RELATIONSHIPS BETWEEN DEEP LEARNING AND LINEAR ADAPTIVE SYSTEMS

Scott C. Douglas

Southern Methodist University
Dept. of Electrical and Computer Engineering
Dallas, Texas 75275 USA
sdouglas@smu.edu

Eric C. Larson

Southern Methodist University
Dept. of Computer Science
Dallas, Texas 75275 USA
eclarson@smu.edu

ABSTRACT

Linear adaptive systems are a well-known staple in numerous signal processing applications. Recently, significant activity and performance gains have been achieved in multilayer neural networks for deep learning applied to practical data processing applications. In this paper, we describe the important relationships and significant differences between the procedures and methods used in linear adaptive systems and those used in multilayer neural networks for deep learning tasks. Input-output structures, cost functions and training criteria, adaptive algorithms, and data processing and optimization strategies are considered. It is the hope of the authors that this discussion will spur further crossover between the two fields, and in particular allow knowledge to be shared and further progress to be made.

Index Terms— adaptive signal processing, adaptive systems, deep learning, machine learning, neural networks.

1. INTRODUCTION

Deep learning systems for data classification and regression have seen an explosion of activity in recent years. Many researchers and system designers have leveraged (a) large data corpuses collected and made available on the internet, (b) the availability of software frameworks for implementing deep learning systems, (c) fast computational resources within parallel hardware such as graphical processing units (GPUs), and (d) access to preconfigured software and hardware through cloud computing. These systems classify images, recognize spoken commands, identify patterns or trends, fill in or replace missing and/or erroneous measurements – the list of applications grows longer each day. As a result, there is a tremendous need within the entire technical community implementing deep learning systems to understand the technology so that it can be successfully tuned.

It may seem surprising, but the underpinnings of today's deep learning systems began to be extensively explored about 70 years ago [2, 15, 16] and the basic methodologies in use today were well-articulated over 30 years ago [18, 19]. Moreover, the adaptive algorithms and methods that spurred the development of multilayer neural networks also created the field of linear adaptive systems, becoming a technology staple in numerous signal processing applications with well-articulated design rules and performance metrics developed over the same period [3]–[14]. The goal of this paper is to describe the important relationships and significant differences between research and practice in linear adaptive systems and that in multilayer neural networks in use for deep learning tasks. It is the hope of the authors that this discussion will spur crossover between the two fields as well as between the corresponding computer science and electrical engineering communities, and in particular allow knowledge to be shared and further progress to be made in modern data processing systems.

This paper is organized into four sections addressing the following issues: input-output structures, cost functions and training criteria, adaptive algorithms, and data processing and optimization strategies. References are organized by date of first edition appearance and by technical area.

2. INPUT-OUTPUT STRUCTURES

Both deep learning systems and linear adaptive systems process measurements or information to predict a desired result. In linear adaptive systems, the processing structure is by definition a linear structure and is most often a weighted combination of L input signal samples $\{x_i(n)\}$ of the form

$$\widehat{d}(n,k) \quad = \quad \sum_{i=1}^{L} w_i(k) x_i(n), \tag{1}$$

where $w_i(k)$ are the L weights of the adaptive system at iteration k and $\widehat{d}(n,k)$ is the system's output for input pattern n and weight iteration k. For systems with multiple outputs, the input-output relation can be compactly expressed in matrix form as

$$\widehat{\mathbf{d}}(n,k) \quad = \quad \mathbf{W}(k)\mathbf{x}(n). \tag{2}$$

Traditionally, linear adaptive systems have been trained using algorithms that employ the current input signal pattern $\mathbf{x}(n)$ in the updates, allowing $k = n$ to be used and thus simplifying the notation to $\widehat{\mathbf{d}}(n,n) = \widehat{\mathbf{d}}(n)$. In the discussions that follow, we will also refer to the linear system output as $\mathbf{y}(n,k) = \widehat{\mathbf{d}}(n,k)$ to allow a common notation.

