Deep Stochastic Configuration Networks: Universal Approximation and Learning Representation

Dianhui Wang,∗ Ming Li
Department of Computer Science and Information Technology
La Trobe University, Melbourne, VIC 3086, Australia
Email: dh.wang@latrobe.edu.au

Abstract

This paper focuses on the development of randomized approaches for building deep neural networks. A supervisory mechanism is proposed to constrain the random assignment of the hidden parameters (i.e., all biases and weights within the hidden layers). Full-rank oriented criterion is suggested and utilized as a termination condition to determine the number of nodes for each hidden layer, and a pre-defined error tolerance is used as a global indicator to decide the depth of the learner model. The read-out weights attached with all direct links from each hidden layer to the output layer are incrementally evaluated by the least squares method. Such a class of randomized learner models with deep architecture is termed as deep stochastic configuration networks (DeepSCNs), of which the universal approximation property is verified with rigorous proof. Given abundant samples from a continuous distribution, DeepSCNs can speedily produce a learning representation, that is, a collection of random basis functions with the cascaded inputs together with the read-out weights. Simulation results with comparisons on function approximation align with the theoretical findings.

1 Introduction

In the past decade, deep neural networks (DNNs) have received extensively considerable attention due to its great potential for dealing with computer vision, pattern recognition, natural language processing, etc. [21, 30]. The success of deep learning is attributed to its representation capability for visual data [4, 18]. In view of some empirical evidence, DNNs are becoming increasingly popular because of a hypothesis that a deep learner model can be exponentially more efficient at representing some functions than a shallow one [15]. It seems that DNNs have a good potential in learning higher-level abstractions (or multiple levels of features), which sounds difficult for models with shallow architecture. Formal analyses of the representation power and learning complexity of DNNs can be found in [11, 25]. In short, the community has reached a common sense that DNNs are much more expressive than the shallow ones.

Although deep learning schemes draw tremendous attention for their overwhelming high performance for some complex data modelling tasks [21], two key issues should be concerned seriously in model design: the architecture determination for DNNs and the training strategies for deep architectures. As usually done, the number of layers and of the nodes at each layer of a DNN are set manually, which poses great threats on the effectiveness of the resulting model. This is because DNNs with moderate depth but fewer number of hidden nodes at each layer cannot exhibit sufficient learning ability (i.e., under fitting), while the large sized architecture (i.e. with quite a few layers and hidden nodes) may cause over-fitting that degrades the generalization capability. Also, it leads

∗Corresponding author.
to unnecessary time cost or even unsatisfactory deep models, if end-users determine the architecture by trial and error approaches. In [1], Alvarez and Salzmann introduced an approach to automatically determine the number of nodes at each layer of the DNN by using group sparsity regularizers. Their method can result in a compact width setting for each layer, but with the prerequisite that the network’s depth is determined in advance. Except for the architecture setting problem, fast training algorithms for DNNs are always welcome. Indeed, training standard multilayer perceptrons by the back-propagation algorithm comes up with a complex non-convex optimization problem, and the simultaneous learning of all parameters leads to a high memory and computation cost for very deep models. Empirically, this gradient-based learning method is less effective for representing highly non-linear and highly-varying functions, since the learning curve easily gets stuck in local minima or lost in plateaus. These obstacles also lie in the process of fine-tuning a pre-trained model obtained by the greedy layer-wise unsupervised learning strategy proposed in [17], as the parameters within all layers need to be optimized according to to an objective cost function. To the best of our knowledge on data sciences, deep learning techniques are bravely and blindly generating DNNs with bulk of data, but with less art on architecture design and fast implementation. Motivated by constructive approaches for building shallow neural networks and randomized methods for fast learning, we make efforts to develop randomized learner model with deep architecture.

This paper is built on recent work reported in [32] where the way used to construct shallow neural networks with randomness (termed as stochastic configuration networks, SCNs) is original, innovative and effective. Extension of SCNs to deep version is not trivial, because it needs to prove that DeepSCNs share the universal approximation property, and also define some proper and meaningful criteria on node adding, layer growth, and construction termination throughout the learning process. Briefly, a DeepSCN starts with a small sized network (e.g, one hidden layer with one hidden node), and stochastically configures its nodes for the current layer until the hidden output matrix fails the full-rank condition, then continues to add the next hidden layer by repeating procedures as done in the last layer, keeps proceeding to deeper layers until an acceptable error tolerance is achieved. Specifically, as the constructive process proceeds, the hidden parameters are randomly assigned under a supervisory mechanism, while the read-out weights linking the hidden nodes to the output nodes are calculated by the least squares method. The success of SCNs can guarantee the convergence of error sequence approaching to certain accuracy, if a moderate number of hidden nodes are generated [32]. Once starting to add a new hidden layer, the last error function acts as a new target function to be approximated. An immediate benefit from DeepSCNs lies in the links between all hidden nodes and the output nodes, which allow end-users to manipulate layer-wise approximations, in other words, one can make options to use these nodes freely after off-line building an accurate DeepSCN model.

