International Journal of Semantic Computing
Vol. 10, No. 3 (2016) 417–439

DOI: 10.1142/S1793351X16500045

Technical Survey

Deep Learning

Xing Hao
EECS, University of California
Irvine, CA 92617, USA
xingh2@uci.edu
http://xinghao.info

Guigang Zhang
Institute of Automation
Chinese Academy of Sciences
Beijing 100190, China
EECS, University of California
Irvine, CA 92617, USA
guigang.zhang@ia.ac.cn, guigangz@uci.edu

Shang Ma
EECS, University of California
Irvine, CA 92617, USA
shangm@uci.edu

Deep learning is a branch of machine learning that tries to model high-level abstractions of data using multiple layers of neurons consisting of complex structures or non-liner transformations. With the increase of the amount of data and the power of computation, neural networks with more complex structures have attracted widespread attention and been applied to various fields. This paper provides an overview of deep learning in neural networks including popular architecture models and training algorithms.

Keywords: Deep learning; neural networks; training.

1. Introduction

The idea of creating machines that think has a very long history which can be traced back to the time of ancient Greece. The invention of programmable computers in 1940s makes the scientists began to seriously explore the possibility of constructing an electronic brain. In 1956, a meeting held at the Dartmouth College formally established the research field of Artificial Intelligence. At the same time, millions of dollars were put into the AI research.

Eventually, researchers found they greatly underestimated the difficulty of this project. In the first 20 years of AI research, people think as long as the machines have

logical reasoning ability, they can be intelligent. Early AI can solve very complex mathematical problems that can be described by a list of formal, mathematical rules like playing cheese, but the real challenge is to solve the problem which humans can solve intuitively, such as image or speech recognition.

To solve these more intuitive problems, only logical reasoning is not enough. Scientists started to believe that we must try to make the machine learn knowledge. Machine learning is to design and analyze algorithms which can let the computer automatically learn knowledge. Machine learning algorithm automatically analyzes and extracts the patterns obtained from data, and use patterns of data to make predictions.

From the perspective of the hierarchical model, the development of machine learning has gone through two major steps since the late 1980s: shallow learning and deep learning.

In the late 1980s, the invention of back propagation algorithm for artificial neural network set off a wave of machine learning based on statistical models. It was found that the use of back propagation algorithm allows an artificial neural network model to learn patterns from a lot of training data, and make predictions based on those patterns. This method beat previous systems in many ways. The artificial neural network at that time is a shallow model containing only one hidden layer.

In 1990s, a variety of shallow machine learning models have been proposed, such as support vector machines (SVM), Boosting, the maximum entropy method including Logistic Regression (LR) and so on. The models have one hidden layer (such as SVM, Boosting), or no hidden layer (such as LR). These models showed great success in both theoretical analysis and applications. Instead, because of the complexity of theoretical analysis, and the high requirement for experience and skills, artificial neural network was relatively unpopular at this period.

Since 2000, the rapid development of the Internet has presented a huge demand for intelligent analysis and prediction of large amounts of data. The shallow machine learning was very successful in this area. The most successful applications include CTR estimation for advertising search systems, page rank systems, spam filtering systems, recommendation systems, and so on.

In 2006, an article by Geoffrey Hinton [1] opened the gate of deep learning in academia and industry. There are two main messages in the article: artificial neural network with more hidden layers has better learning ability; the difficulty in training deep neural network can be overcome by layer-wise pre-training.

Since then, the research of deep learning has continued to heat up. Since 2011, Microsoft and Google have worked on speech recognition using deep neural networks and succeeded to reduce the error rate by 20% to 30%, which is a major breakthrough in the field of speech recognition. In 2012, the deep neural network technology has achieved amazing results in the field of image recognition, and the evaluation result in ImageNet [2] shows that the error rate can be reduced from 26% to 15%.

Today, Google, Microsoft, and other well-known high-tech companies with large data are eager to put resources to deep learning study, because they believe that

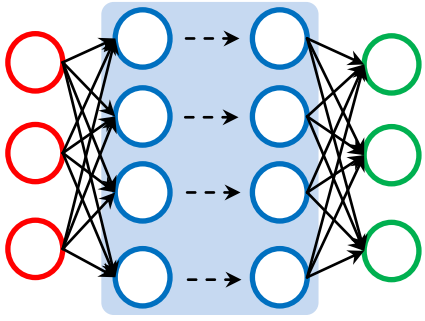


Fig. 1. A simple example of deep neural network.

more complex and powerful models can reveal the wealth of information with the huge amounts of data, and make accurate predictions for future or unknown events.

Different from shallow machine learning, deep learning uses a cascade of layers of nonlinear processing units for feature extraction and transformation. It allows computers to learn from a hierarchical representation of the data where higher level features are derived from lower level features. Figure 1 gives a simple example of deep neural network, in which the inputs of deep learning algorithms are transformed through several hidden layers and the outputs are derived from the computation of the hidden layers.

Given an image, for example, a deep neural network can extract different features of this image in each hidden layer. For example, given the pixels of this image as the input, edges can be identified in the first layer by comparing colors or brightness of neighboring pixels. Then, the second hidden layer can detect corners and contours using the description of edges. After that, the parts of specific objects can be found by searching specific collections of contours and corners. Finally, the objects in the input image can be recognized.

Deep learning has drawn a lot of attentions from academia to industry. In applications, deep learning has made huge progress in speech and image, thus promoting the advancement of artificial intelligence and human-computer interaction. On the other hand, there are still some questions about deep learning which we need to think about.

Compared to the shallow model, the deep model has a better representation ability of nonlinear functions. However, it is difficult to calculate how many resources we need to obtain a better deep model through training and to find the ideal optimization algorithm. Since the deep model is non-convex, theoretical research in this area may be difficult.

In industry, how to take advantage of massively parallel computation to do massive data training is a very important problem. Existing deep neural network training techniques mostly used the stochastic gradient method (SGD). This method unfortunately is unable to run in parallel.

In the rest of this article, we will review the work related to deep learning. As the first step, we will talk about the architectures in Sec. 2 and the training algorithms in

Sec. 3. Following them, we will introduce the most popular deep learning frameworks in Sec. 4, which is then followed by a discussion of the existing difficulties of deep learning research in Sec. 5, the relationship between deep learning and big data in Sec. 6, and some applications of deep learning in Sec. 7.

2. Deep Learning Architectures

The research of deep learning attempts to model large-scale data by multiple processing layers with complex structures. Thus, unlike the architectures of shallow machine learning, deep learning architectures are composed of multiple non-linear transformations as shown in Fig. 1.

There are a huge number of variants of deep architectures, and different architectures can be used to represent different data sources. For example, convolutional neural network is the most popular architecture for image recognition, and recurrent neural network is more applicable to sequential tasks such as handwriting or speech recognition. In this section, we will introduce these architectures and discuss why they are suitable to handle different data.

