LEARNING HIGH-DIMENSIONAL NONLINEAR MAPPING VIA COMPRESSED SENSING

Tomoya Sakai⋆ *Daisuke Miyata*⋆

⋆ Graduate School of Engineering, Nagasaki University

ABSTRACT

This paper proposes an efficient framework for learning a high-dimensional nonlinear mapping using compressed sensing techniques. Given a training data set of the input and output pairs of the mapping to be learned, our framework reduces both the dimensionalities of the input and output spaces by efficient computation of random projection, and then learns a nonlinear mapping between the low-dimensional input and output data. The high-dimensional nonlinear mapping consists of (i) dimensionality reduction by the random projection of the input data, (ii) low-dimensional nonlinear mapping, and (iii) reconstruction of the high-dimensional output data on the basis of a sparse model. The processes (i) and (ii) construct a single hidden layer feedforward neural network, which can efficiently be learned by the extreme learning machine.

Index Terms— Efficient random projection, SLFN, ELM, dimensional scalability

1. INTRODUCTION

In this paper, we aim to develop a framework for learning a nonlinear function from a given set of pairs of input and output vectors with more than thousands of dimensions. A dimensionality-scalable machine learning technique for approximating an arbitrary function would have many potential applications such as nonlinear filtering and image processing whose characteristics depend on some features of a high-dimensional input. If such a nonlinear mapping is so complicated that it is difficult to represent it as a parametric model, a possible way to tailor the mapping is to learn from a large amount of input and output data.

A high-dimensional data set in practice has an intrinsic low-dimensional structure, and so it is possible to reduce the data dimensionality. The reduction techniques, such as the linear and kernel principal component analyses, assume the data to lie on an embedded linear or nonlinear manifold. Finding a data manifold can, however, be computationally as intensive as learning a nonlinear function. Compressed sensing [1, 2, 3, 4, 5] provides us with efficient methods for compression and reconstruction of high-dimensional vectors. Compression by random projection is a method of dimensionality reduction suitable for learning a high-dimensional nonlinear mapping, because it can be performed without any analyses on high-dimensional data or even matrix operations [6].

We show an outline of our framework for learning a nonlinear mapping based on the compressed sensing technique in Section 2. A high-dimensional nonlinear mapping can be approximated by a single hidden layer feedforward neural network (SLFN) and the reconstruction of the high-dimensional output from a low-dimensional output of the SLFN. The SLFN includes the random projection of the high-dimensional input. Such an SLFN can be learned by a simple and efficient algorithm called the extreme learning machine (ELM) [7, 8, 9]. In Section 3, we present a compressive ELM (CELM) algorithm, which includes the random projection of both the input and output to make the ELM scalable in dimensionality. We confirm the scalability of the CELM in Section 4.

2. MACHINE LEARNING FOR HIGH-DIMENSIONAL NONLINEAR MAPPING

2.1. Framework

Let X and Y be real vector spaces. Given the training data set

$$\mathcal{T} = \left\{ (\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)}) \mid \boldsymbol{x}^{(j)} \in X,\, \boldsymbol{y}^{(j)} \in Y,\, j \in \{1, \ldots, n\} \right\}, \quad (1)$$

we consider the problem of learning a mapping function

$$f : X \to Y. \quad (2)$$

Assuming f to be a nonlinear mapping from the input space X to the output space Y, this supervised machine learning task is a nonlinear regression. There is a class of models called the universal approximator [10, 11], such as the projection pursuit [12] and the feedforward neural networks (FNN), which can approximate any continuous function with an arbitrarily small error.

In general, the lower the dimensionalities $d_x = \dim X$ and $d_y = \dim Y$ are, and the smaller the number n of training data is, the lower the computational cost of learning the mapping is. A natural idea is to reduce the dimensionalities of the input and output spaces, X and Y, respectively to low-dimensional spaces P and Q, and instead learn the mapping h from P to Q.

$$h : P \to Q \quad (3)$$

The first author was partially supported by MEXT KAKEN 25330200.

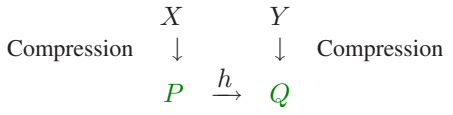

Fig. 1. Compression and learning of a mapping h. Instead of learning f between the high-dimensional input and output spaces, X and Y, consider to learn h between the compressed (dimensionality-reduced) input and output spaces, P and Q.