Our work focuses on framework development for randomized DNNs rather than addresses any specific applications, aiming to offer an advanced solution for the aforementioned challenges. Technical contributions are summarized as follows:

- Establish the universal approximation theory for DeepSCNs;
- Determine the architecture of DeepSCNs automatically without human interventions, and use full-rank based criterion to guide the deployment of DeepSCNs;
- Implement fast learning by stochastically assigning the hidden parameters and evaluating the read-out weights by least squares method.

The remainder of this paper is organized as follows: Section 2 reviews some related work concerning randomness in DNNs and optimization-based learning techniques. Section 3 details DeepSCNs framework with theoretical fundamentals on the universal approximation property, learning representation and algorithmic implementation. Section 4 reports some simulation results with comparisons against SCNs. Section 5 concludes this paper and provides some further research directions.

2 Related Work

Some researchers anticipated the usefulness of randomization in the development of neural networks [29]. In [20], it was found that a random filter in convolutional neural networks can perform slightly worse than a well-designed filter, which usually needs pre-training and discriminative
fine-tuning. Saxe et al. addressed this issue in [27] and showed that the results obtained from the randomized learner are comparable to that after regular pre-training and fine-tuning processes. These experimental results imply that the involvement of randomization in deep architectures has a good potential in enhancing the computational efficiency of the state-of-the-art systems without a significant deterioration in performance. Coates and Ng [8] tried to use random weights in unsupervised learning, and their experimental results suggested that randomization can be helpful to build large sized models very rapidly, with much ease in training and encoding than sparse coding techniques. Randomness was also concerned in the selection of local receptive fields [7], and the pooling operations of deep convolutional neural networks [35]. Arora et al. investigated learning of autoencoders with random weights and demonstrated that it is possible to train them in polynomial time under some restrictions on the network depth [2]. To speed the training process of back-propagation (BP) in DNNs, authors in [22] proposed a random feedback mechanism by multiplying error signals with random weights, which contributes to fast extracting useful information from signals sent through the connections. Motivated by a series of works reported in [9, 24], where they empirically showed some successful learning techniques based on randomization, Giryes et al. [14] theoretically proved that DNNs with random Gaussian weights can perform a stable embedding of the original data, permitting a stable recovery of the data from the learning representation. In [33], the authors stated that the capacity of multilayer perceptrons mainly depends on the number of nodes in the last hidden layer and associated output weights. Indeed, their method came up with the same philosophy of random projection for dimensionality reduction [5]. In [16], some investigations on the functionality of randomized DNNs for deep visualization were empirically conducted. Their findings demonstrate good potential for convolutional neural networks with random weights on visualization.

In [19], a constructive algorithm was proposed for building cascaded networks, where all hidden nodes are directly linked to the output nodes. At each layer, the nodes are successively added by optimizing an objective function (equivalent to searching for the hidden parameters). This suffers from all troubles occurring in gradient-based methods. The used criterion on layer growth is heuristic and lacks association with model’s generalization. In [34], the BP-like algorithm was developed for training this type of cascaded networks with specified architecture. However, their approaches are not efficient for building DNNs and require more human intervention in terms of learning parameter setting.

3 Deep Stochastic Configuration Networks

This section provides some fundamentals of DeepSCNs, including a theoretical result on the universal approximation property, interpretation on learning representation, algorithm description and some technical remarks.

3.1 Universal Approximation Property

Let $L_2(D)$ denote the space of all Lebesgue-measurable vector-valued functions $\mathcal{F} = [f_1, f_2, \ldots, f_m] : R^d \rightarrow R^m$ on a compact set $D \subset R^d$, with the $L_2$ norm defined as

$$
\|\mathcal{F}\| := \left(\sum_{q=1}^{m} \int_D |f_q(x)|^2 dx \right)^{1/2} < \infty,
$$

(1)

and inner product defined as

$$
\langle \mathcal{F}, \mathcal{G} \rangle := \sum_{q=1}^{m} \langle f_q, g_q \rangle = \sum_{q=1}^{m} \int_D f_q(x)g_q(x)dx,
$$

(2)

where $\mathcal{G} = [g_1, g_2, \ldots, g_m] : R^d \rightarrow R^m$.

Given a target function $\mathcal{F} : R^d \rightarrow R^m$, suppose a DeepSCN with $n$ hidden layers and each layer has $L_k$ hidden nodes $(k = 1, 2, \ldots, n)$, has been constructed, that is,

$$
\mathcal{F}_{L_n}^{(n)}(x) = \sum_{k=1}^{n} \sum_{j=1}^{L_k} \beta_j^{(k)} \phi_{k,j} \left( \phi_{k-1}^{(k-1)}(x), \beta_{k-1}^{(k-1)}(x), \beta_j^{(k-1)} \right),
$$

(3)
where $w_j^{(k-1)}$ and $b_j^{(k-1)}$ stand for the hidden parameters within the $k$-th hidden layer, $\phi_{k,j}$ is the activation function used in the $k$-th hidden layer, $x^{(0)} = x$ and $x^{(k)} = \Phi(x^{(k-1)}, W^{(k-1)}, B^{(k-1)}) = [\phi_{k,1}^{(k)}, \phi_{k,2}^{(k)}, \ldots, \phi_{k,L_k}^{(k)}]$, with $W^{(k-1)} = [w_1^{(k-1)}, w_2^{(k-1)}, \ldots, w_{L_k}^{(k-1)}]$ and $B^{(k-1)} = [b_1^{(k-1)}, b_2^{(k-1)}, \ldots, b_{L_k}^{(k-1)}], k = 1, 2, \ldots, n$.