2.1. *Convolutional neural networks (CNNs)*

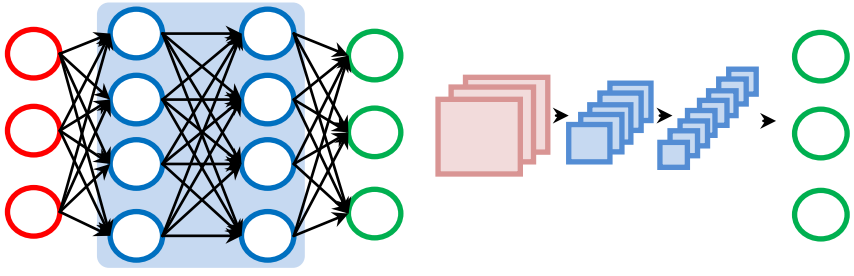
Convolutional neural network (CNN) is a type of feed-forward artificial neural network which uses convolution in at least one of their layers. It was inspired by biological neural networks. CNN combines artificial neural networks and discrete convolution for image processing which can be used to automatically extract features. Thus it is particularly designed for recognizing two-dimensional data, such as images and videos. Images can be directly used as the input of the network, which avoids the complex feature extraction and data reconstruction process in traditional image recognition algorithms.

In 1980, Kunihiko Fukushima proposed a neural network model for a mechanism of visual pattern recognition — "neocognitron" [3], which is the predecessor of CNN.

After that, researchers tried different ways [4–10] to train multi-layer neural networks, and applied them to various scenarios. However, while the breadth and depth of a network increases, the performance of CNN algorithms was limited by computing resources.

After 2006, efficient GPUs became generalized computing devices, which made training larger networks possible. Thus, several models [11–13] have been proposed to train deep convolutional neural networks more efficiently. Today, CNN is used in many areas including image and video understanding [2, 14–17].

Convolutional Neural Networks are modeled as collections of neuron units which are connected in an acyclic graph. Figure 2(a) shows an example of a convolutional neural network with 2 hidden layers. Each layer of this network takes the outputs of the previous layer as the inputs. Therefore cycles are not allowed in the CNN architecture. Typically, the neuron units between adjacent two layers are fully-



(a) CNN architecture (b) Image recognition

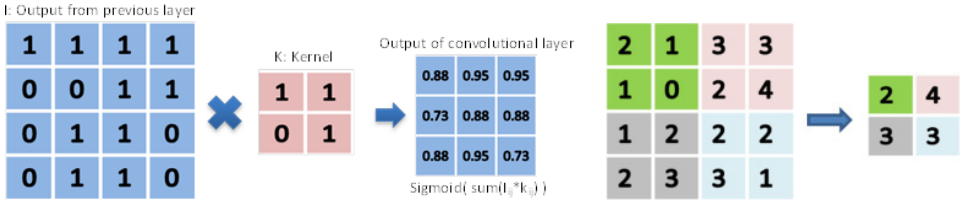
Fig. 2. A convolutional neural network.

connected, but neurons within a single layer share no connection. A deep neural network contains more than 2 layers. There might be 20 or more layers and each layer can be functionally different.

Take image recognition as an example, where CNN arranges its neurons in three dimensions (width, height, depth), as shown in Fig. 2(b). The input of CNN in Fig. 2 (b) is an image, so its width and height are the dimensions of the image, and the depth is 3 (Red, Green, Blue channels). The output of the CNN is a single vector of class scores, arranged along the depth dimension. Between input and output, three main types of layers are used to build CNN architectures including the convolutional layer, the pooling layer, and the fully-connected layer. And there is also a loss function (e.g. SVM/Softmax) in the last layer. Sometimes, other types of layer are used to improve the performance of a CNN such as drop out layer which is used to control the size of CNN.

To summarize in more details:

- The input layer holds the raw pixel values of the image.
- The convolutional layer enhances the features of input data and reduce noise. The input of a convolutional layer is the output from the previous layer. For each input, there is a kernel with weights inside its cells. The kernels inside a convolutional layer have the same size. As shown in Fig. 3(a), the kernel size is 2*2, and the input of this convolutional layer is 4*4. We move the kernel from top-left



(a) Convolutional layer (b) Max pooling

Fig. 3. Different layers of a convolutional neural network.

to bottom-right of the input with step = 1 and gets an output with the size of $(4-2+1)*(4-2+1)=9$. Each number in this output is computed by two steps. The first step is to compute the summation of the weighted numbers covered by the kernel. The second step is a sigmoid function which squeezes the result of the first step to a fraction ranged between 0 and 1. The size of the output of the convolutional layers is decided by the size of the input and the kernel. The number of the output is pre-defined by the user and it determines the number of kernels.

- The pooling layer takes a subsample from the input data. The pooling layer divides the input to several small regions, and it performs some functions such as average or maximum to each region. A max pooling layer is shown in Fig. 3(b). The input with size 4*4 is divided to four regions with size of 2*2, and the output is the max numbers of these four regions. Unlike the convolution layer, there is no overlapping in pooling layer calculation.
- The fully-connected layer takes all neurons from the previous layer and connects them to every single neuron it has. Fully connected layers are not spatially located anymore (we can visualize them to be one-dimensional), so there can be no convolutional layers after a fully connected layer.

To extract the right patterns from data, we need to train a network. The training process is to update the weights in a network to minimize the error generated from the network. We will introduce some training algorithms in Sec. 3.

2.2. *Recurrent neural networks (RNNs)*

RNNs aim to process sequential data. In the traditional feed forward neural network model, like CNNs, data flows from the input layer to the hidden layer and then the output layer. There is no connection between neurons in the same layer. Some problems cannot be fully handled by this kind of neural network. This architecture cannot solve the problem where the input data have relationships with each other. For example, we need the previous word in a sentence to predict the next one, because the words in the same sentence are not independent.

In RNNs, the current output and previous output are relevant. To be more specific, the output of the previous step is stored and used to calculate the current output, that is, the input of the network contains both the data from the input layer and the output of the hidden layers from the previous step. This makes them applicable to tasks such as handwriting recognition [18] and speech recognition [19, 20]. The architecture of RNN is shown in Fig. 4.

In Fig. 4, except for the data flow from the input to the hidden layer and from the hidden layer to the output, which is the same as in CNN, there is also data transferring among neurons in the same hidden layer. A recurrent neural network can be unfolded into an acyclic neural network, as shown in Fig. 5. For example, a recurrent neural network of a five-word sentence can be unfolded to a five-layer neural network with each layer representing a word.

