IEEE Access
practical innovations : open solutions

Received April 20, 2014, accepted May 13, 2014, date of publication May 16, 2014, date of current version May 28, 2014.

Digital Object Identifier 10.1109/ACCESS.2014.2325029

Big Data Deep Learning: Challenges and Perspectives

XUE-WEN CHEN[1], (Senior Member, IEEE), AND XIAOTONG LIN[2]
[1]Department of Computer Science, Wayne State University, Detroit, MI 48404, USA
[2]Department of Computer Science and Engineering, Oakland University, Rochester, MI 48309, USA
Corresponding author: X.-W. Chen (xwen.chen@gmail.com)

ABSTRACT Deep learning is currently an extremely active research area in machine learning and pattern recognition society. It has gained huge successes in a broad area of applications such as speech recognition, computer vision, and natural language processing. With the sheer size of data available today, big data brings big opportunities and transformative potential for various sectors; on the other hand, it also presents unprecedented challenges to harnessing data and information. As the data keeps getting bigger, deep learning is coming to play a key role in providing big data predictive analytics solutions. In this paper, we provide a brief overview of deep learning, and highlight current research efforts and the challenges to big data, as well as the future trends.

INDEX TERMS Classifier design and evaluation, feature representation, machine learning, neural nets models, parallel processing.

I. INTRODUCTION

Deep learning and Big Data are two hottest trends in the rapidly growing digital world. While Big Data has been defined in different ways, herein it is referred to the exponential growth and wide availability of digital data that are difficult or even impossible to be managed and analyzed using conventional software tools and technologies. Digital data, in all shapes and sizes, is growing at astonishing rates. For example, according to the National Security Agency, the Internet is processing 1,826 Petabytes of data per day [1]. In 2011, digital information has grown nine times in volume in just five years [2] and by 2020, its amount in the world will reach 35 trillion gigabytes [3]. This explosion of digital data brings big opportunities and transformative potential for various sectors such as enterprises, healthcare industry manufacturing, and educational services [4]. It also leads to a dramatic paradigm shift in our scientific research towards data-driven discovery.

While Big Data offers the great potential for revolutionizing all aspects of our society, harvesting of valuable knowledge from Big Data is not an ordinary task. The large and rapidly growing body of information hidden in the unprecedented volumes of non-traditional data requires both the development of advanced technologies and interdisciplinary teams working in close collaboration. Today, machine learning techniques, together with advances in available computational power, have come to play a vital role in Big Data analytics and knowledge discovery (see [5]–[8]). They are employed widely to leverage the predictive power of Big Data in fields like search engines, medicine, and astronomy. As an extremely active subfield of machine learning, deep learning is considered, together with Big Data, as the "big deals and the bases for an American innovation and economic revolution" [9].

In contrast to most conventional learning methods, which are considered using shallow-structured learning architectures, deep learning refers to machine learning techniques that use supervised and/or unsupervised strategies to automatically learn hierarchical representations in deep architectures for classification [10], [11]. Inspired by biological observations on human brain mechanisms for processing of natural signals, deep learning has attracted much attention from the academic community in recent years due to its state-of-the-art performance in many research domains such as speech recognition [12], [13], collaborative fultering [14], and computer vision [15], [16]. Deep learning has also been successfully applied in industry products that take advantage of the large volume of digital data. Companies like Google, Apple, and Facebook, who collect and analyze massive amounts of data on a daily basis, have been aggressively pushing forward deep learning related projects. For example, Apple's Siri, the virtual personal assistant in iPhones, offers a wide variety of services including weather reports, sport news, answers to user's questions, and reminders etc. by utilizing deep learning and more and more data collected by Apple services [17]. Google applies deep learning algorithms to massive chunks of messy data obtained from the Internet for Google's translator,

Android's voice recognition, Google's street view, and image search engine [18]. Other industry giants are not far behind either. For example, Microsoft's real-time language translation in Bing voice search [19] and IBM's brain-like computer [18], [20] use techniques like deep learning to leverage Big Data for competitive advantage.

As the data keeps getting bigger, deep learning is coming to play a key role in providing big data predictive analytics solutions, particularly with the increased processing power and the advances in graphics processors. In this paper, our goal is not to present a comprehensive survey of all the related work in deep learning, but mainly to discuss the most important issues related to learning from massive amounts of data, highlight current research efforts and the challenges to big data, as well as the future trends. The rest of the paper is organized as follows. Section 2 presents a brief review of two commonly used deep learning architectures. Section 3 discusses the strategies of deep learning from massive amounts of data. Finally, we discuss the challenges and perspectives of deep learning for Big Data in Section 4.

II. OVERVIEW OF DEEP LEARNING

Deep learning refers to a set of machine learning techniques that learn multiple levels of representations in deep architectures. In this section, we will present a brief overview of two well-established deep architectures: deep belief networks (DBNs) [21]–[23] and convolutional neural networks (CNNs) [24]–[26].

A. DEEP BELIEF NETWORKS

Conventional neural networks are prone to get trapped in local optima of a non-convex objective function, which often leads to poor performance [27]. Furthermore, they cannot take advantage of unlabeled data, which are often abundant and cheap to collect in Big Data. To alleviate these problems, a deep belief network (DBN) uses a deep architecture that is capable of learning feature representations from both the labeled and unlabeled data presented to it [21]. It incorporates both unsupervised pre-training and supervised fine-tuning strategies to construct the models: unsupervised stages intend to learn data distributions without using label information and supervised stages perform local search for fine tuning.

Fig. 1 shows a typical DBN architecture, which is composed of a stack of Restricted Boltzmann Machines (RBMs) and/or one or more additional layers for discrimination tasks. RBMs are probabilistic generative models that learn a joint probability distribution of observed (training) data without using data labels [28]. They can effectively utilize large amounts of unlabeled data for exploiting complex data structures. Once the structure of a DBN is determined, the goal for training is to learn the weights (and biases) between layers. This is conducted firstly by an unsupervised learning of RBMs. A typical RBM consists of two layers: nodes in one layer are fully connected to nodes in the other layer and there is no connection for nodes in the same layer (see Fig.1, for example, the input layer and the first hidden layer H_1 form a RBM) [28]. Consequently, each node is independent of other nodes in the same layer given all nodes in the other layer. This characteristic allows us to train the generative weights W of each RBMs using Gibbs sampling [29], [30].

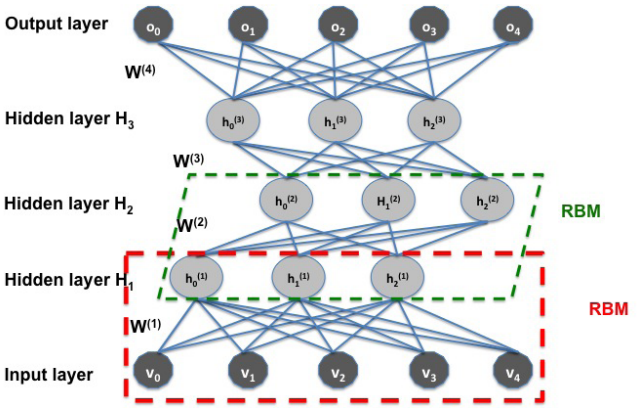

FIGURE 1. Illustration of a deep belief network architecture. This particular DBN consists of three hidden layers, each with three neurons; one input later with five neurons and one output layer also with five neurons. Any two adjacent layers can form a RBM trained with unlabeled data. The outputs of current RBM (e.g., $h_i^{(1)}$ in the first RBM marked in red) are the inputs of the next RBM (e.g., $h_i^{(2)}$ in the second RBM marked in green). The weights W can then be fine-tuned with labeled data after pre-training.

