Stochastic Least Squares Learning for Deep Architectures

Girish Kumar*, Jian Min Sim†, Eng Yeow Cheu‡ and Xiaoli Li§
* NUS High School of Mathematics and Science, Singapore 129957
Email: girishvilla@gmail.com
†Exploit Technologies Pte Ltd, A*STAR (Agency for Science, Technology and Research), Singapore 138671
Email: sim_jian_min@etpl.sg
‡Rolls Royce Singapore Pte Ltd
Email: engyeow.cheu@rolls-royce.com
§Institute for Infocomm Research, A*STAR, Singapore 138632
Email: xlli@i2r.a-star.edu.sg

***Abstract*—In this paper, we present a novel way of pre-training deep architectures by using the stochastic least squares autoencoder (SLSA). The SLSA is based on the combination of stochastic least squares estimation and logistic sampling. The usefulness of the stochastic least squares approach coupled with the numerical trick of constraining the logistic sampling process is highlighted in this paper. This approach was tested and benchmarked against other methods including Neural Nets (NN), Deep Belief Nets (DBN), and Stacked Denoising Autoencoder (SDAE) using the MNIST dataset. In addition, the SLSA architecture was also tested against established methods such as the Support Vector Machine (SVM), and the Naive Bayes Classifier (NB) on the Reuters-21578 and MNIST datasets. The experiments show the promise of SLSA as a pre-training step, in which stacked of SLSA yielded the lowest classification error and the highest F-measure scores on the MNIST and Reuters-21578 datasets respectively. Hence, this paper establishes the value of pre-training deep neural network, by using the SLSA.**

I. Introduction

Deep architectures orginate from Artificial Neural Networks (ANN), a machine learning method inspired by the way biological neurons function [1]. ANNs are typically trained with gradient-descent based methods amongst which, error backpropagation is the most common approach. While this has proven to be feasible, it has a number of limitations such as poor scaling with learning time, and poor resulting local optima. As such, deep architectures have long been difficult to tackle using supervised methods such as error backpropagation, which may not be efficient for practical purposes [2].

A breakthrough occurred in 2006 when Hinton et. al. [3] proposed a novel approach by treating deep, directed networks as a composition of individual units called Restricted Boltzmann Machines (RBMs) [4]. Multiple RBMs are pre-trained using a greedy algorithm layer by layer to obtain good initial weights that can later be stacked together to form a Deep Belief Net (DBN). A DBN can then be fine-tuned using gradient descent [5], [3]. Other approaches include replacing RBMs with autoencoders for pre-training in which Bengio et. al. reported that the results were comparable to that of Deep Belief Nets (DBNs) [6].

However, the approaches used in architectures such as DBNs and SDAEs are still susceptible to local minima. To overcome this issue, we present the SLSA architecture which combines stochastic least squares estimation with constrained logistic sampling. The motivation for the proposed method stems from the similarity between the logistic sampling process and the stochastic nature of neuro-biological process in which the neurons are activated when the membrane potential exceeds the firing threshold [7]. Because the stochastic nature of the logistic sampling process provides a way for solutions to escape local minima more easily, our proposed method was shown to converge much faster (c.f. Section IV). In addition, the least squares approach helped SLSA achieved better results than established methods such as DBN, SVM and NB (c.f. Section IV).

II. Literature Review

Stochastic and deterministic methods have been used to train individual units that can be stacked to make up a deep neural net. The following section aims to highlight the differences between such units which include the Restricted Boltzmann Machine (RBM), Principal Component Analysis (PCA), Kernel PCA (KPCA), stochastic KPCA, stochastic autoencoder, and denoising autoencoder.

A. Restricted Boltzmann Machine

Restricted Boltzmann Machine (RBM), a bipartite graph variant of the boltzmann machine, is essentially an energy-based probability model to infer hidden variables [8]. The bipartite nature of the RBM means that it does not allow connections among units in each layer [9], which makes it efficient in learning [8].

RBMs are trained as probabilistic models by maximizing a log-likelihood criterion [4]. To achieve this, contrastive divergence (CD) is used to efficiently approximate the log-likelihood gradient of RBMs [10], [11], [12]. The RBM learns in an unsupervised fashion with a stochastic element being introduced in the random sampling process. The CD algorithm updates the weights from the following:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \epsilon \left(\boldsymbol{h}_t \boldsymbol{v}_t - \boldsymbol{h}_{t+1} \boldsymbol{v}_{t+1} \right) \tag{1}$$

where the subscript t represents the number of iterations, $\boldsymbol{v}$ is the visible inputs, $\boldsymbol{h}$ is the hidden vector, and ϵ is the learning rate.

