Information Fusion 42 (2018) 146–157

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

A survey on deep learning for big data

Qingchen Zhang[a,b], Laurence T. Yang[*,a,b], Zhikui Chen[c], Peng Li[c]

[a] School of Electronic Engineering, University of Electronic Science and Technology of China China
[b] Department of Computer Science, St. Francis Xavier University,Antigonish, Canada
[c] School of Software Technology, Dalian University of Technology, Dalian,China

ARTICLE INFO

Keywords:
Deep learning
Big data
Stacked auto-encoders
Deep belief networks
Convolutional neural networks
Recurrent neural networks

ABSTRACT

Deep learning, as one of the most currently remarkable machine learning techniques, has achieved great success in many applications such as image analysis, speech recognition and text understanding. It uses supervised and unsupervised strategies to learn multi-level representations and features in hierarchical architectures for the tasks of classification and pattern recognition. Recent development in sensor networks and communication technologies has enabled the collection of big data. Although big data provides great opportunities for a broad of areas including e-commerce, industrial control and smart medical, it poses many challenging issues on data mining and information processing due to its characteristics of large volume, large variety, large velocity and large veracity. In the past few years, deep learning has played an important role in big data analytic solutions. In this paper, we review the emerging researches of deep learning models for big data feature learning. Furthermore, we point out the remaining challenges of big data deep learning and discuss the future topics.

1. Introduction

Recently, the cyber-physical-social systems, together with the sensor networks and communication technologies, have made a great progress, enabling the collection of big data [1,2]. Big data can be defined by its four characteristics, i.e., large volume, large variety, large velocity and large veracity, which is usually called 4V's model [3–5]. The most remarkable characteristic of big data is large-volume that implies an explosive in the data amount. For example, Flicker generates about 3.6 TB data and Google processes about 20,000 TB data everyday. The National Security Agency reports that approximately 1.8 PB data is gathered on the Internet everyday. One distinctive characteristic of big data is large variety that indicates the different types of data formats including text, images, videos, graphics, and so on. Most of the traditional data is in the structured format and it is easily stored in the two-dimensional tables. However, more than 75% of big data is unstructured. Typical unstructured data is multimedia data collected from the Internet and mobile devices [6]. Large velocity argues that big data is generating fast and requires to be processed in real time. The real-time analysis of big data is crucial for e-commerce to provide the online services. Another important characteristic of big data is large veracity that refers to the existence of a huge number of noisy objects, incomplete objects, inaccurate objects, imprecise objects and redundant objects [7]. The size of big data is continuing to grow at an unprecedented rate and is will reach 35 ZB by 2020. However, only having massive data is inadequate. For most of the applications such as industry and medical, the key is to find and extract valuable knowledge from big data for prediction services support. Take the physical devices that suffer mechanical malfunctions occasionally in the industrial manufacturing for an example. If we can analyze the collected parameters of devices effectively before the devices break down, we can take the immediate actions to avoid the catastrophe. While big data provides great opportunities for a broad of areas including e-commerce, industrial control and smart medical, it poses many challenging issues on data mining and information processing. Actually, it is difficult for traditional methods to analyze and process big data effectively and efficiently due to the large variety and the large veracity.

Deep learning is playing an important role in big data solutions since it can harvest valuable knowledge from complex systems [8]. Specially, deep learning has become one of the most active research points in the machine learning community since it was presented in 2006 [9–11]. Actually, deep learning can track back to the 1940s. However, traditional training strategies for multi-layer neural networks always result in a locally optimal solution or cannot guarantee the convergence. Therefore, the multi-layer neural networks have not received wide applications even though it was realized that the multi-layer neural networks could achieve the better performance for feature and representation learning. In 2006, Hinton et al. [12] proposed a two-stage strategy, pre-training and fine-tuning, for training deep learning effectively, causing the first back-through of deep learning. In addition,

* Corresponding author.
E-mail address: ltyang@stfx.ca (L.T. Yang).

http://dx.doi.org/10.1016/j.inffus.2017.10.006
Received 30 August 2017; Accepted 17 October 2017
Available online 11 November 2017

the increase of computing power and data size also contributes to the popularity of deep learning. As the era of big data comes, a large number of samples can be collected to train the parameters of deep learning models. Meanwhile, training a large-scale deep learning model requires high-performance computing systems. Take the large-scale deep belief network with more than 100 million free parameters and millions of training samples developed by Raina et al. [13] for example. With a GPU-based framework, the training time for such the model is reduced from several weeks to about one day. Typically, deep learning models use an unsupervised pre-training and a supervised fine-tuning strategy to learn hierarchical features and representations of big data in deep architectures for the tasks of classification and recognition [14]. Deep learning has achieved state-of-the-art performance in a broad of applications such as computer vision [15,16], speech recognition [17,18] and text understanding [19,20].

In the past few years, deep learning has made a great progress in big data feature learning [21–23]. Compared to the conventional shallow machine learning techniques such as supported vector machine and Naive Bayes, deep learning models can take advantage of many samples to extract the high-level features and to learn the hierarchical representations by combining the low-level input more effectively for big data with the characteristics of large variety and large veracity. In this paper, we review the emerging research work on deep learning models for big data feature learning. We first present four types of most typical deep learning models, i.e., stacked auto-encoder, deep belief network, convolutional neural network and recurrent neural network, which are also the most widely used for big data feature learning, in Section 2. Afterwards, we provide an overview on deep learning models for big data according to the 4V's model, including large-scale deep learning models for huge amounts of data, multi-modal deep learning models and deep computation model for heterogeneous data, incremental deep learning models for real-time data and reliable deep learning models for low-quality data. Finally, we discuss the remaining challenges of deep learning on big data and point out the potential trends.

2. Typical deep learning models

Since deep learning was presented in *Science* magazine in 2006, it has become an extremely hot research topic in the machine learning community. Various deep learning models have been developed in the past few years. The most typical deep learning models include stacked auto-encoder (SAE), deep belief network (DBN), convolutional neural network (CNN) and recurrent neural network (RNN), which are also most widely used models. Most of other deep learning models can be variants of these four deep architectures. In the following parts, we review the four typical deep learning models briefly.

2.1. Stacked auto-encoder (SAE)

A stacked auto-encoder model is usually constructed by stacking several auto-encoders that are the most typical feed-forward neural networks [24–26]. A basic auto-encoder has two stages, i.e., encoding stage and decoding stage, as presented in Fig. 1.

In the encoder stage, the input x is transformed to the hidden layer h

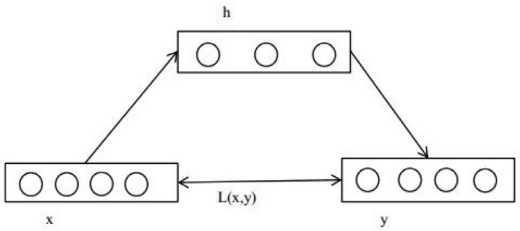

Fig. 1. Basic auto-encoder.

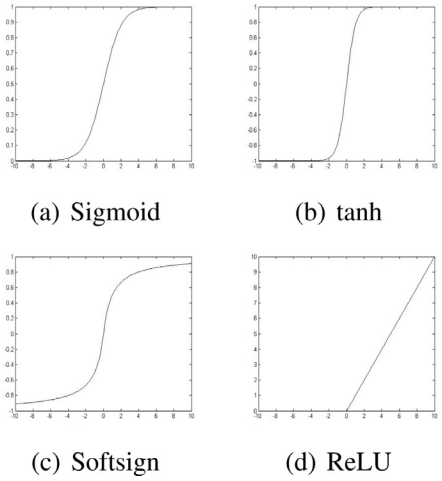

Fig. 2. Functional graph of non-linear activation functions.

via the encoding function f:

$$h = f(W^{(1)}x + b^{(1)}). \tag{1}$$

Afterwards, the hidden representation h is reconstructed back to the original input that is denoted by y in the decoding stage:

$$y = g(W^{(2)}h + b^{(2)}). \tag{2}$$

Typically, the encoding function and the decoding function are non-linear mapping functions. Four widely used non-linear activation functions are the Sigmoid function $f(x) = 1/(1 + e^{-x})$, the tanh function $f(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})$, the softsign function $f(x) = x/(1 + |x|)$ and the ReLu (Rectified Linear Units) function $f(x) = max(0, x)$. The functional graph of the four non-linear activation functions is presented in Fig. 2.

$\theta = \{W^{(1)}, b^{(1)}; W^{(2)}, b^{(2)}\}$ is the parameter set of the basic auto-encoder and it is usually trained by minimizing the loss function J_θ with regard to m training samples:

$$J_\theta = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - x^{(i)})^2, \tag{3}$$

where $x^{(i)}$ denotes the ith training sample.

Obviously, the parameters of the basic auto-encoder are trained in an unsupervised strategy. The hidden layer h is viewed as the extracted feature or the hidden representation for the input data x. When the size of h is smaller than that of x, the basic auto-encoder can be viewed as an approach for data compression.

The basic auto-encoder model has some variants. For example, a regularization named wight-decay is usually integrated into the loss function to prevent the over-fitting:

$$J_\theta = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - x^{(i)})^2 + \lambda\sum_{j=1}^{2}\|W^{(j)}\|, \tag{4}$$

where λ is a hiper-parameter used to control the strength of the weight-decay.