Then, the residual error function is defined by $E_{L_n}^{(n)} = \mathcal{F} - \mathcal{F}_{L_n}^{(n)} = [\mathcal{E}_{L_n,1}^{(n)}, \mathcal{E}_{L_n,2}^{(n)}, \ldots, \mathcal{E}_{L_n,m}^{(n)}]$. Specifically, $\mathcal{E}_{0}^{(1)} = \mathcal{F}$ and $\mathcal{E}_{0}^{(n+1)} = \mathcal{E}_{L_n}^{(n)}, n = 1, 2, \ldots, n$.

The universal approximation property of DeepSCNs is stated in the following Theorem 1.

**Theorem 1.** Suppose that $\text{span}(\Gamma)$ is dense in $L_2$ space and $\forall \phi \in \Gamma$, $0 < \|\phi\| < c$ for some $c \in \mathbb{R}^+$. Given $0 < r < 1$ and a nonnegative decreasing sequence $\{\mu_1\}$ with $\lim_{t \to +\infty} \mu_t = 0$ and $\mu_t < (1 - r)$. For $n = 1, 2, \ldots$, and $j = 1, 2, \ldots, L_n + 1$, denoted by

$$\delta_j^{(n)} = (1 - r - \mu_j)\|\mathcal{E}_{j-1}^{(n)}\|^2.$$  

Stochastically configuring the $j$-th hidden node $\phi_{n,j}$ (within the $n$-th hidden layer) to satisfy the following inequality:

$$\sum_{q=1}^m \langle \mathcal{E}_{j,q}^{(n)}, \phi_{n,j} \rangle^2 \geq c^2 \delta_j^{(n)}, j = 1, 2, \ldots, L_n + 1.$$  

If the random basis functions $\phi_{n,1}, \phi_{n,2}, \ldots, \phi_{n,L_n+1}$ are linearly dependent on a compact set $D_n \subset \mathbb{R}^d$, i.e., for some real numbers $c_1, c_2, \ldots, c_{L_n+1}$ not all zero, the following holds

$$\sum_{j=1}^{L_n+1} c_j \phi_{n,j}(t) = 0, t \in D_n,$$  

fix $L_n$ and start to add the first hidden node $\phi_{n+1,1}$ in the $(n+1)$-th hidden layer according to the following inequality

$$\sum_{q=1}^m \langle \mathcal{E}_{L_n,q}^{(n)}, \phi_{n+1,1} \rangle^2 \geq c^2 \delta_{L_n}^{(n)}.$$  

Keep adding new hidden nodes within the $(n+1)$-th hidden layer based on (5), followed by generating the first hidden node in a new hidden layer via (7) (as (6) holds). After adding one hidden node (either in the present hidden layer or starting a new hidden layer), the read-out weights are evaluated by the least squares method, that is,

$$\beta^* = \arg \min_{\beta} \|\mathcal{F} - \sum_{k=1}^n \sum_{j=1}^{L_k} \beta_j^{(k)} \phi_{k,j}\|^2.$$  

Then, we have $\lim_{n \to +\infty} \|\mathcal{F} - \mathcal{F}_{L_n}^{(n)}\| = 0$, where $\mathcal{F}_{L_n}^{(n)}$ is defined by (1).

**Proof.** Based on Theorem 7 in [1], we can easily obtain that

$$\|\mathcal{E}_{L_n}^{(n)}\|^2 \leq (r + \mu_{L_n})\|\mathcal{E}_{L_n-1}^{(n)}\|^2.$$  

The error after adding one hidden node in the $(n+1)$-hidden layer can be expressed as

$$\|\mathcal{E}_{L_n}^{(n+1)}\|^2 = \|\mathcal{E}_{L_n}^{(n)} - \beta_{n+1}^{(n+1)} \phi_{n+1,1}\|^2,$$

where $\phi_{n+1,1}$ and $\beta_{n+1}^{(n+1)}$ are obtained by using (7) and (8) with , respectively. To identify the relationship between $\mathcal{E}_{1}^{(n+1)}$ and $\mathcal{E}_{L_n}^{(n)}$, we denote $\beta_{1}^{(n+1)} = [\beta_{1,1}^{(n+1)}, \ldots, \beta_{1,m}^{(n+1)}]^T$, where

