Received: 3 May 2020 | Accepted: 26 May 2020
DOI: 10.1002/sta4.294

WILEY

SPECIAL ISSUE PAPER

Deep learning from a statistical perspective

Yubai Yuan[1] | Yujia Deng[2] | Yanqing Zhang[3] | Annie Qu[1]

[1]Department of Statistics, University of California, Irvine, Irvine, CA 92697, USA

[2]Department of Statistics, University of Illinois, Urbana-Champaign, Champaign, IL 61820, USA

[3]Department of Statistics, Yunnan University, Kunming, 650106, China

Correspondence
Annie Qu, Department of Statistics, University of California, Irvine, Donald Bren Hall 2212, Irvine, CA 92697, USA.
Email: aqu2@uci.edu

Funding information
National Science Foundation of US, Grant/Award Numbers: DMS-1821198, DMS-1952406

As one of the most rapidly developing artificial intelligence techniques, deep learning has been applied in various machine learning tasks and has received great attention in data science and statistics. Regardless of the complex model structure, deep neural networks can be viewed as a nonlinear and nonparametric generalization of existing statistical models. In this review, we introduce several popular deep learning models including convolutional neural networks, generative adversarial networks, recurrent neural networks, and autoencoders, with their applications in image data, sequential data and recommender systems. We review the architecture of each model and highlight their connections and differences compared with conventional statistical models. In particular, we provide a brief survey of the recent works on the unique overparameterization phenomenon, which explains the strengths and advantages of using an extremely large number of parameters in deep learning. In addition, we provide a practical guidance on optimization algorithms, hyperparameter tuning, and computing resources.

KEYWORDS
autoencoder, convolutional neural network, generative adversarial network, long short-term memory, overparameterization, restricted Boltzmann machine

1 | INTRODUCTION

In recent years, deep neural networks have achieved great success across all spectrums of machine learning problems such as image segmentation, natural language processing, and object tracking. The development of deep learning can be summarized to three stages (Goodfellow et al., 2016). The first stage is 1940s–1960s, where linear networks with a single neuron were proposed (McCulloch & Pitts, 1943; Rosenblatt, 1958; 1961). However, it lacked nonlinear features, and thus the applications were limited.

The second stage is 1980s–1990s, gaining more successful developments due to the application of backpropagation algorithms (Rumelhart, Hinton, & Williams, 1986) in training neural networks with one or two layers. Furthermore, the prototype of many popular deep models originates in the late second stage, for example, the Boltzmann Machine (Sabour, Frosst, & Hinton, 2018), the long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997), and the convolutional neural network (CNN) (LeCun, Bottou, Bengio, & Haffner, 1998). However, the applications of LSTMs and CNNs were limited at that time because of high computational cost. For example, training a CNN with only two hidden layers could take 3 days (LeCun et al. 1998). Another issue is that the backpropagation algorithm is often trapped in a local optimum because of the nature of nonconvexity of the multilayer network optimization.

The computational issue was unresolved until Hinton proposed the deep belief network (DBN) (Hinton & Salakhutdinov, 2006), which is used as an initialization strategy in deep neural network training. This breakthrough in computation opened the curtain of the third stage of deep learning development. Since 2011, a series of success in computer vision competitions further makes deep learning more popular. For example, AlexNet (Krizhevsky, Sutskever, & Hinton, 2012) won the 2012 ImageNet Recognition Challenge with a more than 10% error rate than the best non-neural-network solution. In 2015, ResNet (He, Zhang, Ren, & Sun, 2016) outperformed the human in classification on the ImageNet dataset for the first time. Generally, two key factors contribute to the resurgence of deep learning. One is the revolutionary gain in

computing power such as the use of graphics processing units (GPUs) and tensor processing units (TPUs), which accelerates the training of neural networks for hundreds or thousands of times (CireşAn, Meier, Gambardella, & Schmidhuber, 2010). The other key factor is the availability of large labelled datasets. The ImageNet dataset (Deng et al. 2009) is launched in 2009 and contains 3.2 million annotated images with 5,247 categories, whereas the WMT 2014 English to French dataset (Bojar et al. 2014) consists of almost 1 billion sentence pairs for machine translation. These datasets make it feasible to generalize deep neural networks by supplying a large amount of training data and therefore improve model prediction accuracy. On the other hand, traditional parametric models fail to fully approximate the complex distribution from these large datasets, whereas deep models allow more flexible model structures and display overwhelming advantages in large-scale data applications.

