Representational Learning with Extreme Learning Machine for Big Data

Liyanaarachchi Lekamalage Chamara Kasun, Hongming Zhou, Guang-Bin Huang and Chi Man Vong

***Abstract*—Restricted Boltzmann Machines (RBM) and auto encoders, learns to represent features in a dataset meaningfully and used as the basic building blocks to create deep networks. This paper introduces Extreme Learning Machine based Auto Encoder (ELM-AE), which learns feature representations using singular values and is used as the basic building block for Multi Layer Extreme Learning Machine (ML-ELM). ML-ELM performance is better than auto encoders based deep networks and Deep Belief Networks (DBN), while in par with Deep Boltzmann Machines (DBM) for MNIST dataset. However ML-ELM is significantly faster than any state−of−the−art deep networks.**

***Index Terms*—Extreme learning machine, Deep Networks, Representational Learning**

I. Introduction

THE generalization capability of a machine learning algorithm depends upon the features of the dataset. Hence engineering the features to represent the salient structure of the data is important. However, feature engineering requires domain knowledge and human ingenuity to generate appropriate features. Hinton, et. al [1] showed that Restricted Boltzmann Machine (RBM) and auto encoders perform feature engineering and can be used to train multiple-layer neural networks. These multiple-layer neural networks are called as deep networks. There are two types of deep networks based on RBM, which are called as Deep Belief Network (DBN) [1] and Deep Boltzmann Machine (DBM) [2]. There are two types of auto encoder based deep networks, which are Stacked Auto Encoders (SAE) [3] and Stacked Denoising Auto Encoders (SDAE) [3]. DBN and DBM are created by stacking RBMs while SAE and SDAE is created by stacking auto encoders. Deep networks outperforms traditional multiple-layer neural networks, Single Layer Feed-forward Neural-networks (SLFN) and Support Vector Machines (SVM) for big data, but are tainted by slow learning speeds.

Extreme Learning Machine (ELM) introduced by Huang, et. al [5] is a SLFN with fast learning speed and good generalization capability. Similar to deep networks, Multi-Layer Extreme Learning Machine (ML-ELM) proposed in this paper performs layer by layer unsupervised learning. This paper introduces Extreme Learning Machine Auto Encoder (ELM-AE) which represents features based on singular values. Resembling deep networks, ML-ELM stacks ELM-AE to create a multi layer neural network.

ML-ELM on the other hand learns significantly faster than existing deep networks while outperforming DBN, SAE, SDAE, and performing in par with DBM for the MNIST [4] dataset.

L.L.C. Kasun, H. Zhou and G.-B. Huang are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798. E-mail: {chamarak001,hmzhou,egbhuang}@ntu.edu.sg.

C.M. Vong is with the Department of Computer and Information Science, University of Macau, Macau. E-mail: cmvong@umac.mo

II. Brief of ELM

Extreme Learning Machine (ELM) proposed by Huang, et. al [5] for SLFN shows that the hidden nodes can be randomly generated. The input data is mapped to L dimensional ELM random feature space and the network output is given by Equation (1):

$$f_L(\mathbf{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} \tag{1}$$

where $\boldsymbol{\beta} = [\beta_1, \cdots, \beta_L]^T$ is the output weight matrix between the hidden nodes and the output nodes. $\mathbf{h}(\mathbf{x}) = [\mathrm{g}_1(\mathbf{x}), \cdots, \mathrm{g}_L(\mathbf{x})]$ are the hidden node outputs (random hidden features) for the input $\mathbf{x}$ and $g_i(\mathbf{x})$ is the output of the i-th hidden node. Given N training samples $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{\mathrm{N}}$, ELM is to resolve the following learning problems:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \tag{2}$$

where $\mathbf{T} = [\mathbf{t}_1, \cdots, \mathbf{t}_N]^T$ are the target labels and $\mathbf{H} = [\mathbf{h}^{\mathrm{T}}(\mathbf{x}_1), \cdots, \mathbf{h}^{\mathrm{T}}(\mathbf{x}_N)]^{\mathbf{T}}$. The output weights $\boldsymbol{\beta}$ can be calculated by Equation (3):

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

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of matrix H.