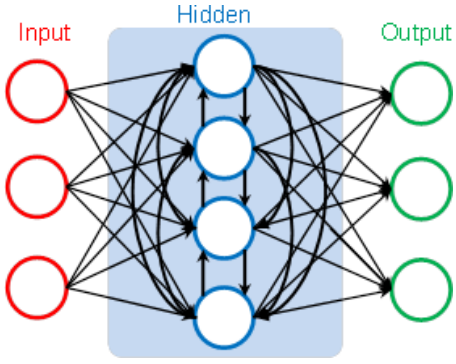


Fig. 4. A recurrent neural network.

In Fig. 5, the input and output of the recurrent neural network is denoted as x and y. t represents the number of steps. For example, x_1 and y_1 is the input and output of step 1, where x_1 might be the first word in one sentence, and y_1 is the predicted next word. s_t, which is the network state of step t, can be calculated using the input of step t x_t and the output of the hidden layer in the previous step s_{t-1} by

$$s_t = f(U * x_t + W * s_{t-1}) \tag{1}$$

where f is normally a nonlinear activation function. s_t can also be viewed as the memory units in the hidden layers which can be used to store the information from the previous steps.

In convolutional neural networks, the parameters of different layers are different. However, in RNN, each layer shares the parameters U, V, and W. That means each step in RNN is doing the same thing, just with different inputs. This greatly reduces the requirement to learn the parameters of the network.

Over the years, researchers have proposed many sophisticated RNNs to deal with the shortcomings of the regular RNN model. Here are some widely used RNNs models:

- Simple RNNs [21] is a special case of RNNs, which is a three-layer network, and it adds context units inside the hidden layer. The connections between the context

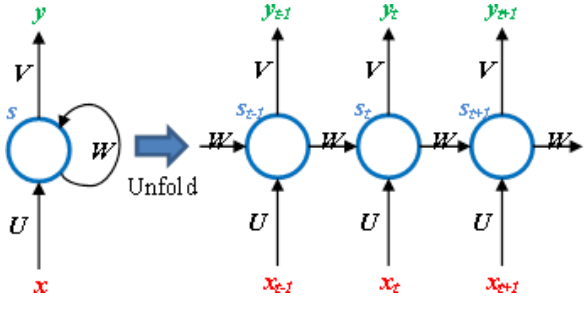


Fig. 5. An unfolded recurrent neural network.

units and the neurons are fixed, and so are the weights. At each step, it is trained like what we do with a feed forward neural network. The output of neurons is saved to the context units which connect to them, and then is used as the input of the neurons for next step.

- Deep Bidirectional RNNs [22] is based on the idea that the current output is not only related with the previous elements, and also related to the future elements. For example, in the case of word prediction, we will need both the left and right context to predict a missing word in one sentence. Bidirectional RNN is a relatively simple RNN, which is composed of two RNNs stacked on top of each other, and the output is determined by the status of the hidden layers in these two RNNs.
- Echo State Networks [23] is a special type of RNN. The basic idea is to use a large-scale randomized connecting cycle network to replace the hidden layers in a classical neural networks, which simplifies the training process.
- Gated Recurrent Unit Recurrent Neural Networks [24] give different weights to previous states according to their differences with the current state, and only a few weights are updated to minimize the loss function.
- Long short-term memory (LSTM) [19] is very popular recently. RNN can deal with the problem when there are some relationships or dependences among the input data. However RNN may lose the ability to learn the connections among the input data while their dependencies are very long. For example, given a sentence "the clouds are in the", RNN can predict the next word is 'sky', but it is difficult for it to predict the next word of the sentence "I grew up in France. I speak fluent". LSTM is proposed to solve this problem. In LSTM, neurons are replaced by blocks. In each block, there is a cell used to remember the information from all the input data. And one component named "forget gate" is used to delete the unimportant information from a cell such as 'the' or 'of'.
- Clockwork RNNs (CW-RNNs) [25] divide the hidden layer into several groups. Each group processes its input according to its own clock frequency.

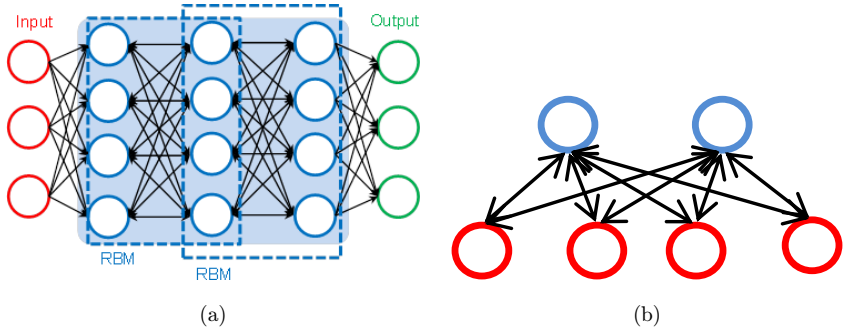
RNN has been proven to be very successful in nature language processing (NLP), including language modeling and generating text [26, 27], machine translation[28–30], and speech recognition [20, 31]. The most successful and widely used RNN model is LSTM.

There are some differences between RNN and CNN. First, the parameters W, U, V are shared for an unfolded RNN. Second, the output of each step depends on not only the current input, but also a number of previous steps of the network. So people use another algorithm to train RNN which is called Backpropagation Through Time (BPTT). We will introduce this algorithm in Sec. 3.

2.3. *Deep belief networks (DBNs)*

The idea of Deep belief networks (DBNs) was proposed by Geoffrey Hinton in 2006 [1]. It not only can be used to recognize and classify data, but also can generate data.

DBN consists of multi-layers of neurons. These neurons are divided into hidden units and visible units. Visible units are used to receive data, while hidden units are



(a) (b)

Fig. 6. (a) Deep Belief Network and (b) Restricted Boltzmann Machine.

used to extract features, so they are also called feature detectors. A DBN is composed of multiple layers of Restricted Boltzmann Machines (RBM). The training of the DBN is processed layer by layer. In each layer, the input data is used to calculate the hidden units, and then the hidden units are given to the next layer as the input data. Figure 6 shows an example deep belief network and restricted Boltzmann machine.

The training process of DBN includes two main steps:

Step 1: Pre-train: train RBM separately in an unsupervised way. Ensure that when feature vectors are mapped to a different space, information is preserved as much as possible. We will introduce this algorithm later in Sec. 3;

Step 2: Fine-Tuning: Using the error from the output layer to update the weights in DBN. This process is supervised training similar to training CNN.

Compared to traditional neural networks, DBN can be trained layer by layer. This significantly reduces the resource needed to train the network.

2.4. *Other architectures*

Besides CNNs, RNNs, and DBNs, there are many other deep network architectures including:

- Deep stacking network (DSN) [32] that consists of multiple stacked blocks, where each block module is simple and easy to train in a supervised way. DSN is mainly used for classification and regression of images and language [33, 34].
- Deep coding network (DPCN) [35, 36] that is a predictive coding scheme which predicts the representation of a layer by using a top-down approach.