$$\beta_{1,q}^{(n+1)} = \frac{\langle \mathcal{E}_{L_n,q}^{(n)}, \phi_{n+1,1} \rangle}{\|\phi_{n+1,1}\|^2}, q = 1, 2, \ldots, m,$$  

where $\phi_{n,1}^{(k-1)}$ and $b_{n,1}^{(k-1)}$ stand for the hidden parameters within the $k$-th hidden layer, $\phi_{k,j}$ is the activation function used in the $k$-th hidden layer, $x^{(0)} = x$ and $x^{(k)} = \Phi(x^{(k-1)}, W^{(k-1)}, B^{(k-1)}) = [\phi_{k,1}^{(k)}, \phi_{k,2}^{(k)}, \ldots, \phi_{k,L_k}^{(k)}]$, with $W^{(k-1)} = [w_1^{(k-1)}, w_2^{(k-1)}, \ldots, w_{L_k}^{(k-1)}]$ and $B^{(k-1)} = [b_1^{(k-1)}, b_2^{(k-1)}, \ldots, b_{L_k}^{(k-1)}], k = 1, 2, \ldots, n$.
Replacing $\beta_1^{(n+1)}$ by $\bar{\beta}_1^{(n+1)}$, with $\bar{\beta}_1,q^{(n+1)}$ evaluated by (11), we have
\[
\|E_1^{(n+1)}\|_2 \leq \|E_L^{(n)} - \bar{\beta}_1^{(n+1)}\phi_{n+1,1}\|_2
\]
\[
= \|E_L^{(n)}\|_2 - \sum_{q=1}^{m} \langle E_{L,q}^{(n)}, \phi_{n+1,1} \rangle_2 \|\phi_{n+1,1}\|_2^2
\]
\[
\leq \|E_L^{(n)}\|_2^2.
\]  
With the help of (9), we can obtain
\[
\|E_{L,n+1}^{(n+1)}\|_2 \leq \|E_1^{(n+1)}\|_2 \leq \|E_L^{(n)}\|_2
\]
\[
\leq \prod_{k=1}^{n} (r + \mu L_k) \|\phi_1^{(n)}\|_2^2
\]
\[
\leq (r + \mu_1) \sum_{k=1}^{n} L_k \|\phi_1^{(1)}\|_2^2.
\]  
Let $n \to +\infty$ for both sides of (13), we have $\lim_{n \to +\infty} \|E_{L,n+1}^{(n+1)}\| = 0$. That completes the proof.

3.2 Learning Representation

The idea of learning internal representation can be traced back to 80’s [26]. In the past decades, many exciting researches and progresses have been reported in literature [12, 23], where attempts are made to meet the following three properties: fidelity, sparsity and interpretability. Learner models can appear in different forms but essentially they must share the universal approximation property so that the fidelity can be achieved. Theorem 1 above ensures that DeepSCNs fit the essential prerequisite well for learning representation. Due to the special architecture of DeepSCNs, learning representation becomes quite straightforward and understandable. The outcomes from the first hidden layer contain two parts: a set of random basis functions with the original inputs, and associated read-out weights between the first hidden layer and the output layer. Apparently, one can define a learning representation from the first hidden layer (indeed, a SCN model), which characterizes the main feature of the target function. Next, the outputs from the first hidden layer, as a new set of inputs, can be fed into the next hidden layer to constructively generate a set of random basis functions with the cascaded inputs, and associated read-out weights between the second hidden layer and the output layer. Again, one can obtain a refined learning representation from the second hidden layer. This procedure keeps going on until some termination criterion is satisfied. Figure 1 shows a typical DeepSCN with $n = 2$, $L_1 = 3$, and $L_2 = 2$, $d = 2$, $m = 1$.

![Typical DeepSCN](image)

Figure 1: A typical DeepSCN with the architecture: 2-3-2-1, i.e., two input nodes, two hidden layers (three hidden nodes in the first hidden layer, two hidden nodes in the second hidden layer), and one output node. Read-out weights (trainable) are shown as red dashed lines, whilst hidden weights and biases (stochastically configured) are shown as fixed lines.

It should be highlighted that DeepSCNs advocate fully connections between each hidden layer and the output layer, which supports the diversity in learning representation for the target function by
using an explicit span of the basis functions stochastically configured (corresponds to \( \phi_{k,j} \) in \( \beta \)), instead of using only the compositional expression from the last hidden layer to conduct signal representation. Note that the set of random basis functions generated by the proposed DeepSCNs are data dependent and with cascaded inputs. According to \( \beta \), DeepSCNs can be regarded as a producer to output learning representations \( \beta \). Indeed, such a learning representation offers end-users more flexibility, for instance, some nodes at certain layer can be discarded according to some criteria, leading to a new learning representation with sparsity.

\[
\mathcal{F} \approx \begin{pmatrix}
\phi_{1,1} & \cdots & \phi_{1,L_1} \\
\beta_{1}^{(1)} & \cdots & \beta_{1}^{(L_1)} \\
\vdots & \ddots & \vdots \\
\phi_{n,1} & \cdots & \phi_{n,L_n} \\
\beta_{n}^{(1)} & \cdots & \beta_{n}^{(L_n)}
\end{pmatrix}
\]

