Training Deep Neural Networks with Gradual Deconvexification

James Ting-Ho Lo Yichuan Gui Yun Peng

***Abstract*—A new method of training deep neural networks including the convolutional network is proposed. The method deconvexifies the normalized risk-averting error (NRAE) gradually and switches to the risk-averting error (RAE) whenever RAE is computationally manageable. The method creates tunnels between the depressed regions around saddle points, tilts the plateaus, and eliminates nonglobal local minima. Numerical experiments show the effectiveness of gradual deconvexification as compared with unsupervised pretraining. After the minimization process, a statistical pruning method is used to enhance the generalization capability of the neural network under training. Numerical results show further reduction of the testing criterion.**

I. Introduction

In this section, we examine the issues involved in training neural networks and motivate a training method based on convexification of the sum of squared errors (SSE) [1–8]. This training method, which is an improved version of that proposed in [5–7], is suited for training all deep learning machines (DLMs) including convolutional neural networks (CNNs) [9] and recurrent neural networks with any feedback structures. We also discuss briefly how convexification can help overcome the saddle points, plateaus and nonglobal local minima on the surface of the SSE.

Training neural networks such as the multilayer perceptron and its variants (e.g., the CNN and other DLMs) is mainly to minimize a training criterion, such as the SSE and cross entropy (CE), constructed with a training dataset. The training criterion is usually nonconvex and has a large number of nonglobal local minima and plateaus. Commonly used minimization methods include the gradient descent, quasi-Newton and conjugate gradient method. Such a method depends on the first- and/or second-order derivative of the training criterion in each iteration and hence slows down on a plateau or saddle point and has difficulty getting out of a nonglobal local minimum.

In addition to the minimization of the training criterion, training a neural network has another objective, namely the maximization of the generalization capability of the neural network. The generalization capability is usually measured by a testing criterion constructed with a test dataset that mimics the training criterion. In the process of minimizing the training criterion, the generalization capability is usually monitored with a cross-validation criterion constructed with a cross-validation dataset. Whenever the cross-validation criterion starts to increase steadily, the minimization process is terminated.

Cross-validation is used to prevent training process from reducing the generalization capability of the neural network. It does not improve the approximation of the underlying function by the neural network. Another way to enhance the generalization capability is pruning or drop-out [10] of neural network weights. After pruning or drop-out, the neural network has less weights, retraining the neural network with the same training method cannot reduce the training criterion or the deviations of the neural network outputs from the corresponding values of the underlying function.

Therefore, the notion that since the training process is to be stopped by cross-validation, it does not matter whether the minimization method used has the ability to avoid nonglobal local minima or whether a global or near global minimum can be found is mistaken. Briefly speaking, failure in minimization does not guarantee success in generalization. In our opinion, generalization capability, which reflects mostly the interpolation and some extrapolation accuracy, should be on the basis of a training method that is able to reach a global or near global minimum. For example, pruning a neural network increases its training criteria. A neural network with a smaller training criterion after the termination of the training process allows more room for pruning and thus for enhancing the generalization capability of the neural network.

Generally speaking, the more approximating resources (e.g., layers and weights) the neural network has, the smaller the values of the training criterion are at local minima. Since the training criterion is bounded from below by 0, the values of most local minima may be close to that of the global local minima. This is true whether the architecture (i.e., approximating resources) of the neural network required for fitting a given training dataset is large or small. However, because of the structural symmetry of the neural networks, nonglobal local minima whose values are not close to that of the global minima exist in large numbers. The chance for a minimization method intended for convex criteria to get trapped in such a nonglobal local minimum is not zero.

Equally important, the more approximating resources the neural network has, the more overfitting errors, especially the misinterpolation error, the neural network has after training. Unless a large number of weights are pruned or drop out, and the cross-validation and testing datasets are large enough to sufficiently reflect all the scenarios in the application of the neural network, the overfitting errors may show up in the application. Therefore, in the early development of neural

James Ting-Ho Lo is with the Department of Mathematics and Statistics of the University of Maryland, Baltimore County (email: jameslo@umbc.edu).

Yichuan Gui and Yun Peng are with the Department of Computer Science and Electrical Engineering of the University of Maryland, Baltimore County (email: {yichgui1, ypeng}@umbc.edu).

The work was supported in part by the U.S.A. National Science Foundation under Grant ECCS1028048 and Grant ECCS1508880, but does not necessarily reflect the position or policy of the U.S.A. Government.

networks, neural networks used are kept as small as feasible even with regularization terms included in the training error criterion. Stimulating results on the surface of the training criterion, which are related to our discussions above, can be found in [11–13].

The purpose of this paper is to develop the gradual deconvexification (GDC) method further [5–7] to avoid plateaus and nonglobal local minima more effectively, speed up minimization process, require no multiple training sessions with different sets of random initial weights, and enhance generalization.

The unsupervised pretraining followed by fine tuning for training DLMs represents the state of the art in training neural networks [10, 14–22]. A large number of more recent articles on the unsupervised pretraining and DLMs can be found on the web. Numerical experiments were performed to compare our new version of GDC to the method of unsupervised pretraining followed by fine tuning on well-known benchmark examples. The numerical results show the deep neural networks (including the CNN) obtained by training with the new GDC, which is stopped by cross-validation and then followed by statistical pruning, are better than those trained with the method of unsupervised discriminative pretraining followed by fine tuning, but are not as good as the deep belief network or deep Boltzmann machine. The differences are small, but significant, showing room for GDC to be further developed.

Because of GDC's consistent performances among different training sessions with different initialization seeds, wide applicability (e.g., to convolutional and recurrent neural networks), conceptual simplicity and mathematical justification, and possible use jointly with other methods such as the discriminative unsupervised pretraining, further development of GDC is at the top of the list of our future work.

II. Preliminary Results

In this section, we review the theory of the risk-averting error (RAE) $J_\lambda(w)$ and the normalized risk-averting error (NRAE) $C_\lambda(w)$ criteria. The former was proposed to convexify the sum of squared errors (SSE) [3]. The latter was used to overcome computer overflow in computing the RAE and its derivatives [4, 5, 8].

