STM JOURNALS

Journal of
Image Processing & Pattern Recognition Progress

ISSN: 2394-1995
Volume 12, Issue 3, 2025
September–December
DOI (Journal): 10.37591/JoIPPRP

https://journals.stmjournals.com/joipprp

Review JoIPPRP

Accelerating Unsupervised Feature Learning: Parallelized Training of Denoising Autoencoders

Nagajayant Nagamani*

Abstract

Unsupervised representation learning has become a cornerstone of contemporary machine learning, enabling algorithms to extract informative features from un-labelled, high-dimensional data. This work investigates the efficacy of stacked denoising autoencoders (SDAEs) trained via parallelized stochastic gradient descent (SGD) as a scalable approach to feature extraction. By strategically leveraging multi-threaded computation, our study systematically examines the trade-offs between increased parallelism, training efficiency, and the preservation of model accuracy. Experiments on the MNIST digit classification benchmark demonstrate that parallelization of the autoencoder training process significantly reduces convergence time, particularly as the dimensionality of the hidden layers grows. Speedup due to multi-threading, while substantial, was occasionally sublinear due to inherent sequential dependencies in the backpropagation algorithm and resource contention during memory access. Moreover, visualizations of learned filters illustrate the model's ability to capture meaningful patterns in the input space, while reconstructions from noisy inputs underline the robustness of the denoising criterion. Classification using stacked representations achieves competitive accuracy compared to state-of-the-art supervised models such as SVMs, highlighting the practical utility of unsupervised pretraining. Beyond SGD, the study discusses the promise of evolutionary optimization algorithms, in particular, genetic algorithms, as highly parallelizable and potentially more robust alternatives for training deep architectures on challenging noisy datasets. Overall, the study emphasizes scalable training techniques for deep unsupervised models and outlines future directions for enhancing training speed, generalizability, and resilience to complex data variations. These findings contribute to the broader field of deep learning by showcasing pathways for improving both the computational efficiency and the qualitative performance of neural network-based feature learning systems.

Keywords: Autoencoders, machine learning, SVM, Supervised learning, algorithm

INTRODUCTION

Autoencoders are widely used for unsupervised representation learning, enabling neural networks to discover compact and meaningful ways of encoding input data. Effective data representation plays a vital role in the success of machine learning algorithms, especially in tasks involving high dimensional inputs. Poor feature representation can lead to overfitting and poor generalization unless a substantial amount of labeled training data is available [1]. To address this, unsupervised pretraining using models like autoencoders can serve as a powerful strategy for transforming raw inputs into representations that simplify downstream supervised learning. An autoencoder consists of a neural network architecture where the

***Author for Correspondence**
Nagajayant Nagamani
E-mail: nagajayant@live.com

Software Engagement, Cognizant, Chennai, Tamil Nadu, India

Received Date: August 11, 2025
Accepted Date: August 19, 2025
Published Date: August 28, 2025

Citation: Nagajayant Nagamani. Accelerating Unsupervised Feature Learning: Parallelized Training of Denoising Autoencoders. Journal of Image Processing & Pattern Recognition Progress. 2025; 12(3): 56–64p.

input and output layers are of the same dimension, and a bottleneck is introduced via a hidden layer with fewer neurons. Let the input vector be $x \in \mathbb{R}^m$ and the hidden layer have n neurons. The encoding operation is defined using a weight matrix $W \in \mathbb{R}^{m \times n}$ and a bias vector $b \in \mathbb{R}^n$, while the decoding operation employs the transpose of the same weight matrix (to reduce trainable parameters) and a separate bias vector $b' \in \mathbb{R}^m$. Employing the sigmoid activation function $s(x)=1/(1+e^{-x})$, the encoding step computes $y=s(Wx+b)$, and the decoding step reconstructs the input as $z=s(W^T y+b')$.

The training objective is to minimize the difference between the original input t and its reconstruction z using a loss function. In this work, we adopt the mean squared error: $E(t, z)=\frac{1}{2}||t-z||^2_2$. Once trained, the decoder portion is discarded, and the encoder (or "autoencoder layer") is used to transform raw input data into a new representation for subsequent machine learning tasks.

Typically, autoencoders enforce $n<m$ to prevent the network from simply learning to replicate the input through identity mapping. However, a variant known as the *denoising autoencoder* can learn informative representations even when $n>m$ [2]. This model operates by introducing noise to the input and training the network to reconstruct the original, clean version. This denoising objective encourages the model to capture the underlying structure of the data rather than just memorizing inputs, making it effective for learning robust and generalizable features.