Another representative variant is sparse auto-encoder [27,28]. To make the learned features sparse, the sparse auto-encoder adds a sparsity constraint into the hidden units, leading to the corresponding loss function as:

$$J_\theta = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - x^{(i)})^2 + \sum_{j}^{n}KL(p\|p_j), \tag{5}$$

where n denotes the number of neurons in the hidden layer and the second item denotes the KL-divergence. Specially, the KL-divergence with regard to the jth neuron is defined as:

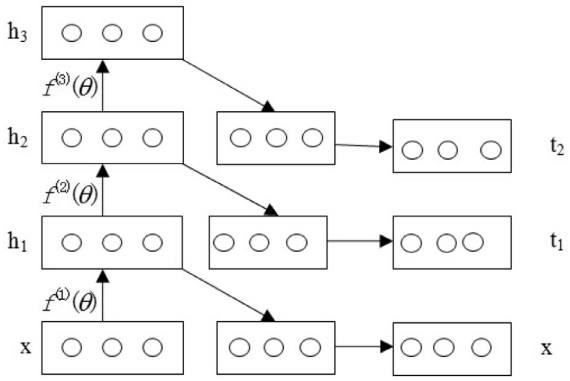

Fig. 3. Stacked auto-encoder for pre-training.

$$KL(p||p_j) = p\log\left(\frac{p}{p_j}\right) + (1-p)\log\left(\frac{1-p}{1-p_j}\right), \tag{6}$$

where p denotes a predefined sparse parameter that is close to 0 and p_j denotes the average activation value of the jth neuron in the hidden layer over all the training samples. Generally, a small p value close to 0 will result in a very sparse hidden representation learned by the auto-encoder.

Several auto-encoders can be stacked to construct a deep learning model, called stacked auto-encoder, to learn hierarchical features or representations for the input, as presented in Figs. 3 and 4.

The stacked auto-encoder is typically trained by two stages, i.e., pre-training and fine-tuning. As shown in Fig. 3, let $X = h_0$ and h_i denote the input layer and the ith hidden layer, respectively. In the pre-training stage, each auto-encoder model is trained in a unsupervised layer-wise manner from bottom to top. In detail, the auto-encoder takes $h_0 = X$ as input and takes Y_0 as output to train the parameters of the first hidden layer, and then h_1 is fed as the input to train the parameters of the second hidden layer. This operation is repeated until the parameters of all the hidden layers are trained. After pre-training, the parameters are set to the initial parameters of the stacked auto-encoder. Some labeled samples are used as the supervised objects to fine-tune the parameters from top to bottom in the fine-tuning stage, as presented in Fig. 4.

According to Hinton et al. [12], this two-stage training strategy can avoid the local optima effectively and achieve a better convergency for deep learning models.

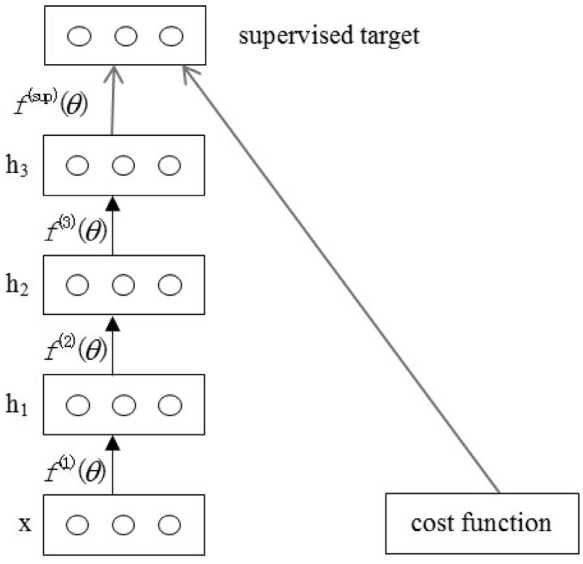

Fig. 4. Stacked auto-encoder for fine-tuning.

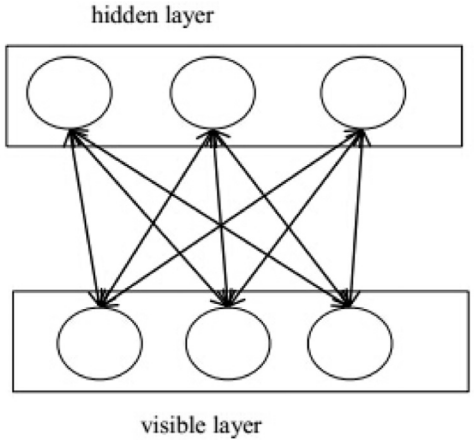

Fig. 5. Restricted Boltzmann machine.

2.2. Deep belief network (DBN)

The first deep learning model that is successfully trained is the deep belief network [12,29]. Different from the stacked auto-encoder, the deep belief network is stacked by several restricted Boltzmann machines. The restricted Boltzmann machine consists of two layers, i.e., visible layer v and hidden layer h, as presented in Fig. 5 [30,31].

A typical restricted Boltzmann machine uses the Gibbs sampling to train the parameters. Specially, the restricted Boltzmann machine uses the conditional probability $P(h|v)$ to calculate the value of each unit in the hidden layer and then uses the conditional probability $p(h|v)$ to calculate the value of each unit in the visible layer. This process is performed repeatedly until convergence.

The joint distribution of the restricted Boltzmann machine with regard to all the units is defined as:

$$p(v, h; \theta) = \frac{\exp(-E(v, h; \theta))}{Z}, \tag{7}$$

where $Z = \sum_v \sum_h \exp(-E(v, h; \theta)$ is used for normalization. E denotes the energy function with the Bernoulli distribution that is calculated via:

$$E(v, h; \theta) = -\sum_{i=1}^{I}\sum_{j=1}^{J} w_{ij} v_i h_j - \sum_{i=1}^{I} b_i v_i - \sum_{j=1}^{J} a_j h_j, \tag{8}$$

where I and J denote the number of the visible units and the hidden units, respectively. $\theta = \{W, b, a\}$ denotes the parameter set of the restricted Boltzmann machine.

The sampling probability of each unit is calculated as follows:

$$p(h_j = 1|v; \theta) = f\left(\sum_{i=1}^{I} w_{ij} v_i + a_j\right), \tag{9}$$

$$p(v_j = 1|h; \theta) = f\left(\sum_{j=1}^{J} w_{ij} h_j + b_i\right), \tag{10}$$

where f is typically a Sigmoid function.

An important variant is the restricted Boltzmann machine with Gauss–Bernoulli distribution whose energy function is calculated as follows [32]:

$$E(v, h; \theta) = -\sum_{i=1}^{I}\sum_{j=1}^{J} w_{ij} v_i h_j + \frac{1}{2}\sum_{i=1}^{I}(v_i - b_i) - \sum_{j=1}^{J} a_j h_j \tag{11}$$

The corresponding conditional probability of each visible unit is calculated via:

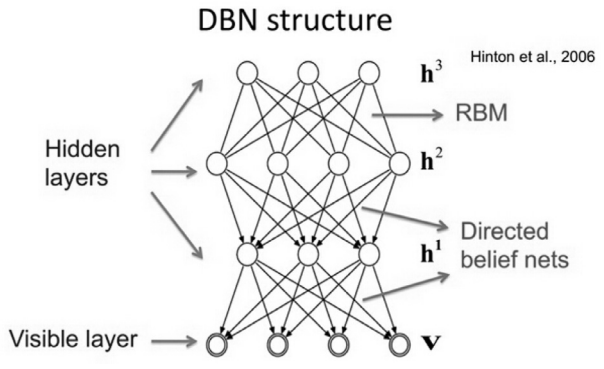

Fig. 6. Deep belief network.

$$p(v_j = 1|h;\theta) = N\left(\sum_{j=1}^{J} w_{ij}h_j + b_i,\ 1\right), \tag{12}$$

where v_i denotes the real-value that satisfies the Gauss distribution with the mean value of $\sum_{j=1}^{J} w_{ij}h_j + b_i$ and the variance of 1. The restricted Boltzmann machine with Gauss-Bernoulli distribution can transform the real random variable to the binary variable.

Several restricted Boltzmann machines can be stacked into a deep learning model, called deep belief network, as presented in Fig. 6 [12].

Similar to the stacked auto-encoder, the deep belief network is also trained by a two-stage strategy. The pre-training stage is used to train the initial parameters in a greedy layer-wise unsupervised manner while the fine-tuning stage uses the supervised strategy to fine-tune the parameters with regard to the labeled samples by adding a softmax layer on the top layer. Deep belief networks have a wide range applications in image classification [33,34] and acoustic modeling [35] and so on [36–38].

2.3. Convolutional neural network (CNN)

Convolutional neural network is the most widely used deep learning model in feature learning for large-scale image classification and recognition [39–43]. A convolutional neural network consists of three layers, i.e., convolutional layer, subsampling layer (pooling layer) and fully-connected layer, as presented in Fig. 7 [44].

The convolutional layer uses the convolution operation to achieve the weight sharing while the subsampling is used to reduce the dimension. Take a 2-dimensional image x as example. The image is firstly decomposed into a sequential input $x = \{x_1, x_2, ...,x_N\}$. To share the weight, the convolutional layer is defined as:

$$y_j = f\left(\sum\nolimits_i K_{ij} \otimes x_i + b_j\right), \tag{13}$$

where y_j denotes the jth output for the convolutional layer and K_{ij} denotes the convolutional kernel with the ith input map x_i. $\otimes$ denotes the discrete convolution operator and b_j denotes the bias. In addition, f denotes the non-linear activation, typically a scaled hyperbolic tangent function.

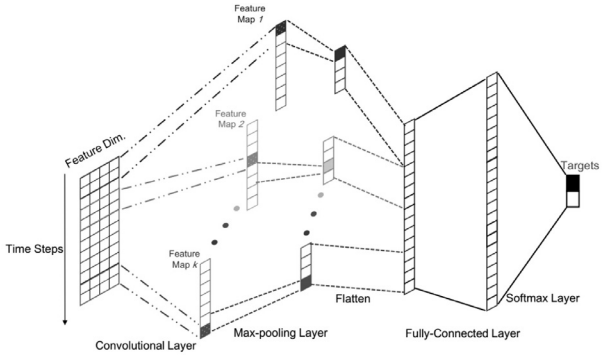

Fig. 7. Convolutional neural network.