It is possible to have different processing structures than those shown above in linear adaptive systems, particularly if the samples in $\mathbf{x}(n)$ are regularly indexed by position and/or time as in a recorded sound, image, or video. Probably the most important set of structures is the use of fast convolution methods involving the fast Fourier transform (FFT) and frequency-selective filters employing subband processing [9], and adaptive algorithms for such structures have seen significant development for audio applications in particular. Another important class of systems in this space are those involving recursive structures, also known as adaptive infinite impulse response (IIR) filters [7], where the number of parameters can typically be reduced for a given modeling capability, although these structures often require monitoring of their parameters in order to maintain input-output stability.

In deep learning, processing architectures are comprised of layered systems in which the m layer's output $\mathbf{y}_m(n,k)$ has the general form

$$\mathbf{y}_m(n,k) = \mathbf{f}_m(\mathbf{W}_m(k)\mathbf{x}_m(n,k) + \mathbf{b}_m(k)) \quad (3)$$

where the matrix $\mathbf{W}_m(k)$ and the vector $\mathbf{b}_m(k)$ contain adjustible parameters and $\mathbf{f}_m(\widehat{\mathbf{d}})$ is a vector-valued nonlinearity acting on the elements of its argument. The use of bias weights in $\mathbf{b}_m(k)$ is a notable difference from the linear adaptive case; signals that are zero on average are common in linear systems and do not require the modelling of signal offsets. An M-layer structure is created by setting

$$\mathbf{x}_1(n,k) = \mathbf{x}(n) \quad (4)$$
$$\mathbf{x}_{m+1}(n,k) = \mathbf{y}_m(n,k) \quad (5)$$
$$\mathbf{y}(n,k) = \mathbf{y}_M(n,k). \quad (6)$$

Almost everything about this structure is open to design, including the number of layers M, the number of output channels at each layer specifying the size of $\mathbf{y}_m(n,k)$, and the choice of nonlinearities in $\mathbf{f}_m(\widehat{\mathbf{d}})$ at each layer. Most notably, however, the *structure* of $\mathbf{W}_m(k)$ is also open to design through the mechanisms of

- *weight sharing,* in which multiple rows of $\mathbf{W}_m(k)$ contain the same weight values at different positions, and
- *sparsity,* in which most of the entries of $\mathbf{W}_m(k)$ are chosen to be zero.

These two mechanisms are typically leveraged in deep learning systems to create systems that perform one- or two-dimensional convolution operations on time-series data [25] or images [20], respectively, as well as reduce the number of outputs at a given layer through sub-sampling, a process that is called *pooling.* Even these pooling mechanisms are open to design, as they can be data-dependent; the most common ones currently in use involve maximization (max pooling) [22], mean calculation (average pooling), L_2-norm evaluation (L_2-pooling), and simple decimation which when combined with convolution is sometimes called *stride convolution* in the deep learning community.

3. COST FUNCTIONS AND TRAINING CRITERIA

Given an input-output structure, both linear adaptive systems and deep neural networks employ cost functions in order to update their adaptive parameters. A *cost function* $J(k)$ is a scalar-valued function of the parameters in $\mathbf{W}(k)$ or $\{\mathbf{W}_m(k), \mathbf{b}_m(k)\}$ and the signal pattern pairs $\{\mathbf{x}(n), \mathbf{y}(n)\}$ that, when minimized, achieves an overall desirable result regarding the input-output pairs $\{\mathbf{x}(n), \mathbf{y}(n,k)\}$ for $1 \le n \le N$.

The choice of cost function is driven by the overall goal of the system. In continuous-valued estimation or *regression*, the goal is to estimate $\mathbf{y}(n)$ using $\mathbf{y}(n,k)$ so that the two are close to each other in some sense. Thus, criteria based on distances between $\mathbf{y}(n)$ and $\mathbf{y}(n,k)$ are often used. The most-often-used cost function in this regard is the sum-of-squared-errors cost function given by

$$J_{SE}(k) = \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{e}(n,k)||^2 \quad (7)$$
$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{L_o}(y_i(n) - y_i(n,k))^2 \quad (8)$$