In this study, we focus on training denoising autoencoders using stochastic gradient descent (SGD). A review of related work is provided in the following section, while the algorithmic details of SGD applied to autoencoder training are outlined in the methodology section. Experimental results are presented using a dataset of handwritten digit images, and the final section discusses potential improvements and future directions for training procedures (Figure 1).

RELATED WORK

The concept of learning internal data representations without explicit supervision has been foundational in the advancement of deep learning. One of the most comprehensive reviews on this subject is provided by Bengio *et al*., who analyze the role of representation learning in machine learning systems [1]. Their work discusses how both shallow and deep models can be used to uncover useful feature hierarchies, with particular attention to autoencoders and their variants. They argue that better representations can significantly reduce the burden on classifiers, especially in high-dimensional settings.

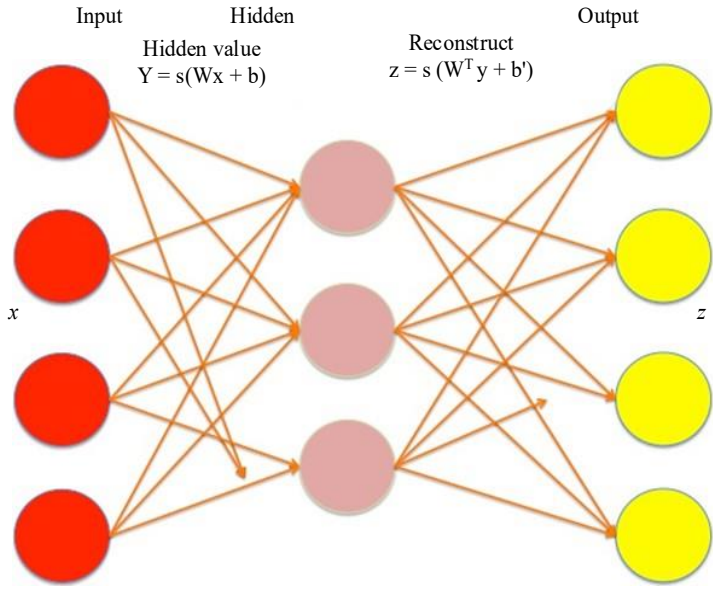

Figure 1. Autoencoder overview showing encoding and decoding with transposed weight matrices.

The development of autoencoders as a core component of deep learning frameworks has seen significant contributions over the years. Vincent *et al*. expanded on the idea of stacked autoencoders and introduced the denoising autoencoder as a robust alternative to traditional unsupervised pretraining techniques [2]. By injecting noise into the inputs and training the network to recover the original signal, they showed that it is possible to guide the learning process toward representations that generalize well, even in overcomplete settings where the hidden layer has more neurons than the input layer.

The challenge of training deep networks with multiple nonlinear layers was notably addressed by Hinton *et al.*, who proposed a greedy layer-wise pretraining strategy [3]. This approach, which is also relevant to our use of stacked denoising autoencoders, demonstrated how unsupervised training of each layer could initialize parameters in a way that facilitates successful fine-tuning with supervised learning. This method marked a turning point in making deep learning practically viable, especially before the advent of large-scale labeled datasets and powerful GPU hardware. The optimization of neural network parameters has traditionally relied on variants of stochastic gradient descent (SGD). Foundational work on the theoretical and practical aspects of backpropagation, a core component of SGD in neural networks, can be found in the works of Hecht-Nielsen and Bottou [4, 5]. These studies laid the groundwork for efficient gradient-based learning, showing how to compute derivatives of loss functions with respect to model parameters using the chain rule.

In addition to classical gradient-based methods, our future research aims to explore evolutionary algorithms, particularly genetic algorithms (GAs), as an alternative training mechanism for autoencoders. GAs offer a population-based search strategy that is well-suited for optimizing complex, high-dimensional spaces where gradients may be unreliable or difficult to compute. Srinivas and Patnaik provide a comprehensive survey of genetic algorithms and their performance on various optimization tasks [6]. Moreover, Cantu-Paz's work on parallel implementations of GAs offers important insights into how such algorithms can be scaled to train neural networks more efficiently in distributed environments [7]. This direction holds promise for reducing training time and exploring broader architectural search spaces in future versions of our model. Collectively, these works provide the theoretical and empirical foundation upon which our current research on denoising autoencoders and alternative optimization methods is built. By integrating insights from representation learning, deep network training, and evolutionary computation, we aim to enhance the robustness, scalability, and training efficiency of unsupervised learning systems.