The subsampling layer aims to reduce the dimension of the feature map. It can typically be implemented by an average pooling operation or a max pooling operation. Afterwards, several fully-connected layers and a softmax layer are typically put on the top layer for classification and recognition.

The deep convolutional neural network usually includes several convolutional layers and subsampling layers for feature learning on large-scale images.

In recent years, convolutional neural networks have also made a great success in language processing and speech recognition and so on [45–50].

2.4. Recurrent neural network (RNN)

The traditional deep learning models such as stacked auto-encoders, deep belief networks and convolutional neural networks do not take the time series into account, so they are not suitable to learn features for the time series data. Take one natural language sentence that is a kind of typical time series data as an example. Since each word is closely related to other words in a sentence, the previous one or more words should be used as inputs when using the current word to predict the next word. Obviously, the feed-forward deep learning models cannot work well for this task since they do not store the information of previous inputs.

The recurrent neural network is a typical sequential learning model. It learns features for the series data by a memory of previous inputs that are stored in the internal state of the neural network. A directed cycle is introduced to construct the connections between neurons, as presented in Fig. 8.

A recurrent neural network includes input units $\{x_0, x_1, ...,x_t, x_{t+1}, ...\}$, output units $\{y_0, y_1, ...,y_t, y_{t+1}, ...\}$ and hidden units $\{s_0, s_1, ...,s_t, s_{t+1}, ...\}$. As shown in Fig. 8, at the time step t, the recurrent neural network takes the current sample x_t and the previous hidden representation s_{t-1} as input to obtain the current hidden representation s_t:

$$s_t = f(x_t, s_{t-1}), \tag{14}$$

where f denotes the encoder function.

One widely used recurrent neural network is vanilla one, which at the time step t is defined as the following forward pass:

$$s_t = f(W_{sx}x_t + W_{ss}s_{t-1} + b_s), \tag{15}$$

$$y_t = g(W_{ys}s_t + b_y), \tag{16}$$

where f and g denote the encoder and decoder, respectively, and $\theta = \{W_{sx}, W_{ss}, b_s; W_{ys}, b_y\}$ denotes the parameter set.

Therefore, the recurrent neural network captures the dependency between the current sample x_t with the previous one x_{t-1} by integrating the previous hidden representation s_{t-1} into the forward pass. From a theoretical point of view, the recurrent neural network can capture arbitrary-length dependencies. However, it is difficult for the recurrent

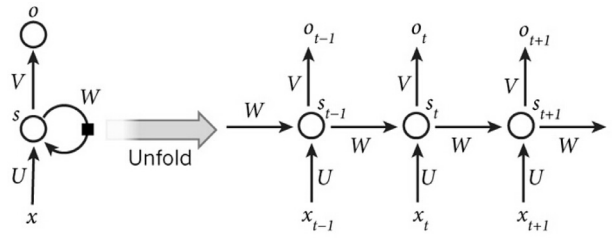

Fig. 8. Recurrent neural network.

neural network to capture a long-term dependency because of the gradient vanishing with the back-propagation strategy for training the parameters. To tackle this problem, some models, such as long short-term memory, have been presented by preventing the gradient vanishing or gradient exploding [51–54].

Multiple recurrent neural networks can be stacked into a deep learning model. The recurrent neural network and its variants have achieved super performance in many applications such as natural language processing, speech recognition and machine translation [55–59].

3. Deep learning models for big data feature learning

Big data is typically defined by the following four characteristics: volume, variety, velocity and veracity. In this section, we review the deep learning models for big data feature learning from four aspects, i.e., deep learning models for huge amounts of data, deep learning models for heterogeneous data, deep learning models for real-time data and deep learning models for low-quality data.

3.1. Deep learning models for huge amounts of data

First and foremost, huge amounts of data poses a big challenge on deep learning models. A big dataset often includes a great many samples, each with a large number of attributes. Furthermore, there are many class types of samples in a big dataset. In order to learn features and representations for large amounts of data, some large-scale deep learning models have been developed. Generally, a large-scale deep learning model involves a few hidden layers, each with a large number of neurons, leading to millions of parameters. It is a typically difficult task to train such large-scale deep learning models.

In recent years, many algorithmic methods have been presented to train large-scale models, which can roughly grouped into three categories, i.e., parallel deep learning models, GPU-based implementation, and optimized deep learning models.

One of the most representative parallel deep learning models is called deep stacking network which is presented by Deng et al. [60]. A deep stacking network is constituted by some modules. Fig. 9 shows a specific example of the deep stacking network with three modules.

In a deep stacking network, each module is also a neural network with a hidden layer and two sets of weights, as presented in Fig. 9. The lowest module consists of three layers from bottom to up. The bottom is a linear layer which uses the original data as input while the hidden layer is a non-linear one with some hidden neurons. Similar to most of deep learning models, the deep stacking network uses the Sigmoid function to map the input to the hidden layer by a weight matrix and a bias vector. The top is also a linear layer, constituted by C output neurons which denote the targets of classification.

The original data is concatenated with the previous output layer(s) and the concatenated vector is used as the input of each module above

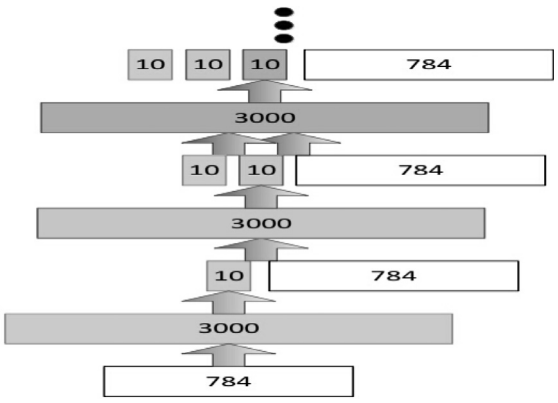

Fig. 9. A deep stacking network with three modules.

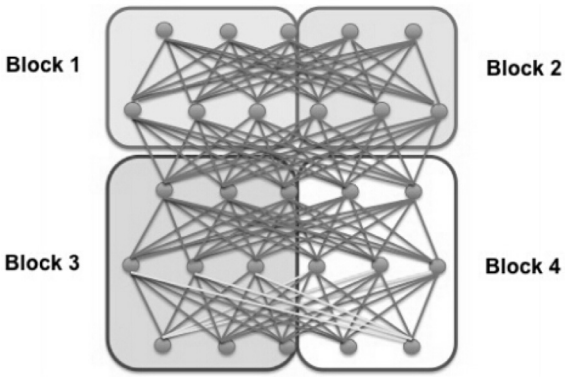

Fig. 10. Example of DistBelief with four blocks.

the lowest module. For example, if each original data object is represented by an n-dimensional vector and there are c class types, the dimension, D, of the input vector of the ith counting from bottom to up, is $D = n + c \times (m - 1)$.

The deep stacking network is efficient for training since it can be paralleled. Furthermore, a tensor deep stacking network was presented to improve the training efficiency further on CPU clusters [61].

Recently, a software framework called DistBelief was developed to train large-scale deep learning models in a large number of machines in parallel [62–64]. DistBelief is efficient to train large-scale models with billions of free parameters and huge amounts of data by combination of data parallelism and model parallelism. In order to achieve the model parallelism, a large deep learning model is partitioned into some small blocks and each block is assigned to a computer for training. Fig. 10 presents one example of DistBelief with four blocks [8].

DistBelief needs to transfer data among the computers for training the deep learning models, which will result in a great deal of communication, especially for the fully-connected network such as stacking auto-encoder and deep belief network. In spite of this, DistBelief still improves the training efficiency significantly by partitioning a large deep model into 144 blocks, as reported in [62].

DistBelief achieves data parallelism by implementing two optimization procedures, i.e., Downpour and Sandblaster. The former is used for online optimization while the latter is used for batch optimization.

DistBelief has obtained a high speedup for training several large-scale deep learning models. For example, it achieved a speedup of 12 × than using only one machine for a convolutional neural network with 1.7 billion parameters and 16 million images on 81 machines. Besides, it also achieved a significant improvement of training efficiency for another deep learning architecture with 14 million images, each one with a size of 200 × 200 pixels, on 1000 machines, each with 16 CPU cores. Therefore, DistBelief is very suitable for big data feature learning since it is able to scale up over many computers, which is the most remarkable advantage of DistBelief [8].

Deep stacking network and DistBelief typically use multiple CPU cores to improve the training efficiency for large-scale deep learning models. Some details about the use of multiple CPU cores for scaling up deep belief networks, such as implementing data layout and using SSE2 instructions, were discussed in [65].

More recently, some large-scale deep learning frameworks based on graphic processors units (GPUs) have been explored. GPUs are typically equipped by great computing power and a big memory bandwidth, so they are suitable for parallel computing for large-scale deep learning models. Some experiments have demonstrated a great advance of large-scale deep learning frameworks based on GPUs.

For example, Raina et al. [13] developed a deep learning framework based on GPUs for parallel training large-scale deep belief networks and sparse coding with more than 100 million parameters and millions of training objects. In order to improve the efficiency for parallelizing the

learning models, Raina et al. [13] used some specialized strategies in their developed learning framework [66,67], For instance, they put the parameters and some training objects into the global memory to reduce the data transfer. Besides, they implemented a parallel Gibbs sampling of hidden and visible neurons by producing two sampling matrices, i.e., $p(h|x)$ and $p(x|h)$. With this framework, a deep belief network constructed by multiple restricted Boltzmann machines, each with 45 million free parameters and 1 million training objects, was sped up by a factor of 70 × .

