Deep Stochastic Configuration Networks with Universal Approximation Property

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 develops a randomized approach for incrementally building deep neural networks, where a supervisory mechanism is proposed to constrain the random assignment of the weights and biases, and all the hidden layers have direct links to the output layer. A fundamental result on the universal approximation property is established for such a class of randomized leaner models, namely deep stochastic configuration networks (DeepSCNs). A learning algorithm is presented to implement DeepSCNs with either specific architecture or self-organization. The read-out weights attached with all direct links from each hidden layer to the output layer are evaluated by the least squares method. Given a set of training examples, 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. An empirical demonstration on a function approximation and a case study are carried out to demonstrate some properties of the proposed deep learner model.

1 Introduction

In the past decade, deep neural networks (DNNs) have received considerable attention due to its great potential for dealing with computer vision, pattern recognition, natural language processing, etc. [1–3]. The success of deep learning is attributed to its representation capability for visual data [4, 5]. In view of some empirical evidence, DNNs are becoming increasingly popular because of a hypothesis that a deep learner model can be more effective and efficient in data representation aspect, compared to shallow models. It seems that DNNs have good potential in learning higher-level abstractions (or multiple levels of features), which sounds difficult for models with shallow architecture. A formal analysis on the representation power and learning complexity of DNNs can be found in [6]. In short, the community has reached a common sense that DNNs are much more expressive than the shallow ones.

DNNs became so popular due to their learning representation power for complex data, however, there are some issues in the design and implementation. For instance, how to smartly choose the architecture of the DNN, improve the learning speed and reduce the computational burden. Usually, one configures a DNN manually based on the trial-and-error performance without technical guidance, which may be time-consuming and potentially impacts the effectiveness of the resulting model. Indeed, DNNs with moderate depth but fewer number of hidden nodes at each layer cannot exhibit sufficient learning ability, while a large sized architecture (i.e. with quite a few layers and hidden nodes) may cause over-fitting that degrades the generalization capability. In [7], Alvarez and Salzmann introduced an approach to automatically determine the number of nodes at each layer of the DNN by using group sparsity regularizer. 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

[*]Corresponding author.

problem, fast training algorithms for DNNs are significant as well. As can be seen that training multilayer perceptrons by using the back-propagation algorithm suffers from several issues such as the weight initialisation, local minima and sensitivity of the learning performance with respect to the learning rate setting. Empirically, this gradient-based learning method could not produce meaningful or interpretable internal representations from each hidden outputs [8]. Therefore, some advanced deep learning techniques have been developed to overcome these obstacles [9]. To the best of our knowledge, there are few deep learning techniques in literature addressing the design and fast implementation simultaneously. In [10], the authors explored the learning capability of autoencoders with random weights scoped in [-1,1], showing the value of the randomized methods in building DNNs with much less computational complexity. Recently, some theoretical results on the universal classification strategy using DNN with random weights were reported in [11], where the authors mathematically showed that DNNs with random Gaussian weights perform a distance-preserving embedding of the data. Motivated by constructive approaches for building shallow neural networks [12, 13] and randomized methods for fast learning [14–21], we make efforts to develop randomized learner model with deep architecture under the philosophy: 'one should walk on the earth rather than in the air'!

This paper further develops the stochastic configuration networks (SCNs) proposed in [22] and establishes the universal approximation property for its deep version, termed as DeepSCNs. From an algorithmic perspective, 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 a certain termination criterion is met, then continues to add the next hidden layer by repeating the same procedures, and keeps proceeding to deeper layers until an acceptable error tolerance is achieved. 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 [23]. Our basic idea behind the DeepSCN is to successively approximate a target function, and the random basis functions for each hidden layer are generated under a set of inequality constraints. Once starting to add a new hidden layer, the obtained random bases remain unchanged and 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. Furthermore, DeepSCN can produce more rich learning representations, which provide an alternative solution for fast building autoencoders. In a nutshell, as results of deep learning for DeepSCNs, a set of data dependent random basis functions are generated for signal representation with piece-wise 'resolutions'.

The remainder of this paper is organized as follows: Section 2 details our DeepSCNs framework with theoretical fundamentals on the universal approximation property and algorithm description. Section 3 reports some simulation results with comparisons against SCNs. Section 4 concludes this paper and provides some further research directions.