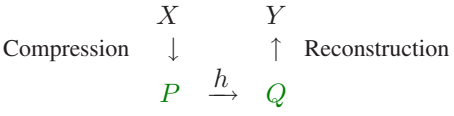

Fig. 2. Mapping from X to Y through h. The mapping from X to Y is achieved by the compression (dimensionality reduction) from X to P, the learned mapping h from P to Q, and reconstruction of Y from Q.

This idea is illustrated in Fig. 1. If it is possible to reconstruct the high-dimensional output data $\boldsymbol{y} \in Y$ from the low-dimensional output data $\boldsymbol{q} \in Q$, the mapping f from X to Y is computable through h as shown in Fig. 2. The compression by the dimensionality reduction, therefore, needs to be computationally more tractable than directly learning f, as well as invertible to reconstruct the high-dimensional output data.

2.2. Compression and Reconstruction

Compressed sensing [1, 2, 3, 4, 5] is a technique for acquisition and reconstruction of a signal with a sparse model. Suppose that the input and output data have the sparse models described as

$$\boldsymbol{x}^{(j)} = \mathbf{D}_x \boldsymbol{\alpha}^{(j)} \tag{4}$$

$$\boldsymbol{y}^{(j)} = \mathbf{D}_y \boldsymbol{\beta}^{(j)}. \tag{5}$$

Here, $\mathbf{D}_x \in \mathbb{R}^{d_x \times k_x}$ and $\mathbf{D}_y \in \mathbb{R}^{d_y \times k_y}$ are the dictionary matrices, a small number of whose column vectors can synthesize $\boldsymbol{x}^{(j)} \in X$ and $\boldsymbol{y}^{(j)} \in Y$, respectively. The sparse vectors $\boldsymbol{\alpha}^{(j)} \in \mathbb{R}^{k_x}$ and $\boldsymbol{\beta}^{(j)} \in \mathbb{R}^{k_y}$, with nonzero components $||\boldsymbol{\alpha}^{(j)}||_0 \ll k_x$ and $||\boldsymbol{\beta}^{(j)}||_0 \ll k_y$, contain the coefficients of the linear combinations for $\boldsymbol{x}^{(j)}$ and $\boldsymbol{y}^{(j)}$, respectively.

Let the compression of the input and output in Fig. 1 be the linear measurement of $\boldsymbol{x}^{(j)}$ and $\boldsymbol{y}^{(j)}$ described as

$$\boldsymbol{p}^{(j)} = \mathbf{R}_x \boldsymbol{x}^{(j)} \tag{6}$$

$$\boldsymbol{q}^{(j)} = \mathbf{R}_y \boldsymbol{y}^{(j)}. \tag{7}$$

If $\mathbf{R}_x \in \mathbb{R}^{d_p \times d_x}$ and $\mathbf{R}_y \in \mathbb{R}^{d_q \times d_y}$ are proper measurement matrices with low coherence to the respective dictionaries, one can find the sparse vector $\boldsymbol{\alpha}^{(j)}$ and $\boldsymbol{\beta}^{(j)}$ for $\boldsymbol{p}^{(j)}$ and $\boldsymbol{q}^{(j)}$, and reconstruct $\boldsymbol{x}^{(j)}$ and $\boldsymbol{y}^{(j)}$ by Eqs. (4) and (5) [4, 13]. The random matrix is in particular known to be a universal measurement for signals obeying the sparse model with any dictionary [2, 4]. Compression of the input $\boldsymbol{x}^{(j)} \in X$ by the random projection using a random matrix as $\mathbf{R}_x$ in Eq. (6) requires $\mathcal{O}(d_p d_x)$ time and $\mathcal{O}(d_p d_x)$ space to generate, store, and multiply the pseudo-random numbers. Fortunately, we have developed an efficient algorithm of the random projection which requires only $\mathcal{O}(d_x \log d_x)$ time and $\mathcal{O}(d_x)$ space [6]. So does for the compression of the output. The procedure of the efficient random projection is shown in Algorithms 1 and 2. Our algorithm achieves the random projection by spectrum spreading and circular convolution with a white random vector. It has been proven that our algorithm satisfies the Johnson-Lindenstrauss lemma [14].