To have better generalization performance and to make the solution more robust, one can add a regularization term as shown in [7].

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1} \mathbf{H}^T\mathbf{T} \tag{4}$$

III. Representation Learning With ELM

A. *Extreme Learning Machine as an Auto Encoder(ELM-AE)*

The main objective of ELM-AE is to represent the input features meaningfully in three different representations.

Compressed representation
: Represent features from a higher dimensional input data space to a lower dimensional feature space.

Sparse representation
: Represent features from a lower dimensional input data space to a higher dimensional feature space.

Equal dimension representation
: Represent features from a input data space dimension equal to feature space dimension.

ELM is modified as follows to perform unsupervised learning: input data is used as output data $\mathbf{t}$=$\mathbf{x}$, random weights and random biases of the hidden nodes are chosen to be orthogonal. Experiments performed by the authors based on a different ELM network indicate that, even when the random weights and random biases are not orthogonal, ELM-AE is capable of learning a useful feature representation. However by choosing the random weights and random biases to be orthogonal higher performance tends to be achieved.

According to ELM theory ELM is an universal approximator [8], hence ELM-AE is also a universal approximator. The network structure of ELM-AE for compressed, sparse and equal dimension representation is shown in Figure 1. In ELM-AE the orthogonal random weights and biases of the hidden nodes project the input data to a different or equal dimension space as shown by Johnson-Lindenstrauss Lemma [6] and calculated by Equation (5):

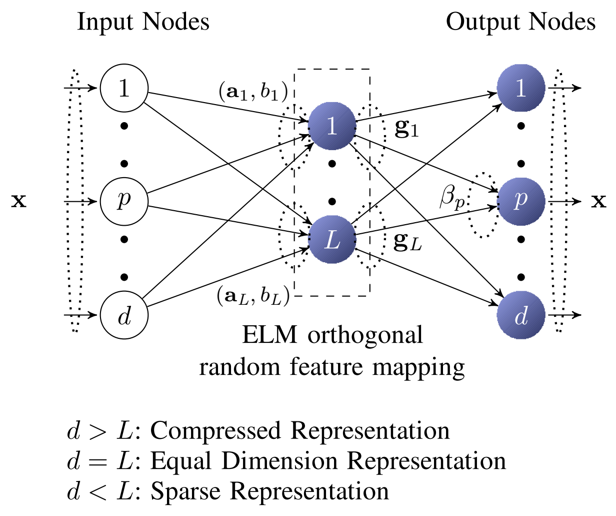


Figure 1. ELM-AE has the same solution as the original ELM except for: 1) The target output of ELM-AE is the same as input $\mathbf{x}$; 2) The hidden node parameters $(\mathbf{a}_i, b_i)$ are made orthogonal after randomly generated. $g_i(\mathbf{x}) = g(\mathbf{a}_i, b_i, \mathbf{x})$ in the i^{th} hidden node for input $\mathbf{x}$.

$$\begin{aligned} \mathbf{h} &= \mathrm{g}(\mathbf{ax} + \mathrm{b}) \\ \mathbf{a}^T\mathbf{a} &= \mathbf{I}, \mathrm{b}^T\mathrm{b} = 1 \end{aligned} \tag{5}$$

Where $\mathbf{a} = [\mathbf{a}_1, \cdots, \mathbf{a}_L]$ are the orthogonal random weight and $\mathbf{b} = [b_1, \cdots, b_L]$ are the orthogonal random bias between the input nodes and hidden nodes.

The output weights $\boldsymbol{\beta}$ of ELM-AE is responsible of learning the transformation from the feature space to input data. For sparse and compressed ELM-AE representations, output weights $\boldsymbol{\beta}$ are calculated by Equation (6):

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1} \mathbf{H}^T\mathbf{X} \tag{6}$$

Where $\mathbf{H} = [\mathbf{h}_1, \cdots, \mathbf{h}_N]$ are the hidden layer outputs of ELM-AE and $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]$ are input data and output data of ELM-AE.