2 Deep Stochastic Configuration Networks

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

2.1 Universal Approximation Property

Let $\mathcal{L}_2(D)$ denote the space of all Lebesgue-measurable vector-valued functions $\mathcal{F} = [f_1, f_2, \ldots, f_m] : R^d \to R^m$ on a compact set $D \subset R^d$, with the $\mathcal{L}_2$ norm defined as

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

and inner product defined as

$$\langle \mathcal{F}, \mathcal{G} \rangle := \sum_{q=1}^{m} \langle f_q, g_q \rangle = \sum_{q=1}^{m} \int_D f_q(x) g_q(x) dx, \tag{2}$$

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

Given a target function $\mathcal{F} : R^d \to R^m$, 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,

$$\mathcal{F}^{(n)}_{S_n}(x) = \sum_{k=1}^{n} \sum_{j=1}^{L_k} \beta^{(k)}_j \phi_{k,j}\Big(x^{(k-1)}; w^{(k-1)}_j, b^{(k-1)}_j\Big), \tag{3}$$

where $S_n = \{L_1, L_2, \ldots, L_n\}$ represents the set of the hidden node numbers, $w^{(k-1)}_j$ and $b^{(k-1)}_j$ 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 $x^{(k)} = \Phi\Big(x^{(k-1)}; W^{(k-1)}, B^{(k-1)}\Big) = [\phi_{k,1}, \phi_{k,2}, \ldots, \phi_{k,L_k}]$, with $W^{(k-1)} = [w^{(k-1)}_1, w^{(k-1)}_2, \ldots, w^{(k-1)}_{L_k}]$ and $B^{(k-1)} = [b^{(k-1)}_1, b^{(k-1)}_2, \ldots, b^{(k-1)}_{L_k}]$, $k = 1, 2, \ldots, n$. Then, the residual error function is defined by $\mathcal{E}^{(n)}_{S_n} = \mathcal{F} - \mathcal{F}^{(n)}_{S_n} = [\mathcal{E}^{(n)}_{S_n,1}, \mathcal{E}^{(n)}_{S_n,2}, \ldots, \mathcal{E}^{(n)}_{S_n,m}]$.

Denoted by $\mathcal{F}^{(n)}_p(x)$ as a DeepSCN model with p nodes at the n-th layer, that is,

$$\mathcal{F}^{(n)}_p(x) = \sum_{k=1}^{n} \sum_{j=1}^{p} \beta^{(k)}_j \phi_{k,j}\Big(x^{(k-1)}; w^{(k-1)}_j, b^{(k-1)}_j\Big), \tag{4}$$

where $p = 1, 2, ..., L_n$. Specifically, $\mathcal{E}^{(1)}_0 = \mathcal{F}$ and $\mathcal{E}^{(n+1)}_0 = \mathcal{E}^{(n)}_{L_n}$, $n = 1, 2, \ldots$.

Theorem 1. Suppose that span(Γ) is dense in $\mathcal{L}_2$ space and $\forall \phi \in \Gamma$, $0 < \|\phi\| < c$ for some $c \in R^+$. Given $0 < r < 1$ and a nonnegative decreasing sequence $\{\mu_l\}$ with $\lim_{l\to+\infty} \mu_l = 0$ and $\mu_l < (1 - r)$. For $n = 1, 2 \ldots$, and $j = 1, 2, \ldots, L_n$, denoted by

$$\delta^{(n)}_{j,q} = (1 - r - \mu_j)\|\mathcal{E}^{(n)}_{j-1,q}\|^2, q = 1, 2, \ldots, m. \tag{5}$$

Stochastically configuring the j-th hidden node $\phi_{n,j}$ within the n-th hidden layer ($j = 1, 2, \ldots, L_n$) to satisfy the following inequalities:

$$\langle \mathcal{E}^{(n)}_{j,q}, \phi_{n,j}\rangle^2 \geq c^2 \delta^{(n)}_{j,q}, q = 1, 2, \ldots, m. \tag{6}$$

Fix the random basis functions $\phi_{n,1}, \phi_{n,2}, \ldots, \phi_{n,L_n}$ and start to add the first hidden node $\phi_{n+1,1}$ in the $(n + 1)$-th hidden layer according to the following inequalities

$$\langle \mathcal{E}^{(n)}_{L_n,q}, \phi_{n+1,1}\rangle^2 \geq c^2 \delta^{(n)}_{L_n,q}, q = 1, 2, \ldots, m. \tag{7}$$

Keep adding new hidden nodes within the $(n+1)$-th hidden layer based on (6), followed by generating the first hidden node in a new hidden layer via (7). 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,

$$\beta^* = \arg\min_{\beta} \|\mathcal{F} - \sum_{k=1}^{n} \sum_{j=1}^{L_k} \beta^{(k)}_j \phi_{k,j}\|^2. \tag{8}$$

Then, we have $\lim_{n\to+\infty} \|\mathcal{F} - \mathcal{F}^{(n)}_{S_n}\| = 0$, where $\mathcal{F}^{(n)}_{S_n}$ is defined by (3).