Other architectures include Spike-and-slab RBMs [37] and Deep Q-networks [38].

3. Training Algorithms

As mentioned in Sec. 2, we need to train or teach a deep learning model before we can use it to do classification or prediction. Training of a machine learning model can be

summarized to several categories:

- Supervised learning learns a function from given training data. When new data arrives, the output can be predicted according this function. The training data set of supervised learning includes both the input and the required output, which are usually labeled by human.
- Unsupervised learning learns from an unlabeled training data set.
- Semi-supervised learning learns from a small amount of labeled data with a large amount of unlabeled data.
- Reinforcement learning learns how to make actions through observations. Each action has an impact on the environment. Reinforcement learning makes judgments based on the feedback from the observed environment.

In this section, we will discuss the first two categories which are widely used in training deep models.

3.1. *Supervised learning*

Supervised learning is to training a network using the existing training samples including the data and its corresponding output. Supervised learning is normally used to do data classification and prediction.

3.1.1. *Backpropagation (BP)*

The backpropagation algorithm is used in many machine learning scenarios to train learning models. It was originally introduced in the 1970s, but its importance was not fully noticed until David Rumelhart's work in 1986 [39]. They described several neural networks where backpropagation works far faster than earlier approaches. Today, the backpropagation algorithm is the most widely used algorithm for unsupervised learning in neural networks.

In this section, we will introduce backpropagation algorithm using a small example as shown in Fig. 7 which is a neural network with two hidden layers. In this

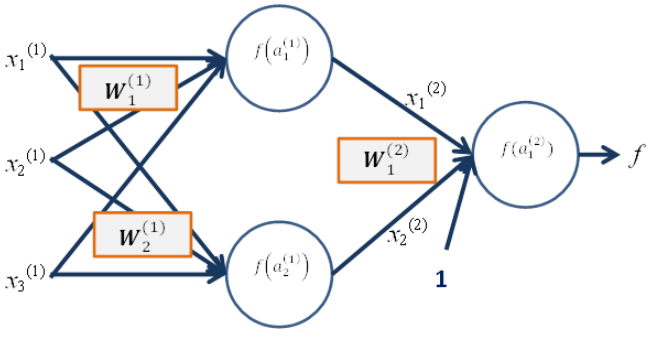


Fig. 7. Example of neural network.

network, each circle denotes a neuron including two steps of calculation. The first step is to compute the weighted summation of the input data. We call the result of this step activation, and denote it as a. The second step is a sigmoid function which squeezes a to range between 0 and 1. We use f to represent this function.

The input of the first layer is a vector with 3 numbers $X^{(1)} = \{x_1{}^{(1)}, x_2{}^{(1)}, x_3{}^{(1)}\}$, and the output of this layer is $f_1{}^{(1)}$ and $f_2{}^{(1)}$, where:

$$\begin{aligned} f_1{}^{(1)} &= f(a_1{}^{(1)}) = 1/(1 + e\hat{}(-a_1{}^{(1)})) \\ &= 1/(1 + e\hat{}(-(x_1{}^{(1)} * w_{11}{}^{(1)} + x_2{}^{(1)} * w_{12}{}^{(1)} + x_3{}^{(1)} * w_{13}{}^{(1)}))) \end{aligned} \tag{2}$$

$$\begin{aligned} f_2{}^{(1)} &= f(a_2{}^{(1)}) = 1/(1 + e\hat{}(-a_2{}^{(1)})) \\ &= 1/(1 + e\hat{}(-(x_1{}^{(1)} * w_{21}{}^{(1)} + x_2{}^{(1)} * w_{22}{}^{(1)} + x_3{}^{(1)} * w_{23}{}^{(1)}))) \end{aligned} \tag{3}$$

and the output of first layer becomes the input of the second layer, that is to say, $x_1{}^{(2)} = f_1{}^{(1)}$ and $x_2{}^{(2)} = f_2{}^{(1)}$. The third input number of the second layer is a constant 1, which is called bias. So the output of the second layer is:

$$\begin{aligned} f_1{}^{(2)} &= f(a_1{}^{(2)}) = 1/(1 + e\hat{}(-a_1{}^{(2)})) \\ &= 1/(1 + e\hat{}(-(x_1{}^{(2)} * w_{11}{}^{(2)} + x_2{}^{(2)} * w_{12}{}^{(2)} + w_{13}{}^{(2)}))) \end{aligned} \tag{4}$$

Suppose we have four data items in our training set, as shown in Table 1 where $\{x_1, x_2, x_3\}$ is the input data and d is its corresponding output.

At the output layer, we use the squared-error function: $\varepsilon = (d-f)^2$ to denote the error of the neural network and update the weights. According to Eqs. (2), (3), and (4), given input $X^{(1)} = \{x_1{}^{(1)}, x_2{}^{(1)}, x_3{}^{(1)}\}$, ε is a function of all the weights $w_{mi}{}^{(l)}$. Thus we can get the minimum of ε using the partial derivative of ε with respect to $w_{mi}{}^{(l)}$: $\nabla\varepsilon(w_{mi}{}^{(l)})$. If we treat all the weights except $w_{m'i'}{}^{(l')}$ as constants, $\varepsilon(w_{m'i'}{}^{(l')})$ is a one-dimensional function, and the minimum of this function is the point where $\nabla\varepsilon(w_{m'i'}{}^{(l')}) = 0$. We can use $\nabla\varepsilon(w_{m'i'}{}^{(l')})$ to determine whether we need to increase or decrease $w_{m'i'}{}^{(l')}$ to get the minimum value of ε: if $\nabla\varepsilon(w_{m'i'}{}^{(l')})$ is less than 0, we need to increase $w_{m'i'}{}^{(l')}$ and if $\nabla\varepsilon(w_{m'i'}{}^{(l')})$ is larger than 0, we need to decrease $w_{m'i'}{}^{(l')}$.

So for each weight we can adjust $w_{mi}{}^{(l)}$ using:

$$w_{mi}{}^{(l)} - c_i{}^{(l)}\nabla\varepsilon(w_{mi}{}^{(l)}) \rightarrow w_{mi}{}^{(l)} \tag{5}$$

Table 1. Training data.

x_1	x_2	x_3	d
1	0	1	0
0	0	1	1
0	1	1	0
1	1	1	1

where ${c_i}^{(l)}$ is the length of the adjust step, which determines how many steps we need before reaching the minimum of the function. If ${c_i}^{(l)}$ is too small, we will take more steps to get the minimum point of the function. If ${c_i}^{(l)}$ is too large, we might swing between the left and right of the minimum point for many times and also need more steps to get the point. This method is called the gradient descent process which has proved to be one of the most efficient ways to get the minimum error for machine learning.