For equal dimension ELM-AE representations, output weights $\boldsymbol{\beta}$ are calculated by Equation (7):

$$\begin{aligned} \boldsymbol{\beta} &= \mathbf{H}^{-1}\mathbf{X} \\ \boldsymbol{\beta}^T\boldsymbol{\beta} &= \mathbf{I} \end{aligned} \tag{7}$$

The Singular Value Decomposition (SVD) of Equation (6) is shown in Equation (8):

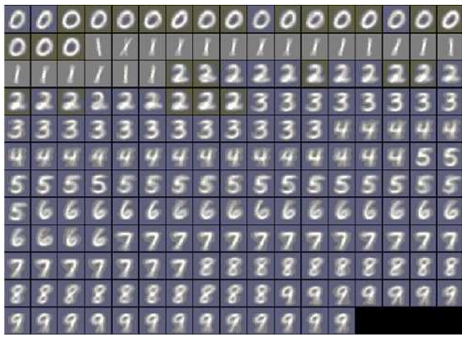
$$\mathbf{H}\boldsymbol{\beta} = \sum_{i=1}^{N} \mathbf{u}_i \frac{\mathbf{d}_i^2}{\mathbf{d}_i^2 + C} \mathbf{u}_i^T \mathbf{X} \tag{8}$$

where $\mathbf{u}$ are the eigenvectors of $\mathbf{HH}^T$, $\mathbf{d}$ are the singular values of $\mathbf{H}$, related to the SVD of input data $\mathbf{X}$. Because $\mathbf{H}$ is the projected feature space of $\mathbf{X}$ squashed via a sigmoid function, we hypothesize that the output weight $\boldsymbol{\beta}$ of ELM-AE will be learning to represent the features of the input data via singular values. To test whether our hypothesis is correct we create 10 mini datasets containing digits 0 to 9 from the MNIST dataset. Next each mini dataset is sent through a ELM-AE (network structure: 784-20-784) and the contents of the output weights $\boldsymbol{\beta}$ (Figure 2(a)) are compared with the manually calculated rank 20 SVD basis (Figure 2(b)) of each mini dataset. According to Figure 2 it can be seen that ELM-AE output weights $\boldsymbol{\beta}$ is better than the manually calculated SVD basis.

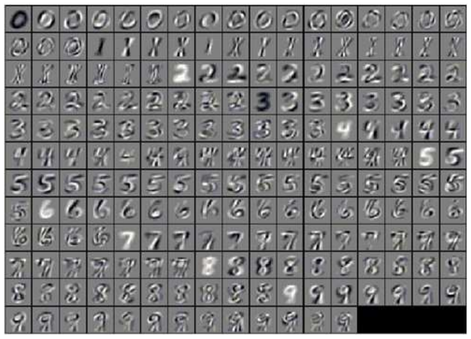
B. *Multi Layer Extreme Learning Machine (ML-ELM)*

Multi layer neural networks perform poorly when trained with BP only. Hence hidden layer weights in a deep network are initialized using layer wise unsupervised training and the whole neural network is fine tuned using BP further. Similar to deep networks, each ML-ELM hidden layer weights are initialized using ELM-AE which performs layer wise unsupervised training. However in contrast to deep networks, ML-ELM doesn't require fine tuning.

ML-ELM hidden layer activation functions can be either linear or nonlinear piecewise. If the number of nodes L^k in



(a) Features represented by the output weights $\boldsymbol{\beta}$ of ELM-AE



(b) Features represented by rank 20 SVD Basis

Figure 2. ELM-AE vs SVD

kth hidden layer is equal to the number of nodes L^{k-1} in the $(k-1)$th hidden layer, g is chosen as linear otherwise, g is chosen as nonlinear piecewise, e.g., sigmoidal function.

$$\mathbf{H}^k = g((\boldsymbol{\beta}^k)^T \mathbf{H}^{k-1}) \tag{9}$$

Where $\mathbf{H}^k$ is the kth hidden layer output matrix. The input layer $\mathbf{x}$ can be considered as the 0th hidden layer where $k=0$. The output of the connections between the last hidden layer and the output node $\mathbf{t}$ is analytically calculated using regularized least squares.