ALGORITHM DESCRIPTION

To begin training, we initialize the weight matrices W and the associated biases b and b' randomly. Each input x is then passed through the network to produce an output z, which is compared to a target value t. In autoencoders, the goal is to reconstruct the input itself, hence $t=x$. The error between z and t is typically measured using the squared loss function $E(t, z)=½||t-z||^2_2$. If the reconstruction error exceeds a threshold, the weights and biases are updated to improve the network's performance. This adjustment is commonly done using the backpropagation algorithm [4], which, when applied on individual training samples, becomes stochastic gradient descent (SGD) (Figure 2). To derive the update rules, consider a three-layer feedforward network consisting of input, hidden, and output layers. The notations used are summarized in Table 1.

Weight Update from Hidden to Output Layer

For each output neuron, the gradient of the error with respect to the output layer weights W_{ij}^{O} is computed as follows:

$$\frac{\partial E}{\partial w_{ij}^{O}} = \frac{\partial E}{\partial z_j} \cdot \frac{\partial z_j}{\partial n_j} \cdot \frac{\partial n_j}{\partial w_{ij}^{O}}$$

$$= \left(z_j - t_j\right) s\left(n_j\right)\left(1 - s\left(n_j\right)\right) \cdot x_i$$

$$= \left(z_j - t_j\right) \cdot z_i\left(1 - z_j\right) \cdot x_i \quad (1)$$

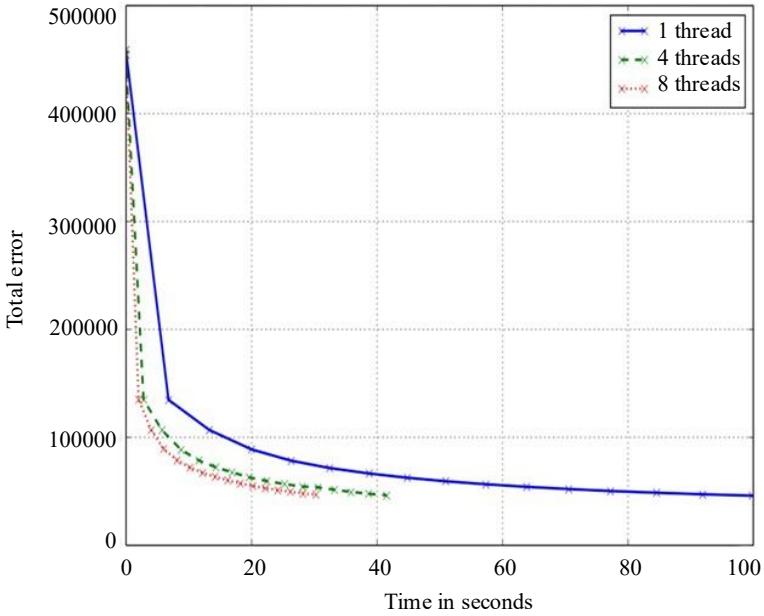

Figure 2. Time vs. error for 500-node autoencoder (15 iters, 1/4/8 threads, 5,000 images).

Table 1. Notation used in derivation of update rules.

Symbol	Description
E	Error computed at the output
x_j	Input node j
y_j	Hidden layer node j
z_j	Output node j
n_j	Pre-activation at node$j$: $\sum_{i=1}^{n} W_{ij}x_i + b_j$
t_j	Target output at node j
W_{ij}^H	Weight from input to hidden layer
$W_{ij}O$	Weight from hidden to output layer
$s(x_j)$	Sigmoid activation: $1/(1+e^{-xj})$
$b_j^{\{H,O\}}$	Biases for hidden/output layer respectively

Using this gradient, the parameters are updated as:

$$\delta_j^O=(z_j-t_j)z_j(1-z_j)$$
$$W_{ijO} \leftarrow W_{ijO} - \eta\delta_{jOxi} \quad (2)$$
$$b_{Oj} \leftarrow b_{Oj} - \eta\delta_{jO}$$

Weight Update from Input to Hidden Layer

Similarly, the gradient of the error with respect to the input to-hidden weights W_{ij}^H is given by:

$$\frac{\partial E}{\partial w_{ij}} = \left(\sum_{k=1}^{m}(z_k - t_k)(1 - z_k)z_k w_{jk}^0\right) y_j(1 - y_j)x_i \quad (3)$$