Another recent developed large-scale deep learning system is the commodity off-the-shelf high performance computing system, which is constituted by 16 GPU servers. Each server consists of 4 NVIDIA GTX680 GPUs, everyone with 4GB memory. This system trains large deep learning models by implementing CUDA kernels with effective memory usage and efficient computation [68]. For instance, Coates et al. [68] makes full use of the matrix sparseness and local receptive field to improve the calculation efficiency for large matrices multiplication. Compare with DistBelief that needs 16 thousand CPU cores to train a large deep learning model with 10 million images and billion parameters, the commodity off-the-shelf high performance computing system achieved almost training efficiency (e.g., about 3 days) for training the same model on three computers.

FPGA-based approaches have also explored for large-scale deep learning models in recent years [69,70]. Chen and Lin reviews the recent progress in large-scale deep learning frameworks in [8].

Generally, large-scale deep learning models can only be trained in high-performance computing servers which are equipped with multiple CPU cores or GPUs, limiting the application of such models on low-end devices. Based on the recent researches indicating that the parameters of large-scale deep learning models especially with fully-connected layers are of high redundance, some methods have been presented to improve the training efficiency by compressing the parameters significantly without a large accuracy drop.

One typical and straightforward method for compressing deep learning models is the use of a low-rank representation of the parameter matrices [71–73]. For example, Sainath et al. [72] applied the low-rank factorization to a deep neural network with 5 hidden layers for speech recognition. Specially, they decomposed the last weight matrix into two smaller matrices to compress the parameters significantly since more than half of the parameters are included in the final layer in the deep learning models for speech recognition. In detail, A is denoted as the weight matrix and A is $m \times n$-dimensional with the rank of r. Sainath et al. [72] decomposed A into B and C, namely $A = B \times C$, which are of dimension with $m \times r$ and $r \times n$, respectively. Obviously, if $mr + rn < mn$, the parameters of the deep learning model are compressed. Furthermore, if we want to compress the parameters of the final layer by a factor of p, the following condition must be satisfied.

$$r < \frac{pmn}{m+n}. \tag{17}$$

The low-rank factorization of the deep learning model is able to constrain the space of search directions, which is helpful to optimize the objective function more efficiently. Some experiments indicated that they could reduce the number of parameters of the deep learning models up to 50% which leads to about 50% speedup without a large loss of accuracy.

Chen et al. [74] employed the Hashing Trick to compress large-scale deep learning models. In detail, they employed a hash function to gather the network connections into several hash groups randomly and the connections in one group share the weights. Fig. 11 shows one example of a network with one hidden layer [74].

In the neural network in Fig. 11, the connections between the input layer and the hidden layer are represented by V^1 while the connections between the hidden layer and the output layer are represented by V^2. With a hash function $h^l(\cdot, \cdot)$ that projects an index (i, j) to a natural number, the item V^l_{ij} is assigned into an item of w^l indexed by $h^l(i, j)$:

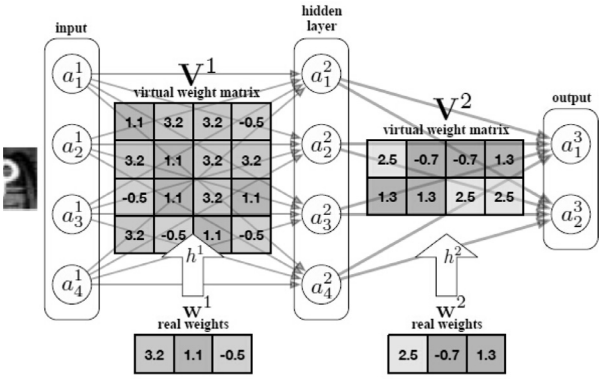

Fig. 11. Example of a neural network which is compressed by the hashing trick.

$$V^l_{ij} = w^l_{h^l(i,j)}. \tag{18}$$

Thus, the connections can be gathered into 3 groups. Furthermore, the connections marked by the same color share the same weights that are denoted as w^1 and w^2. Therefore, the parameters of the neural network in Fig. 11 are compressed to 1/4, i.e., 24 parameters are denoted by 6 parameters. Experiments demonstrated that this approach could compress the parameters of a neural network by a factor of up to 8 × on MNIST without an accuracy drop.

More recently, tensor decomposition schemes have been used to compress the parameters of large-scale deep learning models [75,76]. For example, Novikov et al. [75] proposed a tensorizing learning model based on the tensor-train network. In order to use the tensor-train network to compress the parameters, they converted the neural network to the tensor format. Given a neural network with an N-dimensional input $(N = \prod_{k=1}^{d} n_k)$ and an M-dimensional hidden layer $(M = \prod_{k=1}^{d} m_k)$, they defined a bijection $\mu(l) = (\mu_1(l), \mu_2(l), ..., \mu_d(l))$ to convert the input vector into the tensor format, where $l \in \{1, 2, ..., N\}$ and $\mu_k(l) \in \{1, 2, ..., n_k\}$ denote the coordinates of the input vector b and the coordinate of the corresponding tensor B, respectively. Therefore, the input tensor B can be obtained by $B(\mu(l)) = b_l$. Similarly, they convert the hidden vector into the tensor format. Furthermore, they convert the weight matrix $w \in R^{M \times N}$ into the tensor format W using the bijections $\nu(t) = (\nu_1(t), \nu_2(t), ..., \nu_d(t))$ and $\mu(l) = (\mu_1(l), \mu_2(l), ..., \mu_d(l))$ that projects the index (t, l) of w into the corresponding index of the tensor format W.

Afterwards, they convert the weight tensor W into the tensor-train format G:

$$\begin{aligned} w(t.\ l) &= W((\nu_1(t), \mu_1(l)), ..., \nu_d(t), \mu_d(l)) \\ &= G_1[(\nu_1(t), \mu_1(l)] \cdots G_d[(\nu_d(t), \mu_d(l)] \end{aligned}, \tag{19}$$

where $G_k[(\nu_k(t), \mu_k(l)]$ denote the core matrices of the tensor-train representation for W, with the index $(\nu_k(t), \mu_k(l)$.

Therefore, a linear projection $y = wx + b$ in a fully-connected lay can be transformed into the tensor-train form:

$$\begin{aligned} &Y(i_1, i_2, ..., i_d) = \textstyle\sum_{j_1, ..., j_d} G_1[(\nu_1(t), \mu_2(l)] \cdots \\ &G_d[(\nu_d(t), \mu_d(l)] X(j_1, ..., j_d) + B(i_1, ..., i_d) \end{aligned}. \tag{20}$$

This method could reduce the computational complexity of the forward pass and improve the training efficiency in the back-propagation procedure. Table 1 summarizes the computational complexity and storage complexity of an $M \times N$ tensor-train layer (TT) compared with the original fully-connected layer (FC) [75].

Besides, Lebdev et al. [76] employed the canonical polyadic decomposition to compress the parameters of the convolutional neural network and they achieved a significant speedup for the inference time.

Table 1
Comparison of computational complexity and storage complexity between TT and FC where r denotes the maximal rank of tensor-train network.

Operation	Computational complexity	Storage complexity
FC forward pass	$O(MN)$	$O(MN)$
TT forward pass	$O(dr^2m\max\{M, N\})$	$O(r\max\{M, N\})$
FC backward pass	$O(MN)$	$O(MN)$
TT backward pass	$O(d^2r^4m\max\{M, N\})$	$O(r^3\max\{M, N\})$

3.2. Deep learning models for heterogeneous data

A distinct characteristic of big data is its variety, implying that big data is collected in various formats including structured data and unstructured data, as well as semi-structured data, from a large number of sources. Specially, a great number of objects in big datasets are multi-model. For example, a webpage typically contains image and text simultaneously. Another multi-model example is a multimedia object such as a video clip which includes still images, text and audio. Each modality of multi-modal objects has different characteristic with each other, leading to the complexity of heterogeneous data. Therefore, heterogeneous data poses another challenge on deep learning models.

Some multi-model deep learning models have been proposed for heterogeneous data representation learning. For example, Ngiam et al. [77] developed a multi-modal deep learning model for audio-video objects feature learning. Fig. 12 shows the architecture of the multi-modal deep learning model.

Ngiam et al. [77] used the restricted Boltzmann machines to learn features and representations for audio and video separately. The learned features are concatenated into a vector as the joint representation of the multi-modal object. Afterwards, the joint representation vector is used as the input of a deep auto-encoder model for the tasks of classification or recognition.

Srivastava and Salakhutdinov developed another multi-model deep learning model, called bi-modal deep Boltzmann machine, for text-image objects feature learning, as presented in Fig. 13 [78].

In this model, two deep Boltzmann machines are built to learn features for text modality and image modality, respectively. Similarly, the learned features of the text and the image are concatenated into a vector as the joint representation. In order to perform the classification task, the classifier such as the supported vector machine could be trained with the joint representation as input.

Another multi-modal deep learning model, called multi-source deep learning model, was presented by Ouyang et al. [79] for human pose estimation. Different from above two multi-model models, the multi-source deep learning model aims to learn non-linear representation from different information sources, such as human body articulation and clothing for human pose estimation. In this model, each information source is used as input of a deep learning model with two hidden layers for extracting features separately. The extracted features are then fused for the joint representation.