Algorithm 1 Generating random code: $\mathcal{C} = \textsc{Code}(d)$

Input: d: ambient dimensionality;
Output: $\mathcal{C} = \{\hat{\boldsymbol{w}}, \boldsymbol{s}\}$: a random code;
1 generate $\boldsymbol{s} := [s_1, \ldots, s_d]^\top$ where s_i $(i = 1, \ldots, d)$ are random variable values with mean zero and deviation one, or random signs $\{+1, -1\}$;
2 generate $\boldsymbol{w} := [w_1, \ldots, w_d]^\top$ where w_i $(i = 1, \ldots, d)$ are i.i.d. random variable values with mean zero and deviation one;
3 $\hat{\boldsymbol{w}} := \mathcal{F}[\boldsymbol{w}]$: fast Fourier transform (FFT) of $\boldsymbol{w}$.

Algorithm 2 Efficient random projection: $\boldsymbol{v} = \textsc{Erp}(\boldsymbol{u}, l, \mathcal{C})$

Input: $\boldsymbol{u} \in \mathbb{R}^d$: a vector to be projected; l: dimensionality of the target space; $\mathcal{C} = \{\hat{\boldsymbol{w}}, \boldsymbol{s}\}$: a random code generated by Algorithm 1;
Output: $\boldsymbol{v} \in \mathbb{R}^l$: random projection of $\boldsymbol{u}$;
1 $\boldsymbol{\xi} := \boldsymbol{s}.*\boldsymbol{u}$: spectrum spreading with element-by-element multiplication;
2 $\boldsymbol{\eta} := \mathcal{F}^{-1}\left[\hat{\boldsymbol{w}}. * \mathcal{F}[\boldsymbol{\xi}]^*\right]$: circular convolution;
3 $\boldsymbol{v} := [\eta_1, \ldots, \eta_l]^\top$: pick up l components.

3. EXTREME LEARNING MACHINE FOR HIGH-DIMENSIONAL NONLINEAR MAPPING

3.1. ELM and SLFN

Extreme Learning Machine (ELM) [7, 8, 9] is a simple and fast machine learning algorithm for single hidden layer feedforward neural networks (SLFNs). To map a s_1-dimensional input vector to a s_3-dimensional output vector, an SLFN has three layers: an input layer of s_1 nodes and a bias node, a middle layer of s_2 nodes with activation functionality, and an output layer of s_3 nodes. Propagation from the input $\boldsymbol{x}$ to the output $\boldsymbol{y}$ is expressed as

$$\boldsymbol{y} = \mathbf{W}^{(2)} g\left(\mathbf{W}^{(1)} \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}\right), \tag{8}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{(s_1+1) \times s_2}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{s_2 \times s_3}$ are the matrices of weights of links between the input layer and the middle

layer, and between the middle layer and the output layer, respectively. The activation function g works element-wise on a vector as

$$g(\boldsymbol{z}) = [g(z_1), \ldots, g(z_{s_2})]^\top \quad \in \mathbb{R}^{s_2} \tag{9}$$

A common choice for the activation function is a sigmoidal function:

$$g(z) = \frac{1}{1 + e^{-z}}. \tag{10}$$

Given a training data set $\mathcal{T}$ of the input and output pairs of the mapping f, we want to find $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ of the SLFN which approximates f. The ELM algorithm consists of random projection and the least-squares solution. It has been shown that $\mathbf{W}^{(1)}$ can be a random matrix for an SLFN to be a universal approximator if the activation function is infinitely differentiable [7, 8]. The SLFN training is therefore simplified to finding the least-squares solution $\mathbf{W}^{(2)}$ that minimizes the approximation errors between the n output training data and the SLFN outputs. A sufficient number s_2 of hidden nodes ensures the universal approximation capability [15].

3.2. Compressive ELM

We employ random projection for the compression of both the input and output in Fig. 1. Let us set the low-dimensional mapping h in Eq. (3) to be the activation using g followed by the linear mapping defined by the least-squares solution $\mathbf{W}^{(2)}$. The nonliner mapping from X to Q is then formed as an SLFN, which can efficiently be learned by the ELM algorithm. The low-dimensional input space P plays a role of the middle layer of the SLFN.

Thus we have derived, as shown in Algorithm 3, a compressive extreme learning machine (CELM) algorithm of approximating a high-dimensional nonlinear mapping as an SLFN. It naturally includes the compressed sensing random measurement of the high-dimensional input as the propagation from the input layer to the middle layer. The propagation from the middle layer to the output layer is optimized so that the compressed input training data $\boldsymbol{p}^{(j)}$ after the activation by g approximate the compressed output training data $\boldsymbol{q}^{(j)}$.