A. Risk-Averting Error Criterion

A standard formulation of training a multilayer perceptron (MLP) under supervision follows: given a set of input/output pairs (x_k, y_k), $k = 1, ..., K$, which are assumed to satisfy the equation $y_k = f(x_k) + \xi_k$ where f is an unknown or known function and ξ_k are random noises or zero, finding an MLP $y = \hat{f}(x, w)$ such that the SSE criterion

$$Q(w) := \sum_{k=1}^{K} \left\| y_k - \hat{f}(x_k, w) \right\|^2 \tag{1}$$

is minimized with the variation of the weight vector w, where $\|\cdot\|$ denotes the Euclidean norm and the weights $w \in R^N$.

The RAE criterion [3]

$$J_\lambda(w) := \sum_{k=1}^{K} e^{\left(\lambda \left\| y_k - \hat{f}(x_k, w) \right\|^2\right)} \tag{2}$$

was motivated by emphasizing the larger individual deviations $\left\| y_k - \hat{f}(x_k, w) \right\|$ of the SSE criterion $Q(w)$ in an exponential manner, thereby avoiding such large individual deviations and achieving robust performance. It turned out that under a regularity condition on $\hat{f}(x, w)$, the convexity region of $J_\lambda(w)$ expands monotonically as λ increases. This is stated in the following theorem [3]:

Theorem 1 (RAE Convexity)**.** *Assume that the risk-averting error criterion $J_\lambda(w)$ in (2) is twice continuously differentiable, and that $N < K$.*

$$D_K(w) := \left[\frac{\partial \hat{f}(x_k, w)}{\partial w_{ki}} \right]_{K \times N} \tag{3}$$

is the Jacobian of $\hat{f}(x_k, w)$ and

$$H_\lambda(w) := \left[\frac{\partial^2 J_\lambda(w)}{\partial w_i \partial w_j} \right]_{N \times N} \tag{4}$$

is the Hessian of $J_\lambda(w)$. Denoting the set $\left\{ w \in R^N : \operatorname{rank} D_K(w) = N \right\}$ by Ω, the set $P_\lambda := \{ w \in \Omega \,|\, H_\lambda(w) > 0 \}$ expands monotonically as λ increases.

Remark. Let $C(K, N)$ denote the number of combinations of N objects taken K at a time. Note that the complement Ω^C of Ω is the intersection of the solution sets of $C(K, N)$ algebraic equations defined by setting the determinants of $C(K, N)$ submatrices of $D_K(w)$ equal to zero. As the number K of input/output pairs in the training data increases, the number $C(K, N)$ of solution sets increases rapidly, and the intersection Ω of these solution sets shrinks monotonically.

B. Normalized Risk-Averting Error Criterion

To make advantage of a large convexity region of $J_\lambda(w)$, λ must be large. At the beginning of training, the squared errors $\left\| y_k - \hat{f}(x_k, w) \right\|^2$ of the SSE $Q(w)$ are usually very large. Computing $J_\lambda(w)$ or its derivatives for a large λ or a large $\left\| y_k - \hat{f}(x_k, w) \right\|^2$ often causes computer overflow. To avoid it, the NRAE criterion [4]

$$C_\lambda(w) := \frac{1}{\lambda} \ln \left(\frac{1}{K} J_\lambda(w) \right) \tag{5}$$

must be used.

To simplify mathematical expressions, we use the following symbols:

$$\hat{y}_k(w) := \hat{f}(x_k, w) \tag{6}$$

$$\varepsilon_k(w) := y_k - \hat{y}_k(w) \tag{7}$$

and for each w,

$$S(w) = \underset{k \in \{1, ..., K\}}{\arg\max} \left\| \varepsilon_k(w) \right\|^2 \tag{8}$$

$S(w)$ may contain more than one element if a tie exists. Let

$$M = \min_m \{m \,|\, m \in S(w)\} \tag{9}$$

which is the smallest index among all values in the set $S(w)$. It follows that $\|\varepsilon_k(w)\|^2 \leq \|\varepsilon_M(w)\|^2$ for all $k \in \{1, ..., K\}$. Using the symbol,

$$\eta_k(w) := e^{\lambda\left(\|\varepsilon_k(w)\|^2 - \|\varepsilon_M(w)\|^2\right)} \tag{10}$$

where we note $0 < \eta_k(w) \leq 1$ and $0 < \sum_{k=1}^K \eta_k(w) \leq K$, the NRAE criterion $C_\lambda(w)$ in (5) can be rewritten as

$$C_\lambda(w) = \frac{1}{\lambda} \ln \left(\frac{1}{K} e^{\lambda \|\varepsilon_M(w)\|^2} \sum_{k=1}^K \eta_k(w) \right) \tag{11}$$

$$\leq \frac{1}{\lambda} \ln \frac{1}{K} + \|\varepsilon_M(w)\|^2 + \frac{1}{\lambda} \ln K \tag{12}$$

$$= \|\varepsilon_M(w)\|^2 \tag{13}$$

It indicates that $C_\lambda(w)$ is bounded by the term $\|\varepsilon_M(w)\|^2$, which is independent of λ, and no computer overflow occurs even for $\lambda >> 1$. Evaluation of the derivatives of $C_\lambda(w)$ without computer overflow is examined in the sequel.

Note that $C_\lambda(w)$ is a strictly increasing function $\frac{1}{\lambda} \ln\left(\frac{1}{K}(\cdot)\right)$ of $J_\lambda(w)$. Therefore, NRAE and RAE share the same nonglobal local and global optima. In other words, as the convexity region of RAE expands, NRAE contains less and less nonglobal local minima as well. For a neural network $\hat{f}(x, w)$ that is sufficiently large for $\min_w Q(w)$ to be close to 0 for some w, if a global or near-global minimum of NRAE and RAE is reached, it is also a near-global local minimum of $Q(w)$.

For a neural network $\hat{f}(x, w)$ that is not large enough for $\min_w Q(w)$ to be close to 0, if a global or near-global minimum of NRAE and RAE is reached, it may not be very close to a near-global local minimum of $Q(w)$. For minimizing $Q(w)$, we continue training with NRAE or RAE with gradually smaller and smaller λ until $\lambda = 0$. According to our experiences, this practice converges to $\min_w Q(w)$ all the time. This is consistent with $\lim_{\lambda \to 0} C_\lambda(w) = \frac{1}{K} Q(w)$, which is proven as follows:

$$\lim_{\lambda \to 0} C_\lambda(w) = \lim_{\lambda \to 0} \frac{1}{\lambda} \ln \left(1 + \frac{1}{K} \sum_{k=1}^K \lambda \|\varepsilon_k(w)\|^2 + O(\lambda^2) \right)$$

$$= \lim_{\lambda \to 0} \frac{1}{\lambda} \left(\frac{1}{K} \sum_{k=1}^K \lambda \|\varepsilon_k(w)\|^2 + O(\lambda^2) \right)$$

$$= \frac{1}{K} \sum_{k=1}^K \|\varepsilon_k(w)\|^2 \tag{14}$$

where the Taylor series expansions of exponential and logarithm functions are applied.