3.3 Algorithm Description

Given a training dataset with inputs \( X = \{x_1, x_2, \ldots, x_N\} \), \( x_i = [x_{i,1}, \ldots, x_{i,d}]^T \in \mathbb{R}^d \) and outputs \( T = \{t_1, t_2, \ldots, t_N\} \), where \( t_i = [t_{i,1}, \ldots, t_{i,m}]^T \in \mathbb{R}^m \), \( i = 1, \ldots, N \). Denote \( \mathcal{E}_{L_{n-1}}^{(n)} = \mathcal{E}_{L_{n-1}}^{(n)}(X) = [\mathcal{E}_{L_{n-1},1}^{(n)}(X), \ldots, \mathcal{E}_{L_{n-1},m}^{(n)}(X)]^T \) as the corresponding residual error vector before the \( L_n \)-th new hidden node of the \( n \)-th hidden layer is added, where \( \mathcal{E}_{L_{n-1},q}^{(n)}(X) = [\mathcal{E}_{L_{n-1},q,1}^{(n)}(x_1), \ldots, \mathcal{E}_{L_{n-1},q,1}^{(n)}(x_N)] \in \mathbb{R}^N \), \( q = 1, 2, \ldots, m \).

After adding the \( L_n \)-th new hidden node in the \( n \)-th hidden layer, we get

\[
h_{L_n}^{(n)} := h_{L_n}^{(n)}(X) = [\phi_{n,L_n}(x_{1}^{(n-1)}), \ldots, \phi_{n,L_n}(x_{N}^{(n-1)})]^T,
\]

where \( \phi_{n,L_n}(x_{i}^{(n-1)}) \) is used to simplify \( \phi_{n,L_n}(x_{i}^{(n-1)}; w_j^{(n-1)} \theta_j^{(n-1)}) \), and \( x_i^{(0)} = x_i = [x_{i,1}, \ldots, x_{i,d}]^T \), \( x_i^{(n-1)} = \Phi(x_i^{(n-2)}; W^{(n-1)}, H^{(n-1)}) \) for \( n \geq 2 \). Let \( H_{L_n}^{(n)} = [h_{1}^{(n)}, h_{2}^{(n)}, \ldots, h_{L_n}^{(n)}] \) represent the hidden layer output matrix and denote a temporary variable \( \theta_{L_n}^{(n)} \) to make it convenient in the followed algorithm description.

\[
\theta_{L_n}^{(n)} = \frac{\sum_{q=1}^{m} (\mathcal{E}_{L_{n-1},q}^{(n)} h_{L_n}^{(n)})^2_{L_q}}{(h_{L_n}^{(n)}, h_{L_n}^{(n)})_{L_q}} - \sum_{q=1}^{m} (1 - r - \mu_L) (\mathcal{E}_{L_{n-1},q}^{(n)} \mathcal{E}_{L_{n-1},q}^{(n)})_{L_q},
\]

where \( \langle \cdot, \cdot \rangle_{L_q} \) denotes the dot product and we omit the argument \( X \) in \( \mathcal{E}_{L_{n-1},q}^{(n)} \) and \( h_{L_n}^{(n)} \).

3.4 Further Remark

In essence, the universal approximation property ensures the capacity of DeepSCNs for both data modelling and signal representation. Thus, DeepSCNs can be employed as either predictive models or feature extractors in domain applications. For regression problems, one may be more interested in the predictability of a learner model rather than its learning capability. Unfortunately, it is almost impossible to directly establish some certain correlation between these two performances. That is, a better learning performance does not always imply a sound generalization. In this regard, consistency concept becomes a meaningful metric to assess the goodness of learning machines. From our hands-on experience, DeepSCNs demonstrate very good consistency performance, and it is believed that this merit is associated with both the proposed full-rank terminal criterion and the supervisory mechanism for assigning the random hidden parameters. For classification problems, DeepSCNs can be used directly to put a class label for a testing data by using linear discriminate analysis. Alternatively, we can use DeepSCNs as feature extractors to generate a collection of samples in the DeepSCN feature space, followed by using variety of feature-classifiers to implement classification. In a nutshell, as results of deep learning for DeepSCNs, a set of data dependent random basis functions and \( \beta \) are generated for signal representation with piece-wise ‘resolutions’.

4 Empirical Demonstration

This section reports some simulation results to illustrate advantages of DeepSCNs compared against SCNs in terms of the approximation accuracy, quality of learning representation and model’s
DeepSCN Algorithm (DSCN)

Given inputs $X = \{x_1, x_2, \ldots, x_N\}$, $x_i \in \mathbb{R}^d$ and outputs $T = \{t_1, t_2, \ldots, t_N\}$, $t_i \in \mathbb{R}^n$;
Set $M$ as the maximum number of hidden layers, $L_{max}$ as the maximum number of hidden nodes within the $n$-th hidden layer, $1 \leq n \leq M$, as the expected error tolerance, $T_{max}$ as the maximum times of random configuration; Choose a set of scalars $\mathcal{T} = \{\lambda_{min}; \Delta \lambda; \lambda_{max}\}$.