Proof. It is easy to obtain that

$$\|\mathcal{E}^{(n)}_{L_n}\|^2 \leq (r + \mu_{L_n})\|\mathcal{E}^{(n)}_{L_n-1}\|^2. \tag{9}$$

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

$$\|\mathcal{E}^{(n+1)}_1\|^2 = \|\mathcal{E}^{(n)}_{L_n} - \beta^{(n+1)}_1 \phi_{n+1,1}\|^2, \tag{10}$$

where $\phi_{n+1,1}$ and $\beta^{(n+1)}_1$ are obtained by using (7) and (8) with , respectively. To identify the relationship between $\mathcal{E}^{(n+1)}_1$ and $\mathcal{E}^{(n)}_{L_n}$, we denote $\bar{\beta}^{(n+1)}_1 = [\bar{\beta}^{(n+1)}_{1,1}, \ldots, \bar{\beta}^{(n+1)}_{1,m}]^{\mathrm{T}}$, where

$$\bar{\beta}^{(n+1)}_{1,q} = \frac{\langle \mathcal{E}^{(n)}_{L_n,q}, \phi_{n+1,1}\rangle}{\|\phi_{n+1,1}\|^2}, q = 1, 2, \ldots, m, \tag{11}$$

Replacing $\beta_1^{(n+1)}$ by $\bar{\beta}_1^{(n+1)}$, with $\bar{\beta}_{1,q}^{(n+1)}$ evaluated by (11), we have

$$\begin{aligned}\|\mathcal{E}_1^{(n+1)}\|^2 &\leq \|\mathcal{E}_{L_n}^{(n)} - \bar{\beta}_1^{(n+1)}\phi_{n+1,1}\|^2 \\ &= \|\mathcal{E}_{L_n}^{(n)}\|^2 - \frac{\sum_{q=1}^{m}\langle \mathcal{E}_{L_n,q}^{(n)}, \phi_{n+1,1}\rangle^2}{\|\phi_{n+1,1}\|^2} \\ &\leq \|\mathcal{E}_{L_n}^{(n)}\|^2. \end{aligned} \tag{12}$$

With the help of (9) and (12), we can obtain

$$\begin{aligned}\|\mathcal{E}_{L_{n+1}}^{(n+1)}\|^2 &\leq \|\mathcal{E}_1^{(n+1)}\|_2^2 \leq \|\mathcal{E}_{L_n}^{(n)}\|^2 \\ &\leq \prod_{k=2}^{L_n}(r+\mu_k)\|\mathcal{E}_1^{(n)}\|^2 \\ &\leq (r+\mu_1)^{(\sum_{k=1}^{n} L_k)-1}\|\mathcal{E}_1^{(1)}\|^2, \end{aligned} \tag{13}$$

which implies the universal approximation property because $r + \mu_1 < 1$ and $\sum_{k=1}^{n} L_k$ turns to be infinity as n keeps increasing. This completes the proof.

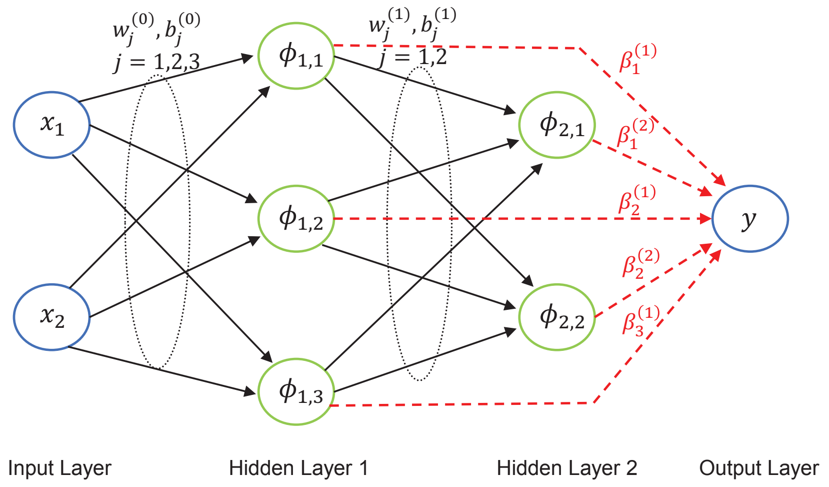


Figure 1: A typical DeepSCN with the architecture: 2-3-2-1, i.e., two input nodes, two hidden layers (three hidden nodes in the first hidden layer, two hidden nods in the second hidden layer), and one output node. Read-out weights (trainable) are shown as red dashed lines, whilst hidden weights and biases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

The idea of learning internal representation can be traced back to 80s [8]. In the past decades, many exciting researches and progresses have been reported in [5], where attempts are made to meet the following three objectives: 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 [24, 25]. Theorem 1 ensures that DeepSCNs fit the essential prerequisite well for learning representation. Due to the nature of DeepSCNs, learning representation becomes quite straightforward but less interpretable. 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, which characterizes the main feature of the target function. Next, the outputs from the first hidden layer, as a new set of inputs, are fed into the next hidden layer to constructively generate a set of random basis functions. 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 $L_2 = 2$, $d = 2$, $m = 1$.

It should be highlighted that DeepSCNs advocate fully connections between each hidden layer and the output layer, which supports a multi-resolution of learning representation rather than using only the compositional expression from the last hidden layer as done in the standard multilayer perceptrons. 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. After generating the random basis functions, we can employ some regularization models such as Lasso [26] to refine the learning representation, which can be done in the layer-wise manner or at one pass. 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.

