Representational Learning with ELMs for Big Data

Liyanaarachchi Lekamalage Chamara Kasun
Nanyang Technological University
20 PUBLICATIONS **1,490** CITATIONS
SEE PROFILE

Hongming Zhou
Nanyang Technological University
13 PUBLICATIONS **5,989** CITATIONS
SEE PROFILE

Guang-Bin Huang
Nanyang Technological University
255 PUBLICATIONS **54,632** CITATIONS
SEE PROFILE

Chi-Man Vong
University of Macau
171 PUBLICATIONS **4,835** CITATIONS
SEE PROFILE

Trans. Neural Networks, vol. 17, no. 4, 2006, pp. 879–892.
2. G.-B. Huang, X. Ding, and H. Zhou, "Optimization Method Based Extreme Learning Machine for Classification," *Neurocomputing*, vol. 74, 2010, pp. 155–163.
3. G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme Learning Machine: Theory and Applications," *Neurocomputing*, vol. 70, 2006, pp. 489–501.
4. G.-B. Huang et al., "Extreme Learning Machine for Regression and Multiclass Classification," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 42, no. 2, 2011, pp. 513–529.

Erik Cambria is an associate researcher at MIT Media Laboratory. Contact him at cambria@media.mit.edu.

Guang-Bin Huang is in the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. Contact him at egbhuang@ntu.edu.sg.

Representational Learning with ELMs for Big Data

Liyanaarachchi Lekamalage Chamara Kasun, Hongming Zhou, and Guang-Bin Huang, *School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore*
Chi Man Vong, *Faculty of Science and Technology, University of Macau*

A machine learning algorithm's generalization capability depends on the dataset, which is why engineering a dataset's features to represent the data's salient structure is important. However, feature engineering requires domain knowledge and human ingenuity to generate appropriate features.

Geoffrey Hinton[1] and Pascal Vincent[2] showed that a restricted Boltzmann machine (RBM) and auto-encoders could be used for feature engineering. These engineered features then could be used to train multiple-layer neural networks, or *deep networks*. Two types of deep networks based on RBM exist: the deep belief network (DBN)[1] and the deep Boltzmann machine (DBM).[3] The two types of auto-encoder-based deep networks are the stacked auto-encoder (SAE)[2] and the stacked denoising auto-encoder (SDAE).[3] DBNs and DBMs are created by stacking RBMs, whereas SAEs and SDAEs are created by stacking auto-encoders. Deep networks outperform traditional multilayer neural networks, single-layer feed-forward neural networks (SLFNs), and support vector machines (SVMs) for big data, but are tainted by slow learning speeds.

Guang-Bin Huang and colleagues[4] introduced the extreme learning machine (ELM) as an SLFN with a fast learning speed and good generalization capability. Similar to deep networks, our proposed multilayer ELM (ML-ELM) performs layer-by-layer unsupervised learning. This article also introduces the ELM auto-encoder (ELM-AE), which represents features based on singular values. Resembling deep networks, ML-ELM stacks on top of ELM-AE to create a multilayer neural network. It learns significantly faster than existing deep networks, outperforming DBNs, SAEs, and SDAEs and performing on par with DBMs on the MNIST[5] dataset.

Representation Learning

The ELM for SLFNs shows that hidden nodes can be randomly generated. The input data is mapped to L-dimensional ELM random feature space, and the network output is

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

where $\beta = [\beta_1, \ldots, \beta_L]^T$ is the output weight matrix between the hidden nodes and the output nodes, $\mathbf{h}(\mathbf{x}) = [g_1(\mathbf{x}), \ldots, g_L(\mathbf{x})]$ are the hidden node outputs (random hidden features) for input $\mathbf{x}$, and $g_i(\mathbf{x})$ is the output of the ith hidden node. Given N training samples $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N}$, the ELM can resolve the following learning problem:

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

where $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T$ are target labels, and $\mathbf{H} = [\mathbf{h}^T(\mathbf{x}_1), \ldots, \mathbf{h}^T(\mathbf{x}_N)]^T$. We can calculate the output weights β from