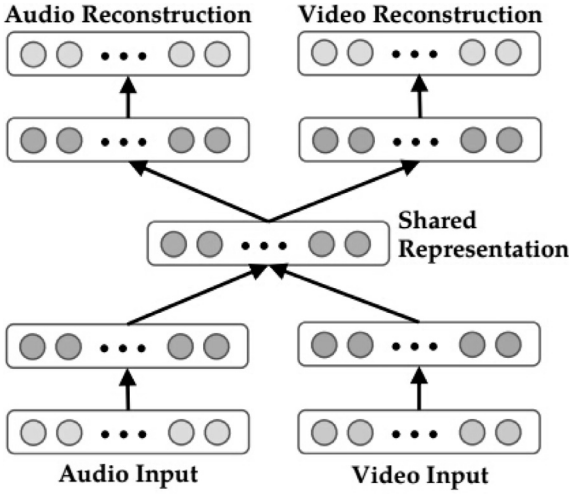

Fig. 12. Architecture of the multi-modal deep learning model.

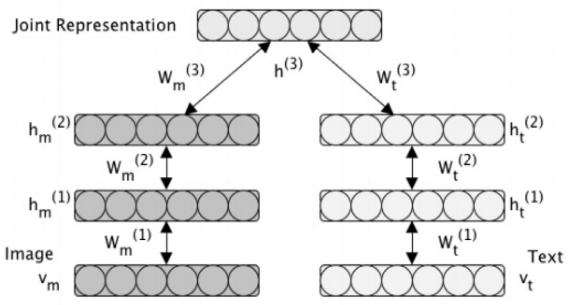

Fig. 13. Architecture of the bi-modal deep Boltzmann machine.

Other representative multi-modal deep learning models include heterogeneous deep neural networks combined with conditional random fields for Chinese dialogue act recognition [80], multi-modal deep neural network with sparse group lasso for heterogeneous feature selection [81] and so on [82–84]. Although they have different architectures, their ideas are similar. Specially, multi-modal deep learning models first learn features for single modality and then combine the learned features as the joint representation for each multi-modal object. Finally, the joint representation is used as input of a logical regression layer or a deep learning model for the tasks of classification or recognition.

Multi-modal deep learning models achieved better performance than traditional deep neural networks such as stacked auto-encoders and deep belief networks for heterogeneous data feature learning. However, they concatenated the learned features of each modality in a linear way, so they are far away effective to capture the complex correlations over different modalities for heterogeneous data. To tackle this problem, Zhang et al. [85,86] presented a tensor deep learning model, called deep computation model, for heterogeneous data.

Specially, they designed a tensor auto-encoder by extending the stacked auto-encoder model to the tensor space based on the tensor data representation. In the tensor auto-encoder model, the input layer X, the hidden layer H, and the parameters $\theta = \{W^{(1)}, b^{(1)}; W^2, b^{(2)}\}$ are represented by tensors. Besides, tensor distance is used to reveal the complex features of heterogeneous data in the tensor space, which yields a loss function with m training objects of the tensor auto-encoder model:

$$J_{TAE}(\theta) = [\frac{1}{m}\sum_{i=1}^{m}(\frac{1}{2}(h_{W,b}(x) - y)^T G(h_{W,b}(x) - y))] + \frac{\lambda}{2}\left(\sum_{p=1}^{J_1\times\cdots\times J_N}\sum_{i_1=1}^{I_1}\cdots\sum_{i_N=1}^{I_N}(W^{(1)}_{pi_1\cdots i_n})^2 + \sum_{q=1}^{I_1\times\cdots\times I_N}\sum_{j_1=1}^{J_1}\cdots\sum_{j_N=1}^{J_N}(W^{(2)}_{qj_1\cdots j_n})^2\right) \quad (21)$$

where G denotes the metric matrix of the tensor distance and the second item is used to avoid over-fitting.

Furthermore, they built a deep computation model by stacking multiple tensor auto-encoder models. Experiments demonstrated that the deep computation model achieved about 2%-4% higher classification accuracy than multi-modal deep learning models for heterogeneous data.

3.3. Deep learning models for real-time data

High velocity is another important characteristic of big data, which

requires to analyze big data in real time. Big data is usually collected at a high speed, posing a challenge on big data real-time processing. Unfortunately, most of deep learning models are of high computational complexity since they typically involves a large number of parameters for big data feature learning, especially large-scale deep neural networks. Therefore, it is difficult for traditional deep learning models to learn features and representations for big data in real time.

In recent years, a lot of incremental learning methods have been presented for high-velocity data feature learning. An kind of incremental learning methods is online learning which only updates the parameters when new objects are arriving with preserving the network structure.

Fu et al. [87] presented an incremental back-propagation learning model based on a concept of bound. A validity bound is defined as a range of the weights that represent the knowledge learned by a neural network. To limit the weights in the validity bound in the updating procedure, a scaling factor s is introduced into weight modifications, yielding a learning rule in the kth iteration:

$$\Delta W_{ji}(k) = s(k)\eta\delta_j(k)O_i(k), \tag{22}$$

where η and δ_i denote the learning rate and the error gradient of the neuron j, O_i denotes the activation level of the neuron i.

Then, s should be set according to:

$$s(k) = \min_{j,i}\left\{1, \frac{[B(p) - |\sum_{t=1}^{k-1}\Delta W_{ji}(t)|]}{|\eta\delta_j(k)O_i(k)|}\right\}, \tag{23}$$

where B denotes a pre-defined bound on the weight modification for an object p.

Fu et al. [87] applied the bounded weight adjustment to implement an incremental learning model. Specially, weights are not modified to prevent the over-training when the new arriving object is covered by the knowledge of the current network. However, it is difficult to define the bound beforehand since it is sensitive to the previous knowledge. Besides, this method is only suitable for two-layer neural network, so it is difficult to apply the bounded weight modification to the incremental deep learning models. Wan and Banta [88] proposed an online learning model based on parameters updating. Specially, an incremental auto-encoder model was presented by updating the current parameters θ to $\theta + \Delta\theta$ to adapt the new arriving objects. θ denotes the previous knowledge trained on the old objects while $\Delta\theta$ denotes the parameters increment on the new arriving objects. Given a new arriving object, to achieve the goal of adaptation that requires the updated parameters could learn the new objects effective, an objective function $J_{adaptation}$ is defined:

$$J_{adaptation} = \frac{1}{2}\Delta x^T\Omega\Delta x, \tag{24}$$

where Ω denotes a weight matrix defined as $\Omega = \left(I - \mu\left(\frac{\partial\varphi(x,\theta)}{\partial\theta}\right)^T\left(\frac{\partial\varphi(x,\theta)}{\partial\theta}\right)\right)^{-1}$ and $\Delta x = \varphi(x, \theta + \Delta\theta) - x$ denotes the post-reconstruction error.

To achieve the goal of preservation that requires the updated model could still learn old objects, an objective function $J_{preservation}$ is defined:

$$J_{preservation} = \frac{1}{2\mu}\theta^T\theta. \tag{25}$$

Furthermore, to obtain a tradeoff between adaptation and preservation, the global objective function is defined as:

$$J(x, \theta + \Delta\theta) = J_{adaptation} + J_{preservation}. \tag{26}$$

There, the parameters increment $\Delta\theta$ can be obtained by minimizing the above function with regard to the new arriving objects.

Online learning methods are efficient for big data feature learning since they do not need to re-train the parameters on the old training objects [89–91]. This scheme is particularly suitable for big data because the data size is too big to hold in the memory. However, this strategy usually preforms poor for dynamic big data whose distribution is changing drastically over time.

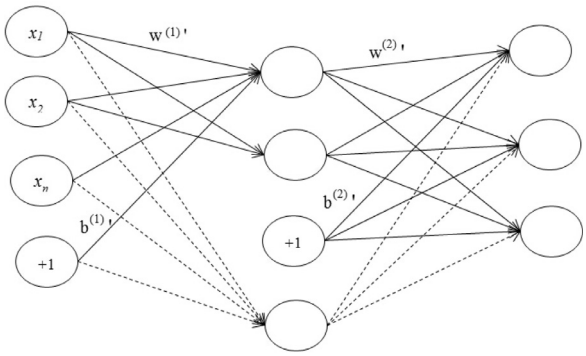

Fig. 14. Structure-based incremental learning method with adding one hidden neuron.

To tackle this problem, another kind incremental learning method based on the structural modification has been presented. For example, a structure-based incremental auto-encoder model was implemented by adding one or more neurons into the hidden layer to adapt new arriving objects [92–94]. Fig. 14 shows an example of this model with adding only one neuron.

After the network structure is modified, the parameters should be also updated accordingly. Let $\theta = \{W^{(1)}, b^{(1)}; W^{(2)}, b^{(2)}\}$ represent the parameters of a neural network with n-dimensional input an m-dimensional hidden layer. Thus, the parameters have the forms:

$$\begin{aligned} &W^{(1)} \in R^{m\times n} \quad b^{(1)} \in R^m \\ &W^{(2)} \in R^{n\times m} \quad b^{(2)} \in R^n. \end{aligned} \tag{27}$$

If only one neuron is added into the hidden layer, the weight matrices $W^{(1)}$ and $W^{(2)}$ add one row and one column, respectively. Also, the bias vector $b^{(1)}$ adds one element. Therefore, the parameters forms are updated as:

$$\begin{aligned} &W^{(1)} \in R^{(m+1)\times n} \quad b^{(1)} \in R^(m+1) \\ &W^{(2)} \in R^{n\times(m+1)} \quad b^{(2)} \in R^n. \end{aligned} \tag{28}$$

If p new neurons are added into the hidden layer, the initial parameters are set to:

$$W^{(1)\prime} = \begin{pmatrix} & W^{(1)} & \\ \mathbf{0} & & \\ (p) & & \\ & \mathbf{0} & \end{pmatrix} \quad b^{(1)} = \begin{pmatrix} & b^{(1)} & \\ \mathbf{0} & & \\ (p) & & \\ & \mathbf{0} & \end{pmatrix}$$
$$W^{(2)\prime} = \begin{pmatrix} & & (p) & \\ W^{(2)} & \mathbf{0} & & \mathbf{0} \end{pmatrix}. \tag{29}$$

Furthermore, the final parameters can be trained by the learning algorithms such as the back-propagation algorithm.