Before fine-tuning, a layer-by-layer pre-training of RBMs is performed: the outputs of a RBM are fed as inputs to the next RBM and the process repeats until all the RBMs are pre-trained. This layer-by-layer unsupervised learning is critical in DBN training as practically it helps avoid local optima and alleviates the over-fitting problem that is observed when millions of parameters are used. Furthermore, the algorithm is very efficient in terms of its time complexity, which is linear to the number and size of RBMs [21]. Features at different layers contain different information about data structures with higher-level features constructed from lower-level features. Note that the number of stacked RBMs is a parameter predetermined by users and pre-training requires only unlabeled data (for good generalization).

For a simple RBM with Bernoulli distribution for both the visible and hidden layers, the sampling probabilities are as follows [21]:

$$p\left(h_j = 1 \mid \mathbf{v}; W\right) = \sigma\left(\sum_{i=1}^{I} w_{ij} v_i + a_j\right) \tag{1}$$

and

$$p\left(v_i = 1 \mid \mathbf{h}; W\right) = \sigma\left(\sum_{j=1}^{J} w_{ij} h_j + b_i\right) \tag{2}$$

where $\boldsymbol{v}$ and $\boldsymbol{h}$ represents a $I \times 1$ visible unit vector and a $J \times 1$ hidden unit vector, respectively; W is the matrix of weights (w_{ij}) connecting the visible and hidden layers; a_j and b_i are bias terms; and $\sigma(\bullet)$ is a sigmoid function. For the case

of real-valued visible units, the conditional probability distributions are slightly different: typically, a Gaussian-Bernoulli distribution is assumed and $p(v_i \mid \mathbf{h}; W)$ is Gaussian [30].

Weights w_{ij} are updated based on an approximate method called contrastive divergence (CD) approximation [31]. For example, the (t + 1)-th weight for w_{ij} can be updated as follows:

$$\Delta w_{ij}(t+1) = c\Delta w_{ij}(t) + \alpha\left(\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{model}\right) \quad (3)$$

where α is the learning rate and c is the momentum factor; $\langle\cdot\rangle_{data}$ and $\langle\cdot\rangle_{model}$ are the expectations under the distributions defined by the data and the model, respectively. While the expectations may be calculated by running Gibbs sampling infinitely many times, in practice, one-step CD is often used because it performs well [31]. Other model parameters (e.g., the biases) can be updated similarly.

As a generative mode, the RBM training includes a Gibbs sampler to sample hidden units based on the visible units and vice versa (Eqs. (1) and (2)). The weights between these two layers are then updated using the CD rule (Eq. 3). This process will repeat until convergence. An RBM models data distribution using hidden units without employing label information. This is a very useful feature in Big Data analysis as DBN can potentially leverage much more data (without knowing their labels) for improved performance.

After pre-training, information about the input data is stored in the weights between every adjacent layers. The DBN then adds a final layer representing the desired outputs and the overall network is fine tuned using labeled data and back propagation strategies for better discrimination (in some implementations, on top of the stacked RBMs, there is another layer called associative memory determined by supervised learning methods).

There are other variations for pre-training: instead of using RBMs, for example, stacked denoising auto-encoders [32], [33] and stacked predictive sparse coding [34] are also proposed for unsupervised feature learning. Furthermore, recent results show that when a large number of training data is available, a fully supervised training using random initial weights instead of the pre-trained weights (i.e., without using RBMs or auto-encoders) will practically work well [13], [35]. For example, a discriminative model starts with a network with one single hidden layer (i.e., a shallow neural network), which is trained by back propagation method. Upon convergence, a new hidden layer is inserted into this shallow NN (between the first hidden layer and the desired output layer) and the full network is discriminatively trained again. This process is continued until a predetermined criterion is met (e.g., the number of hidden neurons).

In summary, DBNs use a greedy and efficient layer-by-layer approach to learn the latent variables (weights) in each hidden layer and a back propagation method for fine-tuning. This hybrid training strategy thus improves both the generative performance and the discriminative power of the network.

B. CONVOLUTIONAL NEURAL NETWORKS

A typical CNN is composed of many layers of hierarchy with some layers for feature representations (or feature maps) and others as a type of conventional neural networks for classification [24]. It often starts with two altering types of layers called convolutional and subsampling layers: convolutional layers perform convolution operations with several filter maps of equal size, while subsampling layers reduce the sizes of proceeding layers by averaging pixels within a small neighborhood (or by max-pooling [36], [37]).

Fig. 2 shows a typical architecture of CNNs. The input is first convoluted with a set of filters (C layers in Fig. 2). These 2D filtered data are called feature maps. After a nonlinear transformation, a subsampling is further performed to reduce the dimensionality (S layers in Fig. 2). The sequence of convolution/subsampling can be repeated many times (predetermined by users).

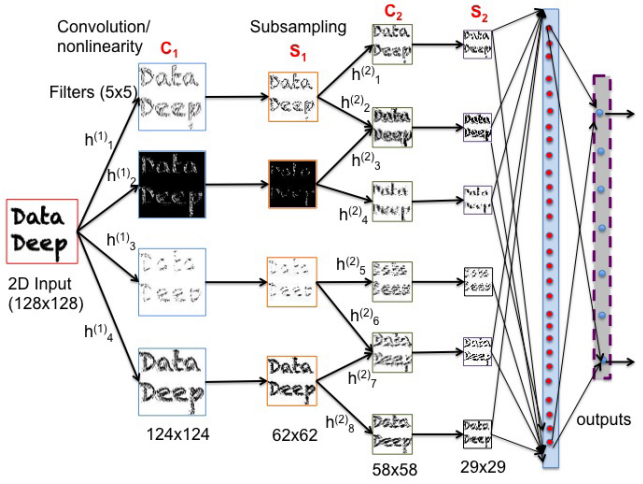

FIGURE 2. Illustration of a typical convolutional neural network architecture. The input is a 2D image, which convolves with four different filters (i.e., $h_i^{(1)}$, i = 1 to 4), followed by a nonlinear activation, to form the four feature maps in the second layer (C_1). These feature maps are down-sampled by a factor of 2 to create the feature maps in layer S_1. The sequence of convolution/nonlinear activation/subsampling can be repeated many times. In this example, to form the feature maps in layer C_2, we use eight different filters (i.e., $h_i^{(2)}$, i = 1 to 8): the first, third, fourth, and sixth feature maps in layer C_2 are defined by one corresponding feature map in layer S_1, each convoluting with a different filter; and the second and fifth maps in layer C_2 are formed by two maps in S_1 convoluting with two different filters. The last layer is an output layer to form a fully connected 1D neural network, i.e., the 2D outputs from the last subsampling later (S_2) will be concatenated into one long input vector with each neuron fully connected with all the neurons in. the next layer (a hidden layer in this figure).

As illustrated in Fig. 2, the lowest level of this architecture is the input layer with 2D $N \times N$ images as our inputs. With local receptive fields, upper layer neurons extract some elementary and complex visual features. Each convolutional layer (labeled Cx in Fig. 2) is composed of multiple feature maps, which are constructed by convolving inputs with different filters (weight vectors). In other words, the value of each unit in a feature map is the result depending on a local receptive field in the previous layer and the filter. This is

followed by a nonlinear activation:

$$y_j^{(l)} = f\left(\sum_i K_{ij} \bigotimes x_i^{(l-1)} + b_j\right) \tag{4}$$

where $y_j^{(l)}$ is the j-th output for the l-th convolution layer C_l; $f(\cdot)$ is a nonlinear function (most recent implementations use a scaled hyperbolic tangent function as the nonlinear activation function [38]: $f(x) = 1.7159 \cdot tanh(2x/3)$). K_{ij} is a trainable filter (or kernel) in the filter bank that convolves with the feature map $x_i^{(l-1)}$ from the previous layer to produce a new feature map in the current layer. The symbol $\bigotimes$ represents a discrete convolution operator and b_j is a bias. Note that each filter K_{ij} can connect to all or a portion of feature maps in the previous layer (in Fig. 2, we show a partially connected feature maps between S_1 and C_2). The sub-sampling layer (labeled Sx in Fig. 2) reduces the spatial resolution of the feature map (thus providing some level of distortion invariance). In general, each unit in the sub-sampling layer is constructed by averaging a 2×2 area in the feature map or by max pooling over a small region.