Pre-training of stacked RBMs, known as a Deep Belief Net [13], is explicitly done by constantly treating the hidden layer of one RBM as the visible inputs to the next RBM to form a stacked RBM architecture according to the desired number of layers. The pre-training stage helps to generate good initial weights for supervised fine-tuning via error backpropagation on the entire architecture.

B. Autoencoder

The autoencoder can be viewed as a two-layer building block for training deep architectures, with the first layer as the mapping for the encoding process and the next layer as the decoding mapping. The encoder maps the input, $\boldsymbol{v}$, into the hidden representation, $\boldsymbol{h}$, while the decoder reconstructs the inputs, $\hat{\boldsymbol{v}}$, from the hidden representation. The general form of the encoding and decoding process can be represented mathematically as follows:

$$f_{encode} = g(\boldsymbol{W}\boldsymbol{v} + \boldsymbol{b}) \quad (2)$$

$$f_{decode} = k(\boldsymbol{V} f_{encode} + \boldsymbol{c}) \quad (3)$$

where $\boldsymbol{W}$ and $\boldsymbol{V}$ are weight matrices, $\boldsymbol{b}$ and $\boldsymbol{c}$ are the bias vectors, and g and k are the mapping functions for the encoder and decoder respectively.

The individual autoencoder is often trained using supervised aproaches such as the different variants of the error backpropagation method. Using the layer-wise training strategy for DBN, the RBM can be replaced by an autoencoder to generate a stacked autoencoder [6]. The main difference between the autoencoder and the RBM lies in the fact that RBM is probabilistic while an autoencoder is deterministic. Hence, the gradient of the log-likelihood of RBM is intractable and has to be approximated by CD algorithm, while that of an autoencoder can be computed with gradient descent.

Replacing g and k with sigmoid functions in Eqn (2) and Eqn (3), the training objective for the stacked autoencoder architecture is then to minimize the reconstruction cross-entropy [6].

$$R = -\sum_i v_i \log p_i(v) + (1 - v_i)\log(1 - p_i(v)) \quad (4)$$

C. Principal Component Analysis

Principal component analysis (PCA), a well-known feature extraction approach introduced in [14], can be viewed as the linear example of the autoencoder in representation learning [15]. PCA can be thought of as a building block, similar to the autoencoder, which can be implemented to make the output to be the same as the input. The only difference is the minimization of the mean squared errors by compressing the input vector in a code, with linear visible and hidden layers.

While PCA is a linear dimensionality reduction technique, which takes n dimensional data and represents them compactly in m principal directions [16], the practical advantage lies in its ability to compress the information contained within the data and to reconstruct the data based on the compressed information at a later stage. On the other hand, autoencoders can allow non-linear dimensionality reduction via non-linear mappings [16]. Backpropagation algorithm can then be used to train these autoencoders via a local search on the weight space [17]. As opposed to PCA, training autoencoders will provide the flexibility in terms of non-linear mappings, a linear learning time and a fairly compact and fast encoding mode [16].

D. Kernel Principal Component Analysis

By introducing the kernel approach to PCA, it is possible to achieve non-linear dimensionality reduction of the data. While the non-linear mapping resembles that in an autoencoder, the main difference lies in the application of the kernel trick. This kernel trick deals with non-linear distribution of inputs by attempting to linearize it via non-linear mapping from the input to the feature space. This means that it is now possible to perform linear PCA in the feature space.

KPCA can similarly be viewed as a building block for learning deep architectures. The advantage over PCA and autoencoder is the computation of the principal components without the need to know the high-dimensional mapping functions.

E. Probabilistic KPCA

In order to obtain stochastic models and non-linear high dimensional mapping of input data, probabilistic versions of KPCA have been proposed [18], [19], [20]. By introducing the probability density, $p(\boldsymbol{v})$, the probabilistic KPCA maximizes the following log-likelihood.

$$L = \prod_{i=1}^{n} p(\boldsymbol{v}_i) \quad (5)$$

This is done by first taking into account a linear combination of a vector $\boldsymbol{y}$ and noise vector $\boldsymbol{n}$ [21].

$$\boldsymbol{v} = U\boldsymbol{y} + \boldsymbol{\mu} + \boldsymbol{n} \quad (6)$$

where U is a matrix to relate the two set of variable vectors $\boldsymbol{y}$ and $\boldsymbol{v}$, $\boldsymbol{n}$ is a noise vector and $\boldsymbol{\mu}$ is a vector to offset non-zero mean. By finding U and expressing $p(\boldsymbol{v})$ in terms of U, Eqn (5) can then be maximized. Introducing a stochastic element in KPCA takes advantge of both the generative property of the RBM and the flexibility of the kernel trick in KPCA in dealing with non-linear input data.