In our example, we will update the weights from the last layer to the first layer; this is why this algorithm is called backpropagation. The derivation of the last layer can be calculated as:

$$\nabla\varepsilon({w_{1i}}^{(2)}) = \partial\varepsilon/\partial {w_{1i}}^{(2)} = \partial(d-f)^2/\partial {w_{1i}}^2 = -2(d-f)f(1-f) * {x_i}^{(2)} \tag{6}$$

So given the step size ${c_i}^{(l)}$, we can update the weights in the second layer using Eq. (6) and the output of the first layer.

The derivation of the first layer can be computed as:

$$\begin{aligned}\nabla\varepsilon({w_{mi}}^{(1)}) &= \partial(d-f)^2/\partial {w_{mi}}^{(1)} \\ &= -2(d-f) * (\partial f({a_m}^{(2)})/\partial {a_m}^{(1)}) * (\partial {a_m}^{(1)}/\partial {w_{mi}}^{(1)})\end{aligned} \tag{7}$$

where ${a_m}^{(1)} = {x_1}^{(1)} * {w_{m1}}^{(1)} + {x_2}^{(1)} * {w_{m2}}^{(1)} + {x_3}^{(1)} * {w_{m3}}^{(1)}$ according to Eqs. (2) and (3). Thus

$$\partial {a_m}^{(1)}/\partial {w_{mi}}^{(1)} = {x_i}^{(1)}. \tag{8}$$

According to Eqs. (2), (3) and (4):

$$\begin{aligned}\partial f({a_m}^{(2)})/\partial {a_m}^{(1)} &= (\partial f({a_m}^{(2)})/\partial {a_m}^{(2)}) * (\partial {a_m}^{(2)}/\partial {a_m}^{(1)}) \\ &= f(1-f) * {w_{mi}}^{(2)} * (\partial {a_m}^{(1)}/\partial {a_m}^{(1)}) \\ &= f(1-f) * {w_{mi}}^{(2)} * {f_m}^{(1)} * (1-{f_m}^{(1)})\end{aligned} \tag{9}$$

So, given the output of the first layer and the second layer, the weights in the first layer can be updated using the updated weights in the second layer using Eqs. (7), (8), (9).

Normally, we use all the training data to update the weights many times until the error in the output layer of all the training data is less than some threshold. Table 2 summarizes the backpropagation algorithm.

Table 2. The Backpropagation algorithm.

Backpropagation
Initial random weights
Do until stopping criteria is met:
For each input (x, y):
Do a feed-forward pass to compute activations at all hidden layers, then at the output layer obtain an output y'
Measure the deviation of y' from the target y
Backpropagate the error through the net and perform weight updates

3.1.2. *Backpropagation through time (BPTT)*

BPTT is a supervised learning algorithm to train RNN. It is similar to the back-propagation algorithm, except that the weights U, V, and W in RNN are shared by each step as shown in Fig. 5. So we need to sum up the contributions of each time step to the gradient.

The states and estimated outputs of each step in RNN are denoted as:

$$s_t = \tanh(U * x_t + W * s_{t-1})$$
$$y_t = \text{softmax}(V * s_t)$$

And the loss function is defined as cross entropy loss:

$$E(d_t, y_t) = -\Sigma d_t \log y_t$$

where d_t is the correct word in step t, and y_t is the estimation or prediction. We usually use the total error of all the words in one sentence to update the weights. Like how we sum up the errors, we sum up the gradients in each time step for one training sample:

$$\partial E / \partial W = \Sigma \partial E_t / \partial W$$

The BPTT algorithm is shown in Table 3.

Table 3. Backpropagation through time algorithm.

Backpropagation through time
Initial random weights
Unfold the network
Do until stopping criteria is met:
x = the zero-magnitude vector
for t from 0 to n – 1:
Set the network inputs to x, a[t], a[t+1], ...
Do a feed-forward pass to compute activations at all hidden layers, then at the output layer obtain an output y'
Measure the deviation of y' from the target y
Backpropagate the error through the net and perform weight updates
Average the weights in each step, so that the parameter is identical
$x = f(x)$

3.2. *Unsupervised learning*

In unsupervised learning, we give our algorithms a large amount of unlabeled data to learn a good feature representation of the input. Unsupervised learning is often used to do data clustering and encoding.

3.2.1. *Autoencoder*

Autoencode [40] is one of the most widely used unsupervised learning algorithms for deep neural networks. Autoencoder is simply a compression encoder, which is to

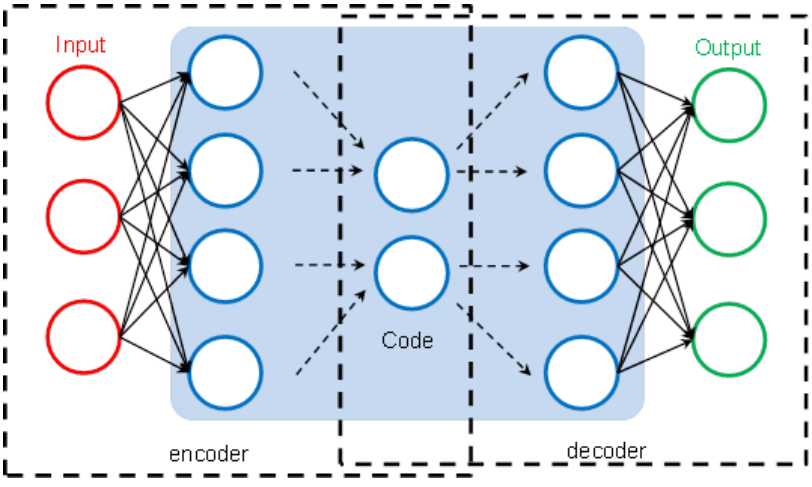


Fig. 8. Autoencoder.

transform things by changing the input to an output which is the same as the input. That is to find a function that enables the function (input) = input, which is called the identity function. Figure 8 shows the schematic structure of an autoencoder.

The intuition behind this algorithm is that the network does not learn the training data between the tag and "map", but instead it learns its own internal data structure and characteristics (and therefore, the hidden layer becomes a feature detector). Usually the number of hidden units is less than the number of input/output layers, forcing a network to learn only the most important features and achieve dimensional reduction.

We want to learn some small nodes in the medium-scale conceptual level data to produce a compressed representation, which somehow captures the core features of the input.

The training algorithm for an autoencoder is summarized in Table 4.

Table 4. Autoencoder.