The CELM is extremely scalable in dimensionality because the compression of both the input and output by the random projection is implemented by the efficient random projection, as shown in Algorithms 1 and 2. No huge random matrices have to be generated and stored. Instead, the CELM returns the the random codes $\mathcal{C}_x$ and $\mathcal{C}_y$. The computational cost of the least-square solution to find the weight matrix $\mathbf{W}^{(2)}$ is also reduced due to the compression of the output space.

The procedure for mapping $\boldsymbol{x} \in X$ to $\boldsymbol{y} \in Y$ is described in Algorithm 4. It obtains the low-dimensional output $\boldsymbol{q}$ of the learned SLFN, and follows reconstruction process of the compressed sensing. Most of the sparse solvers for compressed sensing require operations of signal synthesis and measurement, which correspond to the multiplication by $\mathbf{D}_y$ and random projection in our framework, respectively. Their adjoint operations are also required: multiplication by $\mathbf{D}_y^\top$ and the transposed random matrix $\mathbf{R}_y^\top$. We can implement the operation adjoint to ERP in a similar fashion, although it is not shown in this paper. It is also possible to make a composition of the operations as a compressed dictionary matrix $\boldsymbol{\Theta} = \mathbf{R}_y\mathbf{D}_y$, efficiently by ERP. Its transpose $\boldsymbol{\Theta}^\top = \mathbf{D}_y^\top\mathbf{R}_y^\top$ can be used as the adjoint operations.

Algorithm 3 Compressive ELM: $\{\mathcal{C}_x, \mathbf{W}^{(2)}, \mathcal{C}_y\} = \textsc{Celm}(\mathcal{T}, d_p, d_q)$

Input: $\mathcal{T} = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})\}$: training data set;
d_p: dimensionality of the low-dimensional input space;
d_q: dimensionality of the low-dimensional output space;

Output: $\mathcal{C}_x$: random code for the input space; $\mathbf{W}^{(2)} \in \mathbb{R}^{d_q \times d_p}$: weight matrix between the hidden and output layers; $\mathcal{C}_y$: random code for the output space;

1 generate by Algorithm 1 the random codes:

$$\mathcal{C}_x := \{\hat{\boldsymbol{w}}_x \in \mathbb{R}^{(d_x+1)}, \boldsymbol{s}_x \in \mathbb{R}^{(d_x+1)}\} = \textsc{Code}(d_x + 1)$$

$$\mathcal{C}_y := \{\hat{\boldsymbol{w}}_y \in \mathbb{R}^{(d_y+1)}, \boldsymbol{s}_y \in \mathbb{R}^{(d_y+1)}\} = \textsc{Code}(d_y + 1)$$

for the input and output spaces of the respective dimensionalities $(d_x + 1)$ and $(d_y + 1)$ (bias included);

2 **for** $j = 1, \ldots, n$ **do**

3 perform the random projection of the j-th input data by Algorithm 2: $\boldsymbol{p}^{(j)} := \textsc{Erp}([1,\ {\boldsymbol{x}^{(j)}}^\top]^\top, d_p, \mathcal{C}_x)$;

4 perform the random projection of the j-th output data by Algorithm 2: $\boldsymbol{q}^{(j)} := \textsc{Erp}([1,\ {\boldsymbol{y}^{(j)}}^\top]^\top, d_q, \mathcal{C}_y)$:

5 **end for**

6 construct the input matrix

$$\mathbf{P} := \left[\boldsymbol{p}^{(1)}, \ldots, \boldsymbol{p}^{(n)}\right] \quad \in \mathbb{R}^{d_p \times n};$$

7 apply an activation function g to every element of $\mathbf{P}$ to obtain the output of the hidden layer:

$$\mathbf{H} := g(\mathbf{P}) \quad \in \mathbb{R}^{d_p \times n}; \tag{11}$$

8 compute the Moore-Penrose inverse [16] of $\mathbf{H}$ as

$$\mathbf{H}^+ := \mathbf{V}_r\mathbf{K}_r^{-1}\mathbf{U}_r^\top \quad \in \mathbb{R}^{n \times d_p} \tag{12}$$

where $\mathbf{K}_r$ is the diagonal matrix with $r = \operatorname{rank}\mathbf{H}$ nonzero singular values, and $\mathbf{U}_r$ and $\mathbf{V}_r$ are respectively the matrices of the left and right singular vectors corresponding to the r nonzero singular values;