III. Gradual Deconvexification

Recall that under a rank condition on $\hat{f}(x, w)$, the convexity region of $J_\lambda(w)$ expands monotonically as λ increases. To make advantage of a larger convexity region of $J_\lambda(w)$ at a greater λ and avoid computer overflow, we are tempted to minimize $C_\lambda(w)$ at a λ as large as possible. However, at a very large λ, we encounter the following phenomena called stagnancy of training.

A. Stagnancy of Training at a Too Large λ

As the convexity region of $J_\lambda(w)$ expands as λ increases, it is tempting to start training with $C_\lambda(w)$ at an extremely large value of λ. Let us first see from (11),

$$\lim_{\lambda \to \infty} C_\lambda(w) = \lim_{\lambda \to \infty} \frac{1}{\lambda} \ln \left(\frac{1}{K} \sum_{k=1}^K \eta_k(w) \right) + \|\varepsilon_M(w)\|^2$$

$$= \|\varepsilon_M(w)\|^2 \tag{15}$$

Numerical experiments confirm or reveal the following phenomena:

1) Equation (15) shows that minimum of $C_\lambda(w) \approx \|\varepsilon_M(w)\|^2$ for $\lambda >> 1$ is virtually minimax of $Q(w)$. To minimize $\|\varepsilon_M(w)\|^2$, we use the entire MLP to approximate (x_k, y_k) where $k \in S(w)$, which are at most a small number of input/output pairs in the training dataset, which usually contains a relatively larger number K of input/output pairs. The architecture of the neural network selected for approximating the training dataset is therefore redundant for the approximation of (x_k, y_k). When all the weights are adjusted to achieve the approximation for (x_k, y_k), they tend to become "similar" or "duplicated", thus causing rank deficiency. Without a satisfied rank condition in the basic convexification theorem, it is not clear whether $J_\lambda(w)$ is convexified at an extremely large λ.
2) There are four possible cases in minimizing $C_\lambda(w) \approx \|\varepsilon_M(w)\|^2$ at an extremely large λ:
   a) $C_\lambda(w) \approx \|\varepsilon_M(w)\|^2$ is a plateau or nearly a plateau around the current w: $C_\lambda(w)$ stops decreasing or decreases extremely slowly. It may mistakenly be interpreted as reaching a local or global minimum of $C_\lambda(w)$.
   b) $C_\lambda(w) \approx \|\varepsilon_M(w)\|^2$ contains a local minimum near the current w and the training converges to it.
   c) $C_\lambda(w) \approx \|\varepsilon_M(w)\|^2$ continues to decrease, but some other $\|\varepsilon_k(w)\|^2$ increase: at one point, one of them replaces the current $\|\varepsilon_M(w)\|^2$ as a new $\|\varepsilon_M(w)\|^2$. This usually happens when the surface near the current w is not a plateau or nearly a plateau. Therefore, whether there is a local minimum or rank deficiency is irrelevant at an extremely large lambda.
   d) A group of different $\|\varepsilon_k(w)\|^2$ take turns to be $\|\varepsilon_M(w)\|^2$: if one $\|\varepsilon_M(w)\|^2$ decreases, then the other $\|\varepsilon_k(w)\|^2$ in the group increase. The index M cycles through the group, while such cycling is observed in our numerical experiments.

In the above phenomena, $C_\lambda(w)$ decreases very slowly or simply ceases to decrease. This is called stagnancy of training.

The smallest value of λ at which stagnancy of training with $C_\lambda(w)$ occurs depends on the application. It is unclear how to determine or even estimate it. Therefore, to make advantage of as large a convexity region of RAE (contained in NRAE) as possible, we start training with NRAE $C_\lambda(w)$ at a very large value of λ, say 10^9. To deal with the stagnancy of training, we reduce the value of λ by a preset percentage whenever $C_\lambda(w)$ does not decrease for a preset amount or percentage in a preset number of epochs. As discussed in Section II-B, if $\min_w C_\lambda(w) \approx \min_w Q(w) \approx 0$, we switch training with $C_\lambda(w)$ to training with $Q(w)$; else we continue GDC training with $C_\lambda(w)$ until $\lambda \approx 0$. Experimental results reported in [6, 7] confirm the effectiveness of GDC for preventing the stagnancy of training and avoiding the non-global local minimum in training MLPs in both batch and pairwise training.

As the convexity region of RAE (contained in NRAE) shrinks as λ decreases, this procedure is called GDC stage 1 [6]. In the sequel, GDC stage 2 will be proposed.

We examine a fast evaluation of the gradient of $C_\lambda(w)$ at a very large λ in the subsection below. With this fast evaluation, we may start with a very large λ without incurring too much computational cost.

B. Fast Evaluation of NRAE Gradient at a Large λ

Minimization of $C_\lambda(w)$ in GDC involves the evaluation of the first-order derivative of $C_\lambda(w)$:

$$\begin{aligned}\frac{\partial C_\lambda(w)}{\partial w_i} &= \frac{1}{\lambda J_\lambda(w)}\frac{\partial J_\lambda(w)}{\partial w_i} \\ &= \frac{1}{\lambda J_\lambda(w)}\left[-2\lambda\sum_{k=1}^{K} e^{\lambda\|\varepsilon_k(w)\|^2}\varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i}\right] \\ &= \frac{-2\sum_{k=1}^{K}\eta_k(w)\,\varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i}}{\sum_{k=1}^{K}\eta_k(w)} \end{aligned} \tag{16}$$

where $\|\varepsilon_k(w)\|^2 \le \|\varepsilon_M(w)\|^2$ and $0 < \eta_k(w) \le 1$. The evaluation of $\eta_k(w) = e^{\lambda\left(\|\varepsilon_k(w)\|^2 - \|\varepsilon_M(w)\|^2\right)}$ in (10) involves an exponential operation of $\varepsilon_k(w)$ and $\varepsilon_M(w)$, costing the most computational resources during the training.