The recent deep learning development can be categorized into three main categories: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, there are two main classes for different data types: CNNs and recurrent neural networks (RNNs). CNNs are designed for image-related tasks and applied in image classification, object detection, face recognition, and medical imaging data analyses. RNNs including their variations such as LSTMs and gated recurrent units (GRUs) (Cho et al. 2014) are designed for sequential data such as time series data and acoustic data. RNNs are also applied in dynamic tracking and monitoring sequential systems including traffic forecasting systems (Azzouni & Pujolle, 2017) and driver action prediction systems (Olabiyi, Martinson, Chintalapudi, & Guo, 2017).

Deep unsupervised learning aims at extracting features and learning underlying distributions of input data. Among this category, autoencoders (Hinton & Salakhutdinov, 2006) and Restricted Boltzmann Machines (RBMs) (Srivastava & Salakhutdinov, 2012; Salakhutdinov & Hinton, 2009) are widely applied in text mining and recommender systems, and generative adversarial networks (GANs) are effective for image and video synthesis. In addition, deep unsupervised learning can be integrated into deep supervised learning as a preprocessing technique, which is commonly used in recommender systems (Chu & Tsai, 2017) and natural language processing (Devlin, Chang, Lee, & Toutanova, 2018; Radford et al. 2019). Deep reinforcement learning aims to learn an optimal decision-making strategy in an action-reward system. The most recognized achievements in this field are Alpha-Go (Silver et al. 2017) and Alpha-Zero (Silver et al. 2017). Other applications include autonomous driving techniques (Sallab, Abdou, Perot, & Yogamani, 2017) and game artificial intelligence (Vinyals et al. 2017) to improve game players.

One important trend of deep learning is that complexity of neural networks increases as the layers of neural networks and the total number of parameters keep increasing. Figure 1 illustrates the number of parameters corresponding to representative deep learning models. Generally, neural networks doubled in size every 2.4 years (Goodfellow et al., 2016). The increasing trend is even accelerated more since ResNet (He et al. 2016) has been proposed. Although a few theoretical developments have been established so far, experimental results (Simonyan & Zisserman, 2014; Szegedy et al. 2015) demonstrate that the performance of deep neural networks improves with an increasing number of layers, or equivalently, deeper networks provide lower generalization errors.

In general, deep learning deals with supervised or unsupervised learning problems under a framework similar to statistical methods. However, deep learning goes beyond traditional statistical learning algorithms through increasing the number of processing operations from input data to output data. Specifically, it approximates training examples by a hierarchical composition of multiple nonlinear transformations over latent features from input data. This architecture is extremely effective in capturing highly nonlinear relations between input data and output data. In addition, deep learning models bring new insights on the bias-variance trade-off principle under the overparameterization regime.

In this review, we elaborate the applications of deep learning in three important domains: image analyses, sequence data analyses, and recommender systems. Moreover, we investigate the general connections between statistical models and deep learning models, which are not considered by most of the existing reviews. For the rest of this review, we introduce the basic deep learning architecture, that is, the general feed-forward network in Section 2. In Section 3, we present two popular models in imaging data analyses: the CNN and the GAN. Section 4 illustrates the recurrent neural network designed for sequential data. Section 5 considers two deep models for recommender systems: the autoencoder and the Boltzmann machine.

Section 6 focuses on optimization for training a deep learning model. We summarize the connections between deep learning and statistical models in Section 7. Section 8 provides softwares and platforms for implementing deep learning models. A summary of the advantages and limitations of deep learning is provided in the last section.