9 compute the weight matrix by linear regression as

$$\mathbf{W}^{(2)} := \left[\boldsymbol{q}^{(1)}, \ldots, \boldsymbol{q}^{(n)}\right]\mathbf{H}^+ \quad \in \mathbb{R}^{d_q \times d_p}. \tag{13}$$

Algorithm 4 Mapping: $\boldsymbol{y} = \text{MAP}(\boldsymbol{x}, \{\mathcal{C}_x, \mathbf{W}^{(2)}, \mathcal{C}_y\})$

Input: $\boldsymbol{x} \in \mathbb{R}^d$: an input vector; $\{\mathcal{C}_x, \mathbf{W}^{(2)}, \mathcal{C}_y\}$: a parameter set of SLFN learned by Algorithm 3; $\mathbf{D}_y$: a dictionary matrix;

Output: $\boldsymbol{y}$: an output vector;

1 set d_q and d_p to be the numbers of rows and columns of $\mathbf{W}^{(2)}$;

2 compute the low-dimensional output vector of the SLFN:

$$\boldsymbol{q} := \mathbf{W}^{(2)} g\left(\text{ERP}([1, \boldsymbol{x}^\top]^\top, d_p, \mathcal{C}_x)\right);$$

3 find the sparse solution

$$\boldsymbol{\beta} := \arg\min_{\boldsymbol{b}} ||\boldsymbol{b}||_1 \text{ s. t. } ||\boldsymbol{q} - \text{ERP}(\mathbf{D}_y \boldsymbol{b}, d_q, \mathcal{C}_y)||_2 < \varepsilon. \quad (14)$$

4 $\boldsymbol{y} := \mathbf{D}_y \boldsymbol{\beta}$.

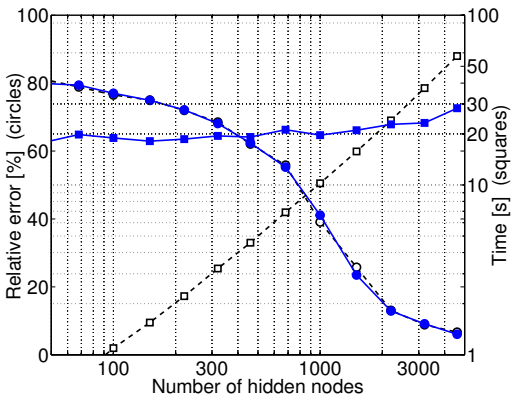

Fig. 3. Computation time and test errors. The open and filled symbols correspond to the ELM and CELM results, respectively. The squares indicate the computation time for training with $n = 10^4$ data with dimensionalities $d_x = d_y = 2^{14}$ (using Core i7 980X in MATLAB 2012b). The circles indicate the relative errors.

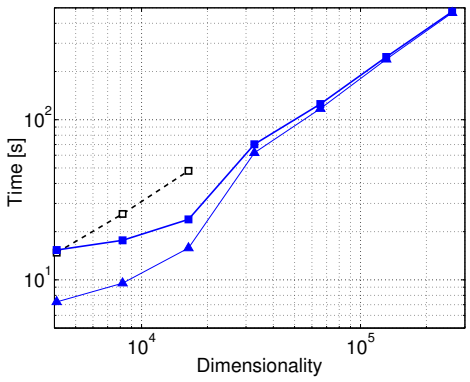

Fig. 4. Computation time with respect to dimensionality. The open and filled squares indicate the computation time for training with $n = 10^4$ data by ELM and CELM, respectively. The triangles indicate the computation time for efficient random projection included in the CELM algorithm.