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

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

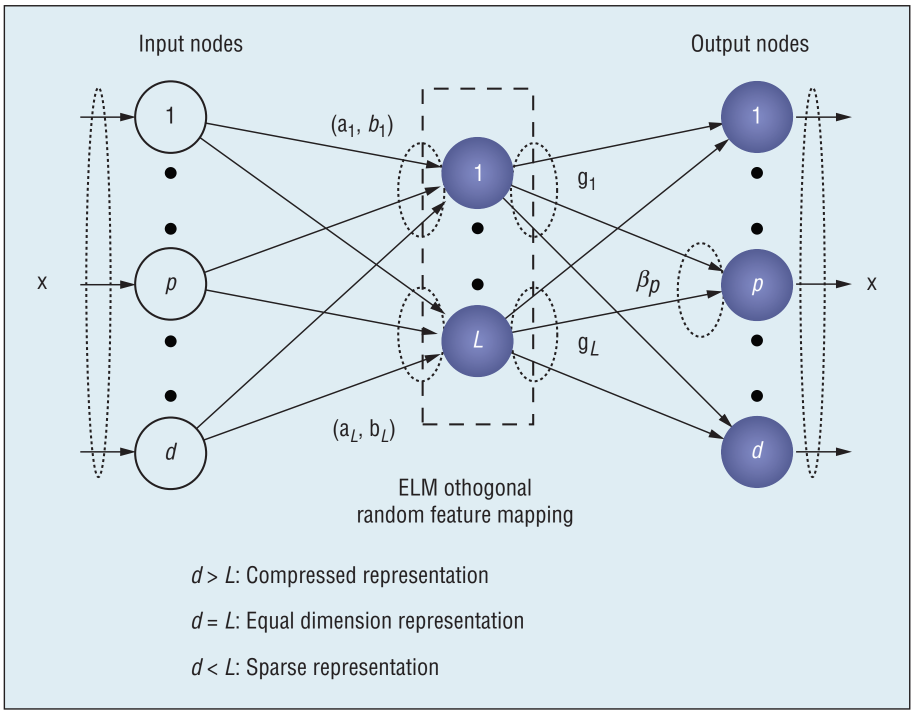
To improve generalization performance and make the solution more robust, we can add a regularization term as shown elsewhere:[6]

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

ELM-AE's main objective to represent the input features meaningfully in three different representations:

- *Compressed*. Represent features from a higher dimensional input data space to a lower dimensional feature space.
- *Sparse*. Represent features from a lower dimensional input data space to a higher dimensional feature space.
- *Equal*. Represent features from an input data space dimension equal to feature space dimension.

The ELM is modified as follows to perform unsupervised learning: input data is used as output data $\mathbf{t} = \mathbf{x}$, and random weights and biases of the hidden nodes are chosen to be orthogonal. Bernard Widrow and colleagues[7] introduced a least mean square (LMS) implementation for the ELM and a corresponding ELM-based auto-encoder that uses nonorthogonal random hidden parameters (weights and biases). Orthogonalization of these



Input nodes
Output nodes
1
p
d
x
(a_1, b_1)
(a_L, b_L)
1
L
g_1
g_L
β_p
1
p
d
x
ELM othogonal random feature mapping
$d > L$: Compressed representation
$d = L$: Equal dimension representation
$d < L$: Sparse representation

Figure 1. ELM-AE has the same solution as the original extreme learning machine except that its target output is the same as input *x*, and the hidden node parameters (a_i, b_i) are made orthogonal after being randomly generated. Here, $g_i(x) = g(a_i, b_i, x)$ is the *i*th hidden node for input *x*.

randomly generated hidden parameters tends to improve ELM-AE's generalization performance.

According to ELM theory, ELMs are universal approximators,[8] hence ELM-AE is as well. Figure 1 shows ELM-AE's network structure for compressed, sparse, and equal dimension representation. 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 the Johnson-Lindenstrauss lemma[9] and calculated as

$$\mathbf{h} = g(\mathbf{a} \cdot \mathbf{x} + \mathrm{b})$$
$$\mathbf{a}^T\mathbf{a} = \mathbf{I},\ \mathrm{b}^T\mathrm{b} = 1, \tag{5}$$

where $\mathbf{a} = [\mathbf{a}_1, \ldots, \mathbf{a}_L]$ are the orthogonal random weights, and $\mathbf{b} = [\mathrm{b}_1, \ldots, \mathrm{b}_L]$ are the orthogonal random biases between the input and hidden nodes.

ELM-AE's output weight β is responsible for learning the transformation from the feature space to input data. For sparse and compressed ELM-AE representations, we calculate output weights β as follows:

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

where $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_N]$ are ELM-AE's hidden layer outputs, and $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$ are its input and output data.