F. Denoising Autoencoder

Denoising autoencoder (DAE) is a stochastic version of the autoencoder. Because of the various limitations of the autoencoder including the inability to guarantee extraction of useful features and that it might learn an identity mapping, one of the strategy is to introduce a bottleneck or sparse representation [15], [22]. The idea of a denoising autoencoder is to alter the reconstruction criterion to make learning good representation more robust under corruption of the inputs, c.f. Eqn (6). As opposed to the vanilla autoencoder, the introduction of input corruption in the denoising autoencoder brings about a more robust representational learning compared to the basic autoencoder as a result of the corruption process.

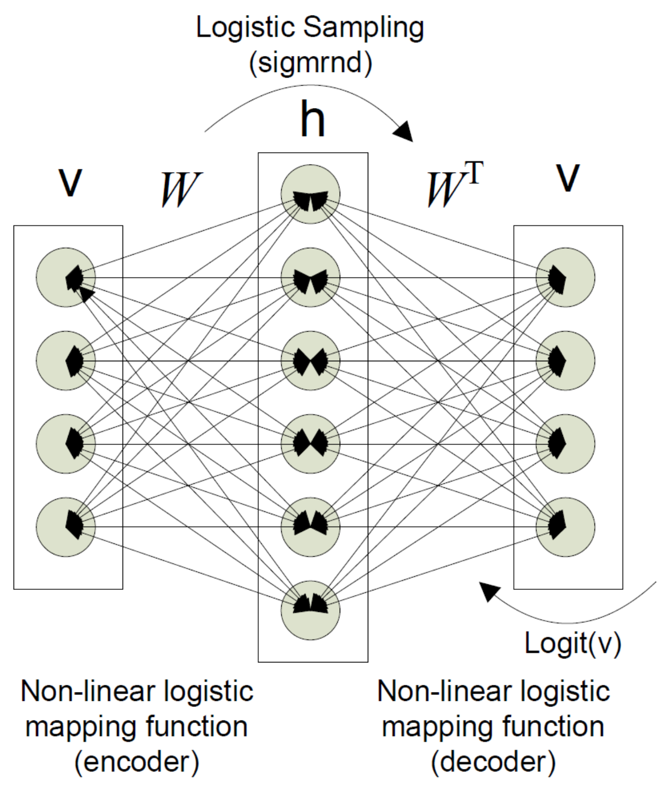
Logistic Sampling (sigmrnd)
h
V
W
W^{T}
V
Logit(v)
Non-linear logistic mapping function (encoder)
Non-linear logistic mapping function (decoder)

Fig. 1. The architecture of the proposed stochastic least squares autoencoder which is based on logistic sampling process.

Denoising autoencoders can also be stacked like a building block, to obtain the stacked denoising autoencoder which is introduced by Vincent et. al. [23].

III. The Proposed Technique

The proposed stochastic least squares autoencoder (SLSA) in Fig. 1 is developed upon the encoding-decoding concept employed by autoencoders in Section II. The main novelty in SLSA is the simple stochastic least squares solution which is derived by Singular Value Decomposition (SVD) on a sparse random matrix, of which, is generated by the logistic sampling process inherently introduced by sampling the sigmoid activation function associated with the hidden nodes. The encoding weights and decoding weights are tied i.e. they are transposes of one another. The logistic sampling process maximizes the likelihood of certain specific data samples during the encoding process and the least squares solution generated by SVD during the decoding process minimizes the sum of squared errors. The recurrent stochastic encoding and least squares decoding process will lead to the asymptotic convergence of the weight.

Similar to the RBM, SLSA can be seen as an energy-based model such that energy of the model is minimized. Hence, to achieve the descent of the energy function with iterations, we will have to establish that the competing natures of both $\boldsymbol{h}$ and W will lead to the asymptotic convergence of the new energy function in Eqn (7) [8].

