Stochastic Deep Learning

Computational Intelligence SS2020
M.Eng Information Technology
Yash Vyas 1266490
vyas@stud.fra-uas.de
Guide: Prof. Dr. Pech

***Abstract*— In the following paper SCN and DeepSCN are explored in brief. Different algorithms introduced on the topic are given. Universal approximation property with its mathematical perspective with proposed theorem and its proof which are explained in brief. Typical architecture of DeepSCN with learning representation are given. Previous works and their result are given in short. MatLab code snapshots are also provided.**

Keywords—SCN, DeepSCN, Learning representation, ensemble, robust , stochastic , randomness, multi-layer perceptron, nodes , Neural Network , Artificial Intelligence, Deep learning

I. INTRODUCTION

Artificial Intelligence research aims at achieving efficiency and robustness by which the human brain represents information. The human brain receives the multitudinous amount of sensory data every second of the day and is still able to prehend critical aspects of this data in a manner that it can be used in the future in brief [1].

In [2] Richard Bellman established dynamic programming theory and pioneered the field of optimal control, asserted that high dimensionality of data is a fundamental hurdle in many sciences and engineering applications. The main problem in the field of pattern classification application is that learning complexity grows exponentially with a linear increase in the dimensionality of data. He coined this phenomenon as the "curse of dimensionality" [2]. The conventional approach of handling the dimensionality problem has been to pre-process the data in such a way that would reduce its dimensionality to that extent that it can be processed effectively. One of the example is classification engine. This phenomenon of dimensionality reduction also known as feature extraction. Because the following administrative unit of many pattern recognition systems shifted to the human-engineered feature extraction process which was many times challenging and highly application dependent [3]. Additionally, if incomplete or wrong features are extracted, the classification process will be limited in performance. Recent advances in the field of neuroscience have provided us with the information representation principles in animal brain One of the key findings has been that the neocortex, which is responsible for many cognitive abilities, does not externally pre-process sensory signal but allows them to pass through a complex hierarchy of components that learn to transpose observation based on the pattern they show [4] [5].

The following advances were responsible for the emergence of deep machine learning, which keeps computational models in the focal point for information representation that shows similar properties to that of neo-cortex. Additionally, with the spatial dimensionality of real-life data, the temporal component plays an important role. The sequence of patterns often conveys some meaning to the observer, on the other hand, independent fragments of this sequence would be difficult in isolation to decipher. Information is often inferred from events or observations that are received as a sequence [6] [7]. Modelling the semantic component of the data plays a critical role in effective information representation. Acquiring the comprehensive dependencies based on patterns in the observations is therefore viewed as a fundamental goal for deep learning [1].

Recent studies in the field of machine learning have proved that neural networks can be successfully used for data representation because of its universal power for nonlinear maps and learning aptitude from a grouping of training sampled [9] [11].During implementation, the following is quite challenging to decide a feasible architecture i.e. number of hidden nodes of the neural network such that above par performance can be achieved in learning and generalization. For solving the stated problem formative approach for the construction of neural networks was used which goes by creating a small-sized network, followed by incrementally producing hidden nodes and output weights until a pre-defined termination condition meeting. Being a harmonic of multiple applications, it is important that the constructive neural networks share the universal approximation property [8].

Stochastic approaches for large scale computing are highly in demand due to their potency and skillfulness [14]. In the field of deep learning for data modeling, stochastic algorithms have shown incontestable potential in creating fast learner models and learning algorithms with much less computational cost [13]

[17]. The basic notion behind these stochastic learning algorithms is the two-step training paradigm that is, randomly specifying the input weights and biases of neural networks and assessing the output weights by solving a linear equation system using the famous least square method and its regularized version [8].
Stochastic neural network has the basic way of assigning the random parameters with an inequality constraint and adaptively selecting the scope of the random parameters, ensuring the universal approximation property of the built stochastic learner model [8].

In the last decade, the deep neural network has been given conventional attention because of its outstanding prospective for dealing with computer vision, pattern recognition, NLP (Natural Language Processing) [20] [21]. The attainment of deep learning is because of the representation of visual data [23] [24]. By existential evidence of DNNs showing more efficiency and effectiveness in the data representation aspect rather than a shallow model, the DNN model is gaining popularity [19].DNN model has a greater ability to learn higher-level abstraction in comparison to the shallow models.
Formal analysis of the representation power and learning complexity of DNN can be referred from [25]. DNNs accretive popularity is due to its learning representation power for complex data, but there are some problems with its design and implementation perspective. For example, how to make an autonomous decision on the architectural design of DNN which improves the learning speed and reduces the computational burden [19].In the previous studies, the configuration of DNN is done manually by trial and error of performance data without technical attributes which may be time-consuming and potentially impact the effectiveness of the resulting model. In reality, DNNs with average depth but less number of hidden nodes at each layer may not be able to show sufficient learning ability, on the other hand, a large-sized architecture may cause over-fitting that degrades the generalization capability [19]. Alvarez and Salzmann instigated a way to spontaneously decide the number of nodes at each layer of DNN using group sparsity regularizer. Their method can produce a tight width setting for each layer with the network's depth determined in advance [26]. It is observed that the training multilayer model with the help of the back-propagation algorithm suffers from several issues such as the weight initialization, local minima, and sensitivity of the learning performance with respect to the learning rate setting. From experimental observation, it is understood that this gradient-based learning method is not able to produce meaningful or interpretable internal representations from each hidden inputs [19][27].Recently learning capability of Autoencoder are observed assigning stochastic weights in the range of [-1, 1], which in turn are used for building DNNs with much less computational complexity [28].

II. FUNDAMENTALS OF DEEPSCN

1) Universal Approximation Property [19]

Let $\mathcal{L}_2(D) =$ *space of all Lebesguemeasureable vector valued* Function $\mathcal{F}$

$$\mathcal{F} = [f_1, .., f_m] : R^d \to R^m \; on \; Compact \; set \; D \subset R^d$$

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

$$Inner \; product \; defined \; as \; \langle \mathcal{F}, G \rangle :$$
$$= \sum_{q=1}^{m} \langle f_q, g_q \rangle$$
$$= \sum_{q=1}^{m} \int f_q(x) g_q(x) dx \qquad (2)$$