And the update rules are:

$$m \qquad !y_j(1-y_j)$$
$$\delta_{jH}=\mathrm{X}(z_k-t_k)(1-z_k)z_kW_{jkO}$$
$$k=1$$
$$W_{ijH} \leftarrow W_{ijH} - \eta\delta_{jHxi}\ b_{Hj} \leftarrow b_{Hj} - \eta\delta_{jH} \quad (4)$$

Alternative Loss Function: Cross-Entropy with Softmax

In cases where softmax activation is used at the output and the error is measured via cross-entropy, the update rules simplify due to the derivative of the softmax-cross entropy combination. The loss function is defined as:

$$E(x,t) = -\sum_{i=1}^{n}[t_i \, ln \, z_i + (1 - t_i) \, ln(l - z_i)] \tag{5}$$

with softmax given by:

$$\sigma(x_j) = \frac{e^{x_j}}{\Sigma_k e^{x_k}} \tag{6}$$

In this case, the gradients and updates for the output layer become:

$$\frac{\partial E}{\partial W_{ij}^O} = (z_j - t_j) y_i$$

$$\delta_j^O = (z_J - t_i)$$

$$W_{ij}^O \Leftarrow W_{ij}^O - \eta \delta_j^O x_i$$

$$b_j^O \Leftarrow b_j^O - \eta^{\delta_j^O} \tag{7}$$

The updates for the hidden layer remain unchanged from the sigmoid activation case.

Backpropagation Overview

Algorithm 1 outlines the overall procedure of backpropagation during training.

Algorithm 1 Backpropagation Algorithm

Initialize weights and biases randomly. For each training iteration, do for all training samples x (randomized), do Perform forward pass to compute z. Compute output error δ_j^O.

Update W_{ij}^O and $b^O{}_j$.

for all hidden layers in reverse do Compute hidden error δ_j^H.

Update W_{ij}^H and b_j^H.

end for end for
end for

Autoencoder-Specific Considerations

In autoencoders, a common simplification involves sharing weights between the encoder and decoder: the decoding weights are often the transpose of the encoding weights, i.e., $W^O=(W^H)^T$. This allows us to write the reconstruction as $z=s(W^O(W^Hx+b)+b')$. To optimize memory, we avoid storing W^O explicitly and instead use transposed access of W^H during decoding [8].

Parallelization in training is constrained by the sequential nature of the backpropagation steps. However, key operations such as matrix-vector multiplications, computation of δ terms, and gradient-based weight updates can be parallelized for efficiency. Finally, once an initial autoencoder layer is trained, its encoded output can serve as input for a subsequent autoencoder, allowing for the construction of a stacked autoencoder architecture that further refines the learned representation. Figure 3 shows the comparison between iteration time, threads and hidden nodes.

EXPERIMENTAL RESULTS

In our experimental setup, we train an autoencoder on the MNIST dataset of handwritten digits. This dataset comprises 60,000 training samples and 10,000 test samples, with each image being a 28×28 grayscale image labeled with a digit from 0 to 9.