1. Initialize $\mathcal{E}_0 = [t_1, t_2, \ldots, t_N]^T$, $0 < r < 1$, three empty sets $\mathcal{H}, \mathcal{W}, \mathcal{D}$.
2. While $\eta \leq M$ AND $\|\mathcal{E}_0\|_F > \epsilon$, Do
   3. While $L_n \leq L_{max}$ AND $\|\mathcal{E}_0\|_F > \epsilon$, Do
     4. For $\lambda \in \mathcal{T}$, Do
        5. For $k = 1, 2, \ldots, T_{max}$, Do
           6. Randomly assign $\omega_{L_n}^{(n-1)}$ and $b_{L_n}^{(n-1)}$ from $[-\lambda, \lambda]^d$ and $[-\lambda, \lambda]$, respectively;
           7. Calculate $h_{L_n}(\omega_{L_n}), \theta_{L_n}^{(n)}$ based on Eq. (15) and (16), set $\mu_{L_n} = (1-r)/(L_n+1)$;
           8. If $\theta_{L_n}^{(n)} \geq 0$
              9. Save $w_{L_n}^{(n-1)}$ and $b_{L_n}^{(n-1)}$ in $\mathcal{W}, \theta_{L_n}^{(n)}$ in $\mathcal{H}$;
             Else go back to Step 5
        10. End If
       11. End For (corresponds to Step 5)
      12. End For (corresponds to Step 4)
     13. If $W$ is not empty
        14. Find $w_{L_n}^{(n-1)*}, b_{L_n}^{(n-1)*}$ maximizing $\theta_{L_n}^{(n)}$ in $\mathcal{W}$, set $H_{L_n}^{(n)} = [h_{L_n}^{(n)*}, \ldots, h_{L_n}^{(n)*}]$;
        15. Break (go to Step 19);
     16. Else Randomly take $\tau \in (0, 1-r)$, renew $r := r + \tau$, return to Step 5;
      17. End If
     18. End For (corresponds to Step 3)
    19. If $H^{(n)}_{L_n}$ is full rank
        20. Set $\mathcal{H} = [\mathcal{H}, H^{(n)}_{L_n}], \beta^* = \mathcal{H}^T, \mathcal{E}^{(n)}_L := \mathcal{H}\beta^* - T, \mathcal{E}^{(n)} = E_{L_n}$;
       21. Else Set $\mathcal{H} = [\mathcal{H}, \tilde{H}^{(n)}_{L_n}], \mathcal{E}^{(n)} = E_{L_n}$;
        22. Calculate $\beta^* = \mathcal{H}^T, \mathcal{E}^{(n)} = \mathcal{H}\beta^* - T, L_n := L_n - 1$;
        23. Break (Go to Step 27)
    24. End If
    25. Renew $\mathcal{E}_0 := \mathcal{E}_{L_n}$, and $L_n := L_n + 1$;
    26. End While (corresponds to Step 3)
    27. Set $\mathcal{E}_{0+1} := \mathcal{E}_{L_n}$, and $\mathcal{L}_{\mathcal{W}} := L_n + 1$;
    28. End While (corresponds to Step 2)
    29. Return $\beta^* = [\beta_{1}^{(1)*}, \ldots, \beta_{L_{max}}^{(1)*}], \omega^* = [w_{1}^{(0)*}, \ldots, w_{L_{max}}^{(n-1)*}], b^* = [b_{1}^{(0)*}, \ldots, b_{L_{max}}^{(n-1)*}]$.

capacity. Here, we also provide a robustness analysis on DSCN algorithm with respect to the learning parameter setting. The following real-valued function is employed in the simulation study:

$$f(x) = 0.2e^{-(10x-4)^2} + 0.5e^{-(80x-40)^2} + 0.3e^{-(80x-20)^2}, \quad x \in [0, 1].$$

In this study, we take 1000 points (randomly generated) as the training data and another 1000 points (randomly generated) as the test data in our experiments. The sigmoidal activation function $g(x) = 1/(1 + \exp(-x))$ is used here for all hidden nodes.