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 has been constructed , that is ,

$$\mathcal{F}_{S_n^{(n)}}(x) = \sum_{k=1}^{n} \sum_{j=1}^{L_k} \beta_j^k \Phi_{kj}\left(x^{k-1}; w_j^{k-1}, b_j^{k-1}\right) \qquad (3)$$

$$S_n = \{L_1, \dots, L_n\} = set \; of \; hidden \; node \; numbers$$

$$w_j^{k-1}, b_j^{k-1} \; stand \; for \; hidden \; parameter \; within \; k^{th} hidden \; layer$$

$$\Phi_{kj} = activation \; function \; used \; in \; k^{th} layer$$

$$x^0 = x \, , x^k = \Phi(x^{k-1}; W^{k-1}, B^{k-1}) = \left[\Phi_{\mathrm{k},1}, \dots, \Phi_{k,L_k}\right]$$

$$Residual \; Error \; function \; \varepsilon_{Sn}^n = \mathcal{F} - \mathcal{F}_{Sn}^n = \left[\varepsilon_{Sn,1}^n, \dots, \varepsilon_{Sn,m}^n\right]$$

$$\mathcal{F}_P^n(x) = \sum_{k=1}^{n} \sum_{j=1}^{L_k} \beta_j^k \Phi_{kj}\left(x^{k-1}; w_j^{k-1}, b_j^{k-1}\right) =$$
$$DeepSCN \; model \; with \; p \; nodes \; at \; n^{th} layer \qquad (4)$$
$$where \; p = 1, \dots, L_n \, ,$$

$$Specifically \; \varepsilon_0^1 = \mathcal{F} \; and \; \varepsilon_0^{n+1} = \varepsilon_{L_n}^n, n = 1,2, \dots$$

2) Theorem and Proof [19]

a) Theorem

Suppose that span Γ is dense in $\mathcal{L}_2$space and $\forall \, \emptyset \in \Gamma$,

$0 < ||\emptyset|| < c$ for some c $\in R^+$.

Given 0 < r < 1 and a non negative decreasing sequence $\{\mu_i\}$ with $\lim_{i \to +\infty} \mu_i = 0$

And $\mu_i < (1-r)$ *For* $n = 1,2, \dots$ *and* $j = 1, \dots, L_n$ *denoted by*

$$\delta_{j,q}^n = \left(1 - r - \mu_i\right) \| \ \varepsilon_{j-1,q}^n \|^2, q = 1, \dots, m \quad (5)$$

Stochastically configuring the j-th hidden node to statisfy the following inequalities:

$$\langle \varepsilon_{j,q}^n, \phi_{n,j} \rangle^2 \ \geq c^2 \ \delta_{j,q}^n \ q = 1,2, \dots, m \quad (6)$$

Fixing random basis function:

$$\langle \varepsilon_{j,q}^n, \phi_{n+1,1} \rangle^2 \ \geq c^2 \ \delta_{j,q}^n \ q = 1,2, \dots, m \quad (7)$$

Keep adding new hidden nodes based on (6) followed by generating the first hidden node in a new layer via (7)
After adding one hidden node, the read out wright are evaluated by the least squares method that is ,

$$\beta^* = \arg\min \| \ \mathcal{F} - \sum_{k-1}^{n} \sum_{j-1}^{L_K} \beta_j^k \phi_{k,j} \ \|^2 \quad (8)$$

Then we have $\lim_{n \to +\infty} \| \ \mathcal{F} - \ \mathcal{F}_{Sn}^n \ \| = 0$

b) Proof:

First we obtain:

$$\| \ \varepsilon_{L_n}^n \ \|^2 \leq (r \ + \ \mu_{Ln}) \ \| \ \varepsilon_{L_n-1}^n \ \|^2 \quad (9)$$

Error can be calculated after adding one hidden layer as:

$$\| \ \varepsilon_1^{n+1} \ \|^2 = \| \ \varepsilon_{L_n}^n - \beta_1^{n+1} \phi_{n+1,1} \ \|^2 \quad (10)$$

Eq 10 parameters are obtained using Eq 7 and 8.

To identify relationship between ε_1^{n+1} and $\varepsilon_{L_n}^{n+1}$:
We denote $\beta_1^{n+1} = \left[\beta_{1,1}^{n+1}, \dots, \beta_{1,m}^{n+1}\right]^T$ *where*

$$\beta_{1,q}^{n+1} = \frac{\langle \ \varepsilon_{Ln,q}, \phi_{n+1,1}, \rangle}{\| \ \phi_{n+1,1} \ \|^2}, q = 1, \dots, m \quad (11)$$

Replacing β_1^{n+1} *evaluated by* 11 *we get*

$$\| \ \varepsilon_1^{n+1} \ \|^2 \leq \| \ \varepsilon_{L_n}^{n+1} - \ \beta_1^{n+1} \phi_{n+1,1} \ \|^2$$

$$= \| \ \varepsilon_{L_n}^{n+1} \ \|^2 - \frac{\sum_{q=1}^{m} \langle \ \varepsilon_{L_n,q}^n, \phi_{n+1,1} \rangle^2}{\| \ \phi_{n+1,1} \ \|^2} \leq \| \ \varepsilon_{L_n}^{n+1} \ \|^2 \quad (12)$$

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

$$\| \ \varepsilon_{L_n}^{n+1} \ \|^2 \leq \| \ \varepsilon_{L_n}^{n+1} \ \|_2^2 \leq \| \ \varepsilon_{L_n}^n \ \|^2$$

$$\leq \prod_{k=2}^{L_N} (r + \mu_k) \| \ \varepsilon_1^n \ \|^2$$

$$\leq (r + \mu_1)(\sum_{k=1}^{n} L_K) - 1 \| \ \varepsilon_1^1 \ \|^2 \quad (13)$$

Which implies the universal approximation property because
$r + \mu_1 <$
$1 \ and \ \sum_{k=1}^{n} L_K \ turns \ to \ be \ infinity \ when \ n$ Keeps increasing

3) Learning Representation:

The thought of acquisition of internal representation is shown is [27]. Learner model can materialize in different form but primarily they must allocate universal approximation property so that the fidelity can be attained [29] [30]. Theorem 1 verify that DeepSCNs fulfil the important conditions for learning representation. The universal approximated property assure the ability of DeepSCNs for both data modelling and signal representation because of following property It can be used as predictive models or feature extractors [19].

DeepSCNs learning representation is less interpretable due to its attributes. The resultant of the first hidden layer contains two parts a set of random basis functions. The first part contains original inputs and the Second part contains readout weights between the first hidden layer and the output layer [19]. Seemingly, learning representation can be defined from the first hidden layer, which distinguishes the important feature of the target function. The random basis function set is assembled constructively using outputs from the first hidden layer as a set of input into the next hidden layer. The following procedure can be used to obtain refined learning representation from the next hidden layer until some termination criterion is satisfied [19].

Difference between standard multilayer perceptrons and DeepSCNs is that standard perceptrons use a compositional expression from last hidden layers while DeepSCNs are using full connections between each hidden layer and output layer for the multi-resolution of learning representation. This kind of learning representation provides users with more flexibility. As an example, some nodes at a definite layer can be abandoned according to some predefined criterion which leads to new learning representation with sparsity. Learning representation can be refined in layer-wise mode or passes mode and models like Lasso can also be used [19] [31].

