S.-X. Zhao, and X.-Z. Wang, "The failure analysis of extreme learning machine on big data and the counter measure," in Machine Learning and Cybernetics (ICMLC), 2015 International Conference on, vol. 2. IEEE, 2015, pp. 849–853.