The key parameters to be decided are weights between layers, which are normally trained by standard backpropagation procedures and a gradient descent algorithm with mean squared-error as the loss function. Alternatively, training deep CNN architectures can be unsupervised. Herein we review a particular method for unsupervised training of CNNs: predictive sparse decomposition (PSD) [39]. The idea is to approximate inputs X with a linear combination of some basic and sparse functions.

$$Z^* = arg\,\|X - WZ\|_2^2 + \lambda\,|Z|_1 + \alpha\,\|Z - D \bullet tanh\,(KX)\|_2^2 \tag{5}$$

where W is a matrix with a linear basis set, Z is a sparse coefficient matrix, D is a diagonal gain matrix and K is the filter bank with predictor parameters. The goal is to find the optimal basis function sets W and the filter bank K that minimize the reconstruction error (the first term in Eq. 5) with a sparse representation (the second term), and the code prediction error simultaneously (the third term in Eq. 5, measuring the difference between the predicted code and actual code, preserves invariance for certain distortions). PSD can be trained with a feed-forward encoder to learn the filter bank and also the pooling together [39].

In summary, inspired by biological processes [40], CNN algorithms learn a hierarchical feature representation by utilizing strategies like local receptive fields (the size of each filter is normally small), shared weights (using the same weights to construct all the feature maps at the same level significantly reduces the number of parameters), and subsampling (to further reduce the dimensionality). Each filter bank can be trained with either supervised or unsupervised methods. A CNN is capable of learning good feature hierarchies automatically and providing some degree of translational and distortional invariances.

III. DEEP LEARNING FOR MASSIVE AMOUNTS OF DATA

While deep learning has shown impressive results in many applications, its training is not a trivial task for Big Data learning due to the fact that iterative computations inherent in most deep learning algorithms are often extremely difficult to be parallelized. Thus, with the unprecedented growth of commercial and academic data sets in recent years, there is a surge in interest in effective and scalable parallel algorithms for training deep models [12], [13], [15], [41]–[44].

In contrast to shallow architectures where few parameters are preferable to avoid overfitting problems, deep learning algorithms enjoy their success with a large number of hidden neurons, often resulting in millions of free parameters. Thus, large-scale deep learning often involves both large volumes of data and large models. Some algorithmic approaches have been explored for large-scale learning: for example, locally connected networks [24], [39], improved optimizers [42], and new structures that can be implemented in parallel [44]. Recently, Deng et al. [44] proposed a modified deep architecture called Deep Stacking Network (DSN), which can be effectively parallelized. A DSN consists of several specialized neural networks (called modules) with a single hidden layer. Stacked modules with inputs composed of raw data vector and the out puts from previous module form a DSN. Most recently, a new deep architecture called Tensor Deep Stacking Network (T-DSN), which is based on the DSN, is implemented using CPU clusters for scalable parallel computing [45].

The use of great computing power to speed up the training process has shown significant potential in Big Data deep learning. For example, one way to scale up DBNs is to use multiple CPU cores, with each core dealing with a subset of training data (data-parallel schemes). Vanhoucke et al. [46] discussed some aspects of technical details, including carefully designing data layout, batching of the computation, using SSE2 instructions, and leveraging SSE3 and SSE4 instructions for fixed-point implementation. These implementations can enhance the performance of modern CPUs more for deep learning.

Another recent work aims to parallelize Gibbs sampling of hidden and visible units by splitting hidden units and visible units into n machines, each responsible for $1/n$ of the units [47]. In order to make it work, data transfer between machines is required (i.e., when sampling the hidden units, each machine will have the data for all the visible units and vice verse). This method is efficient if both the hidden and visible units are binary and also if the sample size is modest. The communication cost, however, can rise up quickly if large-scale data sets are used. Other methods for large-scale deep learning also explore FPGA-based implementation [48] with a custom architecture: a control unit implemented in a CPU, a grid of multiple full-custom processing tiles, and a fast memory.

In this survey, we will focus on some recently developed deep learning frameworks that take advantage of great computing power available today. Take Graphics Processors Units

(GPUs) as an example: as of August 2013, NVIDIA single precision GPUs exceeded 4.5 TeraFLOP/s with a memory bandwidth of near 300 GB/s [49]. They are particularly suited for massively parallel computing with more transistors devoted for data proceeding needs. These newly developed deep learning frameworks have shown significant advances in making large-scale deep learning practical.

Fig. 3 shows a schematic for a typical CUDA-capable GPU with four multi-processors. Each multi-processor (MP) consists of several streaming multiprocessors (SMs) to form a building block (Fig. 3 shows two SMs for each block). Each SM has multiple stream processors (SPs) that share control logic and low-latency memory. Furthermore, each GPU has a global memory with very high bandwidth and high latency when accessed by the CPU (host). This architecture allows for two levels of parallelism: instruction (memory) level (i.e., MPs) and thread level (SPs). This SIMT (Single Instruction, Multiple Threads) architecture allows for thousands or tens of thousands of threads to be run concurrently, which is best suited for operations with large number of arithmetic operations and small access times to memory. Such levels of parallelism can also be effectively utilized with special attention on the data flow when developing GPU parallel computing applications. One consideration, for example, is to reduce the data transfer between RAM and the GPU's global memory [50] by transferring data with large chunks. This is achieved by uploading as large sets of unlabeled data as possible and by storing free parameters as well as intermediate computations, all in global memory. In addition, data parallelism and learning updates can be implemented by leveraging the two levels of parallelism: input examples can be assigned across MPs, while individual nodes can be treated in each thread (i.e., SPs).

A. LARGE-SCALE DEEP BELIEF NETWORKS

Raina et al. [41] proposed a GPU-based framework for massively parallelizing unsupervised learning models including DBNs (in this paper, they refer the algorithms to stacked RBMs) and sparse coding [21]. While previous models tend to use one to four million free parameters (e.g., Hinton & Salakhutdinov [21] used 3.8 million parameters for free images and Ranzato and Szummer used three million parameters for text processing [51]), the proposed approach can train on more than 100 million free parameters with millions of unlabeled training data [41].

Because transferring data between host and GPU global memory is time consuming, one needs to minimize host-device transfers and take advantage of shared memory. To achieve this, one strategy is to store all parameters and a large chunk of training examples in global memory during training [41]. This will reduce the data transfer times between host and globa memory and also allow for parameter updates to be carried out fully inside GPUs. In addition, to utilize the MP/SP levels of parallelism, a few of the unlabeled training data in global memory will be selected each time to compute the updates concurrently across blocks (data parallelism)

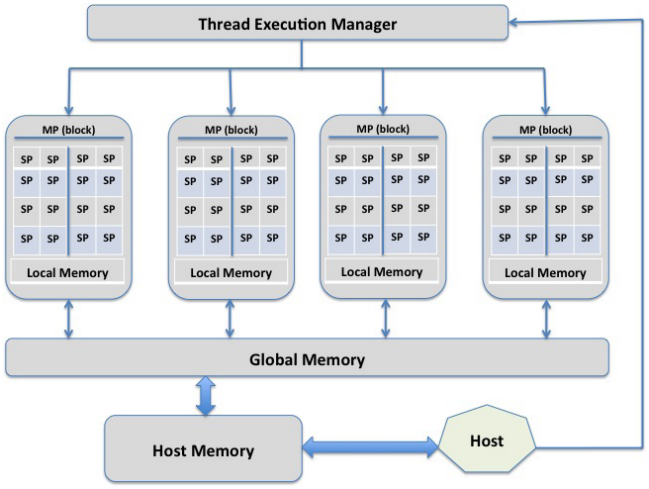

FIGURE 3. An illustrative architecture of a CUDA-capable GPU with highly threaded streaming processors (SPs). In this example, the GPU has 64 stream processors (SPs) organized into four multiprocessors (MPs), each with two stream multiprocessors (SMs). Each SM has eight SPs that share control unit and instruction cache. The four MPs (building blocks) also share a global memory (e.g., graphics double data rate DRAM) that often functions as very-high-bandwidth, off-chip memory (memory bandwidth is the data exchange rate). Global memory typically has high latency and is accessible to the CPU (host). A typical processing flow includes: input data are first copied from host memory to GPU memory, followed by loading and executing GPU program; results are then sent back from GPU memory to host memory. Practically, one needs to pay careful consideration to data transfer between host and GPU memory, which may take considerable amount of time.