III. ALGORITHMS

1) Stochastic Configuration Network

a) SC-1 *[8]*

Algorithm SC-1

Given Inputs $X = \{x_1, x_2, \dots, x_N\}, x_i \in \mathbb{R}^d$ *and outputs* $T = \{t_1, t_2, \dots, t_N\}, t_i \in \mathbb{R}^m$; Set maximum number of hidden nodes L_max expected error tolerance $\mathcal{E}$, maimum times of random configuration $T_{\max}$; *Choose a set of positive scalars* $\Upsilon = \{\lambda_{min} : \triangle \lambda : \lambda_{max}\}$

1. Initialize $e_0 := \{t_1, t_2 \dots, t_N\}^T, 0 < r < 1$, *two empty sets* Ω *and* W ;
2. **WHILE** $L \leq L_{max}$ *AND* $\| e_0 \|_F > \mathcal{E}$, **DO**
   PHASE 1 : Hidden Parameter Configuration (Step 3- 17)
3. **For** $\lambda \in \Upsilon$, **DO**
4. **For** k = 1,2,3,…,T_{Max}, **Do**
5. Randomly assign ω_L *and* b_L *from* $[-\lambda, \lambda]^d$ *and* $[-\lambda, \lambda]$, *respectively*;
6. Calculate $h_L, \xi_{L,q}$ *and* $\mu_L = \frac{(1-r)}{L+1}$
7. **If** $\min\{\xi_{L,1}, \xi_{L,2}, \dots, \xi_{L,m}\} \geq 0$
8. **Save** ω_L *and* b_L *in* $W, \xi_L = \sum_{q=1}^{m} \xi_{L,q}$ *in* Ω *respectively*;
9. **Else** go back to **Step 4**
10. **End If**
11. **End For (**corresponds to **Step 4)**
12. **If** W *is not empty*
13. Find w_L^*, b_L^* *that maximize* ξ_L *in* Ω, *and set* $H_L = [h_1^*, h_2^*, \dots, h_L^*]$;
14. …….**Break** (go to **step 18**);
15. **Else** randomly take $\tau \in (0, 1-r)$, *renew* $r := r + r$, *return to* ***Step*** **4**;
16. **End If**
17. **End For** (corresponds to **Step 3**)
    Phase 2: Output Weights Determination
18. Calculate $\beta_{L,q} = \frac{e_{L-1,q}^T . h_L^*}{h_L^{*T} . h_L^*}$, q = 1,2,…,m;
19. …… $\beta_L = [\beta_{L,1} \dots \beta_{L,m}]^T$;
20. …… $e_L = e_{L-1} - \beta_L h_L^*$;
21. Renew $e_0 := e_L; L := L + 1$;
22. **End While**
23. **Return** $\beta_1, \beta_2, \dots, \beta_L, \omega^* = [\omega_1^*, \dots, \omega_L^*]$ *and* $b^* = [b_1^*, \dots, b_L^*]$

b) SC-2 *[8]*

Algorithm SC-2

Given the same items in **Algorithm SC-1.** Set window size K < Lmax

1. Initialize $e_0 := \{t_1, t_2 \dots, t_N\}^T, 0 < r < 1$, *two empty sets* Ω *and* W ;
2. **WHILE** $L \leq L_{max}$ *AND* $\| e_0 \|_F > \mathcal{E}$, **DO**
3. Proceed **Phase 1 of Algorithm SC-1;**
4. Obtain $H_L = \{h_1^*, h_2^*, \dots, h_L^*\}$;
5. **If** $L \leq K$
6. Calculate $\beta^* = [\beta_1^*, \beta_2^*, \dots, \beta_L^*]^T = H_L T$
7. **Else**
8. Set $H_K = H_L(:, 1: L-K), H_K = H_L(:, L-K+1: L)$;
9. Retrieve $\beta^{previous} = [\beta_1^*, \beta_2^*, \dots, \beta_{L-K}^*]^T$
10. Calculate $\beta^{window} = H_k(T - H_K \beta^{previous})$;
11. Let $\beta^* = [\beta_{1,}^* \beta_{2,}^* \dots, \beta_{L,}^*] := \begin{bmatrix} \beta_{,}^{previous} \\ \beta_{,}^{window} \end{bmatrix}$
12. **End If**
13. Calculate $e_L = e_{L-1} - \beta_L^* h_L^*$;
14. Renew $e_0 := e_L : L := L + 1$;
15. **END While**;
16. **Return** $\beta_1, \beta_2, \dots, \beta_L, \omega^* = [\omega_1^*, \dots, \omega_L^*]$ *and* $b^* = [b_1^*, \dots, b_L^*]$

c) SC-3 *[8]*

Algorithm SC-3

Given the same items in **Algorithm SC-1**

1. Initialize $e_0 := \{t_1, t_2 \dots, t_N\}^T, 0 < r < 1$, *two empty sets* Ω *and* W ;
2. **WHILE** $L \leq L_{max}$ *AND* $\| e_0 \|_F > \mathcal{E}$, **DO**
3. Proceed **Phase 1 of Algorithm SC-1;**
4. Obtain $H_L = \{h_1^*, h_2^*, \dots, h_L^*\}$;
5. Calculate $\beta^* = [\beta_{1,}^* \beta_{2,}^* \dots, \beta_{L,}^*] := H_L^t T$;
6. Calculate $e_L = e_{L-1} - \beta_L^* h_L^*$;
7. Renew $e_0 := e_L : L := L + 1$;
8. **END While**;
9. **Return** $\beta_1, \beta_2, \dots, \beta_L, \omega^* = [\omega_1^*, \dots, \omega_L^*]$ *and* $b^* = [b_1^*, \dots, b_L^*]$

2) Deep Stochastic Configuration Networks *[19]*

$X = \{x_1, \dots, x_N\}, x_i = \{x_{i,1}, \dots, x_{i,d}\} \in R^d$ $X =$ *Input from Training Dataset*

Output $T = \{t_1, \dots, t_N\}, t_i = [t_{i,1}, \dots, t_{i,m}]^T \in R^m, i = 1, \dots, N$

Residual Error Vector Before the L_n*th hidden node of* n^{th} *hidden layer is added*

$$\varepsilon_{L_n-1}^n = \varepsilon_{L_n-1}^n(X) = [\varepsilon_{L_n-1,1}^n(X), \dots, \varepsilon_{L_n-1,m}^n(X)]^T$$

After adding hidden node in hidden layer, we get

$$h_{Ln}^n := h_{Ln}^n(X) = [\phi_{n,L_n}(x_1^{n-1}), \dots, \phi_{n,L_n}(x_N^{n-1})]^T \quad (14)$$

$\phi_{n,L_n}(x_i^{n-1})$ *is used to simplify* $\phi_{n,L_n}(x_1^{n-1}; w_j^{n-1}; b_{n-j}^{n-1})$