2.2 Algorithm Description

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

$$h^{(n)}_{L_n} := h^{(n)}_{L_n}(X) = [\phi_{n,L_n}(x^{(n-1)}_1), \dots, \phi_{n,L_n}(x^{(n-1)}_N)]^{\mathrm{T}}, \tag{14}$$

where $\phi_{n,L_n}(x^{(n-1)}_i)$ is used to simplify $\phi_{n,L_n}(x^{(n-1)}_i; w^{(n-1)}_j, b^{(n-1)}_j)$, and $x^{(0)}_i = x_i = [x_{i,1}, \dots, x_{i,d}]^{\mathrm{T}}$, $x^{(n-1)}_i = \Phi(x^{(n-2)}; W^{(n-1)}, B^{(n-1)})$ for $n \geq 2$.

Let $H^{(n)}_{L_n} = [h^{(n)}_1, h^{(n)}_2, \dots, h^{(n)}_{L_n}]$ represent the hidden layer output matrix and denote a temporary variable $\theta^{(n)}_{L_n,q}$ $(q = 1, 2, \dots, m)$ to make it convenient in the followed algorithm description (pseudo-codes).

$$\theta^{(n)}_{L_n,q} = \frac{\langle \mathcal{E}^{(n)}_{L_n-1,q}, h^{(n)}_{L_n} \rangle^2_{l_2}}{\langle h^{(n)}_{L_n}, h^{(n)}_{L_n} \rangle_{l_2}} - (1-r)\langle \mathcal{E}^{(n)}_{L_n-1,q} \mathcal{E}^{(n)}_{L_n-1,q} \rangle_{l_2}, \tag{15}$$

where $\langle \cdot, \cdot \rangle_{l_2}$ denotes the dot product and we omit the argument X in $\mathcal{E}^{(n)}_{L_n-1,q}$ and $h^{(n)}_{L_n}$.

Whilst building the DeepSCN model, we need to take the over-fitting into account. Thus, in practice, a validation dataset is used to monitor the learning process and prevent the occurrence of the over-fitting. In this way, an appropriate architecture of the DeepSCN model can be automatically determined. Alternatively, end-users can specify the number of nodes for each layer and also the number of the layers in advance. However, this pre-defined configuration method has no guarantee to perform a good learning representation.

To speed up the implementation of DeepSCNs, the generation of the random basis functions at each hidden layer can be done in batch mode. That is, we can select a set of generative nodes that meet the constraint (6). Also, for data modelling tasks with high-dimensional outputs, the condition (6) should be replaced by satisfying $\theta^{(n)}_{L_n} = \sum_{q=1}^{m} \theta^{(n)}_{L_n,q} \geq 0$. This relaxed constraint on the random assignment of the weights and biases still ensures the universal approximation property.

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 the correlation between these two performance. 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 favourable consistency performance. 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.

3 Empirical Demonstration

This section reports some simulation results to illustrate advantages of DeepSCNs compared to SCNs in terms of the approximation accuracy and algebraic property (rank deficiency) of the joint hidden

Algorithm 1: Deep Stochastic Configuration Networks

```
Input  : Training inputs $X = \{x_1, x_2, \ldots, x_N\}$, $x_i \in R^d$, outputs $T = \{t_1, t_2, \ldots, t_N\}$, $t_i \in R^m$; The
         maximum number of hidden layers, $M$; The maximum number of hidden nodes within the $n$-th hidden
         layer, $L_{max}^{(n)}$, $1 \le n \le M$; The expected error tolerance $\epsilon$; The maximum times of random
         configuration $T_{max}$; Two sets of scalars $\Upsilon = \{\lambda_1, \ldots, \lambda_{end}\}$, $\mathcal{R} = \{r_1, \ldots, r_{end}\}$
Output : A DeepSCN Model
1  Initialization: $\mathcal{E}_0^{(1)} := [t_1, t_2, \ldots, t_N]^{\mathrm{T}}$, $\mathcal{H}, \Omega, W := [\ ]$;
2  while $n \le M$ and $\|\mathcal{E}_0^{(1)}\|_F > \epsilon$ do
3      while $L_n \le L_{max}^{(n)}$ and $\|\mathcal{E}_0^{(1)}\|_F > \epsilon$ do
4          for $\lambda \in \Upsilon$ do
5              for $r \in \mathcal{R}$ do
6                  for $k = 1, 2 \ldots, T_{max}$ do
7                      Randomly assign $\omega_{L_n}^{(n-1)}$ and $b_{L_n}^{(n-1)}$ from $[-\lambda, \lambda]^d$ and $[-\lambda, \lambda]$;
8                      Calculate $h_{L_n}^{(n)}$, $\theta_{L_n,q}^{(n)}$ by (14), (15);
9                      if $min\{\theta_{L_n,1}^{(n)}, \ldots, \theta_{L_n,m}^{(n)}\} \ge 0$ then
10                         Save $w_{L_n}^{(n-1)}$ and $b_{L_n}^{(n-1)}$ in $W$, $\theta_{L_n}^{(n)} = \sum_{q=1}^{m} \theta_{L_n,q}^{(n)}$ in $\Omega$;
11                     else
12                         go back to Step 5;
13                     end
14                 end
15                 if $W$ is not empty then
16                     Find $w_{L_n}^{(n-1)*}$, $b_{L_n}^{(n-1)*}$ maximizing $\theta_{L_n}^{(n)}$, set $H_{L_n}^{(n)} = [h_1^{(n)*}, \ldots, h_{L_n}^{(n)*}]$;
17                     Break (go to Step 22);
18                 else
19                     Continue: go to Step 5;
20                 end
21             end
22         end
23         Set $\mathcal{H} := [\mathcal{H}, H_{L_n}^{(n)}]$, $\beta^* = \mathcal{H}^{\dagger} T$, $\mathcal{E}_{L_n}^{(n)} = \mathcal{H}\beta^* - T$, $\mathcal{E}^{(n)} = \mathcal{E}_{L_n}^{(n)}$;
24         Renew $\mathcal{E}_0^{(1)} := \mathcal{E}_{L_n}^{(n)}$, $L_n := L_n + 1$;
25     end
26     Set $\mathcal{E}_0^{(n+1)} = \mathcal{E}^{(n)}$, $\Omega, W = [\ ]$;
27 end
28 Return: $\beta^*$,$\omega^*$,$b^*$
```