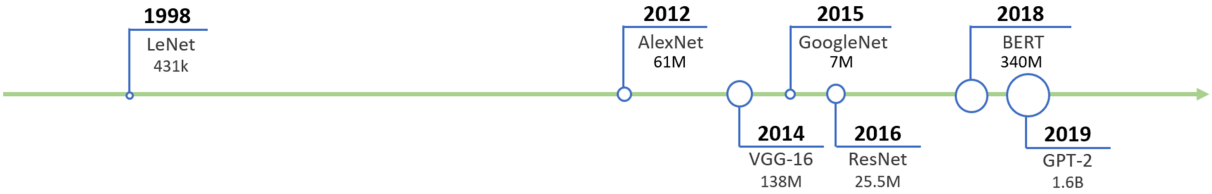

FIGURE 1 Size evolution of deep neural networks, showing the names of deep neural networks along with the year and the total number of parameters

2 | BASIC DEEP ARCHITECTURE: FEED-FORWARD NEURAL NETWORK

In this section, we introduce the general structure of the feed-forward neural network (FNN) (Haykin, 1994). We denote the training dataset as $\{(\boldsymbol{x}_i, y_i)\}_{1 \le i \le n}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is the independent input data and y_i is the dependent response data. In regression problems, y_i belongs to a continuous subset of $\mathbb{R}$, whereas in classification problems, y_i belongs to a discrete set $[K] = \{1, 2, \ldots, K\}$, where K is the number of categories.

The structure of the FNN mainly consists of three parts: input layer, hidden layer and output layer. The input layer is simply the vector $\boldsymbol{x}$. The hidden layer involves a linear transformation or nonlinear transformation. Specifically, for an L-layer FNN, we denote the dimension of the ℓth layer as d_ℓ, and the ℓth layer can be defined recursively by

$$\boldsymbol{h}^{(0)} = \boldsymbol{x}; \quad \boldsymbol{h}^{(\ell)} = \sigma(W^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}), \quad \ell = 1, 2, \ldots, L,$$

where $W^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the weighting matrix, $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$ is the bias, and $\sigma : \mathbb{R}^{d_\ell} \to \mathbb{R}^{d_\ell}$ is a prespecified nonlinear transformation function, called the activation function.

The choice of $\boldsymbol{\sigma}$ includes the sigmoid function, rectified linear unit (ReLU), leaky ReLU, tanh function and others, which are provided in Table 1. Among these activation functions, sigmoid and tanh are smooth functions. However, the computational cost of using these smooth functions is relatively more expensive. Most critically, they may result in a gradient vanishing problem where the derivatives of the first several layers are close to zero during model training, especially when the number of layers is high. The reason is that the derivatives of sigmoid and tanh are between 0 and 1 and could lead to smaller derivatives by multiplications arising from chain rules. In contrast, ReLU has a constant gradient on positive values, which avoids the gradient vanishing problem. Yet the derivative stays zero once the hidden unit is negative, which defeats the purpose of backpropagation. To solve this problem, leaky ReLU (Maas, Hannun, & Ng, 2013) and exponential linear unit (ELU) (Clevert, Unterthiner, & Hochreiter, 2015) add a small gradient to the negative values. Compared with ReLU, these modifications are more robust as they avoid being trapped in the negative values.

The formulation of the output layer is determined by different learning tasks. For regression problems, the function of the output layer can be an affine transformation defined by $g(\boldsymbol{h}^{(L)}) = \boldsymbol{w}_o^T\boldsymbol{h}^{(L)} + b_o$, where $\boldsymbol{w}_o \in \mathbb{R}^{d_o}$ and b_o is a scalar. For multicategory classification problems, the transformation function is selected as a softmax function defined by

$$[\boldsymbol{g}(\boldsymbol{h}^{(L)})]_i = \frac{\exp(\boldsymbol{h}_i^*)}{\sum_{j=1}^{K}\exp(\boldsymbol{h}_j^*)}, \quad \text{where} \quad \boldsymbol{h}^* = W_o\boldsymbol{h}^{(L)} + \boldsymbol{b}_o.$$

In general, the full structure of the FNN can be formulated as $\boldsymbol{f}(\boldsymbol{x}|\Theta) = \boldsymbol{g} \circ \boldsymbol{h}^{(L)} \circ \boldsymbol{h}^{(L-1)} \circ \ldots \circ \boldsymbol{h}^{(1)}(\boldsymbol{x})$, where $\circ$ denotes functional composition and $\Theta = \{W_o, \boldsymbol{b}_o, W^{(\ell)}, \boldsymbol{b}^{(\ell)}\}_{\ell \in [L]}$ is the parameter set corresponding to the affine transformation matrices and bias vectors of all layers. Figure 2

TABLE 1 Commonly used activation functions in deep neural networks

Name	Sigmoid	ReLU	tanh	Leaky ReLU	ELU
Formula	$\frac{1}{1+e^{-x}}$	$\max(0,x)$	$\tanh(x)$	$\max(cx,x)(c>0)$	$\begin{cases} x & x \ge 0 \\ \alpha(e^x-1) & x < 0 \end{cases}$

Abbreviations: ELU, exponential linear unit; ReLU, rectified linear unit.