$x_i^0 = x_i = [x_{i,1}, \dots, x_{i,d}]^T, x_i^{n-1} = \phi(x^{n-2}; W^{n-1}, B^{n-1})$ *for* $n \geq 2$

$H_{Ln}^{n} = [h_1^n, \dots, h_{Ln}^n]$ *represent hidden layer output matrix*

$\theta_{Ln,q}$ $(q = 1,2,..,m)$ *denotes a temporary variable*

$$\theta_{L_n,q} = \frac{\langle \varepsilon_{Ln-1,q}^{n}, h_{Ln}^{n} \rangle^2 l_2}{\langle h_{Ln}^{n} h_{Ln}^{n} \rangle l_2} - (1-r)\langle \varepsilon_{Ln-1,q}^{n} \varepsilon_{Ln-1,q}^{n} \rangle t_2 \quad (15)$$

Where

$\langle .,. \rangle l_2$ *denotes dot product and omit the argument X in* $\in_{L_n-1,q}$ *and* h_{Ln}^{n}

Algorithm : Deep Stochastic Configuration Networks

Given Inputs $X = \{x_1, x_2, \dots, x_N\}, x_i \in \mathbb{R}^d$ *and outputs* $T = \{t_1, t_2, \dots, t_N\}, t_i \in \mathbb{R}^m$; The maximum number of hidden layers M The maximum number of hidden nodes within the nth hidden layer L_{max}^{n} $1 \le n \le M$. The expected error tolerance Ɛ, maimum times of random configuration $T_{\max}$; *Two set of positive scalars* $\Upsilon = \{\lambda_1, \dots, \lambda_{end}\}, R = \{r_1, \dots, r_{end}\}$

Output : DeepSCN Model

1. Initialize $\mathcal{E}_0^1 \coloneqq \{t_1, t_2, \dots, t_N\}^T, H, \Omega, W \coloneqq [\ \ ]$;
2. **While** $n \le M$ *AND* $\| \mathcal{E}_0^1 \|_F > \mathcal{E}$, **Do**
3. **While** $L_n \le L_{max}^{n}$ *and* $\| \mathcal{E}_0^1 \|_F > \mathcal{E}$, **Do**
4. **For** $\lambda \in \Upsilon$, **Do**
5. **For** $r \in$ R **Do**
6. **For** $\boldsymbol{k} = 1,2, \dots, T_{max}$ ***do***
7. Randomly assign ω_{Ln}^{n-1} *and* b_{Ln}^{n-1} *from* $\{-\lambda, \lambda\}^d$ *and* $[-\lambda, \lambda]$
8. Calculate $h_{Ln}^{n}, \Theta_{Ln}^{n}$ *from*
9. **If** $\min\{\Theta_{L1}^{n}, \dots, \Theta_{Lm}^{n}\} \ge 0$ **then**
10. ….**Save** ω_{Ln}^{n-1} *and* b_{Ln}^{n-1} *in* W , $\Theta_{Ln}^{n} = \sum_{q=1}^{m} \Theta_{Lq}^{n}$ *in* Ω
11. **Else** go back to **Step 5**
12. ….**End**
13. **End**
14. **If** W is not empty **then**
15. Find ω_{Ln}^{n-1} *and* b_{Ln}^{n-1} *maximizing* Θ_{Ln}^{n} , *set* $H_{Ln}^{n} = [h_1^n, \dots, h_{Ln}^n]$;
16. **Break** *go to* ***Step*** **22**;
17. **Else Continue to Step 5;**
18. **End**;
19. **End**
20. **End**;
21. $H_{:} \coloneqq \{H, H_{Ln}^{n}\}; \beta^* = H^t T, \epsilon_{Ln}^{n} = H\beta^* - T, \epsilon^n = \epsilon_{Ln}^{n}$
22. Renew $\varepsilon_0^1 \coloneqq \varepsilon_{Ln}^{n}, L_n \coloneqq L_n + 1$
23. **End**
24. Set $\varepsilon_0^{n+1} = \varepsilon^n$, $\Omega, W = [\ ]$;
25. **End**
26. **Return** B^*, ω, b^*

IV. Architecture and Advantages

1) Archetypal Architecture of DeepSCNs

Fig 1 shows archetypal DeepSCN with two input nodes, two hidden layers with three nodes in first layer and two hidden nodes in second layer, one output node. Readout weights are shown with red dashed lines, hidden weights and biases which are stochastically configured are shown by fixed lines. *Fig* 1 *shows* $n = 2. L_1 = 3$ *and* $L_2 = 2$, $d = 2$, $m = 1$

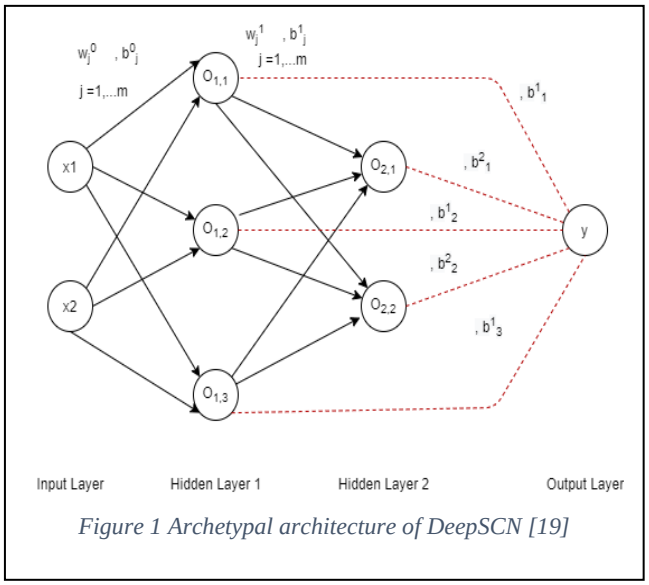


Figure 1 Archetypal architecture of DeepSCN [19]

2) Advantages compared to SCN

Following section is using simulation results and experiment as reported in [19].

For simulation real valued function is defined over [0,1] and 1000 points (randomly generated) were used as training and testing pair.

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

Sigmoidal activation function used for all hidden nodes in the simulation.

$$g(x) = \frac{1}{1 + \exp(-x)}$$

a) Approximation Capability

As per experiments performed in [19], DeepSCN resembles the target function in a defined tolerance range much quickly than SCN. Additionally, DeepSCN beat SCN in the effectiveness domain. On the other hand, both SCN and DeepSCN performance are comparable when consistency relationship is

taken into consideration between learning and generalization

b) Better performance of Hidden output Matrix

Usually, a learner model's ability is measured by using a trade-off metric on learning and generalization performance. Additionally, interpretability is used for the same [19]. Theoretically it is always desirable to have balanced operation ability among learning, generalization, and model complexity [19]. DeepSCN model capacity is associated with algebraic properties of hidden output matrix because of specific configuration in model building
From experiments in [19], it was observed that DeepSCN models have less redundant basis functions compared to SCN models. In SCN models rank deficiency increases when layers are increased

c) Balanced performance with noise induction in external inputs

DeepSCN and SCN framework performance is sensitive to noise in external inputs, changes to the internal structure, and parameter setting. As per observation done in [19] when input sequences are changed DeepSCN as compared to SCN shows less sensitivity which is a sign of better performance in presence of noise in external inputs or changes in internal structure.