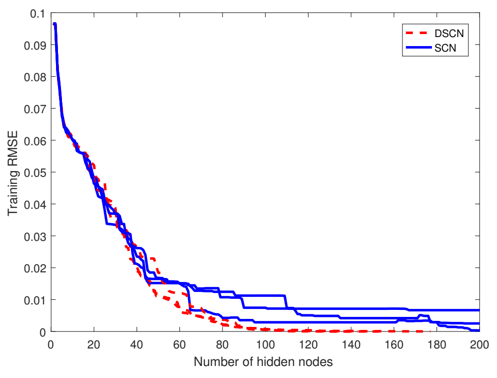
output matrix. Here, we also provide a robustness analysis on DSCN algorithm with respect to the learning parameter setting. The following real-valued function defined over [0,1] is employed in the simulation study:

$$f(x) = 0.2e^{-(10x-4)^2} + 0.5e^{-(80x-40)^2} + 0.3e^{-(80x-20)^2}.$$

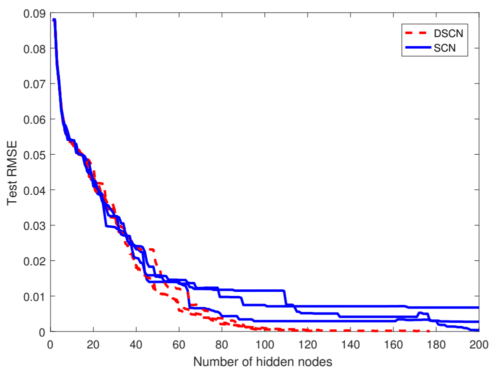
In this study, we take 1000 points (randomly generated) as the training data and another 1000 points (randomly generated) as the test data in our experiments. The sigmoidal activation function $g(x) = 1/(1 + \exp(-x))$ is used for all hidden nodes.

3.1 Approximation Capability

Figure 2 depicts the training and test performance, respectively, where $L = 200$ for SCN, $M = 4$ and $L_{max}^{(n)} = 50$ 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 there is no theoretical justification available so far. Figure 2 shows three independent trials, which indicate that DeepSCN outperforms SCN in terms of effectiveness. In this test, we set $r_k = 1 - 10^{-k}, k = 1, 2, ..., 7$ 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 4.