Note that if $\|\varepsilon_k(w)\|^2 = \|\varepsilon_M(w)\|^2$, then $\eta_k(w) = 1$. Note also that if $\|\varepsilon_k(w)\|^2 < \|\varepsilon_M(w)\|^2$, then $\lim_{\lambda\to\infty} e^{\lambda\left(\|\varepsilon_k(w)\|^2 - \|\varepsilon_M(w)\|^2\right)} = 0$. Therefore, at a very large value of λ, evaluation of $\eta_k(w)$ and $\eta_k(w)\,\varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i}$ is simple: if $\|\varepsilon_k(w)\|^2 < \|\varepsilon_M(w)\|^2$, they are both equal to 0; if $\|\varepsilon_k(w)\|^2 = \|\varepsilon_M(w)\|^2$, set $\eta_k(w) = 1$ and $\eta_k(w)\,\varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i} = \varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i}$ in (16) as presenting in Algorithm 1.

C. Switching Minimization of $C_\lambda(w)$ to Minimization of $J_\lambda(w)$

The logarithm function $\frac{1}{\lambda}\ln\left(\frac{1}{K}J_\lambda(w)\right)$ in $C_\lambda(w)$ tends to neutralize the effects of the exponential functions in $J_\lambda(w)$, and because $\lim_{\lambda\to 0} C_\lambda(w) = \frac{1}{K}Q(w)$, the smaller λ is, the closer $C_\lambda(w)$ is to $Q(w)$. Consequently, the effects of $J_\lambda(w)$ in tilting plateaus and creating tunnels around saddle points on $Q(w)$ are weakened. Therefore, for faster minimization convergence and better training results, we switch minimization of $C_\lambda(w)$ to minimization of $J_\lambda(w)$ as soon as $J_\lambda(w)$ does not cause computer overflow. We include this switching in our GDC procedure. Minimization of $J_\lambda(w)$ is called stage 2 of GDC.

Algorithm 1 Fast NRAE Gradient Evaluation

Require: Initialize the NRAE training with selecting the weight vector w randomly;

for $k = 1$ to K **do**
  if $\|\varepsilon_k(w)\|^2 = \|\varepsilon_M(w)\|^2$ **then**
    Set $\eta_k(w) \leftarrow 1$;
    $T_k(w) \leftarrow \varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i}$;
  else
    $E_k(w) \leftarrow \lambda\left(\|\varepsilon_k(w)\|^2 - \|\varepsilon_M(w)\|^2\right)$;
    if $E_k(w) < \ln F_{\min}$ **then**
      Set $\eta_k(w) \leftarrow 0$ and $T_k(w) \leftarrow 0$;
    else
      $\eta_k(w) \leftarrow e^{E_k(w)}$;
      $T_k(w) \leftarrow \eta_k(w)\,\varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i}$;
    end if
  end if
  Set $k \leftarrow k + 1$;
end for
for $i = 1$ to N **do**
  $\frac{\partial C_\lambda(w)}{\partial w_i} \leftarrow \frac{-2\sum_{k=1}^{K} T_k(w)}{\sum_{k=1}^{K}\eta_k(w)}$;
end for
return $\nabla C_\lambda(w) \leftarrow \left[\frac{\partial C_\lambda(w)}{\partial w_i}\right]_{1\times N}$

Recalling $J_\lambda(w) = \sum_{k=1}^{K} e^{\left(\lambda\|y_k - \hat{f}(x_k,w)\|^2\right)}$ and $\eta_k(w) = e^{\lambda\left(\|y_k - \hat{f}(x_k,w)\|^2 - \|\varepsilon_M(w)\|^2\right)}$, we obtain

$$\begin{aligned} J_\lambda(w) &= e^{\lambda\|\varepsilon_M(w)\|^2}\sum_{k=1}^{K} e^{\lambda\left(\|y_k - \hat{f}(x_k,w)\|^2 - \|\varepsilon_M(w)\|^2\right)} \\ &= e^{\lambda\|\varepsilon_M(w)\|^2}\sum_{k=1}^{K}\eta_k(w) \end{aligned} \tag{17}$$

where $0 < J_\lambda(w) \le Ke^{\lambda\|\varepsilon_M(w)\|^2}$, and the evaluation of $J_\lambda(w)$ does not cause computational overflow if $Ke^{\lambda\|\varepsilon_M(w)\|^2} < F_{\max}$, where $F_{\max}$ denotes the largest positive floating point number that can be handled by the computer. Then, a value λ_c of λ under which the evaluation of $J_\lambda(w)$ does not cause computer overflow is

$$\lambda_c := \frac{\ln F_{\max} - \ln K}{\|\varepsilon_M(w)\|^2} \tag{18}$$

Differentiating $J_\lambda(w)$ yields

$$\begin{aligned}\frac{\partial J_\lambda(w)}{\partial w_i} &= -2\lambda\sum_{k=1}^{K} e^{\lambda\|\varepsilon_k(w)\|^2}\varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i} \\ &= -2\lambda e^{\lambda\|\varepsilon_M(w)\|^2}\sum_{k=1}^{K}\eta_k(w)\,\varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i}\end{aligned} \tag{19}$$

whose evaluation usually does not cause computer overflow, but whose value often very large. Consequently, the gradient

$$\nabla J_\lambda(w) = \left[\frac{\partial J_\lambda(w)}{\partial w_i}\right]_{1\times N} \tag{20}$$

is often very large and changes much from iteration to iteration in the process of minimizing $J_\lambda(w)$. Furthermore, a large gradient causes saturation of sigmoidal activation functions. To avoid such problems in the gradient descent or backpropagation method, we use the normalized gradient $\nabla J_\lambda(w) / \|\nabla J_\lambda(w)\|$, where $\|\cdot\|$ denotes the Euclidean norm.