V. DIFFERENT WORKS AND RESULTS

1) SCN

a) Robust SCN with kernel density

RSCN in [32] is proposed as an extension to original SCN which is introduced in [8] for dealing with robust data regression problems. In [32] it is assumed that the dataset is clean and validated such that the learning of the model will not produce overfitting during the additive construction of SCN. RSCNs uses weighted least square method for assessing output weights, input weights, and biases are incrementally and stochastically generated by satisfying with a set of inequality constraint. In the RSCNs kernel density estimation method is used for setting the penalty weights on training samples such that noisy data have a negligible impact on the learner's model.

b) SCN ensemble with hetrogeneous features

In [33] extension to SCN is proposed where the SCN model is used as base learner while a fast decorrelated neuro-ensemble with heterogeneous features for large scale data analytics on top of it. A Negative-correlation learning strategy is used to evaluate the output weights. For iterative solutions block, Jacobi and Gauss-seidel methods are used for continuous evaluation of output weights on the basis of heterogeneous features.

c) 2-D SCN

In [34] extension to SCN named 2D-SCN is proposed for Image data modeling task. 1D-SCN destroys spatial information of images which results in undesirable performance. In [33] fast building stochastic learner model with matrix inputs are introduced and theoretically, its advantages over SCNs are presented. In the advantages complexity of the random parameter space and the greater ability of generalization are represented.

d) Mixed Distribution based Robust SCN

In [35] new method is proposed for solving problems regarding the construction of a point prediction model is proposed in which robust SCN with the bootstrap ensemble is used for the creation of prediction Interval. Original SCN uses the least square method for output weight calculation while robust SCN uses a mixer of Gaussian and Laplace distribution in the Bayesian framework. Usage of Mixed distributions effectively characterize the complex distribution of real data and their heavy-tailed properties improve the robustness of SCN

e) SCN based adaptive storage

In [36] adaptive power storage replica management system (PARMS) is introduced which meets the requirement of low latency processing application. In the proposed method network traffic and data center's geographical distribution are taken into consideration for the improvement of real-time data processing. SCN is model is used for estimation of traffic state of power data networks, then multiple data replica management algorithms are introduced to lower the effects of limited bandwidth with the fixed underlying infrastructure. Following system is implemented using parallel data computing frameworks for the power industry.

f) Ensemble SCN

In [37] ensemble SCN is proposed for calculation of Prediction Intervals which can adopt to quantify uncertainty related to point prediction. The proposed method improve the modeling stability and computational efficiency. Robustness of ensemble SCNs is based on Bayesian ridge regression and M-estimate. Additionally, hyperparameters of the assumed distribution over noise and output weights of the model are estimated by expectation-maximization algorithm which increases prediction accuracy.

g) SCN based prediction

In [38] SCN based model is proposed for the prediction of component concentrations in sodium aluminate liquor. The proposed model contains SCN as a base model with sodium carbonate concentration are proposed using the predicted value of caustic hydroxide and alumina. Following proposed model is used in Henan Branch of China Aluminum Co. Ltd which establishes the ability of the model for prediction accuracy, compared against the regress model, BP neural networks, RBF neural networks, and random vector functional link model

2) DeepSCNs

a) DeepSCN based prediction

In [39] DSCNs are used for creation of prediction Interval for carbon residual content of crude oil based on the lower-upper bound estimation (LUBE) method. In the proposed method input weights and biases of DSCN are stochastically assigned with a supervisory mechanism and just the output weights need to be assessed which decreases the number of parameters to be optimized. In addition new cost function of DSCN for constructing PIs based on LUBE method is proposed to optimize the mean values of PIs near the targets.