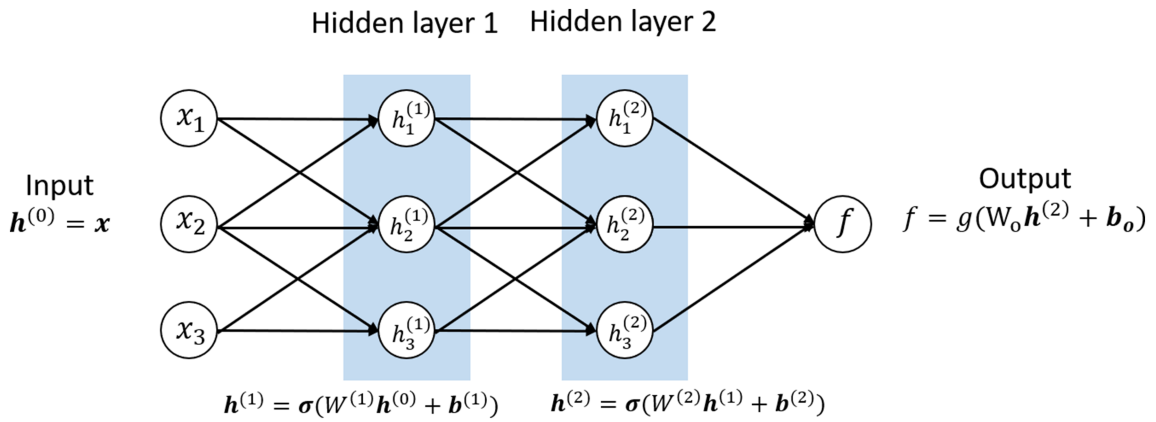

FIGURE 2 General structure of a two-layer feed-forward neural network (FNN) with a scalar output. Each circle represents an element in vectors of input $\boldsymbol{x}$ and hidden units $\boldsymbol{h}^{(\ell)}$, where ℓ=1 and 2

illustrates the structure of a two-layer FNN. For model training, we can estimate Θ by minimizing the empirical loss function. For example, for a regression problem, we use the quadratic loss, $\hat{\Theta} = \text{argmin}_{\Theta} \sum_{i=1}^{n} (\boldsymbol{f}(\boldsymbol{x}_i|\Theta) - y_i)^2$, and for a classification problem, we use the negative log likelihood loss, $\hat{\Theta} = \text{argmin}_{\Theta} - \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbf{1}\{y_i = k\} \log[\boldsymbol{f}(\boldsymbol{x}|\Theta)]_k$. The training of parameters utilizes backpropagation (Rumelhart et al. 1986), which computes the derivatives of parameters in each layer via a chain rule.

Compared with parametric models, the advantages of FNNs are mainly attributed from the nonlinearity induced by the activation function in each layer. Indeed, the universal approximation theorem (Cybenko, 1989; Hornik, Stinchcombe, & White, 1990) states that any Borel measurable function can be approximated by an FNN with one hidden layer under mild conditions on the activation function. Moreover, the derivatives of any Borel measurable function can be approximated by the derivatives of an FNN as well. This theorem provides theoretical foundation for the representation and formulation of the FNN.

3 | DEEP MODELS FOR IMAGING DATA

In this section, we introduce the application of deep neural networks related to imaging data, including CNNs for supervised learning tasks and GANs for unsupervised learning tasks.

3.1 | Convolutional neural networks

CNNs (LeCun et al. 1998) are specialized feed-forward networks designed for imaging data. It has achieved great success in imaging data analysis including classification, object detection, segmentation, and semantic description. Different from the general FNN introduced in Section 2, CNNs use convolution operation instead of matrix multiplication to extract features, which enables CNNs to utilize the grid structure of imaging data.