4. EXPERIMENTAL EVALUATION

We evaluate the scalability of the CELM algorithm using artificial data sets. We generate an artificial training data set of $n = 10^4$ pairs of input vector $\boldsymbol{x}^{(j)}$ and output vector $\boldsymbol{y}^{(j)} = f(\boldsymbol{x}^{(j)}) = \boldsymbol{x}^{(j)} .* \boldsymbol{x}^{(j)}$, where ".*" indicates element-by-element multiplication. Each input vector $\boldsymbol{x}^{(j)}$ is an inverse discrete cosine transform (DCT) of normally distributed random sparse coefficients. To simulate a real output signal with low frequency components, we randomly choose nnz $= 10$ frequencies for the nonzero coefficients from the 40 lowest frequencies for each input vector. Since the nonlinear mapping f is set to compute the output as the squared waveform of the input, the DCT of the output vector $\boldsymbol{y}^{(j)}$ has $\text{nnz}^2+1 = 101$ nonzero random DCT coefficients. We therefore define the dictionary $\mathbf{D}_y$ as the DCT matrix. A test data set is also generated in the same way for the evaluation. We use the regularized orthogonal matching pursuit [17] to solve Eq. (14).

Setting the dimensionality $d_x = d_y = 2^{14}$, we first compare computation time for training by the ELM and our CELM, and evaluate test errors with respect to the number $s_2 = d_p$ of hidden nodes. We fix $s_3 = d_q = 3{,}000$ for the CELM. The results shown in Fig. 3 indicate that a few thousand of hidden nodes are required to approximate f for both the algorithms, and CELM is faster for a larger number of hidden nodes in this experiment. We also evaluate the scalability in the input and output dimensionality up to $d_x = d_y = 2^{18}$. We set $s_2 = d_p = 4{,}000$ to sufficiently approximate f. Figure 4 shows the outperformance of the CELM. Dimensionality reduction accounts for most of the computational cost as the dimensionality increases. While it is hard to run the ELM requiring huge weight matrices of sizes of tens of gigabytes, our CELM works with very high-dimensional data owing to ERP.

5. CONCLUDING REMARKS

Compressed sensing enables us to make it dimensionality-scalable to learn a nonlinear mapping. Our random projection algorithm can efficiently perform the dimensionality reduction of training data. The CELM presented in this paper learns an SLFN, which reduces the dimensionality of the input and computes a low-dimensional output from which a high-dimensional output is reconstructed on the basis of a sparse model.

Further research should include detailed performance evaluation in possible applications such as nonlinear image filtering, tracking, anomaly detection, solving large-scale inverse problem, and so on. In such applications, our CELM should be modified to support incremental learning.

6. REFERENCES

[1] D.L. Donoho, "Compressed sensing," *IEEE Trans. Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[2] E. J. Candès and T. Tao, "Decoding by linear programming," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

[3] Emmanuel J. Candès, Justin Romberg, and Terence Tao, "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory*, vol. 52, no. 2, pp. 489–509, 2006.

[4] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, pp. 21–30, March, 2008.

[5] Kazunori Hayashi, Masaaki Nagahara, and Toshiyuki Tanaka, "A user's guide to compressed sensing for communications systems," *IEICE Transactions*, vol. 96–B, no. 3, pp. 685–712, 2013.

[6] Tomoya Sakai and Atsushi Imiya, "Practical algorithms of spectral clustering: Toward large-scale vision-based motion analysis," in *Machine Learning for Vision-Based Motion Analysis*, Advances in Pattern Recognition, chapter 1, pp. 3–26. Springer, 2011.

[7] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *2004 International Joint Conference on Neural Networks (IJCNN'2004)*, 2004, pp. 25–29.

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

[9] G.-B. Huang, D. H. Wang, and Y. Lan, "Extreme learning machines: A survey," *International Journal of Machine Leaning and Cybernetics*, vol. 2, no. 2, pp. 107–122, 2011.

[10] G. Cybenko, "Approximations by superpositions of sigmoidal functions," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[11] Kurt Hornik, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[12] Jerome H. Friedman and Werner Stuetzle, "Projection pursuit regression," *Journal of the American Statistical Association*, vol. 76, pp. 817–823, 1981.

[13] E. J. Candès and J. Romberg, "Sparsity and incoherence in compressive sampling," *Inverse Problems*, vol. 23, no. 3, pp. 969–985, 2007.

[14] W. Johnson and J. Lindenstrauss, "Extensions of Lipschitz maps into a Hilbert space," *Contemporary Mathematics*, vol. 26, pp. 189–206, 1984.

[15] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

[16] E. H. Moore, "On the reciprocal of the general algebraic matrix," *Bulletin of the American Mathematical Society*, vol. 26, no. 9, pp. 394–395, 1920.

[17] Deanna Needell and Roman Vershynin, "Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit," *Foundations of Computational Mathematic*, vol. 9, no. 3, pp. 317–334, 2009.