VI. MATLAB CODE SNAPSHOTS

a) SCN matlab code [40]

```
classdef SCN
    properties
        Name = 'Stochastic Configuration Networks';
        version = '1.0 beta';
        % Basic parameters (networks structure)
        L       % hidden node number / start with 1
        W       % input weight matrix
        b       % hidden layer bias vector
        Beta    % output weight vector
        % Configurational parameters
        r       % regularization parameter
        tol     % tolerance
        Lambdas % random weights range, linear grid search
        L_max   % maximum number of hidden neurons
        T_max   % Maximum times of random configurations
        % Else
        nB = 1 % how many node need to be added in the network in one loop
        verbose = 50 % display frequency
        COST = 0   % final error
    end
    %% Funcitons and algorithm
    methods
        %% Initialize one SCN model
        function obj = SCN(L_max, T_max, tol, Lambdas, r , nB)
            format long;

            obj.L = 1;

            if ~exist('L_max', 'var') || isempty(L_max)
```


```
classdef SCN
    properties
        Name = 'Stochastic Configuration Networks';
        version = '1.0 beta';
        % Basic parameters (networks structure)
        L       % hidden node number / start with 1
        W       % input weight matrix
        b       % hidden layer bias vector
        Beta    % output weight vector
        % Configurational parameters
        r       % regularization parameter
        tol     % tolerance
        Lambdas % random weights range, linear grid search
        L_max   % maximum number of hidden neurons
        T_max   % Maximum times of random configurations
        % Else
        nB = 1 % how many node need to be added in the network in one loop
        verbose = 50 % display frequency
        COST = 0   % final error
    end
    %% Funcitons and algorithm
    methods
        %% Initialize one SCN model
        function obj = SCN(L_max, T_max, tol, Lambdas, r , nB)
            format long;

            obj.L = 1;

            if ~exist('L_max', 'var') || isempty(L_max)
```


```
            end
            if ~exist('nB', 'var') || isempty(nB)
                obj.nB = 1;
            else
                obj.nB = nB;
            end
        end

        %% inequality equation return the ksi
        function [obj, ksi] = InequalityEq(obj, eq, gk, r_L)
            ksi = ((eq'*gk)^2)/(gk'*gk) - (1 - r_L)*(eq'*eq);
        end
        %% Search for [WB,bB] of nB nodes
        function [WB, bB, Flag] = SC_Search(obj, X, E0)
            Flag = 0;% 0: continue; 1: stop; return a good node /or stop training by set Flag = 1
            WB = [];
            bB = [];
            [~,d] = size(X); % Get Sample and feature number
            [~,m] = size(E0);
            % Linear search for better nodes
            C = []; % container of ksi
            for i_Lambdas = 1: length(obj.Lambdas)  % i index lambda
                Lambda = obj.Lambdas(i_Lambdas);    % Get the random weight and bias range
                % Generate candidates T_max vectors of w and b for selection
                WT = Lambda*( 2*rand(d, [obj.T_max])-1 ); % WW is d-by-T_max
                bT = Lambda*( 2*rand(1, [obj.T_max])-1 ); % bb is 1-by-T_max
                HT = logsig(bsxfun(@plus, X*WT, bT));
```

```
                nr = logsig(bsxfun(@plus, X*wi, bi));
                for i_r = 1:length(obj.r)
                    r_L = obj.r(i_r); % get the regularization value
                    % calculate the Ksi value
                    for t = 1: obj.T_max % searching one by one
                        % Calculate H_t
                        H_t = HT(:,t);
                        % Calculate ksi_1 ... ksi_m
                        ksi_m = zeros(1, m);
                        for i_m = 1:m
                            eq = E0(:,i_m);
                            gk = H_t;
                            [obj, ksi_m(i_m)] = obj.InequalityEq(eq, gk, r_L);
                        end
                        Ksi_t = sum(ksi_m);
                        if min(ksi_m) > 0
                            C = cat(2,C, Ksi_t);
                            WB  = cat(2, WB, WT(:,t));
                            bB  = cat(2, bB, bT(:,t));
                        end
                    end
                    nC = length(C);
                    if nC >= obj.nB
                        break; % r loop
                    else
                        continue;
                    end
                end %(r)
                if nC >= obj.nB
```

```
                        break; % lambda loop
                    else
                        continue;
                    end
                end % (lambda)
                % Return the good node / or stop the training.
                if nC>= obj.nB
                    [~, I] = sort(C, 'descend');
                    I_nb = I(1:obj.nB);
                    WB = WB(:, I_nb);
                    bB = bB(:, I_nb);
                    %HB = HB(:, I_nb);
                end
                if nC == 0 || nC<obj.nB % discard w b
                    disp('End Searching ...');
                    Flag = 1;
                end
            end

            %% Add nodes to the model
            function obj = AddNodes(obj, w_L, b_L)
                obj.W = cat(2,obj.W, w_L);
                obj.b = cat(2,obj.b, b_L);
                obj.L = length(obj.b);
            end

            %% Compute the Beta, Output, ErrorVector and Cost
            function [obj, O, E, Error] = UpgradeSCN(obj, X, T)
```

```
            H = obj.GetH(X);
            obj = obj.ComputeBeta(H,T);
            O = H*obj.Beta;
            E = T - O;
            Error =  Tools.RMSE(E);
            obj.COST = Error;
        end

        %% ComputeBeta
        function [obj, Beta] = ComputeBeta(obj, H, T)
            Beta = pinv(H)*T;
            obj.Beta = Beta;
        end
        %% Regression
        function [obj, per] = Regression(obj, X, T)
            per.Error = [];
            E = T;
            Error =  Tools.RMSE(E);
            disp(obj.Name);
            while (obj.L < obj.L_max) && (Error > obj.tol)
                if mod(obj.L, obj.verbose) == 0
                    fprintf('L:%d\t\tRMSE:%.6f \r', obj.L, Error );
                end
                [w_L, b_L, Flag] = SC_Search(obj, X, E);% Search for candidate node / Hidden Parameters
                if Flag == 1
                    break;% could not find enough node
                end
                obj = AddNodes(obj, w_L, b_L);
```

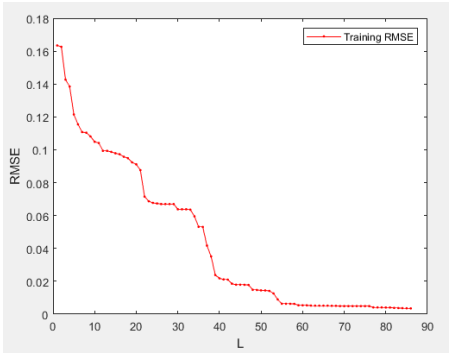
```
            [obj, ~ , E, Error ] = obj.UpgradeSCN(X, T); % Calculate Beta/ Update all
            %log
            per.Error = cat(2, per.Error, repmat(Error, 1, obj.nB));
        end% while
        fprintf('#L:%d\t\tRMSE:%.6f \r', obj.L, Error );
        disp(repmat('*', 1,50));
    end

    %% Classification
    function [obj, per] = Classification(obj, X, T)
        per.Error = []; % Cost function error
        per.Rate = []; % Accuracy Rate
        E = T;
        Error =  Tools.RMSE(E);
        Rate = 0;
        disp(obj.Name);
        while (obj.L < obj.L_max) && (Error > obj.tol)
            if mod(obj.L, obj.verbose) == 0
                fprintf('L:%d\t\t RMSE:%.6f; \t\tRate:%.2f\r', obj.L, Error, Rate);
            end
            [w_L, b_L, Flag] = SC_Search(obj, X, E);
            if Flag == 1
                break;% could not find enough node
            end
            obj = AddNodes(obj, w_L, b_L);
            [obj, ~, E, Error ] = obj.UpgradeSCN(X, T); % Calculate Beta/ Update all
            O = obj.GetLabel(X);
```

```
            Rate = 1- confusion(T',O');
            % Training LOG
            per.Error = cat(2, per.Error, repmat(Error, 1, obj.nB));
            per.Rate = cat(2, per.Rate,  repmat(Rate, 1, obj.nB));
        end% while
        fprintf('#L:%d\t\t RMSE:%.6f; \t\tRate:%.2f\r', obj.L, Error, Rate);
        disp(repmat('*', 1,50));
    end

    %% Output Matrix of hidden layer
    function H = GetH(obj, X)
        H =  obj.ActivationFun(X);
    end
    % Sigmoid function
    function H = ActivationFun(obj,  X)
        H = logsig(bsxfun(@plus, X*[obj.W],[obj.b]));
    end
    %% Get Output
    function O = GetOutput(obj, X)
        H = obj.GetH(X);
        O = H*[obj.Beta];
    end
    %% Get Label
    function O = GetLabel(obj, X)
        O = GetOutput(obj, X);
        O = Tools.OneHotMatrix(O);
    end
    %% Get Accuracy
    function [Rate, O] = GetAccuracy(obj, X, T)
```

```
%% Model Initialization
M = SCN(L_max, T_max, tol, Lambdas, r , nB);
disp(M);

%% Model Training
% M is the trained model
% per contains the training error with respect to the increasing L
[M, per] = M.Regression(X, T);
disp(M);

%% Training error demo
figure;
plot(per.Error, 'r.-');
xlabel('L');
ylabel('RMSE');
legend('Training RMSE');

%% Model output vs target on training dataset
O1 = M.GetOutput(X);
figure;
plot(X,T, 'r.-'); hold on;
plot(X,O1, 'b.-');
xlabel('X');
ylabel('Y');
legend('Training Target', 'Model Output');
```