$$\mathrm{Energy}\left(\boldsymbol{v},\boldsymbol{h}\right) = -\boldsymbol{h}^{\mathrm{T}}W\boldsymbol{v} - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{h} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{v} \tag{7}$$

where $\mathbf{b}$ and $\mathbf{c}$ are the visible and hidden biases respectively.
The first layer of hidden nodes first computes the dot product of the input vector $\boldsymbol{v}$ and the weight matrix $\boldsymbol{W}$. The output of this hidden layer of SLSA as shown in Fig. 1 represents the sparse representative code $\boldsymbol{h}$ of the input data $\boldsymbol{v}$ after sampling from the logistic activation function associated with the nodes. This encoding process is likened to the activation of the biological neuron when its membrane potential exceeds the firing threshold such that a value of 1 represents the corresponding hidden node is activated and a value of 0 non-activation of the node. The input data $\boldsymbol{v}$ is then recovered by gating the dot product of the sparse code $\boldsymbol{h}$ with the transpose of the weight $\boldsymbol{W}^{T}$ through logistic activation function at the decoding stage. Similar to the encoding and decoding process in Eqn (2) and Eqn (3) respectively, the SLSA architecture can be expressed mathematically in Eqn (8),

$$\boldsymbol{v} = \mathrm{sigm}\left(\mathrm{sigmrnd}\left(\boldsymbol{v}\boldsymbol{W}_t + \boldsymbol{b}_t\right)\cdot\boldsymbol{W}_t^{\mathrm{T}}\right) \tag{8}$$

where g is represented by sigmrnd (i.e. logistic sampling), k by sigm (i.e. sigmoid function) and V by $\boldsymbol{W}^{\mathrm{T}}$.

The kernel function g and k in Eqn (2) and Eqn (3) respectively can either represent linear or non-linear mapping functions. The simpler case of affine mapping functions and squared error loss criterion will correspond to PCA. It is possible for the solution to remain in the linear segment of the sigmoid function, and obtaining the PCA subspace is only a likely possibility [24]. In order to lower the possibility of a PCA subspace solution, weights are tied between the encoder and decoder (i.e. the weight matrix used in the decoder is a transpose of that in the encoder). In addition, tying the weights as a constraint helps to avoid having perfect but useless reconstruction via an identity mapping [15], [2].

Hence, the weight update rule is simply reformulated from Eqn (8) into Eqn (9) and Eqn (10),

$$\begin{aligned}\boldsymbol{W}^{\mathrm{T}}_{\boldsymbol{t+1}} &= \mathrm{pinv}\left(\mathrm{sigmrnd}\left(\boldsymbol{v}\boldsymbol{W}_t + \boldsymbol{b}_t\right)\right)\cdot\boldsymbol{x} && (9)\\ \boldsymbol{b}_{t+1} &= -Q\left(\boldsymbol{v}\boldsymbol{W}_{t+1}, q\right) && (10)\end{aligned}$$

where $\boldsymbol{x}$ is defined in Eqn (11), Q is the quantile function, $\boldsymbol{v}\boldsymbol{W}_{t+1}$ and q are inputs to the function. The Quantile function provides a value which is more than fraction q of the vector $\boldsymbol{v}\boldsymbol{W}_{t+1}$. As such, q governs the the faction of input data samples with less than 50% chance of activating the hidden node.

Since logit function extends to negative and positive infinity when $\boldsymbol{v}$ is 0 and 1 respectively, a simple numerical approximation is applied to limit the logit function from producing an infinite value and causing numerical computation error. Here, any value less than -10 will be capped at -10, and any value more than 10 will be capped at 10. This constraint effective clips the logit function within -10 and 10. Why -10 and 10? Firstly, this is because the sigmoid of these values are already very close to 0 and 1 respectively. Secondly, it is less important to further distinguish data points that are already far from the decision margin. Furthermore, the values are further scaled down to -1 and 1. This is done to prevent saturation and more to capture the relative importance of the different features. This numerical approximation is summarized in Eqn (11).

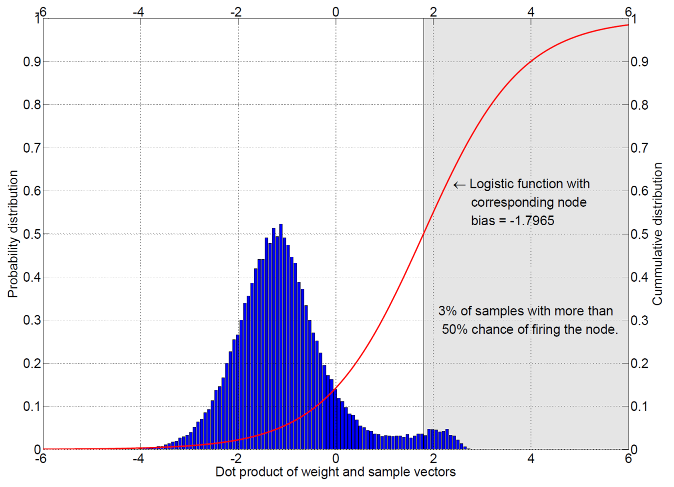

Fig. 2. An example to illustrate the relationship between the bias associated with every hidden node and the sparsity of the hidden layer code $\boldsymbol{h}$.

$$\boldsymbol{x} = \begin{cases} \max(\boldsymbol{y}, -10)/10 & \text{if } \boldsymbol{y} < 0 \\ \min(\boldsymbol{y}, 10)/10 & \text{if } \boldsymbol{y} \geq 0 \end{cases} \tag{11}$$

where $\boldsymbol{y} = \text{logit}(\boldsymbol{v})$.