(Fig. 3). Meanwhile, each component of the input example is handled by SPs.

When implementing the DBN learning, Gibbs sampling [52], [53] is repeated using Eqs. (1-2). This can be implemented by first generating two sampling matrices $P(h|x)$ and $P(x|h)$, with the (i, j)-th element $P(h_j|x_i)$ (i.e., the probability of j-th hidden node given the i-th input example) and $P(x_j|h_i)$, respectively [41]. The sampling matrices can then be implemented in parallel for the GPU, where each block takes an example and each thread works on an element of the example. Similarly, the weight update operations (Eq. (3)) can be performed in parallel using linear algebra packages for the GPU after new examples are generated.

Experimental results show that with 45 million parameters in a RBM and one million examples, the GPU-based implementation increases the speed of DBN learning by a factor of up to 70, compared to a dual-core CPU implementation (around 29 minutes for GPU-based implementation versus more than one day for CPU-based implementation) [41].

B. LARGE-SCALE CONVOLUTIONAL NEURAL NETWORKS

CNN is a type of locally connected deep learning methods. Large-scale CNN learning is often implemented on GPUs with several hundred parallel processing cores. CNN training involves both forward and backward propagation. For parallelizing forward propagation, one or more blocks are assigned for each feature map depending on the size of maps [36]. Each thread in a block is devoted to a single neuron

in a map. Consequently, the computation of each neuron, which includes convolution of shared weights (kernels) with neurons from the previous layers, activation, and summation, is performed in a SP. The outputs are then stored in the global memory.

Weights are updated by back-propagation of errors δ_k. The error signal $\delta_k^{(l-1)}$ of a neuron k in the previous layer $(l-1)$ depends on the error signals $\delta_j^{(l)}$ of some neurons in a local field of the current layer l. Parallelizing backward propagation can be implemented either by pulling or pushing [36]. Pulling error signals refers to the process of computing delta signals for each neuron in the previous layer by pulling the error signals from the current layer. This is not straightforward because of the subsampling and convolution operations: for example, the neurons in the previous layer may connect to different numbers of neurons in the previous layer due to border effects [54]. For illustration, we plot a one-dimensional convolution and subsampling in Fig. 4. As can be seen, the first six units have different number of connections. We need first to identify the list of neurons in the current layer that contribute to the error signals of neurons in the previous layer. On the contrary, all the units in the current layer have exactly the same number of incoming connections. Consequently, pushing the error signals from the current layer to previous layer is more efficient, i.e., for each unit in the current layer, we update the related units in the previous layer.

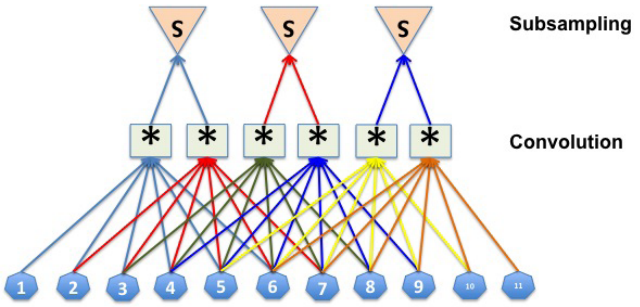

FIGURE 4. An illustration of the operations involved with 1D convolution and subsampling. The convolution filter's size is six. Consequently, each unit in the convolution layer is defined by six input units. Subsampling involves averaging two adjacent units in the convolution layer.

For implementing data parallelism, one needs to consider the size of global memory and feature map size. Typically, at any given stage, a limited number of training examples can be processed in parallel. Furthermore, within each block where comvolution operation is performed, only a portion of a feature map can be maintained at any given time due to the extremely limited amount of shared memory. For convolution operations, Scherer et al. suggested the use of limited shared memory as a circular buffer [37], which only holds a small portion of each feature map loaded from global memory each time. Convolution will be performed by threads in parallel and results are written back to global memory. To further overcome the GPU memory limitation, the authors implemented a modified architecture with both the convolution and subsampling operations being combined into one step [37]. This modification allows for storing both the activities and error values with reduced memory usage while running backpropagation.

To further speedup, Krizhevsky et al. proposed the use of two GPUs for training CNNs with five convolutional layers and three fully connected classification layers. The CNN uses Rectified Linear Units (ReLUs) as the nonlinear function $(f(x) = \max(0, x))$, which has been shown to run several times faster than other commonly used functions [55]. For some layers, about half of the network is computed in a single GPU and the other portion is calculated in the other GPU; the two GPUs communicated at some other layers. This architecture takes full advantage of cross-GPU parallelization that allows two GPUs to communicate and transfer data without using host memory.

C. COMBINATION OF DATA- AND MODEL-PARALLEL SCHEMES

DistBelief is a software framework recently designed for distributed training and learning in deep networks with very large models (e.g., a few billion parameters) and large-scale data sets. It leverages large-scale clusters of machines to manage both data and model parallelism via multithreading, message passing, synchronization as well as communication between machines [56].

For large-scale data with high dimensionality, deep learning often involves many densely connected layers with a large number of free parameters (i.e., large models). To deal with large model learning, DistBelief first implements model parallelism by allowing users to partition large network architectures into several smaller structures (called blocks), whose nodes will be assigned to and calculated in several machines (collectively we call it "a partitioned model"). Each block will be assigned to one machine (see Fig. 5). Boundary nodes (nodes whose edges belong to more than one partitions) require data transfer between machines. Apparently, fully-connected networks have more boundary nodes and often demand higher communication costs than locally-connected structures, and thus less performance benefits. Nevertheless, as many as 144 partitions have been reported for large models in DistBelief [56], which leads to significant improvement of training speed.

DistBelief also implements data parallelism and employs two separate distributed optimization procedures: Downpour stochastic gradient descent (SGD) and Sandblaster [56], which perform online and batch optimization, respectively. Herein we will discuss Downpour in details and more information about Sandblaster can be found in the reference [56].

First, multiple replicas of the partitioned model will be created for training and inference. Like deep learning models, large data sets will be partitioned into many subsets. DistBelief will then run multiple replicas of the partitioned model to compute gradient descent via Downpour SGD on different subsets of training data. Specifically, DistBelief employs a centralized parameter server storing and applying updates for

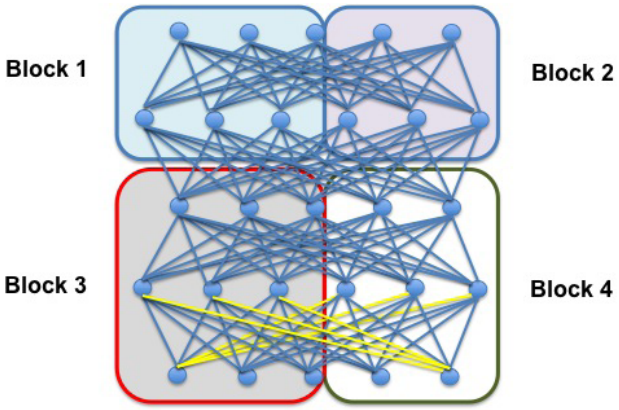

FIGURE 5. DistBelief: models are partitioned into four blocks and consequently assigned to four machines [56]. Information for nodes that belong to two or more partitions is transferred between machines (e.g., the lines marked with yellow color). This model is more effective for less densely connected networks.

all parameters of the models. Parameters are grouped into server shards. At any given time, each machine in a partitioned model needs only to communicate with the parameter server shards that hold the relevant parameters. This communication is asynchronous: each machine in a partitioned model runs independently and each parameter server shard acts independently as well. One advantage of using asynchronous communication over standard synchronous SGD is its fault tolerance: in the event of the failures of one machine in a model copy, other model replicas will continue communicating with the central parameter server to process the data and update the shared weights.