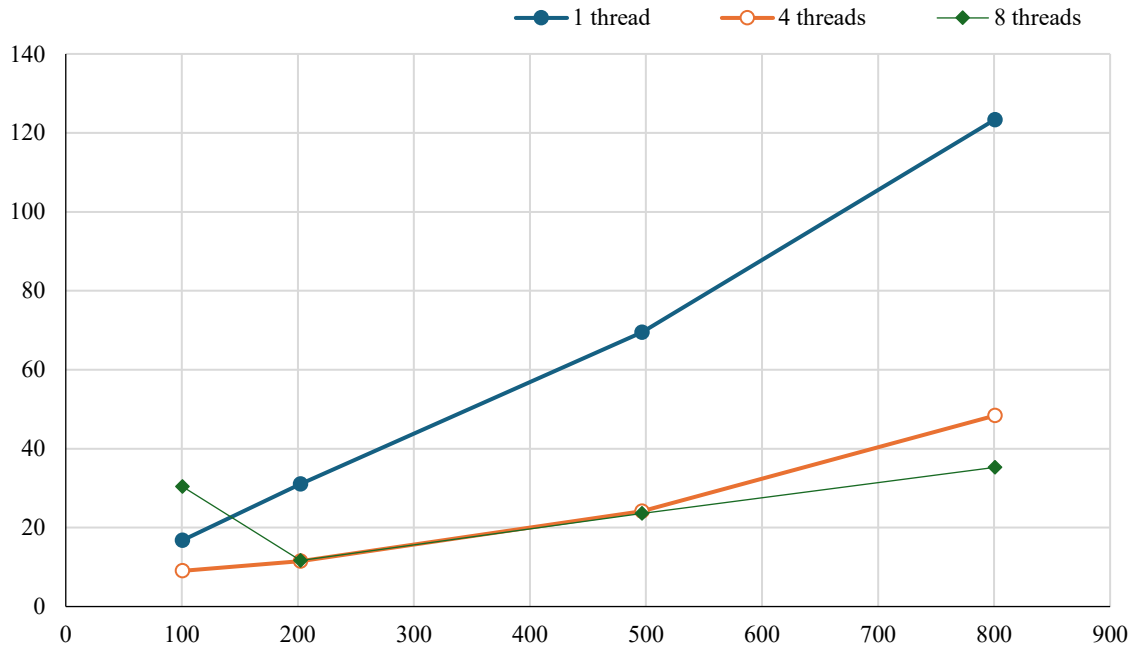

Figure 3. Iteration time vs. threads and hidden nodes (60,000 images).

As a result, each input image is flattened into a 784-dimensional vector. To enhance the model's robustness, we apply the denoising technique proposed by Vincent *et al.*, where each pixel in a training image is independently set to zero with a 25% probability [2].

We first investigate the impact of multi-threading on the rate of error reduction during training. Specifically, we train a single-layer autoencoder with 500 hidden units for 15 full passes over a subset of 5,000 training samples. One training iteration involves processing all 5,000 samples using stochastic gradient descent (SGD) to update the weights. Figure 2 plots training error against elapsed time across different thread counts. Across all thread settings, error reduction is steep in the initial iterations and stabilizes around a similar value after 15 iterations. However, the use of 4 or 8 threads significantly accelerates convergence compared to single-threaded execution. This speedup is not linear due to two key limitations: (1) cache conflicts caused by concurrent reads and writes to different parts of the weight matrix, and (2) the inherently sequential nature of certain steps in backpropagation and SGD (outlined in Algorithm 1), which limits parallelism and introduces overhead.

Figure 3 presents the average time per training iteration (across all 60,000 images) as a function of thread count and the number of hidden units. With a single thread, runtime increases linearly with the number of hidden units. However, with 4 or 8 threads, the gains become sublinear due to overheads from synchronization and parallel operations. Notably, when using 8 threads with only 100 hidden units, training takes longer than single-threaded execution. Only when the hidden unit count reaches 800 does using 8-threads outperform the 4-thread configuration.

Figure 4 illustrates the filters learned after training an autoencoder with 500 hidden units on the complete set of 60,000 training samples. Each hidden unit's filter, represented as a row of the weight matrix, captures the specific input features that the neuron responds to. Since each filter has the same length as the input vector, we reshape and visualize them as 28×28 pixel images. While these filters do not exactly replicate the original digits, they do reflect patterns present in the input data. Additionally, the same figure shows reconstructed outputs from noisy test digits. Most reconstructions are clearly recognizable, indicating effective denoising and meaningful feature extraction by the autoencoder.

Figure 4. Filters of the first 100 hidden units from the trained denoising autoencoder.

Finally, we evaluate a deep neural network consisting of three stacked denoising autoencoders. Each layer has 1,000 hidden units and is trained with corruption levels of 0.1, 0.2, and 0.3 respectively. Each layer undergoes 15 training iterations with a learning rate of 0.001. After unsupervised pretraining, we connect a feedforward neural network on top, consisting of 1,000 input units, 500 hidden units, and 10 output units. This network is then trained in a supervised manner using the indicator vector of the target labels, for 30 iterations with a learning rate of 0.1. The final classification accuracy achieved is 98.04%. For comparison, a Support Vector Machine (SVM) with an RBF kernel, as reported by Vincent *et al*. [2], achieves a slightly higher accuracy of 98.60%.

Algorithm 2: Genetic Algorithm

Initialize a population of N individuals at random. For each generation, do Evaluate each individual using the objective function and assign a fitness score.
Generate αN new individuals ($0<\alpha<1$) by selecting high-fitness individuals and applying crossover and mutation.
Replace the αN least fit individuals with the new offspring.
End for.

FUTURE WORK

In future research, we intend to explore the use of Genetic Algorithms (GAs) for optimizing the weight matrices and bias terms of autoencoders. GAs are population-based metaheuristic optimization techniques that operate without gradient information. Inspired by natural selection, they employ operations such as mutation and crossover to iteratively improve candidate solutions until a near-optimal solution to the objective function is found [9].