(a) Training



(b) Test

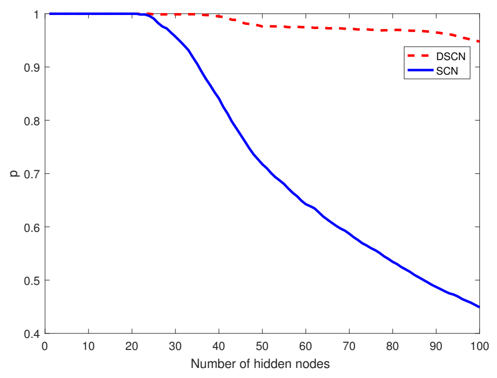
Figure 2: Effectiveness and consistency

3.2 Algebraic Property of the Hidden Output Matrix

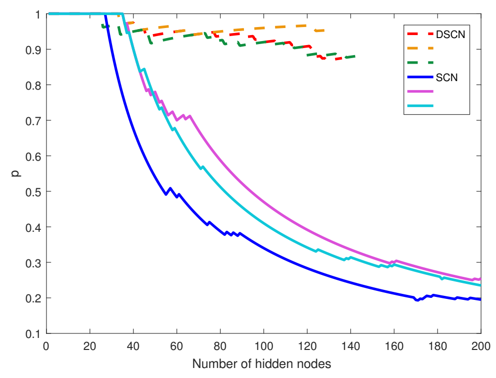
In a common sense, a learner model's capacity can be measured by using a trade-off metric over the learning and generalization performance. In addition, we may evaluate the capacity of a learner model by looking at the interpretability. In fact, it is always desirable to have a learner model with balanced 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. Technically, we found that the degree of the rank-deficiency of the joint hidden output matrix is proportional to the level of redundancy. To do so, the degree of rank-deficiency is computed by $p = \text{rank}(\mathcal{H})/\sum_{n=1}^{j} L_n$, 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 3(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 degree of the rank-deficiency for the SCN becomes much more higher as L becomes larger. Note that the results from DeepSCN reported in Figure 3(a) were obtained with a fixed architecture, i.e., 4 hidden layers with 25 nodes for each. Figure 3(b) depicts six curves of the rank-deficiency for both the SCNs and DeepSCNs, where $M = 4$ and $L_{max}^{(n)} = 50$ for each hidden layer are used in the DSCN algorithm. From these observations, we can infer that the DeepSCN models have much less redundant basis functions compared against the SCN models.

3.3 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 $\Upsilon = \{\lambda_{min} : \Delta\lambda : \lambda_{max}\}$ is not reported here. Figure 4 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 with 4 hidden layers and 25 nodes for each. A remarkable drop-down can be observed after completing the construction of the first layer, which clearly exhibits the advantage of DeepSCN over SCN.

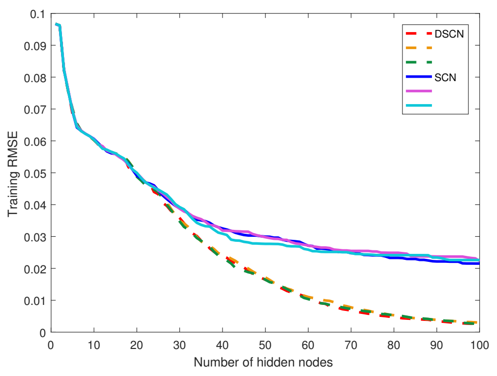


(a) Rank-deficiency property

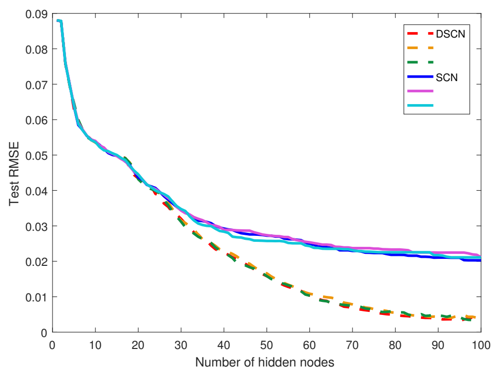


(b) Rank-deficiency property

Figure 3: (a) SCN with $L = 1:100$ and DeepSCN with $M = 4, 1 \leq L_j \leq 25$, $j = 1,2,3,4$; (b) SCN with $L = 1:200$ and DeepSCN with $M = 4, 1 \leq L_j \leq 50$, $j = 1,2,3,4$



(a) Training



(b) Test

Figure 4: Robustness analysis on the learning parameter setting

4 A Case Study: Rotation Angles Predication for Handwritten Digits

This section further investigates the merits of the proposed DeepSCN by predicting the angles of rotation of handwritten digits. We have a collection of synthetic handwritten digits containing 5000 training and 5000 test images of digits with corresponding angles of rotation. Each image represents a rotated digit in grayscale and of normalized size (28×28). Figure 5 displays 16 random sample training digits, retrieved from Neural Network Toolbox in MATLAB R2017b.

To demonstrate the advantages of DeepSCN over SCN framework, we employ two kinds of evaluation metric in the performance comparison: (i) The Percentage of Predictions within an Acceptable error margin (PPA); and (ii) The root-mean-square error (RMSE) of the predicted and actual angles of rotation. In particular for measuring PPA, we need to calculate the prediction error between the predicted and actual angles of rotation, followed by counting the number of predictions within an acceptable error margin from the true angles. In our experiments, we set the threshold to be 10 degrees. Therefore, the PPA value within this threshold can be obtained by

$$PPA = \frac{\#\{|Prediction\ \ Error| < 10\}}{Number\ \ of\ \ Sample\ \ Images}.$$

It is logical to think that higher PPA and/or lower RMSE values can reflect a better capability on this regression task. In Table I, we summarize the training and test results for both DSCN and SCN to illustrate the superiority of our developed DeepSCN framework. Specifically, two options for the architecture setting are examined, that is, $L = 1000$ for SCN, $L_1 = L_2 = L_3 = L_4 = 250$ for DSCN, and $L = 2000$ for SCN, $L_1 = L_2 = L_3 = L_4 = 500$ for DSCN, respectively. As can be