IV. Statistical Neural Network Pruning

In training neural networks, the optimal number of hidden nodes is hard to be determined before each training starts, while it is commonly estimated by the trial-and-error fashion. A network pruning strategy, which first selects a neural network with a large number of hidden nodes then removes the redundant nodes during the training, is the most important way to achieve the proper neural network that uses less trainable parameters and thus has better generalization capability. Although it is difficult to decide which weights or nodes are the least important for pruning the network, several techniques and heuristic approaches have been developed to resolve this issue, such as the iterative pruning algorithm for feedforward neural networks [23], the Karnin's pruning method [24], the pruning based on orthogonal transforms like the SVD and QR with column pivoting (QR-cp) [25], the principal components pruning [26], and methods based on the perturbation analysis of the second-order Taylor expansion of the objective function like optimal brain damage (OBD) [27] and optimal brain surgeon (OBS) [28].

Particularly, the OBD and OBS methods together with their variations are the most popular procedures to prune neural networks through determining the saliency of weights with the aid of Hessian. However, pruning large neural networks with these methods involves intensive calculations of Hessian, which cost enormous amount of computational time and memory spaces on the real-world datasets in practice. A statistical neural network pruning method proposed in [29] described an innovated approach, which can reduce the total amount of computation in OBS with the use of the sensitivity measurement of weights that is closely related to the saliency of weights defined by OBS under certain demonstrated conditions.

The basic idea of the statistical neural network pruning is derived from a hypothesis testing, which is a standard procedure of statistical inferences applied for testing a statistical hypothesis. If the hypothesis testing declares a null and an alternative hypothesis as

$$H_0 : \mu = 0 \tag{21}$$

$$H_1 : \mu \neq 0 \tag{22}$$

where μ is a population mean, while a sample mean x taken from the population that has a normal distribution with variance s^2, then the test z-statistic for the mean under H_0 is

$$z = \frac{x}{s} \tag{23}$$

that describes the distance of the sample mean x away from the population mean $\mu = 0$. A significance level is selected as a probability threshold to test the z-statistic, where the smaller significance level means the stronger evidence to reject the null hypothesis.

The z-statistic that applies to test the hypothesis, which describes the sensitivity of weights as zero, is an estimation of each component of the weight vector in the neural network. The z value is calculated by (23) with the use of the absolute value of the j-th weight as x and the estimation of s as

$$s = \sqrt{P(w) \Big/ \left(\frac{\partial P(w)}{\partial w_i}\right)^2} \tag{24}$$

where $P(w)$ is the mean squared error (MSE) criterion defined as

$$P(w) := \frac{1}{K} Q(w) \tag{25}$$

and $\frac{\partial P(w)}{\partial w_i}$ is the first-order derivative of $P(w)$

$$\frac{\partial P(w)}{\partial w_i} = -\frac{2}{K}\sum_{k=1}^{K} \left\| y_k - \hat{f}(x_k, w) \right\| \frac{\partial \hat{f}(x_k, w)}{\partial w_i} \tag{26}$$

If the tested z value is greater than or equal to a chosen critical value z_c, such a z-statistic is regarded as the sufficient evidence to support that the evaluated component of the weight vector is not zero. Otherwise, the sensitivity of the evaluated weight is zero, which indicates that the corresponding connection associated with the evaluated weight in the network should be pruned.

A statistical pruning (SP) method directly calculates z values for all components of the weight vector and prunes the weight if its corresponding z value is smaller than a critical value z_c. With choosing the proper z_c, the neural network after pruning is able to be retained in R epochs for achieving a better generalization level with less number of weights than the original network. The critical value z_c is commonly chosen as the value with respect to small significance level in the complementary cumulative convention of the standard normal table (also referred to as the Z table), which gives a probability that a statistic is greater than z_c. For example, if we choose $z_\mathrm{c} = 2.00$, the corresponding significant level is 4.55%, which is generally accepted as a sufficiently small probability where any evaluated z value greater than or equal to z_c is considered as a sufficient evidence to reject the null hypothesis in practice. SP is described in Algorithm 2.

Although a proper critical value z_c is generally selected according to a small significance level for rejecting the null hypothesis in practice, the neural network could be pruned incorrectly by an inappropriate selection of z_c. If z_c is too large, the linearization in the derivation of the statistical neural network pruning method is invalid, leading to incorrect pruning results for the significant weights without the linearization or the critical statistic against the null hypothesis. Moreover, if z_c is too large, more sensitive weights with large z values could be removed excessively because of $z < z_\mathrm{c}$. It may cause

Algorithm 2 Statistical Pruning

Require: Obtain the weight vector w from the previously trained neural network, select a critical value z_c, and set R;

for $i = 1$ to N **do** {Perform the pruning}
  Compute $z \leftarrow |w_i|/s$ where $s \leftarrow \sqrt{P(w) \Big/ \left(\frac{\partial P(w)}{\partial w_i}\right)^2}$;
  if $z < z_c$ **then**
    Set $w_i \leftarrow 0$ and $flag_i \leftarrow 0$;
  else
    Set $flag_i \leftarrow 1$;
  end if
end for
Obtain the pruned weight vector w;
for $j = 1$ to R **do** {Retrain the network}
  for $i = 1$ to N **do**
    if $flag_i = 1$ **then**
      Update w_i to ${w_i}^*$ by backpropagation;
    else
      Set ${w_i}^* \leftarrow w_i$;
    end if
  end for
end for
return The optimal weight vector w^*.

the model underfitting, where the weights after pruning with the large z_c are insufficient to properly express the training problem. On the other hand, if z_c is too small, it could cause ineffective pruning, where the weights may not be pruned enough thus the overfitting of the original network is not properly alleviated.

Therefore, we propose a gradual statistical pruning (GSP) method for choosing the best critical value z_c adaptively via repeating the pruning and retraining phases with the aid of the validation data. Like SP, an critical value z_c is initially selected to prune the trained network with the weight vector w_{old} and the error v_{old} for the validation data. Then, the network is retrained after pruning in R epochs. In GSP, we always record the weight vector w_{new} according to the recently lowest validation error as v_{new}, and then w_{new} is pruned by the updated critical value $z_c = z_c + z_{\text{inc}}$, where z_{inc} is an increment for gradually raising z_c as each pruning phase applies. The described pruning and retraining phases will repeat until the maximum critical value z_{max} or a satisfactory validation error v_{opt} is reached. GSP is described in Algorithm 3.