In practice, the Adagrad adaptive learning rate procedure [57] is integrated into the Downpour SGD for better performance. DistBelief is implemented in two deep learning models: a fully connected network with 42 million model parameters and 1.1 billion examples, and a locally-connected convolutional neural network with 16 million images of 100 by 100 pixels and 21,000 categories (as many as 1.7 billion parameters). The experimental results show that locally connected learning models will benefit more from DistBelief: indeed, with 81 machines and 1.7 billion parameters, the method is 12x faster than using a single machine. As demonstrated in [56], a significant advantage of DistBelief is its ability to scale up from single machine to thousands of machines, which is the key to Big Data analysis.

Most recently, the DistBelief framework was used to train a deep architecture with a sparse deep autoencoder, local receptive fields, pooling, and local contrast normalization [50]. The deep learning architecture consists of three stacked layers, each with sublayers of local filtering, local pooling, and local contrast normalization. The filtering sublayers are not convolutional, each filter with its own weights. The optimization of this architecture involves an overall objective function that is the summation of the objective functions for the three layers, each aiming at minimizing a reconstruction error while maintaining sparsity of connections between sublayers. The DistBelief framework is able to scale up the dataset, the model, and the resources all together. The model is partitioned into 169 machines, each with 16 CPU cores. Multiple cores allow for another level of parallelism where each subset of cores can perform different tasks. Asynchronous SGD is implemented with several replicas of the core model and mini-batch of training examples. The framework was able to train as many as 14 million images with a size of 200 by 200 pixels and more than 20 thousand categories for three days over a cluster of 1,000 machines with 16,000 cores. The model is capable of learning high-level features to detect objects without using labeled data.

D. THE COTS HPC SYSTEMS

While DistBelief can learn with very large models (more than one billion parameters), its training requires 16,000 CPU cores, which are not commonly available for most researchers. Most recently, Coates et al. presented an alternative approach that trains comparable deep network models with more than 11 billion free parameters by using just three machines [58]. The Commodity Off-The-Shelf High Performance Computing (COTS HPC) system is comprised of a cluster of 16 GPU servers with Infiniband adapter for interconnects and MPI for data exchange in a cluster. Each server is equipped with four NVIDIA GTX680 GPUs, each having 4GB of memory. With well-balanced number of GPUs and CPUs, COTS HPC is capable of running very large-scale deep learning.

The implementation includes carefully designed CUDA kernels for effective usage of memory and efficient computation. For example, to efficiently compute a matrix multiplication $Y = WX$ (e.g., W is the filter matrix and X is the input matrix), Coates et al. [58] fully take advantage of matrix sparseness and local receptive field by extracting non-zero columns in W for neurons that share identical receptive fields, which are then multiplied by the corresponding rows in X. This strategy successfully avoids the situation where the requested memory is larger than the shared memory of the GPU. In addition, matrix operations are performed by using a highly optimized tool called MAGMA BLAS matrix-matrix multiply kernels [59].

Furthermore, GPUs are being utilized to implement a model parallel scheme: each GPU is only used for a different part of the model optimization with the same input examples; collectively, their communication occurs through the MVAPICH2 MPI. This very large scale deep learning system is capable of training with more than 11 billion parameters, which is the largest model reported by far, with much less machines.

Table 1 summarizes the current progress in large-scale deep learning. It has been observed in several groups (see [41]) that single CPU is impractical for deep learning with a large model. With multiple machines, the running time may not be a big concern any more (see [56]). However,

significant computational resources are needed to achieve the goal. Consequently, major research efforts are towards experiments with GPUs.

TABLE 1. Summary of recent research progress in large-scale deep learning.

Methods	Computing Power	Number of examples and free parameters	Average running time
DBN [41]	NVIDIA GTX 280 GPU with 1 GB mwmory	One million images and 100 minllion parameters	~ 1 day
CNN [55]	Two GTX 580 GPUs, each with 3GB memory	1.2 million high resolution (256 x 256) images and 60 million parameters	~ 5-6 days
DisBelief [56]	1,000 CPUs with Downpour SGD with Adagrad	1.1 billion audio examples and 42 million model parameters	~16 hours
Sparse autoencoder [50]	1,000 CPUs with 16,000 cores	10 million 200 x 200 pixel images and one billion parameters	~3 days
COTS HPC [58]	64 NVIDIA GTX 680 GPUs, each with 4GB memory	10 million 200 x 200 images and 11 billion parameters	~3 days

IV. REMAINING CHALLENGES AND PERSPECTIVES: DEEP LEARNING FOR BIG DATA

In recent years, Big Data has taken center stage in government and society at large. In 2012, the Obama Administration announced a ''Big Data Research and Development Initiative'' to ''help solve some of the Nation's most pressing challenges'' [60]. Consequently, six Federal departments and agencies (NSF, HHS/NIH, DOD, DOE, DARPA, and USGS) committed more than $200 million to support projects that can transform our ability to harness in novel ways from huge volumes of digital data. In May of the same year, the state of Massachusetts announced the Massachusetts Big Data Initiative that funds a variety of research institutions [61]. In April, 2013, U.S. President Barack Obama announced another federal project, a new brain mapping initiative called the BRAIN (Brain Research Through Advancing Innovative Neurotechnologies) [62] aiming to develop new tools to help map human brain functions, understand the complex links between function and behavior, and treat and cure brain disorders. This initiative might test and extend the current limits of technologies for Big Data collection and analysis, as NIH director Francis Collins stated that collection, storage, and processing of yottabytes (a billion petabytes) of data would eventually be required for this initiative.

While the potential of Big Data is undoubtedly significant, fully achieving this potential requires new ways of thinking and novel algorithms to address many technical challenges. For example, most traditional machine learning algorithms were designed for data that would be completely loaded into memory. With the arrival of Big Data age, however, this assumption does not hold any more. Therefore, algorithms that can learn from massive amounts of data are needed.

In spite of all the recent achievement in large-scale deep learning as discussed in Section 3, this field is still in its infancy. Much more needs to be done to address many significant challenges posted by Big Data, often characterized by the three V's model: volume, variety, and velocity [63], which refers to large scale of data, different types of data, and the speed of streaming data, respectively.

A. DEEP LEARNING FROM HIGH VOLUMES OF DATA

First and foremost, high volumes of data present a great challenging issue for deep learning. Big data often possesses a large number of examples (inputs), large varieties of class types (outputs), and very high dimensionality (attributes). These properties directly lead to running-time complexity and model complexity. The sheer volume of data makes it often impossible to train a deep learning algorithm with a central processor and storage. Instead, distributed frameworks with parallelized machines are preferred. Recently, impressive progresses have been made to mitigate the challenges related to high volumes. The novel models utilize clusters of CPUs or GPUs in increasing the training speed without scarifying accuracy of deep learning algorithms. Strategies for data parallelism or model parallelism or both have been developed. For example, data and models are divided into blocks that fit with in-memory data; the forward and backward propagations can be implemented effectively in parallel [56], [58], although deep learning algorithms are not trivially parallel.