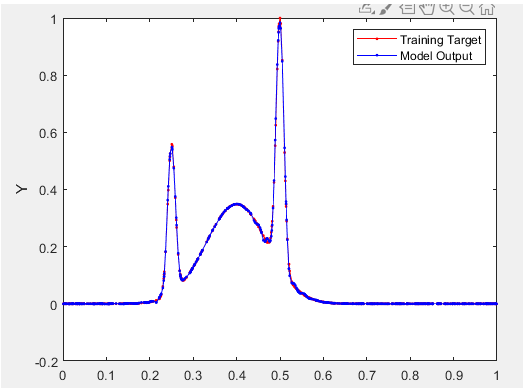
```
%% Model output vs target on test dataset
O2 = M.GetOutput(X2);
figure;
plot(X2, T2, 'r.-'); hold on;
plot(X2, O2, 'b.-');
xlabel('X');
ylabel('Y');
legend('Test Target', 'Model Output');

% The End
```

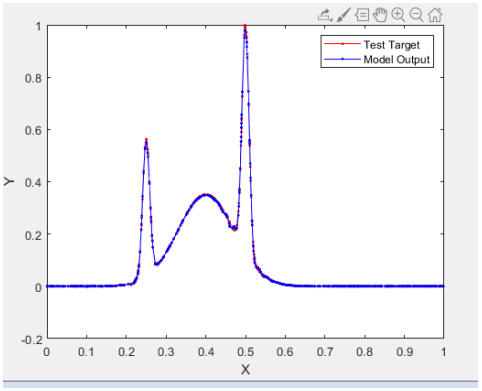
```
        function [Rate, O] = GetAccuracy(obj, X, T)
            O = obj.GetLabel(X);
            Rate = 1- confusion(T',O');
        end
        %% Get Error, Output and Hidden Matrix
        function [Error, O, H, E] = GetResult(obj, X, T)
            % X, T are test data or validation data
            H = obj.GetH(X);
            O = H*(obj.Beta);
            E = T - O;
            Error =  Tools.RMSE(E);
        end

    end % methods
end % class
```



b) Using SCN for Regression

```
% SCN - Regression
clear;
clc;
close all;
format long;
%%  Prepare the training (X, T) and test data (X2, T2)
% X: each row vector represents one sample input.
% T: each row vector represents one sample target.
% same to the X2 and T2.
% Note: Data preprocessing (normalization) should be done before running the program.

load('Toy_Data.mat');

figure;
plot(X,T, 'r.-'); hold on;
plot(X2, T2, 'b.-');
legend('Training', ...
       'Test     ');

%% Parameter Setting
L_max = 250;                    % maximum hidden node number
tol = 0.001;                    % training tolerance
T_max = 100;                    % maximun candidate nodes number
Lambdas = [0.5, 1, 5, 10, ...
    30, 50, 100, 150, 200, 250];% scope sequence
r =  [ 0.9, 0.99, 0.999, ...
    0.9999, 0.99999, 0.999999]; % 1-r contraction sequence
nB = 1;       % batch size
```

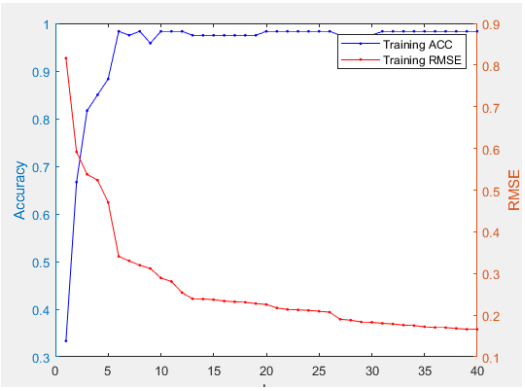




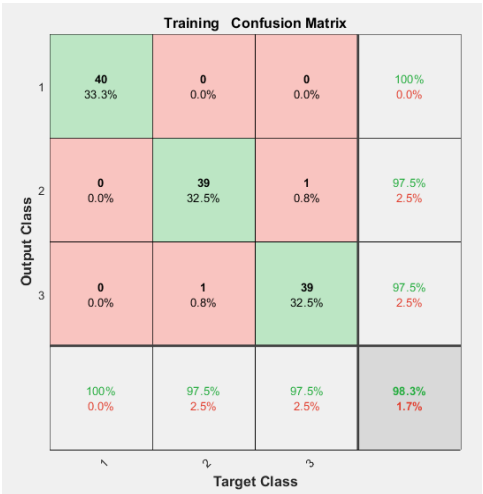

```
%% Model output vs target on training dataset
O1 = M.GetLabel(X);
disp(['Training Acc: ' num2str(M.GetAccuracy(X, T))]);
figure;
plotconfusion(T',O1','Training  ');

%% Model output vs target on test dataset
O2 = M.GetLabel(X2);
disp(['Test Acc: ', num2str(M.GetAccuracy(X2, T2))]);
figure;
plotconfusion(T2',O2','Test  ');

% The End
```

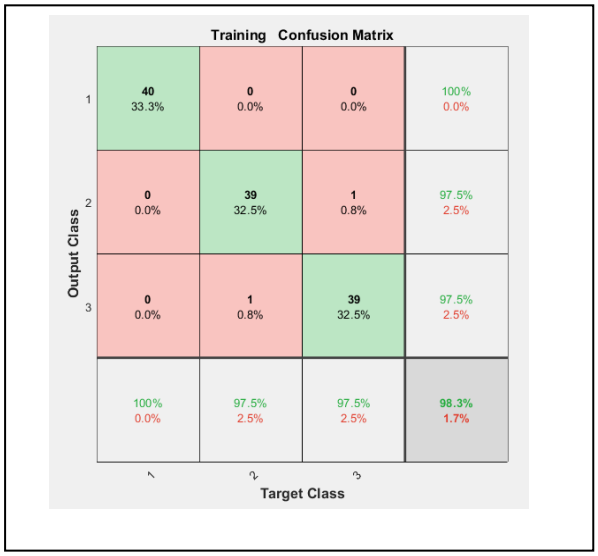
c) Using SCN for Classification

```
% SCN - Regression
clear;
clc;
close all;
format long;
%%  Prepare the training (X, T) and test data (X2, T2)
% X: each row vector represents one sample input.
% T: each row vector represents one sample target.
% same to the X2 and T2.
% Note: Data preprocessing (normalization) should be done before running the program.

load('Toy_Data.mat');

figure;
plot(X,T, 'r.-'); hold on;
plot(X2, T2, 'b.-');
legend('Training', ...
       'Test     ');

%% Parameter Setting
L_max = 250;                    % maximum hidden node number
tol = 0.001;                    % training tolerance
T_max = 100;                    % maximun candidate nodes number
Lambdas = [0.5, 1, 5, 10, ...
    30, 50, 100, 150, 200, 250];% scope sequence
r =  [ 0.9, 0.99, 0.999, ...
    0.9999, 0.99999, 0.999999]; % 1-r contraction sequence
nB = 1;       % batch size
```




```
Lambdas = [0.5, 1, 5, 10, 30, 50,100];% scope sequence
r =  [ 0.9, 0.99, 0.999, ...
    0.9999, 0.99999, 0.999999]; % 1-r contraction sequence
nB = 1;       % batch size

%% Model Initialization
M = SCN(L_max, T_max, tol, Lambdas, r , nB);

%% Model Training
% M is the trained model
% per contains the training error and accuracy with respect to the increasing L
[M, per] = M.Classification(X, T);
disp(M);

%% Training error and accuracy demo
figure;
yyaxis left;
plot(per.Rate, 'b.-'); hold on;
ylabel('Accuracy');
yyaxis right;
plot(per.Error, 'r.-');
xlabel('L');
ylabel('RMSE');
xlabel('L');
legend( 'Training ACC', 'Training RMSE');
```