A. Quantile Parameter

As proposed by Hinton [4], the method to minimize the training error would be to use parameters such as size of mini-batches, learning rate, momentum, weight decay, initial weights and biases, and the number of hidden units. We propose an additional parameter q to determine the sensitivity of each neuron to the data samples. In this paper, the quantile parameter q was used as an important parameter, which is defined as the fraction of samples which will have less than 50% likelihood of firing the hidden node. This quantile parameter q provides a way to adjust the sparsity of the random hidden code $\boldsymbol{h}$ through the bias term $\boldsymbol{b}$.

The bias term $\boldsymbol{b}$ determines the mean of the sigmoid activation functions for the hidden nodes. Increasing the bias term increases the likelihood for a data sample to fire the hidden node. Thus, increasing the bias term is also equivalent to shifting the logistic function towards the left. Decreasing the bias term does the inverse. In the example as shown in Fig. 2, the quantile parameter q is set to 0.97 and the bias for one of hidden nodes is -1.7965 as determined by Eqn (10). Here, only 3% of the data samples have more than 50% likelihood to activate the particular node.

B. Logistic Sampling

The stochastic nature of the SLSA arises from the logistic sampling process introduced through the sigmrnd function in Equation 8. Through the sigmrnd function, we sample hidden neurons using a sigmoidal cummulative distribution function (CDF). Since the sigmoid function is the CDF of the logistic distribution, we model the data points as following a logistic distribution. Note that the SLSA can be used to model data following other distributions via modifying the activation function of the neurons and the weight-update rule.

C. Singular Value Decomposition (SVD)

The term in Eqn (9), $\text{sigmrnd}\,(\boldsymbol{v}\boldsymbol{W}_t + \boldsymbol{b}_t)$, is computed using the Moore-Penrose pseudo-inverse method [25] which is based on the concept of Singular Value Decomposition (SVD) [26]. Taking the SVD on the term result in Eqn (12),

$$\text{sigmrnd}\,(\boldsymbol{v}\boldsymbol{W}_t + \boldsymbol{b}_t) = \boldsymbol{U}\sum\boldsymbol{V}^* \tag{12}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are real or complex unitary matrices, $\sum$ is a rectangular diagonal matrix with non-negative real values on the diagonals and $\boldsymbol{V}^*$ is the conjugate transpose of $\boldsymbol{V}$. Hence, computing the Moore-penrose pseudoinverse is simply,

$$\text{pinv}\,(\text{sigmrnd}\,(\boldsymbol{v}\boldsymbol{W}_t + \boldsymbol{b}_t)) = \boldsymbol{V}\text{pinv}\left(\sum\right)\boldsymbol{U}^* \tag{13}$$

The pseudoinverse of the diagonal matrix, $\sum$, is computed by taking the reciprocals of the diagonal elements. Since the rank of the term $\text{sigmrnd}\,(\boldsymbol{v}\boldsymbol{W}_t + \boldsymbol{b}_t)$ and hence the rank of the new weight matrix $\boldsymbol{W}_{t+1}$, as shown in Eqn (9), can be obtained by reading off from its SVD, this can be determined by the number of diagonal elements of $\sum$ not exceedingly close to zero. Since the rank of the matrix corresponds to the number of linearly independent solutions, the number of unique solutions in a system of equations consisting of each dimension that is made up of the pre-determined number of neurons. Hence, the usage of SVD in the computation of the pseudoinverse leads to each dimension of a sample selecting the unique neurons that are able to describe it.

IV. Experimental Results

In this section, we evaluate our proposed stacked stochastic least squares autoencoders technique SLSA and compare it against 3 existing state-of-the-art techniques, namely, stacked denoising autoencoders (SDAE), Deep Belief Net (DBN), and Neural Nets (NN). Note the implementations of SDAE, DBN, and NN were obtained from a MATLAB package developed by Rasmus (Palm, 2012). Stacking is done similar to the DBN and SDAE. The hidden layer activation of the first layer will be the input to the second layer. Each layer is trained greedily, one after another.