V. Experimental Settings

To experimentally verify the effectiveness of the GDC method in training neural networks on a real-world dataset, we evaluate it by training CNNs and MLPs on the MNIST dataset without data augmentation. The standard MNIST dataset contains 60,000 training samples and 10,000 testing samples of handwritten digits from 0 to 9. Each sample has 784 features, which are obtained from a 28 $\times$ 28 black and white image.

Algorithm 3 Gradual Statistical Pruning

Require: Obtain the weight vector w_{old} and its related validation error v_{old} from the previously trained neural network, choose a desired validation error v_{opt}, select z_{max} and z_{inc}, and set $z_c << z_{\text{max}}$;

while $z_c < z_{\text{max}}$ or $v_{\text{old}} > v_{\text{opt}}$ **do**
  Start SP with w_{old} and z_c;
  Record the lowest validation error v_{new} during the retraining phase in SP;
  Save the corresponding weight vector as w_{new};
  if $v_{\text{new}} < v_{\text{old}}$ **then**
    Let $w_{\text{old}} \leftarrow w_{\text{new}}$ and $v_{\text{old}} \leftarrow v_{\text{new}}$;
  end if
  $z_c \leftarrow z_c + z_{\text{inc}}$;
  Set $w^* \leftarrow w_{\text{old}}$;
end while
return The optimal weight vector w^*.

Each feature value is generated by the anti-aliasing normalized gray level of the corresponding pixel in an image. To properly use the MNIST dataset as the same as the benchmark methods employed, we randomly choose 50,000 out of 60,000 training samples and apply the left 10,000 samples as the validation set to select the best performed weights with the lowest validation error, then we use the full 60,000 training samples to keep training the optimal weights until the convergence. At last, we adopt the optimal weights to evaluate the 10,000 testing samples and provide the final test error rate.

For the experiment of CNNs, we apply the GDC method to train LeNet-5 [9] for classifying handwritten digits on the MNIST dataset. LeNet-5 is generally considered as a classic learning model with deep architectures, which comprise 8 layers including 1 input layer, 3 convolutional layers, 2 pooling layers, 1 fully connected layer, and 1 output layer in sequence. In our experiments, we apply the same experimental settings, including the organization of training and testing samples, the architecture of LeNet-5, and the selection of convolutional and pooling operations, as described in [9] to evaluate the GDC method on the pairwise mode. In addition, we perform the GDC method on the pairwise mode with using the same parameters as the GDC method to train LeNet-5 for demonstrating the advantages of GDC compared to GDC.

To further demonstrate the advantage of the GDC method comparing to more deep learning methods that are aided by unsupervised layer-wise pre-training, we perform GDC simply in the supervised manner to train distinct MLPs on the MNIST dataset and compare the achieved test error rates to benchmark results reported in training MLPs, stacked autoencoders (SAEs), deep belief networks (DBNs), and deep Boltzmann machines (DBMs) under the same experimental settings. Particularly, we perform GDC on one shallow MLP with the 784-1000-10 architecture, one deep MLP with the 784-500-1000-10 architecture, and another deep MLP with the 784-500-500-1000-10 architecture.

As for training parameters of the GDC and GDC methods,

we set the initial value of λ as 10^4, the maximum training epochs for the deconvexification as 10 and the deconvexification rate of λ as 0.9. For the pairwise NRAE and RAE training sessions in both GDC and GDC, we fix the global learning rate and the momentum term in stochastic gradient descent (SGD) equal to 0.0001 and 0.5, respectively. For the SSE training session in GDC when $\lambda < 1$, we apply the learning rate decay to decrease the global learning rate by 50% in every 10 training epochs, i.e., 0.0001 for the first 10 epochs, then 0.00005 for the next 10 epochs, and so on. The training session is considered as convergence once the training error rate in ten consecutive epochs is less than 0.01%. For all experiments, we use $F_{max} = 10^{300}$ and $F_{min} = 10^{-300}$ to perform GDC.

Some general parameters in training neural networks are chosen based on the suggestions in [30]: initial weights are randomly selected from a uniform distribution between $-2.4/F_u$ and $2.4/F_u$, where F_u is the number of input nodes of the connected unit u; all input and output values in the training dataset are normalized into $[-1, 1]$; the activation function in each training node is chosen as the hyperbolic tangent function $\varphi(v) = a\tanh(bv)$, where $a = 1.7159$ and $b = 2/3$.

In order to remove the network redundancy and improve the generalization of LeNet-5 and MLPs, we perform GSP to prune the networks trained by GDC. For parameters of GSP, we choose $z_{\text{c}} = 0.2$, $z_{\text{max}} = 3.0$, $z_{\text{inc}} = 0.1$, $R = 10$, and $v_{\text{opt}} = 0.80\%$ in our experiments. Since GSP provides the best pruned and retrained network with the aid of the validation data, the network does not need to convergence during the retraining phase after pruning, and it would avoid excessive training time if the network is large. Therefore, in each retraining phase, we always apply SGD by using the fixed global learning rate and the momentum term as 0.00001 and 0.5 without employing the learning rate decay.

VI. Results and Discussion

A. Experiments with LeNet-5 on MNIST dataset

LeNet-5 is a well-known CNN that was first published by Yann LeCun in 1998 [9]. The GDC method without and with statistical pruning is applied to train LeNet-5. The numerical results are listed in Table I in comparison with those in [9]. Note that GDC followed by RAE and pruning produced a LeNet-5 with a test error rate of 0.84%, which is close to the two LeNet-5's trained with stochastic diagonal Levenberg-Marquardt (SDLM) method on the MNIST dataset with image distortion added. **The results are encouraging**.

B. Experiments with MLPs on MNIST dataset

The purpose of the experimental results and comparisons that are discussed here is to show how a GDC training method compares to well-known methods. We are not trying to come up with a neural network or a committee of neural networks to beat everyone else on the test error rates regardless of the size of the neural network. Therefore, we select neural network architectures of reasonable sizes from the table on the well-known webpage http://yann.lecun.com/exdb/mnist/ and use GDC to train neural networks of the selected architectures. We then compare the resultant test error rates with the corresponding ones in the same table.

Numerical results show that GDC without switching to RAE produces a good MLP(784-300-10) in each of 5 training sessions starting with 5 different initialization seeds on the MNIST dataset without added data with distortion [6]. The 5 resultant test error rates (2.61%, 2.67%, 2.70%, 2.73%, 2.88%) are listed in Table II in comparison with those obtained with MLPs of the same or similar architectures.