4.1 Effectiveness and Consistency

Figure 2 depicts the training and test performances, respectively, where $L = 200$ for SCN, $M = 4$ and $L_{max} = 50$ for each layer in DeepSCN. It is clear that DeepSCN approximates the target function within a given tolerance more faster than SCN. Also, the consistency relationship between learning and generalization can be observed. As a matter of fact, both shallow and deep SCNs share this nice consistency property. It should be clearly pointed out that our statement on the consistency comes from a large number of experimental observations, and theoretical justifications are being expected. Figure 2 shows three independent trials, which indicate that DeepSCN outperforms SCN in terms of effectiveness. In this test, we set $\tau_k = 1 - 10^{-k}, k = 1, 2, \ldots, 7$ in Step 16 of DSCN algorithm. With different settings, the performance on both effectiveness and consistency could be further verified by the results from Figure 5.
From a mathematical perspective, both SCN and DeepSCN can produce a set of random basis functions. The difference between them lies in the way how to generate the random basis functions, which relate to the input variables and random parameters of learner models. It is difficult to quantify the quality of learning representations. In this paper, we favourably consider a learning representation if the distribution of the read-out weights in (14) has sound statistical characteristics, for instance, lower expectation and standard deviation. In Figure 3, two normalized distributions of the read-out weights (converted into [-1,1] on the basis of 200 coefficients from both SCN and DeepSCN) are plotted, and the approximation results for the test dataset are displayed beside the distributions, respectively. As can be seen that DeepSCN has a set of concentrated read-out weights and performs much better in generalization than SCN, which indeed has a quite poor distribution of the read-out weights with a few of peaking values. In this scenario, DeepSCN has four hidden layers and \( L_{max} = 25 \) is applied for each layer (without use of the full-rank terminal condition).

4.3 Model Capacity versus Rank Deficiency

In common sense, a learner model’s capacity can be measured by using a trade-off metric over the learning and generalization performance. In addition, we may evaluate the capacity of a learner model by looking at the interpretability. In fact, it is always desirable to have a learner model with balanced performance among learning, generalization and model complexity. For DeepSCN framework, due to the specific configuration in model building, the model capacity is not only associated with the number of nodes for each hidden layer, but also closely related to some algebraic properties of the hidden output matrix at each hidden layer. Technically, it is very interesting and useful to look into the relationship between the rank-deficiency property of the joint hidden output matrix and the model’s generalization capability. To do so, the degree of rank-deficiency is computed by \( p = \frac{\text{rank}(\mathcal{H})}{\sum_{n=1}^{j} L_n} \), where \( \mathcal{H} \) represents the hidden output matrix composed of all existing hidden nodes, \( \sum_{n=1}^{j} L_n \) represents the (current) number of hidden nodes after adding \( j (j = 1, 2, 3, 4) \) hidden layers and \( L_j \) hidden nodes \( 1 \leq L_j \leq 25 \). The values of \( p \) for both SCN and DeepSCN are plotted in Figure 4 (a), in which it can be observed that \( \mathcal{H} \) of DeepSCN is with higher rank even when the number of hidden nodes is approaching to 100, however, the rank-deficiency level of SCN becomes much more higher as \( L \) is larger. Note that the results from DeepSCN reported in Figure (a) were obtained with a fixed architecture (i.e., 4 hidden layers with 25 nodes for each) and without use of the full-rank terminal criterion. Figure 4 (b) depicts six curves of rank-deficiency for SCN and DeepSCN, where \( M = 4 \) and \( L_{max}^{(n)} = 50 \) for each hidden layer are used in DSCN algorithm. By referring to the previous results in comparison with SCN, we infer that the DeepSCN model capacity has stronger correlation to the rank of the joint hidden output matrix.
4.4 Robustness Analysis

In general, robustness refers to the ability of a system to maintain its performance whilst subjected to noise in external inputs, changes to internal structure and/or shifts in parameter setting. Obviously, the quality or performance of DeepSCNs depends upon the parameter setting. In order to investigate the robustness of DSCN algorithm, in this paper we limit our study on the learning parameter $r$ with 11 values only. A similar robustness analysis on another set of key learning parameters $\Upsilon = \{\lambda_{\text{min}} : \Delta \lambda : \lambda_{\text{max}}\}$ is not reported here. Figure 5 shows the training and test performances for both SCN and DeepSCN with three different settings of $r$, that is, randomly taking 10 real numbers from the open interval $(0.9, 0.99)$ and arranging them in increasing order to form a set $r_1$, and set $r = \{r_1, 1 - 10^{-6}\}$. As can be seen that both SCN and DeepSCN perform robustly against this learning parameter setting (i.e., not sensitive to the uncertain sequence $r$). In this case, we fixed the architecture of DeepSCN (i.e., with 4 hidden layers and 25 nodes for each) and kept the full-rank terminal criterion free. A remarkable drop-down can be observed after completing the construction of the first layer, which clearly exhibits the advantage of DeepSCN over SCN.

5 Conclusions

Although many empirical proofs demonstrate great potential of deep neural networks for learning representation, it is still blind for end-users to deploy the network architecture so that the resulting deep learner model has sufficient capacity to approximately represent signals in some ways. Except for concerns on the architecture, one cares much about fast learning algorithms. In this paper, our proposed DeepSCNs, as a class of randomized deep neural networks, offer a fast and feasible pathway for problem solving. Before ending up this paper, we would like to highlight and summarize again our technical contributions to the working field as follows:
The supervisory mechanism characterised by the inequalities (5) and (7) is the key for developing DeepSCN framework, which was originally proposed in [32] and could be used as a unique feature to distinguish our SCN framework from other randomized learner models.