Autoencoder
Initial random weights
Do until stopping criteria is met:
For each input x :
Do a feed-forward pass to compute activations at all hidden layers, then at the output layer obtain an output x'
Measure the deviation of x' from the input x
Backpropagate the error through the net and perform weight updates

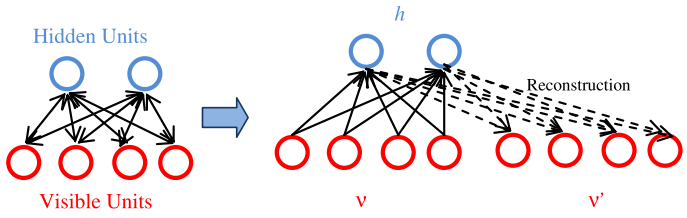


Fig. 9. Unfold RBM.

3.2.2. *Training restricted Boltzmann machine*

Actually, when RBM is unfolded, it is similar to autoencoder, as shown in Fig. 9. The training of RBM is also similar to the training of autoencoder. We first calculate the value of hidden units h using the vector from the visible units v and then reconstruct the visible units v'. Using the difference between v and v', we can update the weights between the visible and hidden units.

The detailed algorithm for training RBM is shown in Table 5.

Table 5. Restricted Boltzmann machine.

Restricted Boltzmann machine
Initial random weights
Do until stopping criteria is met:
For each input vector $X = \{x_1, x_2, \ldots.\}$:
Calculate h_j using X, and denote positive$(w_{ij}) = x_i * h_j$
Reconstruct X', and denote negative$(w_{ij}) = h_j * x_i$
Perform weight updates: $w_{ij} = w_{ij} + \alpha * (\text{positive}(w_{ij}) - \text{negative}(w_{ij}))$

When RBMs are put together layer by layer where the visible units of the upper layer are the hidden units of the bottom layer, they form DBN, as mentioned in Sec. 2.

4. Deep Learning Frameworks

There are many open source deep learning frameworks from startups in this area. We will talk about these frameworks in this section.

Caffe (Convolutional Architecture for Fast Feature Embedding) [41] was developed by the Berkeley Vision and Learning Center. It may be the most popular toolkit in computer vision. Caffe has an excellent convolution neural network implementation, but the support for recurrent neural networks is poor. Caffe provides an interface for C++, Python, and Matlab and can be run on both CPU and GPU.

Tensorflow [42] was developed by Google. It has an ideal recurrent neural network API and implementation. Tensorflow supports heterogeneous distributed

computing, and it can be run on different platforms, from mobile phones, a single CPU/GPU to distributed systems with hundreds of GPUs. It supports C++ and Python, but cannot be implemented on Windows.

CNTK [43] was developed by Microsoft which was firstly only used for Cortana personal assistant of Microsoft and Skype Translator services, and was released in April 2015. CNTK allows systems to recognize human speech.

Theano [44] was born in Montreal Polytechnic. Many deep learning packages were derived from Theano including Blocks and Keras. It provides a Python interface and can support most of the neural network models.

Torch [45] has existed for more than ten years. It is implemented on LuaJIT, which has better speed compared to C++, C# Java. The only problem is that Lua is not a mainstream programming language.

Besides, there are many other deep learning frameworks such as Brainstorm [46] which can be used to handle super-deep neural networks with hundreds of hidden layers, Deeplearning4j [47] which is a deep learning framework for Java, and MXnet [48] which emphasizes the efficiency of memory usage, and so on.

5. Deep Learning and Big Data

In the industry most people think shallow machine learning is more effective to handle big data. For example, in many big data applications, the simplest linear model has been widely used. But recently the progress of deep learning prompted us to rethink this idea. In the case of big data, perhaps only more complex models or models with high expression ability can fully exploit the wealth of information hidden in the vast amounts of data. Maybe we can learn more valuable information and knowledge using deep learning.

The shallow model has an important feature — extract patterns or features from sample data by experience, and the shallow model is mainly responsible for classification or prediction. So the quality of extracted patterns becomes the bottleneck in overall system performance. Thus, generally more effort of a development team is devoted to discover better patterns. To find a good pattern requires developers to have a deep understanding of the problem. Thus, artificially designing sample patterns may not be a good approach.

On the other hand, deep learning aims to learn more useful features from big data via constructing deep models with multiple hidden layers and training vast amounts of data. Different from shallow machine learning, deep learning emphasizes the depth of the model and the importance of training, so that it can improve the accuracy of classification or prediction by extracting more appropriate features. Compared with the artificial designing method, deep learning use training of big data to learn features, which has a better ability to express the features of rich information. So, in the next few years, we may see more and more big data applications using deep learning instead of a linear shallow model.

6. Applications of Deep Learning

Deep learning is an important breakthrough achieved in the field of artificial intelligence over the past decade especially in speech recognition, natural language processing, computer vision, image and video analysis, and multimedia. In this section, we list some of the application examples and summarize their work.

6.1. *Image understanding*

The most influential breakthrough of computer vision occurred in 2012 when Hinton's team used deep learning to win ImageNet image classification champion [2]. Their results generated a great deal of shock in the field of computer vision and opened a gate to deep learning. And in ImageNet ILSVRC 2013 competition, all the top 20 teams used deep learning technology. Among them the winner was Rob Fergus of New York University (Rob Fergus) and the error rate was 11.197%. Their model is called Clarifai [49]. In ILSVRC 2014 the winner GoogLeNet [16] reduced the error rate to 6.656%.

Object detection is one task that is more difficult than object classification. An image may include a number of objects belonging to different classes and object detection is to determine the location and type of each object. In ImageNet ILSVRC 2013 the organizer added the object detection tasks which were required to detect objects belonging to 200 class in 40,000 images. The winners used manual design features, and the mean Averaged Precision (mAP) was only 22.581%. In ILSVRC 2014, deep learning increased the number to 43.933%. Other influential work that employed deep learning to machine vision includes RCNN [50], Overfeat [51], GoogLeNet [16], DeepID-Net [52], network in network [53], VGG [54], and spatial pyramid pooling in deep CNN [55].

Another important breakthrough in deep learning on object recognition is facial recognition. The biggest challenge of facial recognition is to distinguish between intra-class variation due to different light, gesture and facial expressions and extra-class variations caused by different identities. Distribution of these two changes is nonlinear and highly complex. Traditional linear models cannot distinguish them effectively. The objective of deep learning is to obtain new feature representations by multiple layers of nonlinear transformation. These new features have to remove as much as possible intra-class variations, while retaining extra-class variations. In IEEE International Conference on Computer Vision and Pattern Recognition Conference 2014, DeepID [56] and DeepFace [57] used face recognition as the supervisory signal and achieved 97.45% and 97.35% recognition rate on the LFW (Labeled Faces in the Wild) face database.

6.2. *Speech understanding*

Automatic Speech Recognition (ASR), which is aimed to enable natural human–machine interaction, has been an intensive research area for decades. There are many

commercial systems for speech recognition such as AT&T Watson [58], Microsoft Speech Server [59], Google Speech API [60] and Nuance Recognizer [61].