where $||\cdot||^2$ denotes the Euclidean distance and $\mathbf{e}(n,k) = \mathbf{y}(n) - \mathbf{y}(n,k)$ are errors at the outputs of the system for an L_o-output system. The sum-of-squared-errors cost function is popular in both linear adaptive systems and deep learning because (a) it is simple to compute, (b) it is generally well-behaved in parameter space, and (c) training results can often be easily interpreted or judged. In linear adaptive systems, the choice is further motivated by two additional properties: (1) The cost function is both convex and unimodal in the parameters in $\mathbf{W}(k)$, such that it is easily optimized. (2) The gradient of the cost function is linear in the parameters in $\mathbf{W}(k)$, and thus analysis of the adaptive system's behavior based on this cost function is greatly simplified. Non-squared-error-based criteria are possible but are generally only beneficial when the training signals in $\mathbf{y}(n)$ have deviations from a linear model that can be characterized, such as by their statistical distribution [10]. The sum-of-squared-errors cost is often chosen in deep learning tasks for regression as well, although both convexity and unmodality are typically lost due to the nonlinearities in $\mathbf{f}_m(\widehat{\mathbf{d}})$ and the layered input-output structure.

Much of the current work in deep learning focuses on *classification,* in which the goal is to identify a discrete category or class associated with the input signal patterns $\mathbf{x}(n)$. Typically, this leads to a structured desired pattern $\mathbf{y}(n)$ that is also discrete-valued, and the choice

$$y_i(n) = \begin{cases} 1 & \text{if } \mathbf{x}(n) \text{ corresponds to class } i \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

is common. Classification tasks typically are interpreted within a probabilistic framework and fall within the well-studied field of statistical pattern recognition [17]. In such a framework, the outputs in $\mathbf{y}(n,k)$ of a deep neural network are often interpreted as probabilities, driving the choices of both the nonlinearities in the output layer $\mathbf{f}_M(\widehat{\mathbf{d}})$ and the criterion $J(k)$ for training. A common choice for both yields the *cross-entropy loss*

$$J(k) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{L_o} y_i(n)\log y_i(n,k) \quad (10)$$

$$\mathbf{y}(n,k) = \mathbf{f}_M(\widehat{\mathbf{d}}_M(n,k)) \quad (11)$$

where $\mathbf{f}_M(\widehat{\mathbf{d}})$ is the *softmax output function* with entries [22]

$$f_{iM}(\widehat{\mathbf{d}}) = \frac{\exp(\widehat{d}_i)}{\sum_{j=1}^{L_o}\exp(\widehat{d}_j)}. \quad (12)$$

By contrast, linear adaptive systems for classification tasks are much more constrained in their modeling abilities due to their linear input-output structures. Such systems often use simple decision nonlinearities such as quantizers at their system output so that the linear nature of their structure can be easily observed in the signal $\widehat{d}_i(n,k)$ prior to the decision element. The field of *adaptive equalization* and *adaptive array processing* in digital communications is concerned with these methods, where signals and adaptive parameters are assumed to be complex-valued due to the frequency-domain processing involved. The criteria often employed in such systems are related to the sum-of-squared-errors cost in (8), however, again due to the benefits in performance characterization and system tuning that such choices provide.

While not a focus of this paper, unsupervised criteria for training both linear adaptive systems and deep neural networks have been studied, where only the input patterns $\mathbf{x}(n)$ are available for training. In deep learning, the most common choice for this occurs in the problem of *autoencoding*, in which the goal is to set $\mathbf{y}(n) = \mathbf{x}(n)$ and train the system to approximate the input pattern at its output. Such training makes sense when the deep neural network has one or more low-dimensional intermediate outputs $\mathbf{y}_m(n,k)$, and has been used in a *pretraining* process for adjusting parameters prior to signal modeling of the pair $\{\mathbf{x}(n),\mathbf{y}(n)\}$. In linear adaptive systems, similar methods form the bases of *principal component analysis* [11] and *independent component analysis* [12], which form the underlying basis of applications such as *subspace tracking* and *blind source separation,* respectively.