- The scopes of random parameters at each layer of DeepSCNs could be updated adaptively. Indeed, this must be done to make the set of basis functions rich and functional.
- The termination criterion, i.e., the full-rank condition, for keep adding hidden nodes at the same layer is original and insightful. It is strongly believed that this criterion is an appropriate option although there is no theoretical justification in place at the moment.

Further researches on this topic move into two main clusters: theoretical aspects, including further explorations on various supervisory mechanisms, scientific justifications on the proposed full-rank criterion, studies on the existence of the global random basis functions (GRBFs) for a collection of functions, and looking into more complicated supervisory mechanisms to generate GRBFs, development of learning representation with sparsity and interpretability; practical aspects, comparing with other deep neural networks for gaining more understandings on merits and limits of DeepSCNs, and applying DeepSCNs for real world applications.

References
[1] J. Alvarez and M. Salzmann  Learning the number of neurons in deep networks. In Advances in Neural Information Processing Systems, pages 2262–2270, 2016.
[2] S. Arora, A. Bhaskara, R. Ge, and T. Ma. Provable bounds for learning some deep representations. In Proceedings of the 31th International Conference on Machine Learning, pages 584–592, 2014.

[3] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13:281–305, 2012.

[4] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8):1798–1828, 2013.

[5] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining, pages 245–250, 2001.

[6] A. Choromanska, M. Henaff, M. Mathieu, G. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In Proceedings of the 18th International Conference on Artificial Intelligence and Statistics, pages 192–204, 2015.

[7] A. Coates and A. Ng. Selecting receptive fields in deep networks. In Advances in Neural Information Processing Systems, pages 2528–2536, 2011.

[8] A. Coates and A. Ng. The importance of encoding versus training with sparse coding and vector quantization. In Proceedings of the 28th International Conference on Machine Learning, pages 921–928, 2011.

[9] D. Cox and N. Pinto. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. In Proceedings of 2011 IEEE International Conference on Automatic Face & Gesture Recognition and Workshops, pages 8–15, 2011.

[10] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Advances in Neural Information Processing Systems, pages 2933–2941, 2014.

[11] R. Eldan and O. Shamir. The power of depth for feedforward neural networks. arXiv preprint arXiv:1512.03965, 2015.

[12] Y. C. Eldar and G. Kutyniok. Compressed sensing: theory and applications. Cambridge University Press, 2012.

[13] L. Gatys, A. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In Advances in Neural Information Processing Systems, pages 262–270, 2015.

[14] R. Giryes, G. Sapiro, and A. Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? IEEE Transactions on Signal Processing, 64(13):3444–3457, 2016.

[15] I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. MIT Press, 2016.

[16] K. He, Y. Wang, and J. Hopcroft. A powerful generative model using random weights for the deep image representation. In Advances In Neural Information Processing Systems, pages 631–639, 2016.

[17] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. Neural Computation, 18(7):1527–1554, 2006.

[18] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 313(5786):504–507, 2006.

[19] M. Islam, M. Sattar, M. Amin, X. Yao, and K. Murase. A new constructive algorithm for architectural and functional adaptation of artificial neural networks. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(6):1590–1605, 2009.

[20] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. Lecun. What is the best multi-stage architecture for object recognition? In Proceedings of the 12th IEEE International Conference on Computer Vision, pages 2146–2153, 2009.

[21] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. Nature, 521(7553):436–444, 2015.

[22] T. Lillicrap, D. Cownden, D. Tweed, and C. Akerman. Random feedback weights support learning in deep neural networks. arXiv preprint arXiv:1411.0247, 2014.

[23] C. M. O’Brien. Statistical Learning with Sparsity: The Lasso and Generalizations. Wiley Online Library, 2016.
[24] N. Pinto, D. Doukhan, J. DiCarlo, and D. Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PloS Computational Biology*, 5(11):e1000579, 2009.

[25] N. Roux and Y. Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, 20(6):1631–1649, 2008.

[26] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning internal representations by backpropagating errors, *Nature* 323 (1986) 533-536.

[27] A. Saxe, P. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1089–1096, 2011.

[28] A. Saxe, J. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proceedings of International Conference on Learning Representations*, 2014.

[29] S. Scardapane and D. Wang. Randomness in neural networks: An overview. *WIREs Data Mining and Knowledge Discovery*, 2017, 7:e1200. doi: 10.1002/widm.1200.

[30] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of International Conference on Learning Representations*, 2014.

[32] D. Wang and M. Li. Stochastic configuration networks: Fundamentals and algorithms. *arXiv preprint arXiv:1702.03180*, 2017.

[33] B. Widrow, A. Greenblatt, Y. Kim, and D. Park. The no-prop algorithm: A new learning algorithm for multilayer neural networks. *Neural Networks*, 37:182–188, 2013.

[34] B. M. Wilamowski. Neural network architectures and learning algorithms. *IEEE Industrial Electronics Magazine*, 3(4):56–63, 2009.

[35] M. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.