The most recent deep learning framework can handle a significantly large number of samples and parameters. It is also possible to scale up with more GPUs used. It is less clear, however, how the deep learning systems can continue scaling significantly beyond the current framework. While we can expect the continuous growth in computer memory and computational power (mainly through parallel or distributed computing environment), further research and effort on addressing issues associated with computation and communication management (e.g., copying data or parameters or gradient values to different machines) are needed for scaling-up to very large data sets. Ultimately, to build the future deep learning system scalable to Big Data, one needs to develop high performance computing infrastructure-based systems together with theoretically sound parallel learning algorithms or novel architectures.

Another challenge associated with high volumes is the data incompleteness and noisy labels. Unlike most conventional datasets used for machine learning, which were highly curated and noise free, Big Data is often incomplete resulting from their disparate origins. To make things even more complicated, majority of data may not be labeled, or if labeled, there exist noisy labels. Take the 80 million tiny

image database as an example, which has 80 million low-resolution color images over 79,000 search terms [64]. This image database was created by searching the Web with every non-abstract English noun in the WordNet. Several search engines such as Google and Flickr were used to collect the data over the span of six months. Some manual curation was conducted to remove duplicates and low-quality images. Still, the image labels are extremely unreliable because of search technologies.

One of the unique characteristics deep learning algorithms possess is their ability to utility unlabeled data during training: learning data distribution without using label information. Thus, the availability of large unlabeled data presents ample opportunities for deep learning methods. While data incompleteness and noisy labels are part of the Big Data package, we believe that using vastly more data is preferable to using smaller number of exact, clean, and carefully curated data. Advanced deep learning methods are required to deal with noisy data and to be able to tolerate some messiness. For example, a more efficient cost function and novel training strategy may be needed to alleviate the effect of noisy labels. Strategies used in semi-supervised learning [65]–[68] may also help alleviate problems related to noisy labels.

B. DEEP LEARNING FOR HIGH VARIETY OF DATA

The second dimension for Big Data is its variety, i.e., data today comes in all types of formats from a variety sources, probably with different distributions. For example, the rapidly growing multimedia data coming from the Web and mobile devices include a huge collection of still images, video and audio streams, graphics and animations, and unstructured text, each with different characteristics. A key to deal with high variety is data integration. Clearly, one unique advantage of deep learning is its ability for representation learning – with either supervised or unsupervised methods or combination of both, deep learning can be used to learn good feature representations for classification. It is able to discover intermediate or abstract representations, which is carried out using unsupervised learning in a hierarchy fashion: one level at a time and higher-level features defined by lower-level features. Thus, a natural solution to address the data integration problem is to learn data representations from each individual data sources using deep learning methods, and then to integrate the learned features at different levels.

Deep learning has been shown to be very effective in integrating data from different sources. For example, Ngiam et al. [69] developed a novel application of deep learning algorithms to learn representations by integrating audio and video data. They demonstrated that deep learning is generally effective in (1) learning single modality representations through multiple modalities with unlabeled data and (2) learning shared representations capable of capturing correlations across multiple modalities. Most recently, Srivastava and Salakhutdinov [70] developed a multimodal Deep Boltzmann Machine (DBM) that fuses two very different data modalities, real-valued dense image data and text data with sparse word frequencies, together to learn a unified representation. DBM is a generative model without fine-tuning: it first builds multiple stacked-RBMs for each modality; to form a multimodal DBM, an additional layer of binary hidden units is added on top of these RBMs for joint representation. It learns a joint distribution in the multimodal input space, which allows for learning even with missing modalities.

While current experiments have demonstrated that deep learning is able to utilize heterogeneous sources for significant gains in system performance, numerous questions remain open. For example, given that different sources may offer conflicting information, how can we resolve the conflicts and fuse the data from different sources effectively and efficiently. While current deep learning methods are mainly tested upon bi-modalities (i.e., data from two sources), will the system performance benefits from significantly enlarged modalities? Furthermore, at what levels in deep learning architectures are appropriate for feature fusion with heterogeneous data? Deep learning seems well suited to the integration of heterogeneous data with multiple modalities due to its capability of learning abstract representations and the underlying factors of data variation.

C. DEEP LEARNING FOR HIGH VELOCITY OF DATA

Emerging challenges for Big Data learning also arose from high velocity: data are generating at extremely high speed and need to be processed in a timely manner. One solution for learning from such high velocity data is online learning approaches. Online learning learns one instance at a time and the true label of each instance will soon be available, which can be used for refining the model [71]–[76]. This sequential learning strategy particularly works for Big Data as current machines cannot hold the entire dataset in memory. While conventional neural networks have been explored for online learning [77]–[87], only limited progress on online deep learning has been made in recent years. Interestingly, deep learning is often trained with stochastic gradient descent approach [88], [89], where one training example with the known label is used at a time to update the model parameters. This strategy may be adapted for online learning as well. To speed up learning, instead of proceeding sequentially one example at a time, the updates can be performed on a mini-batch basis [37]. Practically, the examples in each mini-batch are as independent as possible. Mini-batches provide a good balance between computer memory and running time.

Another challenging problem associated with the high velocity is that data are often non-stationary, i.e., data distribution is changing over time. Practically, non-stationary data are normally separated into chunks with data from a small time interval. The assumption is that data close in time are piece-wise stationary and may be characterized by a significant degree of correlation and, therefore, follow the same distribution [90]–[97]. Thus, an important feature of a deep learning algorithm for Big Data is the ability to learn the data as a stream. One area that needs to be explored is deep online learning – online learning often scales naturally and

is memory bounded, readily parallelizable, and theoretically guaranteed [98]. Algorithms capable of learning from non-i.i.d. data are crucial for Big Data learning.

Deep learning can also leverage both high variety and velocity of Big Data by transfer learning or domain adaption, where training and test data may be sampled from different distributions [99]–[107]. Recently, Glorot et al. implemented a stacked denoising auto-encoder based deep architecture for domain adaption, where one trains an unsupervised representation on a large number of unlabeled data from a set of domains, which is applied to train a classifier with few labeled examples from only one domain [100]. Their empirical results demonstrated that deep learning is able to extract a meaningful and high-level representation that is shared across different domains. The intermediate high-level abstraction is general enough to uncover the underlying factors of domain variations, which is transferable across domains. Most recently, Bengio also applied deep learning of multiple level representations for transfer learning where training examples may not well represent test data [99]. They showed that more abstract features discovered by deep learning approaches are most likely generic between training and test data. Thus, deep learning is a top candidate for transfer learning because of its ability to identify shared factors present in the input.

Although preliminary experiments have shown much potential of deep learning in transfer learning, applying deep learning to this field is relatively new and much more needs to be done for improved performance. Of course, the big question is whether we can benefit from Big Data with deep architectures for transfer learning.

In conclusion, Big Data presents significant challenges to deep learning, including large scale, heterogeneity, noisy labels, and non-stationary distribution, among many others. In order to realize the full potential of Big Data, we need to address these technical challenges with new ways of thinking and transformative solutions. We believe that these research challenges posed by Big Data are not only timely, but will also bring ample opportunities for deep learning. Together, they will provide major advances in science, medicine, and business.

REFERENCES

[1] National Security Agency. *The National Security Agency: Missions, Authorities, Oversight and Partnerships* [Online]. Available: http://www.nsa.gov/public_info/_files/speeches_testimonies/2013_08_09_the_nsa_story.pdf

[2] J. Gantz and D. Reinsel, *Extracting Value from Chaos*. Hopkinton, MA, USA: EMC, Jun. 2011.

[3] J. Gantz and D. Reinsel, *The Digital Universe Decade—Are You Ready*. Hopkinton, MA, USA: EMC, May 2010.

[4] (2011, May). *Big Data: The Next Frontier for Innovation, Competition, and Productivity*. McKinsey Global Institute [Online]. Available: http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation

[5] J. Lin and A. Kolcz, "Large-scale machine learning at twitter," in *Proc. ACM SIGMOD*, Scottsdale, Arizona, USA, 2012, pp. 793–804.