IV. Performance Evaluation

The MNIST is a commonly used dataset for testing performance of deep networks which contains, 28 x 28 images of handwritten digits with 60000 training samples and 10000 testing samples. The original MNIST dataset is used without any distortions to test the performance of ML-ELM with respect to DBN, DBM, SAE, SDAE, random feature ELM, gaussian kernel ELM and the results are shown in Table I. The experiments were carried out a in a laptop with a core i7 3740QM 2.7 GHz processor and 32 GB RAM running MATLAB 2013a. Gaussian-kernel ELM requires a larger memory than 32GB, hence executed in a high-performance computer with dual Xeon E5-2650 2 GHz processors and 256 GB RAM running MATLAB 2013a. The symbol ”*” marked in Table I indicates that the corresponding algorithm was executed in the high-performance computer. ML-ELM, ELM, Kernel-ELM was executed 10 times and the average values are reported. The ML-ELM (network structure: 784-700-700-15000-10 with ridge parameters 10^{-1} for layer 784-700, 10^3 for layer 700-15000 and 10^8 for layer 15000-10) with sigmoidal hidden layer activation function generates an accuracy of 99.03. The DBN, DBM network structures used respectively were 748-500-500-2000-10 and 784-500-1000-10. As a two layer DBM network produces better results than a three layer DBM network [2], the two layer DBM network is tested. We achieved the same test results for DBN as reported in [1] and also for DBM as reported in [2].

Algorithms	Testing Accuracy% (dev%)	Training Time
ML-ELM	99.03 ($\pm$0.04)	444.655s
ELM (random features)	97.39 ($\pm$0.1)	545.95s
* ELM (Gaussian kernel)	98.75	790.96s
DBN	98.87	20580s
DBM	99.05	68246s
SAE [3]	98.6	-
SDAE [3]	98.72	-

Table I
Performance comparison of ML-ELM with state-of-the-art Deep Networks

As shown in Table I, ML-ELM performs in par with DBM and outperforms SAE,SDAE, DBN, ELM with random feature and gaussian kernel ELM. Furthermore, ML-ELM has the least training time with respect to deep networks. ML-ELM achieves fast training time due to the following reasons;

1) In contrast to deep networks ML-ELM does not require fine tuning.
2) ELM-AE output weights can be determined analytically, unlike in RBM or traditional auto encoders where iterative algorithms are required.
3) ELM-AE learns to represent features via singular values unlike RBM and traditional auto encoders where actual representation of data is learned.

V. Conclusion

It could be seen that ELM-AE is a special case of ELM where the input is equal to output and the randomly generated weights are chosen to be orthogonal. The representation capability of ELM-AE may provide a good solution to multi-layer feed forward neural networks. ELM based multi-layer network seems to provide better performance than state-of-the-art deep networks.

References

[1] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[2] R. Salakhutdinov and H. Larochelle “Efficient learning of deep boltzmann machines,” in *International Conference on Artificial Intelligence and Statistics*, 2010.