GDC with switching to RAE was used to train an MLP(784-1000-10) on the MNIST dataset. The test error rate of the resultant neural network is 1.37%. After statistical pruning, the test error rate was reduced to 1.34%. These 2 test error rates are also listed in Table II in comparison with those obtained with MLPs of the same or similar architectures. Notice that the test error rate of MLP(784-1000-10) trained as a DBN has a test error rate of 1.30%.

In [17], MLP(784-X-X-X-10)'s trained with supervised pretraining, auto-associator pretraining and DBN have a test error rate of 2.00%, 1.40%, 1.20%, respectively, where X denotes a number between 500 and 1000. In comparison, MLP(784-500-500-1000-10)'s trained with GDC w/ RAE, GDC w/ RAE + pruning have a test error rate of 1.29% and 1.27% respectively. Smaller MLP(784-500-1000-10)'s trained with GDC w/ RAE, GDC w/ RAE + pruning have a test error rate of 1.31% and 1.29% respectively. However, an MLP(784-500-500-1000-10) trained as a DBM has a test error rate of 1.01%, and an MLP(784-500-1000-10) trained as a DBM has a very good test error rate of 0.95% [19].

C. Pruning LeNet-5 and MLPs

The experimental result shown in Table I present a new test error rate 0.84%, which is lower than the original test error rate 0.90%, with applying GSP after GDC on LeNet-5. This new test error rate is also comparable to a benchmark result 0.80% achieved by applying huge data distortion as reported in [9]. Since GSP is directly applied to LeNet-5 after GDC without any image distortions, the experimental result we achieved confirms the effectiveness of GSP for improving the generalization of LeNet-5. Furthermore, the test error rate 0.84% is achieved when the critical value $z_{\text{c}} = 2.0$ in GSP. Meanwhile, the significance level corresponding to $z_{\text{c}} = 2.0$ is 4.55%, which is generally accepted as a sufficiently small probability to reject the null hypothesis in practical hypothesis testing. According to that significance level, the pruning percentage is close to 60%, which means more than a half of trainable weights in LeNet-5 could be pruned in GSP. It illustrates that GSP is capable to provide a succinct network by dramatically reducing trainable weights of the original LeNet-5 and achieve a lower test error rate compared to the network without pruning and data augmentation.

Table II demonstrates the best performances of shallow and deep MLPs achieved by GSP after applying GDC. As similar as what we observed on LeNet-5, both shallow and deep MLPs are able to be pruned by GSP, achieving consistently

TABLE I
TEST ERROR RATES ACHIEVED BY LENET-5 ON THE MNIST DATASET

Training Method	Neural Network	Test Error Rate
SSE + SDLM	LeNet-5	0.95%
GDC w/o RAE	**LeNet-5**	**0.93%**
GDC w/ RAE	**LeNet-5**	**0.90%**
SSE + SDLM (huge distortions)	LeNet-5	0.85%
GDC w/ RAE + Pruning	**LeNet-5**	**0.84%**
SSE + SDLM (distortions)	LeNet-5	0.80%

TABLE II
TEST ERROR RATES ACHIEVED BY MLPS ON THE MNIST DATASET

Training Method	Neural Network	Test Error Rate
SSE	MLP(784-300-10)	4.7%
SSE (distortions)	MLP(784-300-10)	3.6%
GDC w/o RAE	**MLP(784-300-10)**	**2.61%, 2.67%, 2.70%, 2.73%, 2.88%**
SSE	MLP(784-1000-10)	4.5%
SSE	MLP(784-300-100-10)	3.05%
SSE	MLP(784-500-150-10)	2.95%
CE	MLP(784-1000-10)	1.78%
CE	MLP(784-800-10)	1.60%
CE + Weight Regularizer	MLP(784-1000-10)	1.68%
CE + Denoising SAE	MLP(784-1000-10)	1.57%
GDC w/ RAE	**MLP(784-1000-10)**	**1.37%**
GDC w/ RAE + Pruning	**MLP(784-1000-10)**	**1.34%**
Deep Belief Network	MLP(784-1000-10)	1.30%
Supervised Pretraining	MLP(784-X-X-X-10)	2.00%
Stacked Auto-encoder	MLP(784-X-X-X-10)	1.40%
GDC w/ RAE	**MLP(784-500-1000-10)**	**1.31%**
GDC w/ RAE + Pruning	**MLP(784-500-1000-10)**	**1.29%**
GDC w/ RAE	**MLP(784-500-500-1000-10)**	**1.29%**
GDC w/ RAE + Pruning	**MLP(784-500-500-1000-10)**	**1.27%**
Deep Belief Network	MLP(784-X-X-X-10)	1.20%
Deep Boltzmann Machine	MLP(784-500-500-1000-10)	1.01%
Deep Boltzmann Machine	MLP(784-500-1000-10)	0.95%

lower test error rates compared to the original networks before pruning. In addition, the test error rates are achieved when the critical values of z_c are adapted between 1.8 and 2.2. The significance levels according to these z_c values are below 10%, and the pruning percentages are correspondingly between 20% and 30%. Such experimental results present the effectiveness of GSP in pruning shallow and deep MLPs for a better generalization with a smaller network architecture.

VII. CONCLUSION

Gradual deconvexification (GDC) comprising two stages is proposed for training neural networks. In stage 1, we minimize normalized risk-averting error (NRAE) with a very large risk-sensitivity index λ and gradually decrease it. When λ is large, we use a fast method to evaluate the NRAE and its gradient. As soon as the evaluation of risk-averting error (RAE) does not cause computer overflow, we switch to minimizing RAE with a fixed λ as stage 2 until convergence or termination by cross-validation.

Stage 1 allows the use of very large λ to make advantage of a large convexity region of RAE contained in NRAE. Stage 1 brings the squared errors down so that RAE becomes computationally manageable. Stage 2 allows further use of a large convexity region and continued much tilting of the plateaus and tunneling around saddle points on SSE. An important advantage of the propose method is the consistent

performances among different training sessions with different initialization seeds. Multiple training sessions with different initialization seeds and selection of the best one obtained are needed no longer. Another advantage is it wide applicability. It can be applied for virtually any data fitting including training convolutional and recurrent neural networks and estimating statistical nonlinear regression models. The third advantage is its conceptual simplicity and mathematical justification. The fourth advantage is the possibility of its use jointly with other methods such as the discriminative unsupervised pretraining.