Since its launched in 2009, Google Voice [62] transcription has used Gaussian Mixture Model (GMM) acoustic models, the state of the art in speech recognition, for 30+ years. Sophisticated techniques like adapting the models to the speaker's voice have augmented this relatively simple modeling method. Then around 2012, Deep Neural Networks (DNNs) revolutionized the field of speech recognition. These multi-layer networks distinguish sounds better than GMMs by using "discriminative training," differentiating phonetic units instead of modeling each one independently. Technologies have improved rapidly with Recurrent Neural Networks (RNNs), and especially LSTM RNNs that were first launched in Android's speech recognizer in May 2012. Compared to DNNs, LSTM RNNs have additional recurrent connections and memory cells that allow them to "remember" the data they've seen so far.

6.3. *Video understanding*

Deep learning for the classification of video applications is still in its infancy. The deep learning model obtained from ImageNet can be used to describe the static image features of videos. The difficulty is how to describe the dynamic characteristics. The most direct approach is considering video as three-dimensional images, and directly using a convolution neural network [63]. But this idea apparently did not take into account the differences of the time dimension and the space dimension. Another simple but more effective idea is to pre-calculate the spatial statistics of optical flows or other dynamic features as the input of convolution [64–66]. There are also studies of using deep an auto-encoder to extract dynamic features in a nonlinear manner [65]. In the latest research work [67], Long Short-Term Memory (LSTM) attracted widespread attention, which is a special type of recurrent neural networks to model the variable-range dependencies entailed in the task of video understanding.

6.4. *Nature language processing (NLP)*

In addition to voice and image, another application of deep learning is natural language processing (NLP). The first NLP application using neural networks is the language model. In 2003, Yoshua Bengio [68] proposed to map a word to a vector apace using embedding, and then use a non-linear neural network to represent the N-Gram model. Today, deep learning has been used in many NLP applications such as text generation and translation.

Overall, the progress of deep learning made on NLP is not as impressive as on voice or image. Compared to voice and image, language is the only artificial signal that is produced and processed by the human brain. So we believe there is still much to explore for NLP using deep learning.

6.5. *Complex image recognition*

Complex image recognition is a difficult area using the existing shallow learning algorithms. In the real world, we often have to process lots of complex image recognition problems. For example, how to recognize handwriting?

6.6. *Target recognition*

Often, we can find some objects especially some high speed moving objects, but it is very difficult to know what they are. One reason is that we can only crawl very little information when finding objects. And so, we need to develop some new methods to recognize these targets. Deep learning may do this work well in the future.

7. The Difficulty of Deep Learning

Although deep learning has made significant progress and been used in many applications in the last few decades, there are still some difficulties that we need to consider and solve in the future.

7.1. *Theoretical difficulties*

Although deep learning has achieved a great success in practice, there is still much work to be done on theoretical analysis. For example, when will the network convergence? How to achieve better local features? What invariance extracted from transform in each layer and what information is lost in this procedure? Since the deep model is non-convex, theoretical research in this area may be difficult. Recently Mallat did quantitative analysis for deep network using wavelet [69], which may be an important direction.

7.2. *Modeling difficulties*

Since theoretical analysis is so difficult for current deep models, is it possible to design a new model which has the same strong learning ability as the existing models and at the same time is easier to analyze? Besides, as we mentioned in Sec. 2, different architectures are only suitable for different applications. So another question is how to design a new unified model that can handle different types of data such as language, speech and image? Those are the questions we need to think while promoting the study of theoretical analysis and computing the ability of current deep models.

7.3. *Industry difficulties*

For Internet companies, how to take advantage of massively parallel computing platforms to train big data is a most important problem when using deep learning technologies. Traditional big data platforms such as Hadoop are not suitable for deep learning computation with frequent iterations due to their high latency. Existing DNN training techniques mostly use stochastic gradient method (SGD) as the

training method. This method is unlikely to be parallelized among multiple computers. Even training with GPU, traditional DNN learning process will still take very long time. With the increasing amount of big data on Internet, this slow speed training is unable to meet the needs of Internet applications. Google DistBelief is a parallel computing platform for deep model learning using an asynchronous algorithm. With the development of massive data training technology, the accuracy of speech and image recognition can be further improved in the future.

References

[1] G. E. Hinton, Learning multiple layers of representation, *Trends in Cognitive Sciences* **11**(10) (2007) 428–434.

[2] A. Krizhevsky, I. Sutskever and G. E. Hinton, Imagenet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems* **25** (2012) 1097–1105.

[3] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biological Cybernetics* **36**(4) (1980) 193–202.

[4] L. E. Atlas, T. Homma and R. J. Marks II, An artificial neural network for spatio-temporal bipolar patterns: Application to phoneme classification, in *Proc. Neural Information Processing Systems*, 1988, p. 31.

[5] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, in *Proc. of IEEE* **86**(11) (1998) 2278–2324.

[6] S. Behnke, *Hierarchical Neural Networks for Image Interpretation* (Springer Science & Business Media, 2003), p. 2766.

[7] P. Y. Simard, D. Steinkraus and J. C. Platt, Best practices for convolutional neural networks applied to visual document analysis, *ICDAR* **3** (2003) 958–962.

[8] D. Graupe, R. W. Liu and G. S. Moschytz, Applications of neural networks to medical signal processing, in *Proc. 27th IEEE Conference on Decision and Control*, 1988, pp. 343–347.

[9] D. Graupe, B. Vern, G. Gruener, A. Field and Q. Huang, Decomposition of surface EMG signals into single fiber action potentials by means of neural networks, in *Proc. IEEE International Symposium on Circuits and Systems*, 1989, pp. 1008–1011.

[10] Q. Huang, D. Graupe, Y. F. Huang and R. Liu, Identification of firing patterns of neuronal signals, in *Proc. 28th IEEE Conference on Decision and Control*, 1989, pp. 266–271.

[11] K. Chellapilla, S. Puri and P. Simard, High performance convolutional neural networks for document processing, in *Proc. 10th International Workshop on Frontiers in Handwriting Recognition*, 2006.

[12] D. Strigl, K. Kofler and S. Podlipnig, Performance and scalability of GPU-based convolutional neural networks, in *PDP*, 2010, pp. 317–324.

[13] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella and J. Schmidhuber, Flexible, high performance convolutional neural networks for image classification, in *Proc. International Joint Conference on Artificial Intelligence*, 2011, p. 1237.

[14] C. Szegedy, D. Erhan and A. T. Toshev, Object detection using deep neural networks, U.S. Patent 9,275,308. 2016.

[15] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath and B. Kingsbury, Deep neural networks for

acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Processing Magazine* **29**(6) (2012) 82–97.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[17] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, Large-scale video classification with convolutional neural networks, in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[18] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke and J. Schmidhuber, A novel connectionist system for unconstrained handwriting recognition, *IEEE Trans. Pattern Analysis and Machine Intelligence* **31**(5) (2009) 855–868.