3.4. Deep learning models for low-quality data

Another emerging challenge for big data feature learning arises from its veracity. Specially, low-quality data is prevalent in big data, implied by the fact that there are a large number of incomplete objects, noise, inaccurate objects, imprecise objects and redundant objects in big data. Low-quality data is resulted from many reasons. For example, a large amount of data is collected from sensors. If some sensors are broken, we may collect some incomplete objects. The transmission fault of the network may also result in some noise in big data. The topics about big data quality have been studied in literatures [95–98].

Most of deep learning models do not take low-quality data into account. In other words, most of deep learning models are designed for high-quality data feature learning. In the past few years, some methods have been proposed to learn features for low-quality data. Vincent et al.

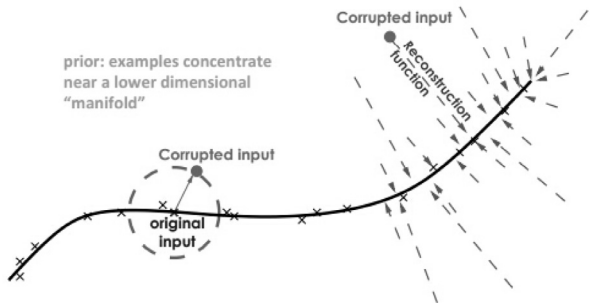

Fig. 15. Denoising auto-encoder.

[99] presented a denoising auto-encoder model which could learn features and representations for data with noise. Specially, the denoising auto-encoder model trains the parameters by reconstructing the original input from the corrupted input, as shown in Fig. 15.

To reconstruct the original input, the objective function of the denoising auto-encoder model is defined as:

$$J = \sum_t E_{q(\tilde{x}|x^{(t)})}[L(x^{(t)}, g_\theta(f_\theta(\tilde{x})))], \tag{30}$$

where $\tilde{x}$ denote the corrupted instances of the original input $x^{(t)}$ based on the corruption process $q(\tilde{x}|x^{(t)})$ and $E_{q(\tilde{x}|x^{(t)})}[\cdot]$ averages over the instances $\tilde{x}$.

The parameters of the objective function can be trained by the gradient descent strategy. $\tilde{x}$ can be obtained by adding isotropic Gaussian noise or pepper noise. Furthermore, Vincent et al. [100] developed stacked denoising auto-encoder models for feature learning on data with noise.

Zhang et al. [101] proposed an imputation auto-encoder model to learn features for incomplete objects, as shown in Fig. 16.

The simulated incomplete object x' is obtained by setting a part of attributes values of the original object x to 0. The imputation auto-encoder model takes the incomplete object x' as input and output the reconstructed object z:

$$z = g_\theta(f_\theta(x)). \tag{31}$$

The parameters θ are trained by minimizing the following objective function:

$$J = L(x, z). \tag{32}$$

Furthermore, they built a deep imputation network for incomplete objects feature learning by stacking several imputation auto-encoders, as presented in Fig. 17.

More recently, Wang and Tao presented a non-local auto-encoder to learn reliable features for corrupted data [102]. Their work is motivated by the neurological observation that similar input should stimulate human brains to produce the similar response. Therefore, the neural networks should yield similar hidden representations for the similar input objects. In detail, suppose that h_1, h_2 and h_3 are the learned representations of x_1, x_2 and x_3, respectively. If $\|x_1 - x_2\| < \|x_1 - x_3\|$, $\|h_1 - h_2\|$ should be smaller than $\|h_1 - h_3\|$. However, the neural network cannot always guarantee the relationship $\|h_1 - h_2\| < \|h_1 - h_3\|$ since the neural networks use the non-linear

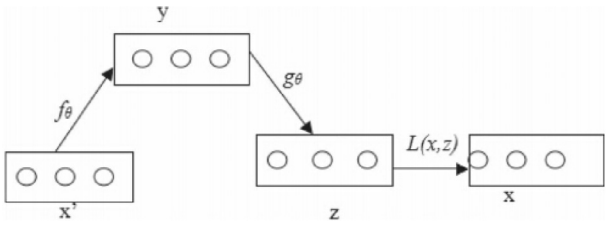

Fig. 16. Imputation auto-encoder model.

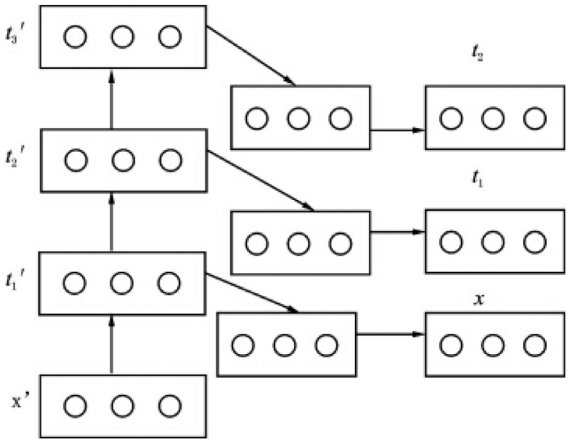

Fig. 17. Deep imputation network.

function such as the Sigmoid function as activation function. To learn reliable representations, Wang and Tao presented a regularization term and added the regularization term into the objective function. Given a training object x, some corrupted instances x_i can be obtained by adding the noise into x. Let h and h_i denote the learned hidden representations of x and x_i, respectively. The regularization term is defined as:

$$\sum_i \omega_i \|h - h_i\|_p, \tag{33}$$

where ω_i denotes the weight of the ith corrupted instance.

The objective function of the non-local auto-encoder model can be obtained by applying the regularization term:

$$J = \left\|x - g_\theta(f_\theta(x))\right\|_2^2 + \lambda \sum_i \omega_i \|h - h_i\|_p. \tag{34}$$

Experiments demonstrated that the non-local auto-encoder model achieved super performance in image denoising and restoration.

4. Summary and perspectives

Deep learning is currently the most active topic of machine learning. In recent years, deep learning models have been widely used in many fields such as computer vision, health monitoring and natural language processing. The explosion of big data offers enough training objects, which helps to improve the performance of deep learning. Furthermore, high-performance computing devices and architectures such as graphic processing units and CPU cluster enable the training of large-scale deep learning models for big data feature leaning. Today, deep learning models enjoy the success with a great many parameters, typically millions of parameters, together with a large number of training objects. While big data brings enough training objects, it also poses some challenges on deep learning. Therefore, in the past few years, many deep learning models have been developed for big data learning. In this paper, we provide a survey of big data deep learning models. Big data is typically defined by the four V's model: volume, variety, velocity and veracity, which implies huge amount of data, various types of data, real-time data and low-quality data, respectively. Therefore, we summarized the deep learning models for big data learning from four aspects accordingly. In detail, we reviewed large-scale deep learning models for huge amount of data, multi-modal deep learning models and deep computation models for heterogeneous data, incremental deep learning models for real-time data, and reliable deep learning models for low-quality data. From the previous studies, we can see that deep learning models have made a great progress in big data feature learning. However, big data deep learning is still in its infancy, i.e.,

there are still some remaining challenges to be addressed for big data learning.

First and foremost, some large-scale deep learning models with millions or billions of free parameters have been trained to learn features for high volumes of data, which uses CPU clusters and GPUs to improve the training efficiency. Besides, some parallelizing schemes including data parallelism and model parallelism also contribute to the training of large-scale deep learning models. For instance, high volumes of data that cannot be loaded into memory are partitioned into blocks for parallel training while the back-propagation is run in parallel on multiple CPU cores. Therefore, some success have been achieved for large-scale deep learning models to address the challenges about huge amounts of data. However, the scale of current deep learning models that can be trained for big data is significantly depending on the growth of the high-performance computing architectures such as CPU clusters and GPUs. Unfortunately, the gain in computational performance is lagging far behind the growth rate of big data. Therefore, with the continuing growth of big data, much more large-scale deep learning models are needed to be built. Such large models with big data may no longer be trained effective depending on current techniques and computing power. So, one future direction to address this issue is to develop new learning frameworks and computing infrastructures. In addition, how to compress the large-scale deep learning models without scarifying accuracy further is another research direction for big data feature learning.

Secondly, two advanced deep learning models, i.e., multi-modal deep learning model and deep computation model, have been developed for heterogeneous data feature learning. Multi-modal deep learning modes extract the features for each modality separately and then concatenate the extracted features as the joint representation of the heterogeneous objects. Afterwards, the joint representation is used as the input of the specified deep learning model for the tasks of classification and recognition. Deep computation models utilize tensors to represent the heterogeneous objects for extending the stacking auto-encoders to the tensor space. Deep computation models enjoy their success with the tensor-based big data representation methods that can reveal the correlations over different modalities. However, most of current multi-modal deep learning models focus on bi-modal data. Most objects in big data have more than two modalities such as a video clip that usually contains three modalities, i.e., image, audio and text. Besides, current multi-modal deep learning models concatenate learned features of each modality in a simply linear way, which often results in a poor result. Therefore, exploring effective fusion methods for learned features to improve the multi-modal deep learning models is a future direction. While deep computation models achieve super performance than multi-modal deep learning models, they include much more parameters, leading to a high computational complexity. How to reduce the computational complexity of deep computation models needs to be addressed.