We first introduce the unique operators involved in CNNs, including convolution and pooling. Let $X \in \mathbb{R}^{c_{in} \times d_1 \times d_2}$ be a multicolor image; we define the filter as $K \in \mathbb{R}^{c_{out} \times c_{in} \times p_1 \times p_2}$, where c_{in} is the number of matrices of input data and c_{out} is the number of matrices of output data. Then, the convolution between X and K is defined elementwisely as

$$[X * K]_{u,i,j} = \sum_{v=1}^{c_{in}} \sum_{m=1}^{p_1} \sum_{n=1}^{p_2} X_{v,(i-1)s+m,(j-1)s+n} K_{u,v,m,n},$$

and the max pooling function is defined as

$$[\pi(X)]_{u,i,j} = \max_{(m,n) \in [p_1] \times [p_2]} X_{u,(i-1)s+m,(j-1)s+n},$$

where s is a positive integer called stride. Different K results in different image processing procedures such as blurring, sharpening, and edge detection. In CNNs, the filters are viewed as parameters and are trained to extract features. Figure 3 illustrates a convolution with $c_{in} = c_{out} = 1$, $d_1 = d_2 = 4$, $p_1 = p_2 = 3$, and $s = 1$.

In general, CNNs consist of a feature extractor and a classifier. The feature extractor is a combination of convolution layers and pooling layers $\boldsymbol{g}^{(\ell-1)} = \boldsymbol{\sigma}(\boldsymbol{h}^{(\ell-1)} * K^{(\ell-1)})$, $\boldsymbol{h}^{(\ell)} = \boldsymbol{\pi}(\boldsymbol{g}^{(\ell-1)})$, where $\boldsymbol{\sigma}$ is an activation function, selected as ReLU by default. The outputs of the convolution layer $\boldsymbol{g}^{(\ell)}$ are also called feature maps. Note that the convolution layer is not necessarily followed by a pooling layer. The output of the feature extractor is then vectorized and connected with the classifier, which consists of a general multilayer FNN or a fully connected layer. Figure 4 illustrates the structure of CNNs with a single convolution layer and a single fully connected layer.

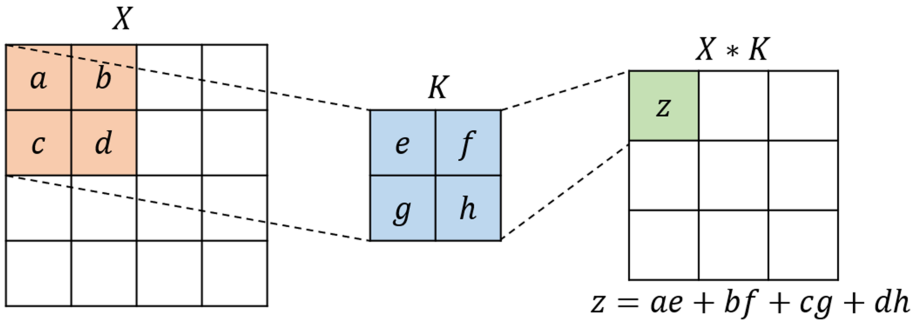

FIGURE 3 Illustration of convolution operator between 4 × 4 input data and a 2 × 2 filter with stride equal 1. The output is of size 3 × 3