Statistically pruning connections after stage 2 is also proposed in this paper that reduces data overfitting and enhances the generalization capability of the neural network under training.

Numerical experiments show the efficacy of stage 1, stage 2, and pruning. The proposed method comprising GDC and pruning outperforms discriminative training methods using SSE or cross entropy in all of our examples including convolutional networks on benchmark datasets. However, the proposed method does not do as well as the generative training methods such as the deep belief network (DBN) and deep Boltzmann machine (DBM). The differences are small but significant.

The numerical experiments convinced us that the differences are those differences in the generalization capabilities of the neural network resulting from training with GDC and those of DBNs and DBMs. Therefore, on the top of the list of future work is the exploration of methods for reducing data overfitting such as drop-out and network regularization. The mathematical simplicity of GDC is expected to prove useful here.

////// Gradual deconvexification (GDC) comprising two stages is proposed for training neural networks. An important advantage of the propose method is the consistent performances among different training sessions with different initialization seeds. Multiple training sessions with different initialization seeds and selection of the best one obtained are needed no longer. Another advantage is it wide applicability. It can be applied for virtually any data fitting including training convolutional and recurrent neural networks and estimating statistical nonlinear regression models. The third advantage is its conceptual simplicity and mathematical justification. The fourth advantage is the possibility of its use jointly with other methods such as the discriminative unsupervised pretraining.

Statistically pruning connections after GDC is also proposed in this paper that reduces data overfitting and enhances the generalization capability of the neural network under training.

The proposed method comprising GDC and pruning outperforms discriminative training methods using SSE or cross entropy in all of our examples including convolutional networks on benchmark datasets. However, the proposed method does not do as well as the generative training methods such as the deep belief network (DBN) and deep Boltzmann machine (DBM). The differences are small but significant.

The numerical experiments convinced us that the differences are those differences in the generalization capabilities of the neural network resulting from training with GDC and those of DBNs and DBMs. Therefore, the most important future work for us is the exploration of methods for reducing data overfitting such as drop-out and network regularization. //////

REFERENCES

[1] J. T.-H. Lo and D. Bassu, "An adaptive method of training multilayer perceptrons," in *Proceedings of the 2001 International Joint Conference on Neural Networks*, vol. 3, Jul 2001, pp. 2013–2018.

[2] J. T.-H. Lo and D. Bassu, "Robust identification of dynamic systems by neurocomputing," in *Proceedings of the 2001 International Joint Conference on Neural Networks*, vol. 2, Jul 2001, pp. 1285–1290.

[3] J. T.-H. Lo, "Convexification for data fitting," *Journal of Global Optimization*, vol. 46, no. 2, pp. 307–315, Feb 2010.

[4] J. T.-H. Lo, Y. Gui, and Y. Peng, "Overcoming the local-minimum problem in training multilayer perceptrons with the NRAE training method," in *Advances in Neural Networks - ISNN 2012*, vol. Part I, July 2012, pp. 440–447.

[5] J. T.-H. Lo, Y. Gui, and Y. Peng, "Overcoming the local-minimum problem in training multilayer perceptrons with the NRAE-MSE training method," in *Advances in Neural Networks - ISNN 2013*, vol. Part I, July 2013, pp. 83–90.

[6] J. T.-H. Lo, Y. Gui, and Y. Peng, "Overcoming the local-minimum problem in training multilayer perceptrons by gradual deconvexification," in *Proceedings of the 2013 International Joint Conference on Neural Networks*, August 2013, pp. 1–6.

[7] Y. Gui, J. T.-H. Lo, and Y. Peng, "A pairwise algorithm for training multilayer perceptrons with the normalized risk-averting error criterion," in *Proceedings of the 2014 International Joint Conference on Neural Networks*, July 2014, pp. 358–365.

[8] J. T.-H. Lo, Y. Gui, and Y. Peng, "The normalized risk-averting error criterion for avoiding nonglobal local minima in training neural networks," *Neurocomputing*, vol. 149, no. A, pp. 3–12, February 2015.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[11] R. Pascanu, Y. Dauphin, S. Ganguli, and Y. Bengio, "On the saddle point problem for non-convex optimization," in *arXiv*, May 2014, p. 1405.4604v2.

[12] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2014, pp. 2933–2941.

[13] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *arXiv*, January 2015, p. 1412.0233v3.

[14] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *The Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, March 2003.

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

[16] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI," in *Large-Scale Kernel Machines*. Cambrigde, MA, USA: MIT Press, 2007.

[17] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*, NIPS. Cambrigde, MA, USA: MIT Press, 2007, pp. 153–160.

[18] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, Jan 2009.

[19] R. Salakhutdinov and G. E. Hinton, "Deep boltzmann machines," *Journal of Machine Learning Research - Proceedings Track*, vol. 5, pp. 448–455, 2009.

[20] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[21] G. E. Hinton, "A practical guide to training restricted boltzmann machines," *Lecture Notes in Computer Science in Neural Networks: Tricks of the Trade*, vol. 7700, pp. 599–619, 2012.

[22] G. Hinton, L. Deng, D. Yu, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. S. G. Dahl, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, November 2012.

[23] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 519–531, May 1997.

[24] E. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 239–242, June 1990.

[25] P. P. Kanjilal and D. N. Banerjee, "On the application of orthogonal transformation for the design and analysis of feedforward networks," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1061–1070, 1995.

[26] A. U. Levin, T. K. Leen, and J. E. Moody, "Fast pruning using principal components," in *Advances in Neural Information Processing Systems 6*, J. Cowan, G. Tesauro, and J. Alspector, Eds. Morgan Kaufmann, 1994, pp. 35–42.

[27] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan Kaufmann, 1990, pp. 598–605.

[28] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Proceedings of the 1993 IEEE International Conference on Neural Networks*, 1992, pp. 293–299.

[29] J. T.-H. Lo, "Statistical method of pruning neural networks," in *Proceedings of the 1999 International Joint Conference on Neural Networks*, vol. 3, 1999, pp. 1678–1680.

[30] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller, "Efficient backprop," *Lecture Notes in Computer Science in Neural Networks: Tricks of the Trade*, vol. 1524, pp. 9–50, 1998.