Figure 5. Denoising autoencoder reconstructions: noisy inputs (odd) and outputs (even).

Compared to Stochastic Gradient Descent (SGD), GAs offer two primary advantages in the context of training autoencoders. First, SGD often converges to local minima due to the non-convex nature of the loss landscape, whereas GAs are better suited for escaping such traps and can potentially identify global optima. Second, parallelization of SGD presents certain challenges, as demonstrated in our experiments where linear speedup was not achieved. Conversely, GAs naturally lend themselves to parallelization in two ways [7]: (a) evaluation of individuals in the population can be distributed across processors, and (b) mutation and crossover operations are element-wise and can be parallelized easily (Figure 5).

Algorithm 2 outlines the basic structure of a Genetic Algorithm. In our context, each individual in the population represents a set of weights and biases. The objective function is the same as the reconstruction loss described earlier, and an individual's fitness corresponds to how well it minimizes this loss.

Furthermore, we plan to assess the generalizability of our autoencoder architecture on more challenging image datasets by Vincent *et al*. [2], such as *bg-rand* and *bg-img-rot*, which introduce variations like background noise and image rotations [10].

CONCLUSION

In this work, we developed and evaluated a deep learning framework using stacked denoising autoencoders (SDAEs) to learn robust feature representations from corrupted input data. Our approach was tested on the MNIST dataset, and we showed that the autoencoder was capable of learning meaningful internal representations that could effectively denoise and reconstruct handwritten digits. The reconstructions were qualitatively accurate, and the filters learned by the hidden layers demonstrated a clear understanding of the underlying data structure.

Furthermore, the performance of our stacked architecture was benchmarked on a classification task, where the final accuracy reached 98.04%, which is competitive with state of-the-art methods like Support Vector Machines with RBF kernels. This highlights the effectiveness of SDAEs not just for unsupervised feature learning, but also as a pretraining mechanism for supervised learning tasks.

Our experiments also examined the computational aspects of training, particularly the impact of multi-threading on the convergence rate and iteration time of stochastic gradient descent (SGD). Although parallelization improved training speed, the speedup was sublinear due to inherent overheads and sequential components of the backpropagation algorithm. This limitation points to the need for more parallel-friendly optimization strategies.

Given these findings, one key direction for future exploration is the integration of evolutionary algorithms, such as Genetic Algorithms (GAs), as an alternative to SGD for training deep models. GAs offer potential advantages, including global search capabilities and a high degree of parallelizability, making them well-suited for distributed or high-performance computing environments. In addition, their gradient-free nature can help overcome issues with non-convex loss surfaces, which are common in deep learning models.

We also recognize the need to validate the generalizability and robustness of our model on more complex and noisy datasets. The MNIST dataset, while widely used, is relatively clean and may not reflect real-world data complexities. Thus, future experiments will include datasets with more challenging variations, such as those with background clutter, transformations, and occlusions.

In conclusion, this study affirms the viability of stacked denoising autoencoders for both representation learning and classification. It also opens promising avenues for optimization and scalability improvement.

REFERENCES

1. Bengio Y, Courville A, Vincent P. Representation learning: A review and new perspectives. IEEE Trans Pattern Anal Mach Intell. 2013 Mar 7; 35(8): 1798–828.
2. Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol PA, Bottou L. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J Mach Learn Res. 2010 Dec 1; 11(12): 3371–3408.
3. Hinton GE, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. Neural Comput. 2006 Jul; 18(7): 1527–54.
4. Hecht-Nielsen R. Theory of the backpropagation neural network. In: Neural networks for perception. Academic Press; Massachusetts. 1992 Jan 1; 65–93.
5. Bottou L. Stochastic gradient learning in neural networks. Proceedings of Neuro-Nımes. 1991 Nov; 91(8): 12.
6. Srinivas M, Patnaik LM. Genetic algorithms: A survey. Computer. 2002 Aug 6; 27(6): 17–26.
7. Cantú-Paz E. A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis. 1998 May; 10(2): 141–71.
8. Cantu-Paz E. Efficient and accurate parallel genetic algorithms. New York, NY: Springer Science & Business Media; 2000 Nov 30.
9. Amari SI. Backpropagation and stochastic gradient descent method. Neurocomputing. 1993 Jun 1; 5(4–5): 185–96.
10. Ranzato MA, Krizhevsky A, Hinton G. Factored 3-way restricted Boltzmann machines for modeling natural images. In Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings. 2010 Mar 31; 621–628.