In the experiments, we have tested the above techniques on two different yet commonly used classification problems, i.e. 1) digit classification using MNIST benchmark dataset, and 2) text classification using Reuters benchmark datasets. For both experiments, we have employed two layers of network consisting of 1500 nodes in the first layer and 500 nodes in the second layer, with the quantile parameter q set to 0.97 (Eqn 10). For the length of the training time, 50 epochs were used for pre-training, while 500 epochs were used for training the neural net. For fair comparison, the same parameter values have been used for all the architectures. The detailed results are described in following Sections IV-A and IV-B respectively.

A. Digit Classification Problem

We have performed the experiments using the standard MNIST benchmark digit classification dataset, with $60,000$ training examples and $10,000$ test examples [27].

Fig. 3 shows an example of the visualization of the digits using one layer of 100 hidden nodes that are pre-trained with

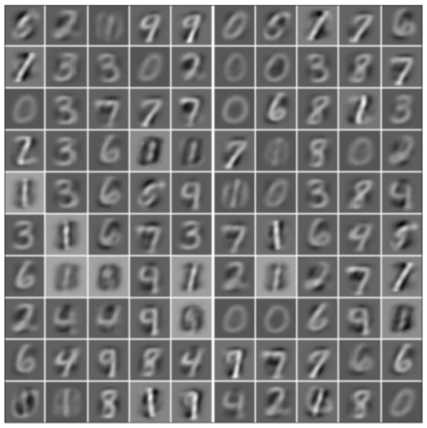

Fig. 3. A visualization of the digits using one layer of 100 hidden nodes pre-trained with SLSA.

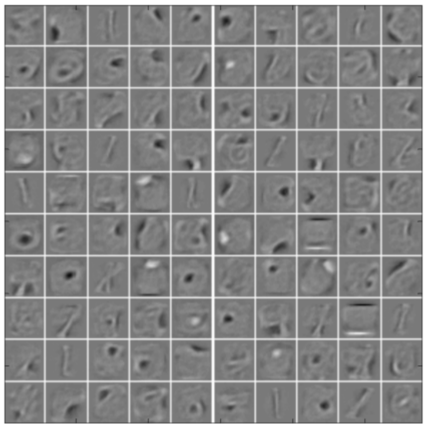

Fig. 4. A visualization of the digits using one layer of 100 hidden nodes pre-trained with RBM.

TABLE I. Error Rates on the MNIST test set

	DBN	SDAE	NN	SLSA
Error Rate (%)	1.3	1.3	1.4	1.3

SLSA. In contrast, Fig. 4 shows the same visualization with pre-training by RBMs. While the main difference lies with the resulting clarity of the images, it also emphasizes that the RBMs needed more layers to be trained.

Table I shows that the SLSA architecture is able to achieve 0.1% less error rates than the NN respectively, and has the same lowest error rate with SDAE and SBN. Compared with all the other architectures (Fig.6), SLSA is able to converge much faster. This is could be due to the gradient descent approach used by the other architectures which leads to the bigger number of epochs when solution gets stuck in a local minima. On the other hand, the logistic sampling process facilitates SLSA escape from local minima and thus converge quickly. Fig. 5 shows the typical asymptotic convergence of the energy function against the length of pre-training with SLSA, indicating that SLSA is able to quickly reduce the average energy within just around 15 epochs. The ripples highlight the stochastic nature of the logistic sampling process.

Fig. 6 and Fig. 7 further illustrate the value of pre-training

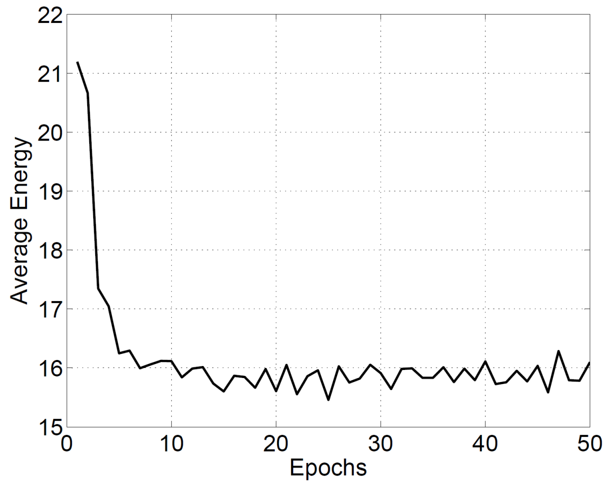

Fig. 5. The plot of average energy against number of epochs for SLSA using a layer of 100 hidden nodes.