While deep learning models for large volumes of data and heterogeneous data have made a great progress, only a limited progress on incremental deep learning models for high velocity of data has been made in the past few years. Some incremental learning algorithms based on parameters updating or structure updating have been proposed to adapt the new arriving objects. Most of these algorithms have been proved effective for traditional learning models with only one hidden layer. It is less clear, however, whether they can be applied to deep learning models and how they can be applied to deep architectures. For the structure-based incremental learning methods that modify the network structure by adding one or more neurons into the hidden layer, how many neurons should be added every time? Furthermore, the continuous incorporation of new neurons in the hidden layer will lead to a high redundancy of the network. So, how to optimize the network structure is another future work.

Last but not least, only several reliable deep learning models have been proposed to learn features and representations for low-quality data. Furthermore, they focus on only noisy data or incomplete data feature learning. Except for noisy objects and incomplete objects, there are a large number of redundant objects, imprecise objects and outdated objects in big data. With an explosive increase of low-quality data, reliable deep learning models for low-quality data need to be explored urgently.

References

[1] L. Kuang, L.T. Yang, Y. Liao, An integration framework on cloud for cyber physical social systems big data, IEEE Trans. Cloud Comput. (2015). 10.1109/TCC.2015.2511766

[2] Y. Li, X. Qi, M. Keally, Z. Ren, G. Zhou, D. Xiao, S. Deng, Communication energy modeling and optimization through joint packet size analysis of BSN and wifi networks, IEEE Trans. Parallel Distrib. Syst. 24 (9) (2013) 1741–1751.

[3] C.L.P. Chen, C. Zhang, Data-intensive applications, challenges, techniques and technologies: a survey on big data, Inf. Sci. 275 (2014) 314–347. 2014

[4] X. Wu, X. Zhu, G.-Q. Wu, W. Ding, Data mining with big data, IEEE Trans. Knowl. Data Eng. 26 (1) (2014) 97–107.

[5] G.B. Orgaz, J.J. Jung, D. Camacho, Social big data: recent achievements and new challenges, Inf. Fusion 28 (2016) 45–59.

[6] A. Samuel, M.I. Sarfraz, H. Haseeb, S. Basalamah, A. Ghafoor, A framework for composition and enforcement of privacy-aware and context-driven authorization mechanism for multimedia big data, IEEE Trans. Multimedia 17 (9) (2015) 1484–1494.

[7] B. Saha, D. Srivastava, Data quality: the other face of big data, Proceedings of IEEE International Conference on Data Engineering, (2014), pp. 1294–1297. IEEE

[8] X. Chen, X. Lin, Big data deep learning: challenges and perspectives, IEEE Access 2 (2014) 514–525.

[9] Y. Bengio, A. Courville, P. Vincent, Representation learning: a review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1798–1828.

[10] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521(7553) 436–444.

[11] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.

[12] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.

[13] R. Raina, A. Madhavan, A. Ng, Large-scale deep unsupervised learning using graphics processors, Proceedings of International Conference on Machine Learning, (2009), p. 873 C880. ACM

[14] V. Sze, Y. Chen, T. Yang, J. Emer, Efficient processing of deep neual networks: a tutoria and survey, 2017, arXiv:1703.09039.

[15] S.M.S. Islam, S. Rahman, M.M. Rahman, E.K. Dey, M. Shoyaib, Application of deep learning to compuer vision: a comprehensive study, Proceedings of International Conference on Informatics, Electronics and Vision, (2016), pp. 592–597. IEEE

[16] N. Kruger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A.J. Rodriguez-Sanchez, L. Wiskott, Deep hierarchies in the primate visual cortex: what can we learn for computer vision, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1847–1871.

[17] L. Deng, G. Hinton, B. Kingsbury, New types of deep neual network learning for speech recognition and related applications: an overview, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, (2013), pp. 26–31. IEEE

[18] D. Chen, B.K. Mak, Multitask learning of deep neural networks for low-resource speech recognition, IEEE/ACM Trans. Audio Speech Lang. Process. 23 (7) (2015) 1172–1183.

[19] N. Majumder, S. Poria, A. Gelbukh, E. Cambria, Deep learning-based document modeling for personality detection from text, IEEE Intell. Syst. 32 (2) (2017) 74–79.

[20] Z. Jiang, L. Li, D. Huang, L. Lin, Training word embeddings for deep learning in biomedical text mining tasks, Proceedings of IEEE International Conference on Bioinformatics and Biomedicine, (2015), pp. 625–628. IEEE

[21] A. Khumoyun, Y. Cui, L. Hanku, Spark based distributed deep learning framework for big data applications, Proceedings of International Conference on Information Science and Communications Technologies, (2016), pp. 1–5. IEEE

[22] M.A. Alsheikh, D. Niyato, S. Lin, H. Tan, Z. Han, Mobile big data analytics using deep learning and apache spark, IEEE Netw. 30 (3) (2016) 22–29.

[23] B.M. Wilamowski, B. Wu, J. Korniak, Big data and deep learning, Proceedings of IEEE Jubilee International Conference on Intelligent Engineering Systems, (2016), pp. 11–16. IEEE

[24] J. Gehring, Y. Miao, F. Metze, A. Waibel, Extracting deep bottleneck features using stacked auto-encoders, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, (2013), pp. 26–31. IEEE

[25] R. Weng, J. Lu, Y. Tan, J. Zhou, Learning cascaded deep auto-encoder networks for face alignment, IEEE Trans. Multimedia 18 (10) (2016) 2066–2078.

[26] M. Sun, X. Zhang, H.V. Hmme, T.F. Zheng, Unseen noise estimation using separable deep auto encoder for speech enhancement, IEEE/ACM Trans. Audio Speech Lang. Process. 24 (1) (2016) 93–104.

[27] I. Goodfellow, Q. Le, A. Saxe, A. Ng, Measuring invariances in deep networks, Proceeding of Neural Information and Processing System, (2009), pp. 646–654. MIT

[28] A. Ng, Sparse autoencoder, CS294A Lecture notes 72(2011) 1–19.

[29] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18(7) 1527–1554.

[30] H. Larochelle, M. Mandel, R. Pascanu, Y. Bengio, Learning algorithms for the classification restricted boltzmann machine, J. Mach. Learn. Res. 13 (2012) 643–669.

[31] R. Salakhutdinov, A. Mnih, G. Hinton, Restricted boltzmann machines for collaborative filtering, Proceedings of International Conference on Machine learning, (2007), pp. 791–798. ACM

[32] K. Cho, A. Ilin, T. Raiko, Improved learning of gaussian-bernoulli restricted boltzmann machines, Proceedings of International Conference in Artificial Neural Networks, (2011), p. 10C17. MIT

[33] T. Li, J. Zhang, Y. Zhang, Classification of hyperspectral image based on deep belief networks, Proceedings of IEEE International Conference on Image Processing, (2014), pp. 5132–5136. IEEE

[34] J. Niu, X. Bu, Z. Li, Y. Wang, An improved bilinear deep belief network algorithm for image classification, Proceedings of International Conference on Computational Intelligence and Security, (2014), pp. 189–192. IEEE

[35] A. Mohamed, G.E. Dahl, G. Hinton, Acoustic modeling using deep belief networks, IEEE Trans. Audio Speech Lang. Process. 20 (1) (2012) 14–22.

[36] X. Zhang, J. Wu, Deep belief networks based voice activity detection, IEEE Trans. Audio Speech Lang. Process. 21 (4) (2013) 697–710.

[37] W. Huang, G. Song, H. Hong, K. Xie, Deep architecture for traffic flow prediction: deep belief networks with multitask learning, IEEE Trans. Intell. Transp. Syst. 15 (5) (2014) 2191–2201.

[38] R. Sarikaya, G.E. Hinton, A. Deoras, Application of deep belief networks for natural language understanding, IEEE/ACM Trans. Audio Speech Lang. Process. 22 (4) (2014) 778–784.

[39] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, F. Li, Large-scale video classification with convolutional neural networks, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, (2014), pp. 1725–1732. IEEE

[40] E. Maggiori, Y. Tarabalka, G. Charpiat, P. Alliez, Convolutional neural networks for large-scale remote-sensing image classification, IEEE Trans. Geosci. Remote Sens. 55 (2) (2017) 645–657.

[41] Y. Han, J. Kim, K. Lee, Deep convolutional neural networks for predominant instrument recognition in polyphonic music, IEEE/ACM Trans. Audio Speech Lang. Process. 25 (1) (2017) 208–221.

[42] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Proceedings of Advances in Neural Information Processing Systems, (2012), pp. 1097–1105. MIT

[43] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv:1409.1556.

[44] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, R.X. Gao, Deep learning and its applications to machine health monitoring: a survey, 2016, arXiv:1612.07640.

[45] N. Kalchbrenner, E. Grefenstette, P. Blunsom, A convolutional neural network for modeling sentences, 2014, arXiv:1404.2188.

[46] H. Shi, T. Ushio, M. Endo, K. Yamagami, N. Horii, A multichannel convolutional neural network for cross-language dialog state tracking, Proceedings of IEEE Spoken Language Technology Workshop, (2016), pp. 559–564. IEEE

[47] O. Abdel-Hamid, A. Mohamed, H. Jiang, G. Penn, Applying convolutional neural networks concepts to gybrid NN-HMM model for speech recognition, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, (2012), pp. 4277–4280. IEEE

[48] Y. Qian, M. Bi, T. Tan, K. Yu, Very deep convolutional neural networks for noise robust speech recognition, IEEE/ACM Trans. Audio Speech Lang. Process. 24 (12) (2016) 2263–2276.