[6] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 703–710, 2010.

[7] A. Ng *et al.*, "Map-reduce for machine learning on multicore," in *Proc. Adv. Neural Inf. Procees. Syst.*, vol. 19. 2006, pp. 281–288.

[8] B. Panda, J. Herbach, S. Basu, and R. Bayardo, "MapReduce and its application to massively parallel learning of decision tree ensembles," in *Scaling Up Machine Learning: Parallel and Distributed Approaches*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[9] E. Crego, G. Munoz, and F. Islam. (2013, Dec. 8). Big data and deep learning: Big deals or big delusions? *Business* [Online]. Available: http://www.huffingtonpost.com/george-munoz-frank-islam-and-ed-crego/big-data-and-deep-learnin_b_3325352.html

[10] Y. Bengio and S. Bengio, "Modeling high-dimensional discrete data with multi-layer neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12. 2000, pp. 400–406.

[11] Y. Marc'Aurelio Ranzato, L. Boureau, and Y. LeCun, "Sparse feature learning for deep belief networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 20. 2007, pp. 1185–1192.

[12] 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, Lang. Process.*, vol. 20, no. 1, pp. 30–41, Jan. 2012.

[13] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[14] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 791–798.

[15] D. Cireşan, U. Meler, L. Cambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Comput.*, vol. 22, no. 12, pp. 3207–3220, 2010.

[16] M. Zeiler, G. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *Proc. IEEE Int. Conf. Comput. Vis.*, Nov. 2011, pp. 2018–2025.

[17] A. Efrati. (2013, Dec. 11). How 'deep learning' works at Apple, beyond. *Information* [Online]. Available: https://www.theinformation.com/How-Deep-Learning-Works-at-Apple-Beyond

[18] N. Jones, "Computer science: The learning machines," *Nature*, vol. 505, no. 7482, pp. 146–148, 2014.

[19] Y. Wang, D. Yu, Y. Ju, and A. Acero, "Voice search," in *Language Understanding: Systems for Extracting Semantic Information From Speech*, G. Tur and R. De Mori, Eds. New York, NY, USA: Wiley, 2011, ch. 5.

[20] J. Kirk. (2013, Oct. 1). Universities, IBM join forces to build a brain-like computer. *PCWorld* [Online]. Available: http://www.pcworld.com/article/2051501/universities-join-ibm-in-cognitive-computing-research-project.html

[21] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[22] Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

[23] V. Nair and G. Hinton, "3D object recongition with deep belief nets," in *Proc. Adv. NIPS*, vol. 22. 2009, pp. 1339–1347.

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

[25] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing almost from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Nov. 2011.

[26] P. Le Callet, C. Viard-Gaudin, and D. Barba, "A convolutional neural network approach for objective video quality assessment," *IEEE Trans. Neural Netw.*, vol. 17, no. 5, pp. 1316–1327, Sep. 2006.

[27] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[28] G. Hinton, "A practical guide to training restricted Boltzmann machines," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep. UTML TR 2010-003, 2010.

[29] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1327–1554, 2006.

[30] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Neural Inf. Process. Syst.*, 2006, pp. 153–160.

[31] G. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, 2002.

[32] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol "Extracting and composing robust features with denoising autoencoders," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1096–1103.

[33] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *J. Mach. Learn. Res.*, vol. 10, pp. 1–40, Jan. 2009.
[34] H. Lee, A. Battle, R. Raina, and A. Ng, "Efficient sparse coding algorithms," in *Proc. Neural Inf. Procees. Syst.*, 2006, pp. 801–808.
[35] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Interspeech*, 2011, pp. 437–440.
[36] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proc. 22nd Int. Conf. Artif. Intell.*, 2011, pp. 1237–1242.
[37] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Proc. Int. Conf. Artif. Neural Netw.*, 2010, pp. 92–101.
[38] Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*, G. Orr and K. Muller, Eds. New York, NY, USA: Springer-Verlag, 1998.
[39] K. Kavukcuoglu, M. A. Ranzato, R. Fergus, and Y. LeCun, "Learning invariant features through topographic filter maps," in *Proc. Int. Conf. CVPR*, 2009, pp. 1605–1612.
[40] D. Hubel and T. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *J. Physiol.*, vol. 195, pp. 215–243, Mar. 1968.
[41] R. Raina, A. Madhavan, and A. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proc. 26th Int. Conf. Mach. Learn.*, Montreal, QC, Canada, 2009, pp. 873–880.
[42] J. Martens, "Deep learning via Hessian-free optimization," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010.
[43] K. Zhang and X. Chen, "Large-scale deep belief nets with MapReduce," *IEEE Access*, vol. 2, pp. 395–403, Apr. 2014.
[44] L. Deng, D. Yu, and J. Platt, "Scalable stacking and learning for building deep architectures," in *Proc. IEEE ICASSP*, Mar. 2012, pp. 2133–2136.
[45] B. Hutchinson, L. Deng, and D. Yu, "Tensor deep stacking networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1944–1957, Aug. 2013.
[46] V. Vanhoucke, A. Senior, and M. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learn. Unsupervised Feature Learn. Workshop*, 2011.
[47] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.
[48] C. Farabet *et al.*, "Large-scale FPGA-based convolutional networks," in *Machine Learning on Very Large Data Sets*, R. Bekkerman, M. Bilenko, and J. Langford, Eds. Cambridge, U.K.: Cambridge Univ. Press, 2011.
[49] *CUDA C Programming Guide, PG-02829-001_v5.5*, NVIDIA Corporation, Santa Clara, CA, USA, Jul. 2013.
[50] Q. Le *et al.*, "Building high-level features using large scale unsupervised learning," in *Proc. Int. Conf. Mach. Learn.*, 2012.
[51] M. Ranzato and M. Szummer, "Semi-supervised learning of compact document representations with deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 792–799.
[52] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, no. 6, pp. 721–741, Nov. 1984.
[53] G. Casella and E. George, "Explaining the Gibbs sampler," *Amer. Statist.*, vol. 46, no. 3, pp. 167–174, 1992.
[54] P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proc. 7th ICDAR*, 2003, pp. 958–963.
[55] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. NIPS*, 2012, pp. 1106–1114.
[56] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. NIPS*, 2012, pp. 1232–1240.
[57] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011.
[58] A. Coats, B. Huval, T. Wng, D. Wu, and A. Wu, "Deep Learning with COTS HPS systems," *J. Mach. Learn. Res.*, vol. 28, no. 3, pp. 1337–1345, 2013.
[59] S. Tomov, R. Nath, P. Du, and J. Dongarra. (2011). MAGMA users guide. ICL, Univ. Tennessee, Knoxville, TN, USA [Online]. Available: http://icl.cs.utk.edu/magma
[60] (2012). *Obama Administration Unveils 'Big Data' Initiative Announces $200 Million in New R&D Investments*. Office of Science and Technology Policy, Executive Office of the President, Washington, DC, USA [Online]. Available: http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_press_release_final_2.pdf
[61] K. Haberlin, B. McGilpin, and C. Ouellette. *Governor Patrick Announces New Initiative to Strengthen Massachusetts' Position as a World Leader in Big Data*. Commonwealth of Massachusetts [Online]. Available: http://www.mass.gov/governor/pressoffice/pressreleases/2012/2012530-governor-announces-big-data-initiative.html
[62] *Fact Sheet: Brain Initiative*, Office of the Press Secretary, The White House, Washington, DC, USA, 2013.
[63] D. Laney, *The Importance of 'Big Data': A Definition*. Stamford, CT, USA: Gartner, 2012.
[64] A. Torralba, R. Fergus, and W. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008.
[65] J. Wang and X. Shen, "Large margin semi-supervised learning," *J. Mach. Learn. Res.*, vol. 8, no. 8, pp. 1867–1891, 2007.
[66] J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," in *Proc. 25th Int. Conf. Mach. Learn.*, Helsinki, Finland, 2008.
[67] K. Sinha and M. Belkin, "Semi-supervised learning using sparse eigenfunction bases," in *Proc. Adv. NIPS*, 2009, pp. 1687–1695.
[68] R. Fergus, Y. Weiss, and A. Torralba, "Semi-supervised learning in gigantic image collections," in *Proc. Adv. NIPS*, 2009, pp. 522–530.
[69] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Ng, "Multimodal deep learning," in *Proc. 28th Int. Conf. Mach. Learn.*, Bellevue, WA, USA, 2011.
[70] N. Srivastava and R. Salakhutdinov, "Multimodal learning with deep Boltzmann machines," in *Proc. Adv. NIPS*, 2012.
[71] L. Bottou, "Online algorithms and stochastic approximations," in *On-Line Learning in Neural Networks*, D. Saad, Ed. Cambridge, U.K.: Cambridge Univ. Press, 1998.
[72] A. Blum and C. Burch, "On-line learning and the metrical task system problem," in *Proc. 10th Annu. Conf. Comput. Learn. Theory*, 1997, pp. 45–53.
[73] N. Cesa-Bianchi, Y. Freund, D. Helmbold, and M. Warmuth, "On-line prediction and conversation strategies," in *Proc. Conf. Comput. Learn. Theory Eurocolt*, vol. 53. Oxford, U.K., 1994, pp. 205–216.
[74] Y. Freund and R. Schapire, "Game theory, on-line prediction and boosting," in *Proc. 9th Annu. Conf. Comput. Learn. Theory*, 1996, pp. 325–332.
[75] N. Littlestone, P. M. Long, and M. K. Warmuth, "On-line learning of linear functions," in *Proc. 23rd Symp. Theory Comput.*, 1991, pp. 465–475.
[76] S. Shalev-Shwartz, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, no. 2, pp. 107–194, 2012.
[77] T. M. Heskes and B. Kappen, "On-line learning processes in artificial neural networks," *North-Holland Math. Library*, vol. 51, pp. 199–233, 1993.
[78] R. Marti and A. El-Fallahi, "Multilayer neural networks: An experimental evaluation of on-line training methods," *Comput. Operat. Res.*, vol. 31, no. 9, pp. 1491–1513, 2004.
[79] C. P. Lim and R. F. Harrison, "Online pattern classification with multiple neural network systems: An experimental study," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 33, no. 2, pp. 235–247, May 2003.
[80] M. Rattray and D. Saad, "Globally optimal on-line learning rules for multi-layer neural networks," *J. Phys. A, Math. General*, vol. 30, no. 22, pp. L771–776, 1997.
[81] P. Riegler and M. Biehl, "On-line backpropagation in two-layered neural networks," *J. Phys. A*, vol. 28, no. 20, pp. L507–L513, 1995.
[82] D. Saad and S. Solla, "Exact solution for on-line learning in multilayer neural networks," *Phys. Rev. Lett.*, vol. 74, no. 21, pp. 4337–4340, 1995.
[83] A. West and D. Saad, "On-line learning with adaptive back-propagation in two-layer networks," *Phys. Rev. E*, vol. 56, no. 3, pp. 3426–3445, 1997.
[84] P. Campolucci, A. Uncini, F. Piazza, and B. Rao, "On-line learning algorithms for locally recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 10, no. 2, pp. 253–271, Mar. 1999.
[85] N. Liang, G. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1411–1423, Nov. 2006.
[86] V. Ruiz de Angulo and C. Torras, "On-line learning with minimal degradation in feedforward networks," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 657–668, May 1995.