For equal dimension ELM-AE representations, we calculate output weights β as follows:

$$\beta = \mathbf{H}^{-1}\mathbf{T}$$
$$\beta^T\beta = \mathrm{I}. \tag{7}$$

Singular value decomposition (SVD) is a commonly used method for feature representation. Hence we believe that ELM-AE performs feature representation similar to SVD. Equation 6's singular value decomposition (SVD) is

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

where $\mathbf{u}$ are eigenvectors of $\mathbf{HH}^T$, and $\mathbf{d}$ are 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 ELM-AE's output weight β will learn to represent the features of the input data via singular values. To test if our hypothesis is correct, we created 10 mini datasets containing digits 0 to 9 from the MNIST dataset. Then we sent each mini dataset through an ELM-AE (network structure: 784-20-784) and compared the contents of the output weights β (Figure 2a) with the manually calculated rank 20 SVD (Figure 2b) for each mini dataset. As Figure 2 shows, ELM-AE output weight β and the manually calculated SVD basis.

Multilayer neural networks perform poorly when trained with back propagation (BP) only, so we initialize hidden layer weights in a deep network by using layer-wise unsupervised training and fine-tune the whole neural network with BP. Similar to deep networks, ML-ELM hidden layer weights are initialized with 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 the *k*th 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, such as a sigmoidal function:

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

where $\mathbf{H}^k$ is the *k*th 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.

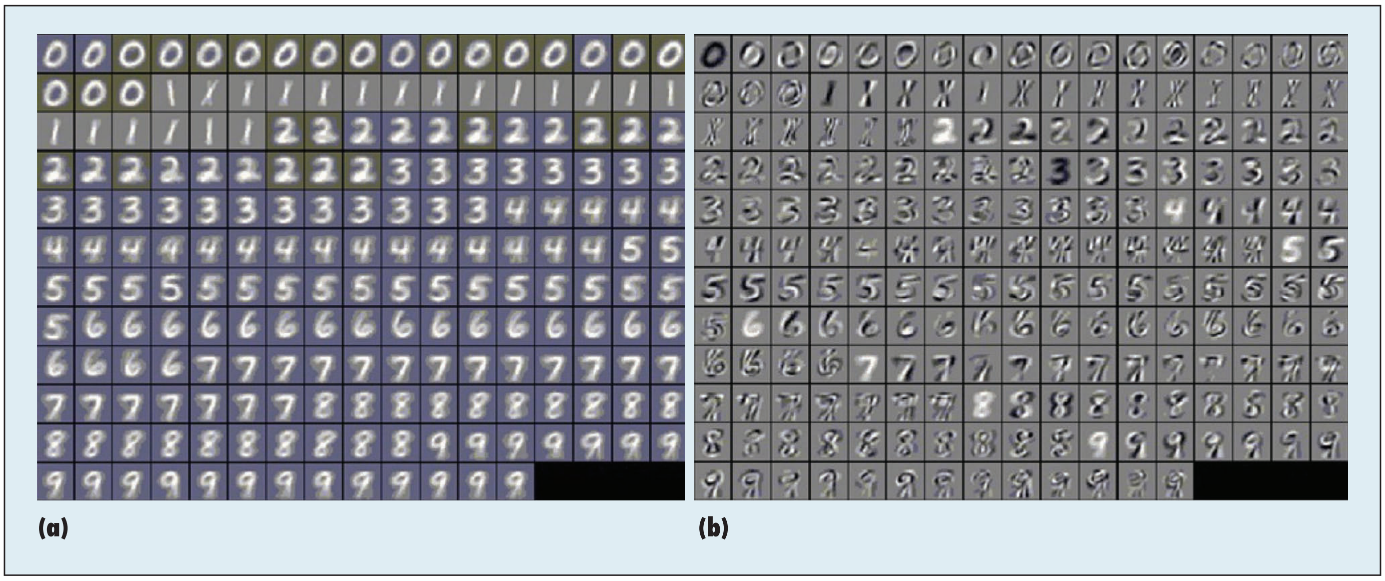


Figure 2. ELM-AE vs. singular value decomposition. (a) The output weights β of ELM-AE and (b) rank 20 SVD basis shows the feature representation of each number (0–9) in the MNIST dataset.