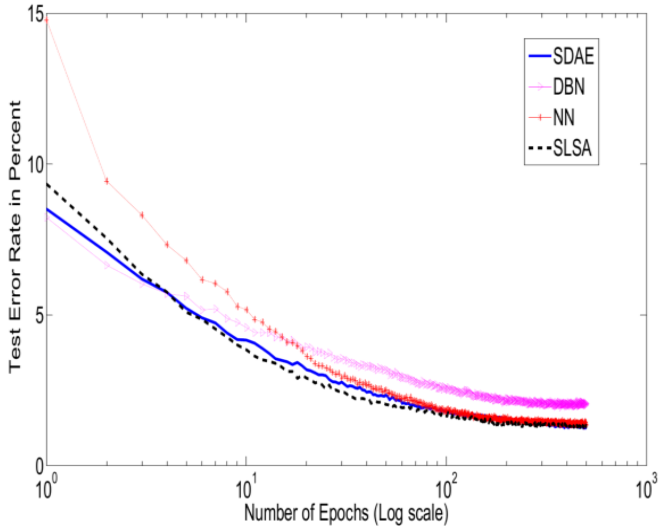

Fig. 6. Plot of convergence rates of different methods for test errors using the dropout method [28] during the supervised learning phase.

with individual units such as RBM, DAE and SLSA, as described in Sections II-A and II-F, in which SLSA, DBN and SDAE performed better than NN in terms of the rate of convergence. Without pre-training, NN is unable to achieve a lower error rate due to its gradient descent approach that could cause the solution obtained to fall into a poor local minima. In addition, the results shown in Table I and Fig. 6 highlight the value of pre-training specifically with the proposed method of using SLSA, in which the error rate is similar to the best result obtained by SDAE and the rate of convergence is the best among SDAE, DBN and NN.

B. *Text Classification Problem*

We have also evaluated the proposed SLSA technique by comparing it with existing techniques using the Reuters-21578 text collection, which is commonly used as a benchmark dataset for evaluating text classification methods. In addition to the existing techniques like DBN, SDAE, we also incorporate two state-of-the-art classification methods, namely, Support

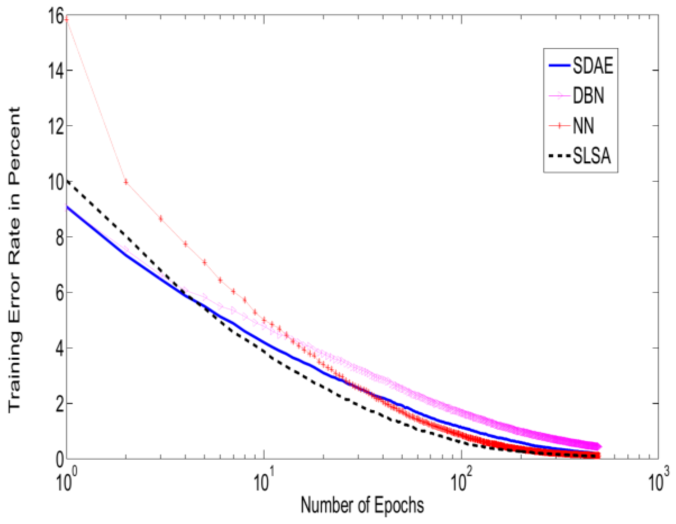

Fig. 7. Plot of convergence rates of different methods for training errors.

TABLE II. CATEGORIES AND NUMBERS OF DOCUMENTS IN REUTERS.

acq	corn	crude	earn	grain	interest	money	ship	trade	wheat
2369	238	578	3964	582	478	717	286	486	283

Vector Machines (SVM) and Naïve Bayes classifier (NB), which have been proven to perform very well on text classification tasks.

The Reuters-21578 collection contains 21578 text documents. We have used the most populous 10 out of the 135 topic categories, namely acq, corn, crude, earn, grain, interest, money-fx, ship, trade, and wheat. Table II provides the number of documents in each of the ten categories. In our experiments, we build binary classification tasks using one-vs-others fashion, where each category is used as the positive class, and the rest of the 9 categories as the negative class, which gives us 10 datasets. In addition, we have employed 5-fold cross validation to evaluate the F-measure, which is the commonly used evaluation metric in text classification, of various techniques. The results for NB and SVM are obtained from [29].

The detailed results are shown in Table III. We observe that deep architectures, including DBN, SDAE, SLSA, as well as NN, perform significantly better than SVM and NB for most Reuters data sets where SVM and NB need more training data (like in the categories acq and corn) to build more accurate classification models. Our proposed SLSA is able to achieve consistently better results across all the Reuters data sets.