4. ADAPTIVE ALGORITHMS

Given an input-output structure and a cost function $J(k)$, an adaptive algorithm is used to adjust the parameters $\mathbf{W}(k)$ or $\{\mathbf{W}_m(k),\mathbf{b}_m(k)\}$ so that the value of $J(k)$ is minimized. For both linear adaptive systems and deep neural networks, by far the most popular algorithm choice is *gradient descent*, in which each parameter is adjusted according to the negative derivative of the cost with respect to the parameter, or

$$w_{ij}(k+1) = w_{ij}(k) - \eta(k)\frac{\partial J(k)}{\partial w_{ij}(k)} \quad (13)$$

for the entries of $\mathbf{W}(k)$ (the adjustments for $\mathbf{W}_m(k)$ and $\mathbf{b}_m(k)$ are similarly-described), where $\eta(k)$ is the step size at iteration k. Widrow and Hoff are credited for discovering this algorithm in modern signal processing contexts [2], where it is referred to as the *least-mean-square (LMS) algorithm.* The extension of this method for training multilayer neural networks is credited to Werbos [18], although it was popularized over a decade later by Rumelhart, Hinton, and Williams through their well-known book chapter [19] as the *backpropagation algorithm.* The notation in [19] is still widely-used today to describe the approach.

In both cases, the adjustments amount to accumulating terms that correspond to the products of error terms and input signal terms, where both of these terms are defined relative to the parameters being adjusted. For the linear case, the LMS algorithm is

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \eta(k)\frac{1}{N}\sum_{n=1}^{N}\mathbf{e}(n,k)\mathbf{x}^T(n), \quad (14)$$

The algorithm would potentially "stop" when the condition $\partial J(k)\partial w_{ij}(k) = 0$, corresponding to a perfectly-flat point in the multidimensional error surface defined by the cost $J(k)$, although in practice the algorithm will continue adjustment forever without any other intervention. Such an approach is well-founded from the standpoint of estimation theory, in which error and input signal decorrelation (orthogonalization) is a provable condition for identification of an optimum set of parameters, often referred to as the *Wiener solution* in deference to the pioneering work in [1].

It is important to note that, while simple to describe and implement, gradient descent is a heuristic search procedure, and better performance may be achievable through either modifications of the gradient descent algorithm, or an alternative approach altogether. In linear adaptive systems, an alternative class of methods that directly minimize $J_{SE}(k)$ at each iteration can be recursively-implemented, leading to extremely-fast convergence in many cases. Such *recursive least-squares (RLS)* procedures make use of system linearity and, in some cases, the shift-input structure of the input signal pattern $\mathbf{x}(n)$, to reduce the computational load. Some of these techniques employ carefully-crafted linear algebraic relations that test the limits of numerical accuracy and system stability, requiring clever feedback mechanisms to stabilize [8]. An accessible listing of many of these algorithms, along

with their associated references, can be found in [13]. Unfortunately, many of these RLS procedures do not directly apply to nonlinear input-output structures nor to the processing of non-convolutive models; and thus their applicability to deep learning is likely limited. Thus, it is often the case that modifications to the LMS algorithm prove the best choice for linear adaptive system design. The most popular modified method is the *normalized LMS (NLMS) algorithm*, which is stable for a fixed range of step sizes $\eta(k)$ – very convenient for system tuning [4, 5, 6, 14].

In deep learning, the sheer number of weights in such systems – many modern-day DNNs have millions of parameters – means computational complexity of any adaptive algorithm is a critical issue. As such, modifications of gradient descent are a common choice for system implementations, largely due to issues of computational simplicity. The nonlinear nature of the layered structures within DNNs means, however, that even well-behaved cost functions such as $J_{SE}(k)$ are *non-convex* in parameter space, and the design of any modifications of gradient descent are closely-tied to other issues in system implementation. Perhaps the most important recent developments in training algorithms within DNNs are

1. *batch normalization,* a strategy for adjusting the input patterns within the processing structure to normalize signal variances and reduce signal mean values [29],

2. *dropout,* a system-averaging technique whereby a fraction of nodes are randomly removed within training, resulting in networks with better generalization capabilities [27], and

3. *weight initialization* strategies that address overall system gain issues for improved training [24, 30].