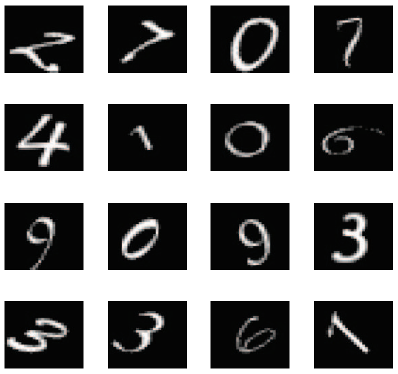
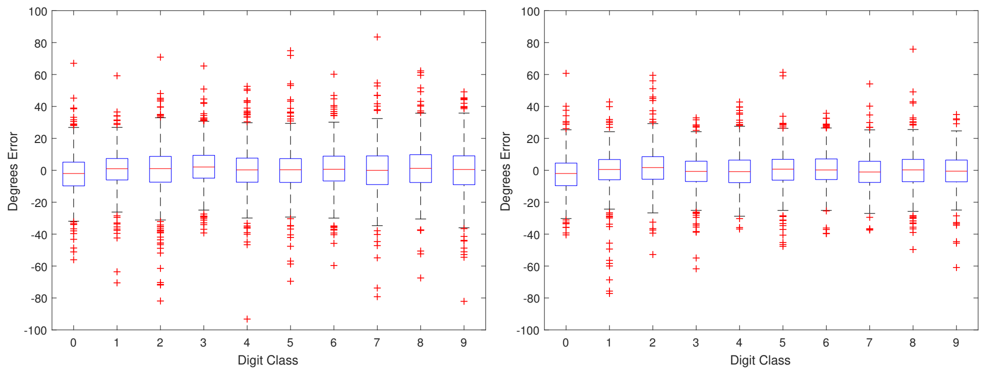


Figure 5: Samples of Handwritten Digits with Angles of Rotation

Table 1: Performance Comparison

Algorithms	PPA (%)		RMSE	
	Training	Test	Training	Test
DSCN[1]	**82.21**	**65.21**	**7.0265**	**14.8231**
SCN[1]	78.14	60.35	7.5623	15.9176
DSCN[2]	**84.73**	**66.36**	**5.2324**	**12.0971**
SCN[2]	80.12	60.00	5.6797	14.7775
$L = 1000$ for SCN[1], $L_1 = L_2 = L_3 = L_4 = 250$ for DSCN[1]				
$L = 2000$ for SCN[2], $L_1 = L_2 = L_3 = L_4 = 500$ for DSCN[2]				

seen in Table I, DSCN outperforms SCN in these two cases with higher PPA and lower RMSE values. It should be noted that all the results shown here are the averaged values based on five independent trials. Based on our experience, their standard deviations have no significant difference, so we omit this information here without any confusion.



(a) SCN: $L = 2000$ (b) DSCN: $L_1 = L_2 = L_3 = L_4 = 500$

Figure 6: Display Box Plot of Residuals for Each Digit Class

In Figure 6, we use the box-and-whisker diagram to show the difference between DSCN and SCN on residual distribution for test images (for each digit class). Intuitively and obviously, DSCN works much more favorably than SCN as there are less records with abnormal degree error, as marked in red plus symbol. In particular, for the digit '1', '2', '5', '7', '8', DSCN shows more robustness in predicting the associated angles of rotation. The digit classes with highest accuracy have a mean close to zero and little variance, such as '3' and '6'. The learner model produced by SCN is more prone to result in larger predicted angles, i.e, a plenty of examples with degree error outside the interval

[-60,60]. With consideration of length limitation, we only provide the results of the architecture setting with $L = 2000$ for SCN, $L_1 = L_2 = L_3 = L_4 = 500$ for DSCN. Based on our simulations, similar findings can be found in the case that $L = 1000$ for SCN, $L_1 = L_2 = L_3 = L_4 = 250$ for DSCN.

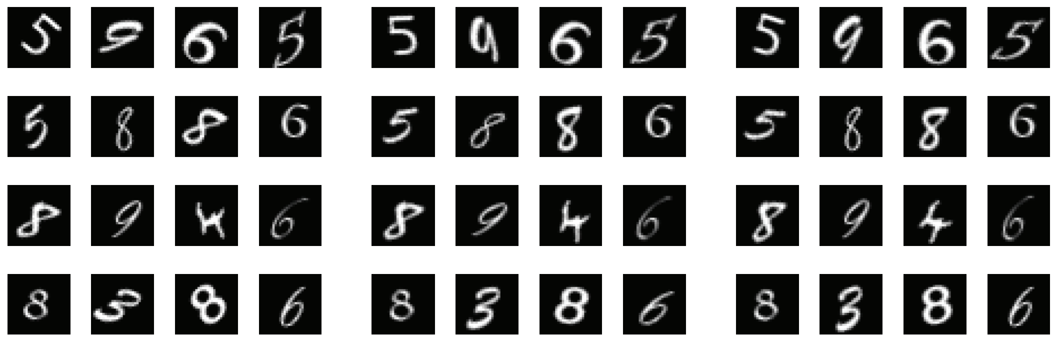

(a) Original (b) Corrected by SCN (c) Corrected by DSCN

Figure 7: Demonstration of Original Digits and Their Corrected Rotations: (b) $L = 2000$ for SCN; (c) $L_1 = L_2 = L_3 = L_4 = 500$ for DSCN.