V. CONCLUSION

In this paper, we have shown that the SLSA is better than the other pre-training techniques used in SDAE or DBN in finding a good guess of the initial set of weights, which were used for the subsequent stage of localized optimization technique. This was supported by the low classification error rate on the MNIST dataset and the consistently best F-measure scores across all the Reuters data sets. In addition, the training and test errors converge faster than other state-of-the-arts, as shown in the experiments. As such, this paper establishes the value of the pre-training phase, in particular using the SLSA, whose novelty stems from the stochastic least squares to generate a weight update. Furthermore, the introduction of the simple numerical trick in the logistic sampling process and an additional parameter, quantile, help to provide more robustness to the SLSA architecture. Using the general framework of stochastic least squares approach and the numerical trick to constraint the sigmoid function, further work could include using a different activation function such as the Gaussian to train different variation of the SLSA architecture to achieve faster convergence and more accurate results.

TABLE III. THE EXPERIMENTAL RESULTS FOR REUTERS DATA SETS.

	SVM	NB	DBN	SDAE	**SLSA**
acq	95.0	91.8	97.0	96.7	**97.5**
corn	8.5	38.1	96.8	96.7	**97.4**
crude	75.2	80.5	96.8	96.7	**97.4**
earn	97.7	95.0	96.8	96.7	**97.4**
grain	22.5	61.2	96.8	96.6	**97.5**
interest	59.5	61.5	96.7	96.7	**97.4**
money-fx	66.1	72.8	96.8	96.7	**97.4**
ship	46.5	68.8	96.9	96.8	**97.5**
trade	74.6	74.7	96.7	96.6	**97.4**
wheat	12.6	41.5	96.8	96.8	**97.5**

REFERENCES

[1] B. Yegnanarayana, *Artificial Neural Networks*. PHI Learning Pvt. Ltd., 2004.

[2] L. Arnold, S. Rebecchi, S. Chevallier, and H. Paugam-Moisy, "An introduction to deep learning," *ESANN*, 2011.

[3] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, 2006.

[4] G. E. Hinton, "A practical guide to training restricted boltzmann machines," *Momentum*, 2010.

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, 2006.

[6] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in Neural Information Processing Systems*, 2007.

[7] E. Izhikevich, "Simple model of spiking neurons," *Neural Networks, IEEE Transactions on*, 2003.

[8] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends in Machine Learning*, November 2009.

[9] R. Salakhutdinov and G. E. Hinton, "Deep boltzmann machines," in *International Conference on Artificial Intelligence and Statistics*, 2009.

[10] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, 2002.

[11] O. Woodford, "Notes on constrastive divergence," Department of Engineering Science, University of Oxford, Tech. Rep.

[12] M. A. Carreira-Perpinan and G. E. Hinton, "On constrastive divergence learning," *Artificial Intelligence and Statistics*, 2005.

[13] C. Sammut and G. Webb, *Encyclopedia of Machine Learning*. Springer, 2010.

[14] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, 1933.

[15] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *JMLR*, December 2010.

[16] G. E. Hinton, "Lecture 15: From pca to autoencoders," http://class.coursera.org/neuralnets-2012-001/lecture/index/, September 2013.

[17] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep machine learning - a new frontier in artificial intelligence research," *IEEE Computational Intelligence Magazine*, November 2010.

[18] Z. Zhang, G. Wang, D.-Y. Yeung, and J. T. Kwok, “Probabilistic kernel principal component analysis,” Department of Compputer Science, The Hong Kong University of Science and Technology, Tech. Rep.

[19] M. Alvarez and R. Henao, “Probabilistic kernel principal component analysis through time,” *Neual Information Processing, Springer Berlin Heidelberg*, 2006.

[20] L. Neil, “Probabilistic non-linear principal component analysis with gaussian process latent variable models,” *JMLR 6*, 2005.

[21] H. Hoffmann, “Unsupervised learning of visuomotor associations,” Universitat Bielefeld, Technische Fakultat, Tech. Rep., 2005.

[22] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, August 2013.

[23] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *25th ICML*, 2008.

[24] H. Boulard and Y. Kamp, “Auto-association by multi-layer perceptrons and singular value decomposition,” *Biological Cybernetics*, 1988.

[25] A. Arthur and A. Albert, *Regression and the Moore-Penrose pseudoinverse*. New York Academic Press, 1972.

[26] G. G. H. and C. Reinsch, *Singular value decomposition and least squares solutions*. Numerische Mathematik, 1970.

[27] R. B. Palm, “Prediction as a candidate for learning deep hierarchical models of data,” Technical University of Denmark, Palm, Tech. Rep., 2012.

[28] G. E. Hinton, N. Krivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv preprint arXiv:1207.0580, 2012.

[29] B. Liu, W. Lee, P. Yu, and X. Li, “Partially supervised classification of text documents,” *ICML Vol. 2.*, 2002.