[49] O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, D. Yu, Convolutional neural networks for speech recognition, IEEE/ACM Trans. Audio Speech Lang. Process. 22 (10) (2014) 1533–1545.

[50] P. Swietojanski, A. Ghoshal, S. Renals, Convolutional neural networks for distant speech recognition, IEEE Signal Process. Lett. 21 (9) (2014) 1120–1124.

[51] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1725–1780.

[52] F.A. Gers, J. Schmidhuber, F. Cummins, Learning to torget: continual prediction with istm, Neural Comput. 12 (10) (2000) 2451–2471.

[53] F.A. Gers, N.N. Schraudolph, J. Schmidhuber, Learning precise timing with istm recurrent networks, J. Mach. Learn. Res. 3 (2002) 115–143.

[54] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014, 1412.3555.

[55] X. Chen, X. Liu, Y. Qian, M.J.F. Gales, P.C. Woodland, CUED-RNNLM-an open-source toolkit for efficient training and evaluation of recurrent neural network language models, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2016, pp. 6000–6004.

[56] T. Fu, Y. Han, X. Li, Y. Liu, X. Wu, Integrating prosodic information into recurrent neual network language model for speech recognition, Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, IEEE, 2015, pp. 1194–1197.

[57] A. Hermanto, T.B. Adji, N.A. Setiawan, Recurrent neural network language model for english-indonesian machine translation: Experimental study, Proceedings of International Conference on Science in Information Technology, IEEE, 2015, pp. 132–136.

[58] X. Zhang, G. Xie, C. Liu, Y. Bengio, End-to-end online writer identification with recurrent neural network, IEEE Trans. Hum. Mach. Syst. 47 (2) (2017) 285–292.

[59] J. Chien, Y. Ku, Bayesian recurrent neural network for language modeling, IEEE Trans. Neural Netw. Learn. Syst. 27 (2) (2016) 361–374.

[60] L. Deng, D. Yu, J. Platt, Scalable stacking and learning for building deep architectures, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2012. 2133 C2136

[61] B. Hutchinson, L. Deng, D. Yu, Tensor deep stacking networks, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1944–1957.

[62] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q.V. Le, A.Y. Ng, Large scale distributed deep networks, Proceedings of Advances in Neural Information Processing Systems, MIT, 2012, pp. 1223–1231.

[63] G. Heigold, V. Vanhoucke, A. Senior, P. Nguyen, M. Ranzato, M. Devin, J. Dean, Multilingual acoustic models using distributed deep neural networks, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2013, pp. 8619–8623.

[64] G. Heigold, E. McDermott, V. Vanhoucke, A. Senior, M. Bacchiani, Asynchronous stochastic optimization for sequence training of deep neural networks, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2014, pp. 5587–5591.

[65] V. Vanhoucke, A. Senior, M. Mao, Improving the speed of neural networks on CPUs, Proceedings of Deep Learning and Unsupervised Feature Learning Workshop, MIT, 2011, pp. 1–8.

[66] S. Geman, D. Geman, Stochastic relaxation, gibbs distributions, and the bayesian restoration of images, IEEE Trans. Pattern Anal. Mach. Intell. 6 (6) (1984) 721–741.

[67] G. Casella, E. George, Explaining the gibbs sampler, Am. Stat. 46 (3) (1992) 167–174.

[68] A. Coats, B. Huval, T. Wng, D. Wu, A. Wu, Deep learning with COTS HPC systems, J. Mach. Learn. Res. 28 (2013) 1337–1345.

[69] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-based accelerator design for deep convolutional neural networks, Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM, 2015, pp. 161–170.

[70] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, X. Zhou, DLAU: a scalable deep learning accelerator unit on FPGA, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 36 (3) (2017) 513–517.

[71] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, N. de Freitas, Predicting parameters in deep learning, Proceedings of Advances in Neural Information Processing Systems, MIT, 2013, pp. 2148–2156.

[72] T.N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, B. Ramabhadran, Low-rank matrix factorization for deep neural network training with high-dimensional output targets, Proceedings of IEEE International Conference of Acoustics, Speech, and Signal Processing, IEEE, 2013. Pp. 6655 C6659

[73] J. Xue, J. Li, Y. Gong, Restructuring of deep neural network acoustic models with singular value decomposition, Proceedings of Conference of the International Speech Communication Association, Springer, 2013. Pp. 2365 C2369

[74] W. Chen, J.T. Wilson, S. Tyree, K.Q. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, Proceedings of International Conference on Machine Learning, ACM, 2015. Pp. 2285 C2294

[75] A. Novikov, D. Podoprikhin, A. Osokin, D. Vetrov, Tensorizing neural netwroks, Proceedings of Advances in Neural Information Processing Systems, MIT, 2015, pp. 442–450.

[76] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, V. Lempitsky, Speeding-up convolutional neural networks ssing fine-tuned CP-decomposition, 2014, 1412.6553.

[77] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, A.Y. Ng, Multimodal deep learning, Proceedings of International Conference on Machine Learning, ACM, 2011, pp. 689–696.

[78] N. Srivastava, R. Salakhutdinov, Multimodal learning with deep boltzmann machines, Proceedings of Advances in Neural Information Processing Systems, MIT, 2012, pp. 2222–2230.

[79] W. Ouyang, X. Chu, X. Wang, Multi-source deep learning for human pose estimation, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2014, pp. 2337–2344.

[80] Y. Zhou, Q. Hu, J. Liu, Y. Jia, Combining heterogeneous deep neural networks with conditional random fields for chinese dialogue act recognition, Neurocomputing 168 (2015) 408–417.

[81] L. Zhao, Q. Hu, W. Wang, Heterogeneous feature selection with multi-modal deep neural networks and sparse group LASSO, IEEE Trans. Multimedia 17 (11) (2015) 1936–1948.

[82] A. Wang, J. Lu, J. Cai, T. Cham, G. Wang, Large-margin multi-modal deep learning for RGB-d object recognition, IEEE Trans. Multimedia 17 (11) (2015) 1887–1898.

[83] N. Neverova, C. Wolf, G. Taylor, F. Nebout, Moddrop: adaptive multi-modal gesture recognition, IEEE Trans. Pattern Anal. Mach. Intell. 38 (8) (2016) 1692–1706.

[84] S. Rastegar, M.S. Baghshah, H.R. Rabiee, S.M. Shojaee, MDL-CW: a multimodal deep learning framework with crossweights, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2016, pp. 2601–2609.

[85] Q. Zhang, L.T. Yang, Z. Chen, Deep computation model for unsupervised feature learning on big data, IEEE Trans. Serv. Comput. 9 (1) (2016) 161–171.

[86] Q. Zhang, L.T. Yang, Z. Chen, Privacy preserving deep computation model on cloud for big data feature learning, IEEE Trans. Comput. 65 (5) (2016) 1351–1362.

[87] L.M. Fu, H. Huang, J.C. Principe, Incremental backpropagation learning networks, IEEE Trans. Neural Netw. 7 (3) (1996) 757–761.

[88] S. Wan, L.E. Banta, Parameter incremental learning algorithm for neural networks, IEEE Trans. Neural Netw. 17 (6) (2006) 1424–1438.

[89] S. Shalev-Shwartz, Online learning and online convex optimization, Found. Trends Mach. Learn. 4 (2) (2012) 107–194.

[90] L. Zhao, J. Chen, F. Chen, W. Wang, C. Lu, N. Ramakrishnan, Simnest: social media nested epidemic simulation via online semi-supervised deep learning, Proceedings of IEEE International Conference on Data Mining, IEEE, 2015, pp. 639–648.

[91] L. Yu, H. Chen, Q. Dou, J. Qin, P.A. Heng, Integrating online and offline three-dimensional deep learning for automated polyp detection in colonoscopy videos,

IEEE J. Biomed. Health Inf. 21 (1) (2017) 65–75.
[92] F. Bu, Z. Chen, Q. Zhang, Incremental updating method for big data feature learning, Comput. Eng. Appl. 51 (12) (2015) 21–26.
[93] R. Elwell, R. Polikar, Incremental learning in nonstationary environments with controlled forgetting, Proceedings of International Joint Conference on Neural Networks, IEEE, 2009, pp. 771–778.
[94] K. Chen, Q. Huo, Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2016, pp. 5880–5884.
[95] C. Bizer, P. Boncz, M.L. Brodie, O. Erling, The meaningful use of big data: four perspectives – four challenges, ACM SIGMOD Rec. 40 (4) (2012) 56–60.
[96] O. Kwon, N. Lee, B. Shi, Data quality management, data usage experience and acquisition intention of big data analytics, Int. J. Inf. Manage. 34 (3) (2014) 387–394.
[97] D. Becker, T.D. King, B. McMullen, Big data, big data quality problem, Proceedings of IEEE International Conference on Big Data, IEEE, 2015, pp. 2644–2653.
[98] P. Ciancarini, F. Poggi, D. Russo, Big data quality: a roadmap for open data, Proceedings of IEEE International Conference on Big Data Computing Service and Applications, IEEE, 2016, pp. 210–215.
[99] P. Vincent, H. Larochelle, Y. Bengio, P. Manzagol, Extracting and composing robust features with denoising autoencoders, Proceedings of International Conference on Machine learning, ACM, 2008, pp. 1096–1103.
[100] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (2010) 3371–3408.
[101] F. Bu, Z. Chen, Q. Zhang, Incomplete big data impputation algorithm based on deep learning, Microelectr. Comput. 31 (12) (2014) 173–176.
[102] W. R, D. Tao, Non-local auto-encoder with collaborative stabilization for image restoration, IEEE Trans. Image Process. 25 (5) (2016) 2117–2129.