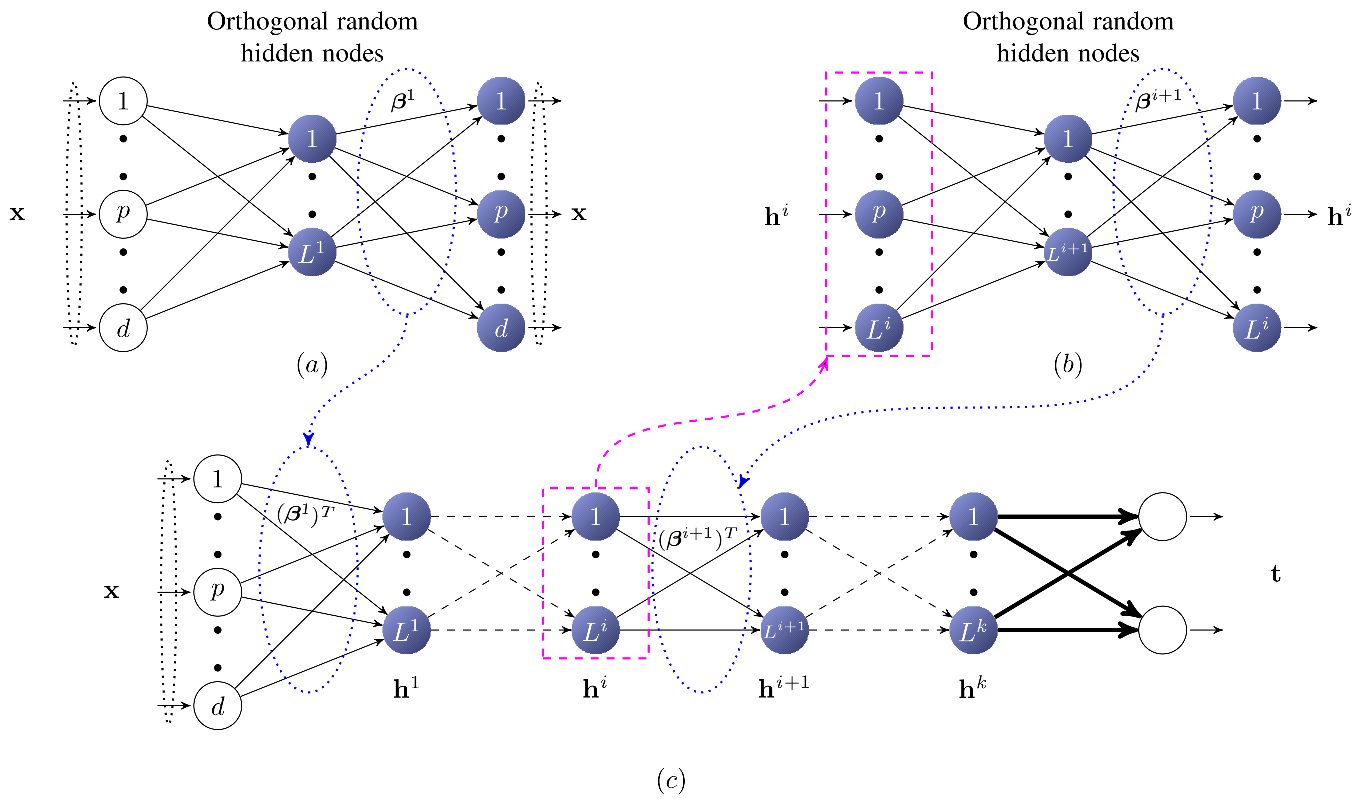

Orthogonal random hidden nodes
$\mathbf{x}$
$\boldsymbol{\beta}^1$
(a)
Orthogonal random hidden nodes
$\mathbf{h}^i$
$\boldsymbol{\beta}^{i+1}$
(b)
$(\boldsymbol{\beta}^1)^T$
$(\boldsymbol{\beta}^{i+1})^T$
$\mathbf{h}^1$
$\mathbf{h}^i$
$\mathbf{h}^{i+1}$
$\mathbf{h}^k$
$\mathbf{t}$
(c)

Figure 3. Adding Layers in ML-ELM. (a) ELM-AE output weights $\boldsymbol{\beta}^1$ with respect to input data $\mathbf{x}$ are the 1st layer weights of ML-ELM. (b) The output weights $\boldsymbol{\beta}^{i+1}$ of ELM-AE, with respect to i^{th} hidden layer output $\mathbf{h}^i$ of ML-ELM are the $(i+1)^{\text{th}}$ layer weights of ML-ELM. (c) The ML-ELM output layer weights are calculated using regularized least squares.

[3] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio and P. A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[4] 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.

[5] G.-B. Huang, Q.-Y. Zhu and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, pp. 489–501, 2006.

[6] W. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," in *Conference in modern analysis and probability*, vol. 26, pp. 189–206, 1984.

[7] G.-B. Huang, H. Zhou, X. Ding and R. Zhang,"Extreme Learning Machine for Regression and Multiclass Classification," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 42, no. 2, pp. 513–529, 2012.

[8] G.-B. Huang, L. Chen and C.-K. Siew,"Universal Approximation Using Incremental Constructive Feedforward Networks with Random Hidden Node," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 879-892, 2006