Performance Evaluation

The MNIST is commonly used for testing deep network performance; the dataset contains images of handwritten digits with 60,000 training samples and 10,000 testing samples. Table 1 shows the results of using the original MNIST dataset without any distortions to test the performance of ML-ELM with respect to DBNs, DBMs, SAEs, SDAEs, random feature ELMs, and Gaussian kernel ELMs.

Table 1. Performance comparison of ML-ELM with state-of-the-art deep networks.

Algorithms	Testing accuracy % (standard deviation %)	Training time
Multi-layer extreme learning machine (ML-ELM)	99.03 ($\pm$0.04)	444.655 s
Extreme learning machine (ELM random features)	97.39 ($\pm$0.1)	545.95 s
ELM (ELM Gaussian kernel); run on a faster machine	98.75	790.96 s
Deep belief network (DBN)	98.87	20,580 s
Deep Boltzmann machine (DBM)	99.05	68,246 s
Stacked auto-encoder (SAE)	98.6	–
Stacked eenoising auto-encoder (SDAE)	98.72	–

We conducted the experiments on a laptop with a core i7 3740QM 2.7-GHz processor and 32 Gbytes of RAM running Matlab 2013a. Gaussian-kernel ELMs require a larger memory than 32 Gbytes, so we executed on a high-performance computer with dual Xeon E5-2650 2-GHz processors and 256 Gbytes of RAM running Matlab 2013a. 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 generated an accuracy of 99.03. We used DBNs and DBM network structures 748-500-500-2000-10 and 784-500-1000-10, respectively, to generate the results shown in Table 1. As a two-layer DBM network produces better results than a three-layer one,[3] we tested the two-layer network.

As Table 1 shows, ML-ELM performs on par with DBMs and outperforms SAEs, SDAEs, DBNs, ELMs with random feature, and Gaussian kernel ELMs. Furthermore, ML-ELM has the least amount of required training time with respect to deep networks:

- In contrast to deep networks, ML-ELM doesn't require fine-tuning.
- ELM-AE output weights can be determined analytically, unlike RBMs and traditional auto-encoders, which require iterative algorithms.
- ELM-AE learns to represent features via singular values, unlike RBMs and traditional auto-encoders, where the actual representation of data is learned.

ELM-AE can be seen as a special case of ELM, where the input is equal to output, and the randomly generated