ACKNOWLEDGMENT

I would like to thank Prof Dr. Pech, for the continuous support and guidance provided for this seminar paper. Furthermore, I would also like to put my sincere gratitude for providing all the necessary information and useful links which proved to be very helpful for the successful completion of this seminar paper.

REFERENCES

[1] D. C. R. P. K. Itamar Arel, "Deep Machine Learning – A new Frontier in Artificial Intelligence research".

[2] R. Bellman, "Dynamic Programming," Princetion Press, 1957.

[3] P. H. a. D. S. R. Duda, "Pattern Recognition,," in *Wiley*, NewYork, 2000.

[4] T. Lee and D. Mumford, "“Hierarchical Bayesian inference in the visual cortex,," J.Opt, 2003, pp. 1434-1448.

[5] D. M. R. R. a. V. L. T. Lee, "“The role of the primary visual cortex in higher level vision,”," 1998., pp. 2429–2454,.

[6] G. W. a. H. Bülthoff, " “Learning to recognize objects,”," in *Trends Cogn. Sci., vol. 3,*, 1999., p. 23–31.

[7] G. W. a. E. Rolls, "“Invariant face and object recognition in the visual system,”," in *prog. Neurobiol., vol. 51,*, 1997, pp. 167-194.

[8] M. L. Dianhui Wang, "Stochastic Configuration Networks – Fundamental and Algorithm".

[9] H. C. T. Chen, " Universal approximation to nonlinear operators by neural networks with arbitrary," IEEE, 1995.

[10] G. Cybenko, Approximations by superpositions of a sigmoidal function, Mathematics of Control, Signals, 303-314, 1989.

[11] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks* , 1988, pp. 593-605.

[12] M. S. H. W. K. Hornik, "Multilayer feedforward networks are universal approximators," in *Neural Networks* , 1989, pp. 359-366.

[13] H. J. M. LukošEviˇcIus, "Reservoir computing approaches to recurrent neural network training,," 2009, pp. 127-149.

[14] M. W. Mahoney, "Randomized algorithms for matrices and data," in *Foundations and Trends in Machine*, 2011, pp. 123-224.

[15] I. S. J. Park, " Universal approximation using radial basis function networks,," in *Neural Computation*, 1991, pp. 246-257.

[16] G. E. H. R. J. W. D. E. Rumelhart, "Learning internal representations by backpropagating," 1986, pp. 533-536.

[17] D. W. S. Scardapane, in *Randomness in neural networks: An overview, WIREs Data Mining and*, 2017.

[18] D. Wang, " Randomized algorithms for training neural networks," in *Information Sciences* , 2016, pp. 364-365.

[19] M. L. Dianhui Wang, "Deep Stochastic Configuration Networks with Universal Approximation Property".

[20] Y. B. a. G. H. Y. LeCun, in *Deep learning. Nature,*, 2015, pp. 436–444,.

[21] J. Schmidhuber, "Deep learning in neural networks: An overview," in *Neural Network*, 2015, pp. 85-117.

[22] Y. B. a. A. C. I. Goodfellow, "Deep Learning," MIT press, p. 2016.

[23] G. H. a. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks.," in *Information Sciences*, 2006, pp. 504-507.

[24] A. C. a. P. V. Y. Bengio, in *Representation learning: A review and new perspectives.*, 2013, pp. 1798-1828.

[25] R. E. a. O. Shamir, "The power of depth for feedforward neural networks," 2015.

[26] J. A. a. M. Salzmann, "Learning the number of neurons in deep networks," in *In Advances in Neural Information Processing System*, 2016, pp. 2262-2270.

[27] G. E. H. R. J. W. D. E. Rumelhart, "Learning internal representations by backpropagating errors, Nature," 1986, pp. 533-536.

[28] A. B. R. G. a. T. M. S. Arora, "Provable bounds for learning some deep representations," in *In*

Proceedings of the 31th International Conference on Machine Learning,, 2014, pp. 584–592,.

[29] G. Cybenko, "Approximation by superpositions of a sigmoidal function," in *Mathematics of Control, Signals and Systems*, 1989, pp. 303-314.

[30] M. S. a. H. W. K. Hornik, " Multilayer feedforward networks are universal approximators," in *Neural Networks*, 1989, pp. 359-366.

[31] R. Tibshirani, " Regression shrinkage and selection via the lasso," in *Journal of the Royal Statistical series*, 1996, pp. 267-288.

[32] M. L. Dianhui Wang ∗, "Robust stochastic configuration networks with kernel density estimation for uncertain data regression".

[33] C. C. Dianhui Wang ∗, "Stochastic configuration networks ensemble with heterogeneous features for large-scale data analytics".

[34] D. W. Ming Li, "Two Dimensional Stochastic Configuration Networks for Image Data Analytics".

[35] J. Jun Lu and Jinliang Ding, "Mixed-Distribution-Based Robust Stochastic Configuration Networks for Prediction Interval Construction".

[36] Q. H. D. W. Changqin Huang, "Stochastic Configuration Networks Based Adaptive Storage Replica Management for Power Big Data Processing".

[37] J. D. ,. X. D. T. C. Jun Lu, "Ensemble Stochastic Configuration Networks for Estimating Prediction Intervals: A Simultaneous Robust Training Algorithm and Its Application".

[38] W. W. •. D. Wang, "Prediction of component concentrations in sodium aluminate liquor using stochastic configuration networks".

[39] J. D. Jun Lu, "Construction of prediction intervals for carbon residual of crude oil based on deep stochastic configuration networks".

[40] "http://www.deepscn.com/software.php," [Online]. Available: http://www.deepscn.com/software.php.