[87] M. Choy, D. Srinivasan, and R. Cheu, "Neural networks for continuous online learning and control," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1511–1531, Nov. 2006.
[88] L. Bottou and O. Bousequet, "Stochastic gradient learning in neural networks," in *Proc. Neuro-Nimes*, 1991.
[89] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub-gradient solver for SVM," in *Proc. Int. Conf. Mach. Learn.*, 2007.
[90] J. Chien and H. Hsieh, "Nonstationary source separation using sequential and variational Bayesian learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 5, pp. 681–694, May 2013.
[91] M. Sugiyama and M. Kawanabe, *Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation*. Cambridge, MA, USA: MIT Press, Mar. 2012.
[92] R. Elwell and R. Polikar, "Incremental learning in nonstationary environments with controlled forgetting," in *Proc. Int. Joint Conf. Neural Netw.*, 2009, pp. 771–778.
[93] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.
[94] C. Alippi and M. Roveru, "Just-in-time adaptive classifiers—Part I: Detecting nonstationary changes," *IEEE Trans. Neural Netw.*, vol. 19, no. 7, pp. 1145–1153, Jul. 2008.
[95] C. Alippi and M. Roveru, "Just-in-time adaptive classifiers—Part II: Designing the classifier," *IEEE Trans. Neural Netw.*, vol. 19, no. 12, pp. 2053–2064, Dec. 2008.
[96] L. Rutkowski, "Adaptive probabilistic neural networks for pattern classification in time-varying environment," *IEEE Trans. Neural Netw.*, vol. 15, no. 4, pp. 811–827, Jul. 2004.
[97] W. de Oliveira, "The Rosenblatt Bayesian algorithm learning in a nonstationary environment," *IEEE Trans. Neural Netw.*, vol. 18, no. 2, pp. 584–588, Mar. 2007.
[98] P. Bartlett, "Optimal online prediction in adversarial environments," in *Proc. 13th Int. Conf. DS*, 2010, p. 371.
[99] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning," *J. Mach. Learn. Res.*, vol. 27, pp. 17–37, 2012.
[100] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *Proc. 28th Int. Conf. Mach. Learn.*, Bellevue, WA, USA, 2011.
[101] G. Mesnil *et al.*, "Unsupervised and transfer learning challenge: A deep learning approach," *J. Mach. Learn. Res.*, vol. 7, pp. 1–15, 2011.
[102] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
[103] S. Gutstein, O. Fuentes, and E. Freudenthal, "Knowledge transfer in deep convolutional neural nets," *Int. J. Artif. Intell. Tools*, vol. 17, no. 3, pp. 555–567, 2008.
[104] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proc. 11th Annu. Conf. Comput. Learn. Theory*, 1998, pp. 92–100.
[105] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: Transfer learning from unlabeled data," in *Proc. 24th ICML*, 2007.
[106] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Trans. Neural Netw.*, vol. 22, no. 2, pp. 199–210, Feb. 2011.
[107] G. Mesnil, S. Rifai, A. Bordes, X. Glorot, Y. Bengio, and P. Vincent, "Unsupervised and transfer learning under uncertainty: From object detections to scene categorization," in *Proc. ICPRAM*, 2013, pp. 345–354.

XUE-WEN CHEN (M'00–SM'03) is currently a Professor and the Chair with the Department of Computer Science, Wayne State University, Detroit, MI, USA. He received the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, USA, in 2001. He is currently serving as an Associate Editor or an Editorial Board Member for several international journals, including IEEE Access, *BMC Systems Biology*, and the IEEE Transactions on Information Technology in Biomedicine. He served as a Conference Chair or Program Chair for a number of conferences such as the 21st ACM Conference on Information and Knowledge Management in 2012 and the 10th IEEE International Conference on Machine Learning and Applications in 2011. He is a Senior Member of the IEEE Computer Society.

XIAOTONG LIN is currently a Visiting Assistant Professor with the Department of Computer Science and Engineering, Oakland University, Rochester, MI, USA. She received the Ph.D. degree from the University of Kansas, Lawrence, KS, USA, in 2012, and the M.Sc. degree from the University of Pittsburgh, Pittsburgh, PA, USA, in 1999. Her research interests include large scale machine learning, data mining, high-performance computing, and bioinformatics.