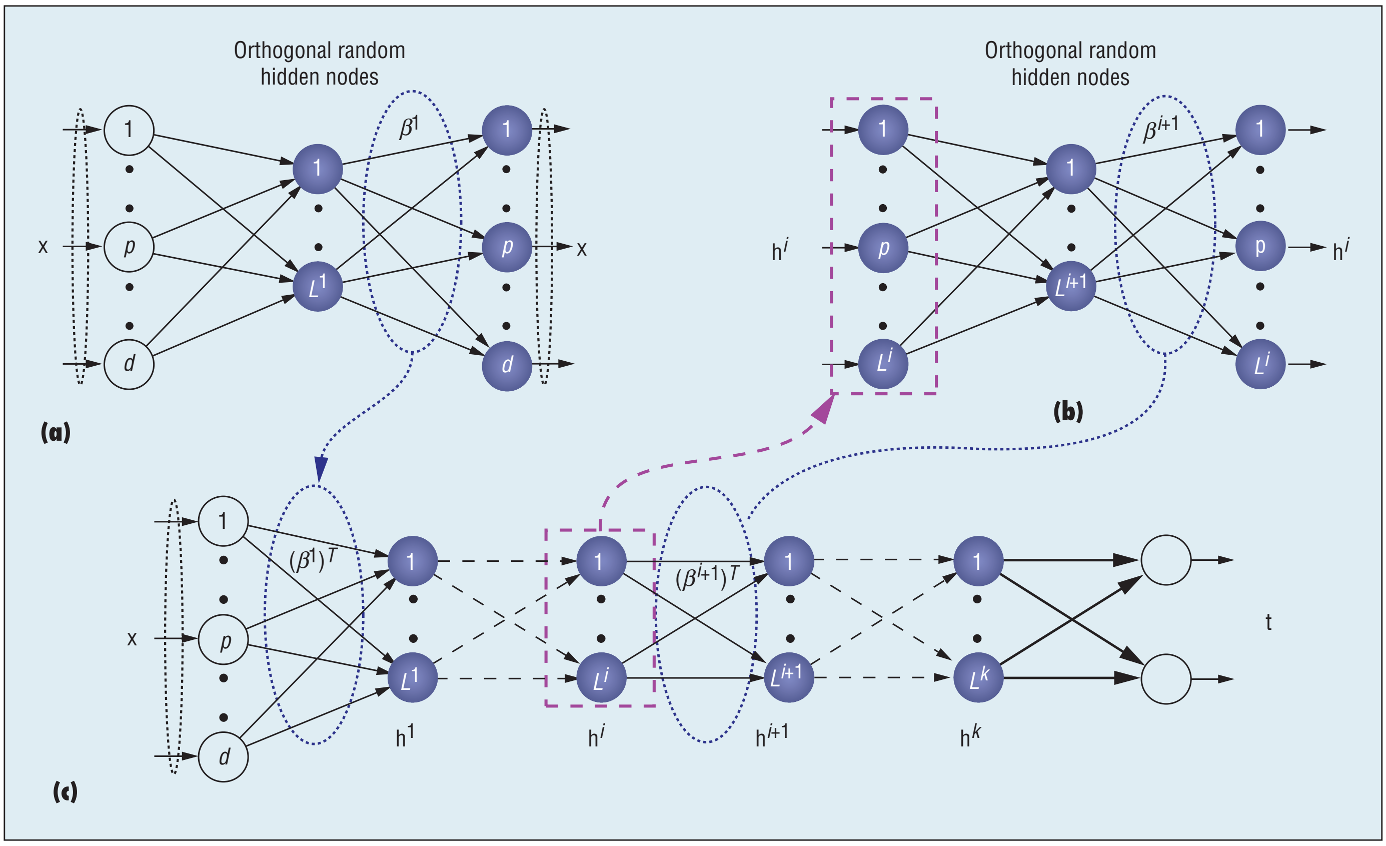


Figure 3. Adding layers in ML-ELM. (a) ELM-AE output weights β^1 with respect to input data *x* are the first-layer weights of ML-ELM. (b) The output weights β^{i+1} of ELM-AE, with respect to *i*th hidden layer output h^i of ML-ELM are the (*i* + 1)th layer weights of ML-ELM. (c) The ML-ELM output layer weights are calculated using regularized least squares.

weights are chosen to be orthogonal (see Figure 3). ELM-AE's representation capability might provide a good solution to multilayer feed-forward neural networks. ELM-based multilayer networks seem 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, 2006, pp. 504–507.
2. P. Vincent et al., "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *J. Machine Learning Research*, vol. 11, 2010, pp. 3371–3408.
3. R. Salakhutdinov and H. Larochelle "Efficient Learning of Deep Boltzmann Machines," *J. Machine Learning Research*, vol. 9, 2010, pp. 693–700.
4. G.-B. Huang, Q.-Y. Zhu and C.-K. Siew, "Extreme Learning Machine: Theory and Applications," *Neurocomputing*, vol. 70, 2006, pp. 489–501.
5. Y. LeCun et al., "Gradient-Based Learning Applied to Document Recognition," *Proc. IEEE*, vol. 86, no. 11, 1998, pp. 2278–2324.
6. G.-B. Huang et al.,"Extreme Learning Machine for Regression and Multiclass Classification," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 42, no. 2, 2012, pp. 513–529.
7. B. Widrow et al., "The No-Prop Algorithm: A New Learning Algorithm for Multilayer Neural Networks," *Neural Networks*, vol. 37, 2013, pp. 182–188.
8. G.-B. Huang, L. Chen, and C.-K. Siew, "Universal Approximation Using Incremental Constructive Feedforward Networks with Random Hidden Node," *IEEE Trans. Neural Networks*, vol. 17, no. 4, 2006, pp. 879–892.
9. W. Johnson and J. Lindenstrauss, "Extensions of Lipschitz Mappings into a Hilbert Space," *Proc. Conf. Modern Analysis and Probability*, vol. 26, 1984, pp. 189–206.

Liyanaarachchi Lekamalage Chamara Kasun is at the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. Contact him at chamarak001@e.ntu.edu.sg.

Hongming Zhou is at the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. Contact him at hmzhou@ntu.edu.sg.

Guang-Bin Huang is at the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. Contact him at egbhuang@ntu.edu.sg.

Chi Man Vong is in the Faculty of Science and Technology, University of Macau. Contact him at cmvong@umac.mo.