[19] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Computation* **9**(8) (1997) 1735–1780.

[20] A. Graves and N. Jaitly, Towards end-to-end speech recognition with recurrent neural networks, *ICML* **14** (2014) 1764–1772.

[21] J. L. Elman, Finding structure in time, *Cognitive Science* **14**(2) (1990) 179–211.

[22] M. Schuster and K. K. Paliwal, Bidirectional recurrent neural networks, *IEEE Transactions on Signal Processing* **45**(11) (1997) 2673–2681.

[23] H. Jaeger, The "echo state" approach to analysing and training recurrent neural networks with an erratum note, German National Research Center for Information Technology GMD Technical Report **148** (2001) 34.

[24] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, G. Bougare, H. Schwenk and Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.

[25] J. Koutnik, K. Greff, F. Gomez and J. Schmidhuber, A Clockwork RNN, in *Proc. 31st International Conference on Machine Learning*, Beijing, 2014, pp. 1845–1853.

[26] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký and S. Khudanpur, Recurrent neural network based language model, *Interspeech* **2** (2010) 3.

[27] I. Sutskever, J. Martens and G. E. Hinton, Generating text with recurrent neural networks, in *Proc. 28th International Conference on Machine Learning*, 2011, pp. 1017–1024.

[28] L. Shujie *et al.*, A recursive recurrent neural network for statistical machine translation, *ACL* (1) (2014).

[29] S. Liu, N. Yang, M. Li and M. Zhou, A recursive recurrent neural network for statistical machine translation, *ACL* (1) (2014) 1491–1500.

[30] M. Auli, M. Galley, C. Quirk and G. Zweig, Joint language and translation modeling with recurrent neural networks, *EMNLP* **3**(8) (2013) 1044–1054.

[31] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney and K. Saenko, Translating videos to natural language using deep recurrent neural networks, arXiv preprint arXiv:1412.

[32] L. Deng and D. Yu, Deep convex net: A scalable architecture for speech pattern classification, *Interspeech* (2011).

[33] L. Deng, X. He and J. Gao, May. Deep stacking networks for information retrieval, in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 3153–3157.

[34] J. Li, H. Chang and J. Yang, Sparse deep stacking network for image classification, arXiv preprint arXiv:1501.00777. 2015.

[35] R. Chalasani and J. C. Principe, Deep predictive coding networks, arXiv preprint arXiv:1301.3541. 2013.

[36] S. Zhou, S. Zhang and J. Wang, August. Deep sparse coding network for image classification, in *Proc.7th ACM International Conference on Internet Multimedia Computing and Service*, 2015, p. 24.
[37] A. C. Courville, J. Bergstra and Y. Bengio, A spike and slab restricted boltzmann machine, *AISTATS* **1** (2011) 5.
[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski and S. Petersen, Human-level control through deep reinforcement learning, *Nature* **518**(7540) (2015) 529–33.
[39] D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning representations by back-propagating errors, *Cognitive Modeling* **5**(3) (1998) 1.
[40] P. Vincent, H. Larochelle, Y. Bengio and P. A. Manzagol, Extracting and composing robust features with denoising autoencoders, in *Proc. 25th International Conference on Machine Learning*, 2008, pp. 1096–1103.
[41] Caffe: http://caffe.berkeleyvision.org/
[42] Tensorflow: http://tensorflow.org/
[43] CNTK: https://www.cntk.ai/
[44] Theano: https://github.com/Theano/Theano
[45] Torch: http://torch.ch/
[46] Brainstorm: https://github.com/IDSIA/brainstorm
[47] Deeplearning4j: http://deeplearning4j.org
[48] MXnet: https://github.com/dmlc/mxnet
[49] Clarifai: http://www.clarifai.com/
[50] R. Girshick, J. Donahue, T. Darrell and J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, (2014), p. 580–587.
[51] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus and Y. LeCun, Overfeat: Integrated recognition, localization and detection using convolutional networks, arXiv preprint arXiv:1312.6229. 2013.
[52] W. Ouyang, P. Luo, X. Zeng, S. Qiu, Y. Tian, H. Li, S. Yang, Z. Wang, C. Qian, Z. Zhu, R. Wang, C. Loy, X. Wang and X. Tang, Deepidnet: Multi-stage and deformable deep convolutional neural networks for object detection, arXiv preprint arXiv:1409.3505, 2014.
[53] M. Lin, Q. Chen and S. Yan, Network in network, arXiv preprint arXiv:1312.4400. 2013.
[54] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556. 2014.
[55] K. He, X. Zhang, S. Ren and J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, in *Proc. European Conference on Computer Vision*, 2014, pp. 346–361.
[56] Y. Sun, X. Wang and X. Tang, Deep learning face representation from predicting 10,000 classes, in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1891–1898.
[57] Y. Taigman, M. Yang, M. A. Ranzato and L. Wolf, Deepface: Closing the gap to human-level performance in face verification, in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708.
[58] V. Goffin, C. Allauzen, E. Bocchieri, D. Hakkani-Tür, A. Ljolje, S. Parthasarathy, M. G. Rahim, G. Riccardi and M. Saraclar, The AT&T WATSON Speech Recognizer, *ICASSP* (1) (2005) 1033–1036.
[59] A. Dunn, Pro Microsoft Speech Server 2007: Developing Speech Enabled Applications with. NET, Apress, 2007.
[60] J. Adorf, Web Speech API, KTH Royal Institute of Technology, 2013.

[61] Nuance Communication Inc.: Speech recognition solutions. http://www.nuance.com/for-individuals/by-solution/speech-recognition/index.htm
[62] Google Voice: https://www.google.com/googlevoice/about.html
[63] S. Ji, W. Xu, M. Yang and K. Yu, 3D convolutional neural networks for human action recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**(1) (2013) 221–231.
[64] K. Simonyan and A. Zisserman, Two-stream convolutional networks for action recognition in videos, *Advances in Neural Information Processing Systems* **27** (2014) 568–576.
[65] J. Shao, K. Kang, C. C. Loy and X. Wang, Deeply learned attributes for crowded scene understanding, in *2015 IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4657–4666.
[66] K. Kang and X. Wang, Fully convolutional neural networks for crowd segmentation, arXiv preprint arXiv:1411.4464. 2014.
[67] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko and T. Darrell, Long-term recurrent convolutional networks for visual recognition and description, in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2625–2634.
[68] Y. Bengio, R. Ducharme, P. Vincent and C. Jauvin, A neural probabilistic language model, *Journal of Machine Learning Research* **3** (2003) 1137–1155.
[69] J. Bruna and S. Mallat, Invariant scattering convolution networks, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**(8) (2013) 1872–1886.