With the predicted results, we can rotate the sample digits according to their predicted angles of rotation, aiming to correct the digits into a regular position. In Figure 7, we randomly selected 16 test samples and display their original images, the corrected digits by SCN, and the corrected ones by DSCN, respectively. It is obvious that DSCN can contribute to a better rotation correction than SCN, showing that the proposed DeepSCN framework have more advantages over our previously developed SCNs [22]. So far, we have extensively compared DSCN with SCN on this specified data modelling task. As for the paper length limitation, we have no extra space to add more robustness analysis on these two algorithms. Additionally, more in-depth theoretical analysis (in terms of our innovative SCN methodology) why deep is better more shallow are desirable in our future work.

5 Concluding Remarks

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 forms. Except for concerns on the architecture determination, one cares much about learning performance in terms of effectiveness and efficiency. In this paper, our proposed DeepSCNs, as a class of randomized deep neural networks, offer a fast and feasible pathway for problem solving. Our technical contributions to the working field can be summarized as follows:

- The proposed supervisory mechanism (6) is the key for developing DeepSCN framework, which could be used as a unique feature to distinguish our work from other randomized learner models;
- The scopes of random parameters at each layer of DeepSCNs could be updated adaptively, which make the set of basis functions rich and functional;
- The construction of DeepSCNs is data dependant, and the proposed algorithm has much less human intervention provided that a set of appropriate learning parameters is given;
- The removal of the redundant nodes becomes possible with some post-processing steps, which implies that we can make a better learning representation with sparsity.

There exist a lot of opportunities to further develop the proposed framework in either theoretical aspects or real-world applications. For instance, one can explore some algebraic properties of the DeepSCN to evaluate the quality of the resulting model, seek for different forms of supervisory mechanisms, develop some advanced learning schemes for random representation, study convolutional or recurrent versions of the proposed DeepSCN. For applications, the DeepSCN model has great potential for extracting random features with lower computational cost and fast implementation, which is very important for big data classification with hierarchical or cascade structure.

Before ending up this paper, we would like to share our philosophy behind developing randomized methods for DNNs. The territory of the weights and biases of the randomized learner models must be data-dependant and it can be characterised by certain constraints in relation to the model's capacity. From our understandings on the randomized learner's quality, it is highly desired to make the joint hidden output matrix full-rank. Unfortunately, such an essential algebraic condition may not be achieved by simply assigning the random hidden parameters from a fixed scope such as [-1,1] or a pre-defined probability distribution. Therefore, the way to generate the random parameters at the hidden layers with guarantee for both universal approximation property and sound generalization capability should be done.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

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

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

[5] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

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

[7] J. Alvarez and M. Salzmann Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2262–2270, 2016.

[8] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning internal representations by backpropagating errors, Nature, vol. 323, pp. 533–536, 1986.

[9] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[10] 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.

[11] R. Giryes, G. Sapiro, and A. Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, vol. 64, no. 13, pp. 3444–3457, 2016.

[12] A. R. Barron, Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.

[13] T. Y. Kwok and D. Y. Yeung, Objective functions for training new hidden units in constructive neural networks. *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1131–1148, 1997.

[14] S. Gallant and D. Smith, "Random cells: an idea whose time has come and gone... and come again," In *Proceedings of of IEEE International Conference on Neural Networks*, San Diego, CA, USA, 1987, pp. 671–678.

[15] D. S. Broomhead and D. Lowe, Multi-variable functional interpolation and adaptive networks. *Complex Systems*, vol. 2, pp. 321–355, 1988.

[16] Y. H. Pao and Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities. *Computer*, vol. 25, no. 5, pp. 76–79, 1992.

[17] R. S. Sutton and S. D. Whitehead, Online learning with random representations. In *Proceedings of the 10th International Conference on Machine Learning*, Amherst, MA, USA, 1993, pp. 314–321.

[18] B. Igelnik and Y. H. Pao, Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 1320–1329, 1995.

[19] A. N. Gorban, I. Y. Tyukin, D. V. Prokhorov, and K. I. Sofeikov, Approximation with random bases: Pro et contra. *Information Sciences*, vol. 364, pp. 129–145, 2016.

[20] M. Li and D. Wang, Insights into randomized algorithms for neural networks: Practical issues and common pitfalls. *Information Sciences*, vol. 382–383, pp. 170–178, 2016.

[21] S. Scardapane and D. Wang, Randomness in neural networks: An overview. *WIREs Data Mining and Knowledge Discovery*, vol. 7, no. 2, e1200. doi: 10.1002/widm.1200, 2017.

[22] D. Wang and M. Li, Stochastic configuration networks: Fundamentals and algorithms. *IEEE Transactions on Cybernetics*, vol. 47, no. 10, pp. 3466–3477, 2017.

[23] P. Lancaster and M. Tismenetsky, *The Theory of Matrices: With Applications*. Elsevier, 1985.

[24] G. Cybenko, Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[25] K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators. *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[26] R. Tibshirani, Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288 , 1996.