5. DATA PROCESSING AND OPTIMIZATION STRATEGIES

Given an input-output structure, a chosen cost function, and an adaptive algorithm, the final task is to process the data in $\{\mathbf{x}(n), \mathbf{y}(n)\}$ to obtain the parameters that produce relevant outputs $\mathbf{y}(n,k)$. In this case, the overall methodologies in linear adaptive systems and deep learning methods are often different due to different goals and outcomes. Moreover, the mechanisms used to optimize the overall performance of the system via exploration of parameter changes, structural changes, and data processing changes are also different.

Most linear adaptive systems function in an online setting, where new measurements $\{\mathbf{x}(n), \mathbf{y}(n)\}$ for additional training are constantly being created. Thus, the goal is to set up a system that continually improves its ability to model the input-output relationship over time. Thus, system parameters such as the number of parameters in the model and the choice of step size sequence $\eta(k)$ are designed to balance a multitude of factors including initial convergence speed, final estimation accuracy or amount of misadjustment in the system's output, and tracking behavior in time-varying signal conditions. Note that the goal for the system may not be to recover an accurate value $\mathbf{y}(n,k)$. For example, in adaptive noise canceling [3], the error signal $\mathbf{e}(n,k) = \mathbf{y}(n) - \mathbf{y}(n,k)$ is the quantity of interest, because $\mathbf{x}(n)$ is correlated to undesirable signals that are contained in $\mathbf{y}(n)$ that are to be cancelled out. Mechanisms for efficient computation via block-based operations can be used but generally do not impact performance significantly. Weight initialization is typically not critical. Moreover, a system is typically designed to perform well in a range of measurement scenarios, using data that is collected at the time the system is put into service – and when the system is turned off, the parameter values are often discarded.

In contrast, neural networks for deep learning are typically implemented in a two-phase process. In the *training phase* where $\{\mathbf{x}(n), \mathbf{y}(n)\}$ pairs are available, the values of $\{\mathbf{W}_m(k), \mathbf{b}_m(k)\}$ are adjusted to a desired cost function performance, at which point the parameters are "frozen." In the *evaluation phase,* these trained parameters are used without alteration to compute new outputs $\mathbf{y}(n,k)$ given input patterns $\mathbf{x}(n)$ for which no training pattern $\mathbf{y}(n)$ is available. Thus, there is great value in both the parameters computed during training and the architecture used to compute the system outputs. In this context, the goal is to achieve not only good training performance, but also a performance in the evaluation phase that is as close as possible to that observed in the training phase. The term *generalization* describes this ability of a neural network to produce useful outputs for new input patterns, and it is a requirement for success [17].

The choice of architecture (numbers of layers, nonlinearities, weight sharing and so on) [28, 31], the use of mechanisms for efficient computation such as small training block sizes ("mini-batch"), and the setting of parameters such as the initial weight values [24, 30] and the step size sequence $\eta(k)$ all affect this performance translation from training to evaluation in deep neural networks and need to be carefully tuned. When combined, these considerations typically lead to an iterative approach whereby the training data is partitioned into an N_T-sample *training set* and an N_V-sample *validation set*, where $N_T + N_V = N$. After the system is trained on the training set, performance is evaluated on the validation set to ensure that a sufficient level of performance is achieved. Then, system parameters are adjusted iteratively to improve overall system performance on the validation set through repeated training runs. Thus, overall system design in deep learning becomes iterative, where changes are typically evaluated numerically in various combinations. The sheer number of possible outcomes leads to significant use of computational resources, and achieving good performance requires good strategies combined with both insight and patience. Experience plays a critical role, and knowledge development and aggregation has been a focus within the field for some time [23]. General design rules in deep learning are challenging to obtain, although there has been recent theoretical progress on bounding the overall performance of such techniques [32].

6. REFERENCES

Linear Adaptive Systems:

[1] N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series, with Engineering Applications* (New York: Wiley, 1949).

[2] B. Widrow and M. Hoff, Jr., "Adaptive switching circuits," *IRE WESCON Convention Record,* pt. 4, pp. 96-104, 1960.

[3] B. Widrow, *et al*, "Adaptive noise cancelling: Principles and applications," *Proc. IEEE,* vol. 63, no. 12, pp. 1692-1716, Dec. 1975.

[4] B. Widrow and S.D. Stearns, *Adaptive Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall, 1986).

[5] J. R. Treichler, C.R. Johnson, Jr., and M.G. Larimore, *Theory and Design of Adaptive Filters,* 2nd ed. (Upper Saddle River, NJ: Prentice-Hall, 2001).

[6] S. Haykin, *Adaptive Filter Theory,* 5th ed. (Upper Saddle River, NJ: Pearson, 2014).

[7] J.J. Shynk, "Adaptive IIR filtering," *IEEE ASSP Mag.,* vol. 6, no. 2, pp. 4-21, April 1989.

[8] D.T.M. Slock and T. Kailath, "Numerically stable fast transversal filters for recursive least squares adaptive filtering," *IEEE Trans. Signal Processing,* vol. 38, no. 1, pp. 92-114, Jan. 1991.

[9] J.J. Shynk, "Frequency-domain and multirate adaptive filtering," *IEEE Signal Processing Mag.,* vol. 9, no. 1, pp. 14-37, Jan. 1992.

[10] S.C. Douglas and T.H.-Y. Meng, "Stochastic gradient adaptation under general error criteria," *IEEE Trans. Signal Processing,* vol. 42, no. 6, pp. 1335-1351, June 1994.

[11] K.I. Diamantaras and S.-Y. Kung, *Principal Component Neural Networks: Theory and Applications* (New York: Wiley, 1996).

[12] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis* (New York: Wiley, 2002).

[13] S.C. Douglas and R. Losada, "Adaptive filters in MATLAB: From novice to expert," *Proc. 2nd. Signal Processing Education Workshop,* Callaway Gardens, GA, paper 4.9, Oct. 2002.

[14] A.H. Sayed, *Fundamentals of Adaptive Filtering* (Hoboken, NJ: Wiley, 2003).

Neural Networks and Deep Learning:

[15] D.E. Hebb, *The Organization of Behavior: A Neuropsychological Theory* (New York: Wiley, 1949).

[16] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review,* vol. 65, no. 6, pp. 386-408, Nov. 1958.

[17] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification,* 2nd ed. (New York: Wiley, 2001).

[18] P.J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. thesis, Harvard University, 1974.

[19] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error backpropagation," in *Parallel Distributed Processing, Vol. 1: Foundations*, D.E. Rumelhart and J.L. McClelland, eds. (Cambridge, MA: Bradford Books, 1986), pp. 318-362.

[20] Y. LeCun, *et al*, "Handwritten digit recognition with a back-propagation network," *Proc. Advances in Neural Information Processing Syst. 2 (NIPS 1989),* Denver, CO, pp. 396-404, Nov. 1989.

[21] S. Haykin, *Neural Networks and Learning Machines,* 3rd. ed (Upper Saddle River, NJ: Pearson, 2009).

[22] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in the cortex," *Nature Neuroscience,* vol. 2, pp. 1019-1025, Nov. 1999.

[23] P.Y. Simard, D. Steinkraus, and J.C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," *Proc. 7th Int. Conf. Document Analysis and Recognition,* vol. 2, pp. 958-963, Aug. 2003.

[24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Proc. Machine Learning Research,* vol. 9, pp. 249-256, May 2010.

[25] G.E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. Audio, Speech, Language Processing,* vol. 20, no. 1, pp. 30-42, Jan. 2012.

[26] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet classification with deep convolutional neural networks," *Proc. 25th Int. Conf. Neural Information Processing Systems,* vol. 1, pp. 1097-1105, Dec. 2012.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Slakahutdinov, "Dropout: A simple way to prevent neural network over fitting," *J. Machine Learning Research,* vol. 15, pp. 1929-1958, June 2014.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Proc. 2015 Int. Conf. on Learning Representations,* San Diego, CA, `arXiv:1409.1556,` 14 pp., May 2015.

[29] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Proc. Machine Learning Research,* vol. 37, pp. 448-456, July 2015.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," *Proc. 2015 Int. Conf. Computer Vision,* pp. 1026-1034, Dec. 2015.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. 29th IEEE Conf. Computer Vision and Pattern Recog.,* pp. 770-778, June 2016.

[32] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," *Proc. Machine Learning Research,* vol. 48, pp. 1225-1234, June 2016.