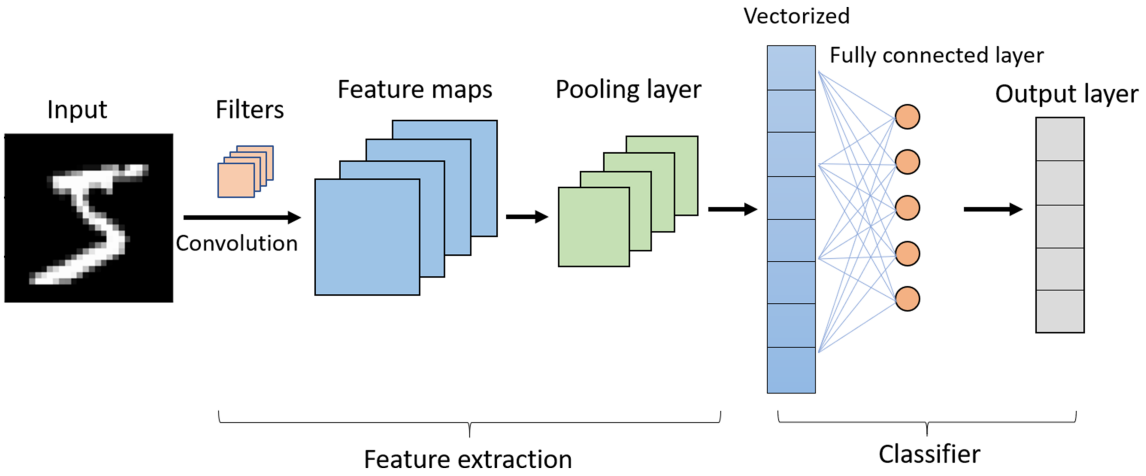

FIGURE 4 Illustration of a convolutional neural network (CNN) with a single convolution layer, a single pooling layer, and a single fully connected layer

The reason why the convolution operator benefits the performance of CNNs on image data is attributed to two parts. On one hand, the features extracted by CNNs are not location sensitive because of the translation invariance of convolution operators (Goodfellow et al., 2016), and thus CNNs are more robust to the translation of signal pixels compared with the general FNN and traditional regression settings, where the index of the features are usually fixed. On the other hand, using a convolution layer reduces the computational cost and storage requirement of the model as the size of filters is much smaller than the input data. The reduced number of parameters avoids overfitting and thus improves the generalization capability of the model.

Despite the popularity of CNNs in the computer vision field, there are also limitations. First, CNNs ignore the orientation and the relative location information of these features. However, these information may be important in tasks such as face recognition, where the presence of all facial features does not imply a human face. Second, CNNs perform poorly when image data are highly heterogeneous across the subjects or the number of training samples is limited (Tang, Bi, & Qu, 2019). In practice, this problem can be alleviated by data augmentation, that is, enlarging the training dataset manually through zooming, rotating, and cropping. Nonetheless, CNNs are still not able to recognize variations of images that are not presented in the augmented data. Finally, CNNs are vulnerable to adversarial attacks, which are specifically designed noise added to training data. Some of these attacks are imperceptible to human beings (Goodfellow, Shlens, & Szegedy, 2014) or only minor changes of lighting and orientation settings (Kurakin, Goodfellow, & Bengio, 2016) but could result in poor performance of a fine-trained CNN. How to defend CNNs from adversarial attacks remains an open problem. For implementations of image classification through CNNs, please see the Supplementary Material (Yuan, Deng, Zhang, & Qu, 2020).

3.2 | Generative adversarial networks

The GAN is an unsupervised learning method that estimates the density of population distribution. It has shown great capability to sample from complex distributions and achieved great success in image-related tasks such as style transfer (Karras, Laine, & Aila, 2019), image synthesis (Tran, Yin, & Liu, 2018), and object detection (Ehsani, Mottaghi, & Farhadi, 2018).

Given the observed data $\{\boldsymbol{x}_i\}_{1\le i\le n}\in\mathbb{R}^p$, the GAN learns the underlying distribution density of X, P_X, with two deep neural networks, namely, the discriminator D and the generator G. The discriminator $D:\mathbb{R}^p\to\mathbb{R}$ is trained to determine whether the input data is sampled from the population or generated from G, whereas the generator G aims to minimize the difference between the generated data distribution and the population data distribution, so that D eventually cannot distinguish these two distributions. To that end, the generator G samples from a prior noise distribution $\boldsymbol{z}\sim P_Z$ as input and outputs a transformed vector $G(\boldsymbol{z})\in\mathbb{R}^p$. Mathematically, the GAN solves the following min-max problem:

$$\min_{\theta_G}\max_{\theta_D} E_{\boldsymbol{x}\sim P_X}\log\{D(\boldsymbol{x})\}+E_{\boldsymbol{z}\sim P_Z}\log(1-D\{G(\boldsymbol{z})\}), \tag{1}$$

where θ_D and θ_G are the parameters in D and G, respectively. Figure 5 illustrates an example of GANs in generating images that resemble handwritten digits. We denote the implicit probability distribution of $G(\boldsymbol{z})$ over the data space induced by the generator as P_g; then, the target function (1) can be written as to minimize the Jensen-Shannon divergence between P_g and P_X. However, Jensen-Shannon divergence may not be suitable for multimodal distributions because of potential missing modes (Che, Li, Jacob, Bengio, & Li, 2016; Nguyen, Yosinski, & Clune, 2016), where the

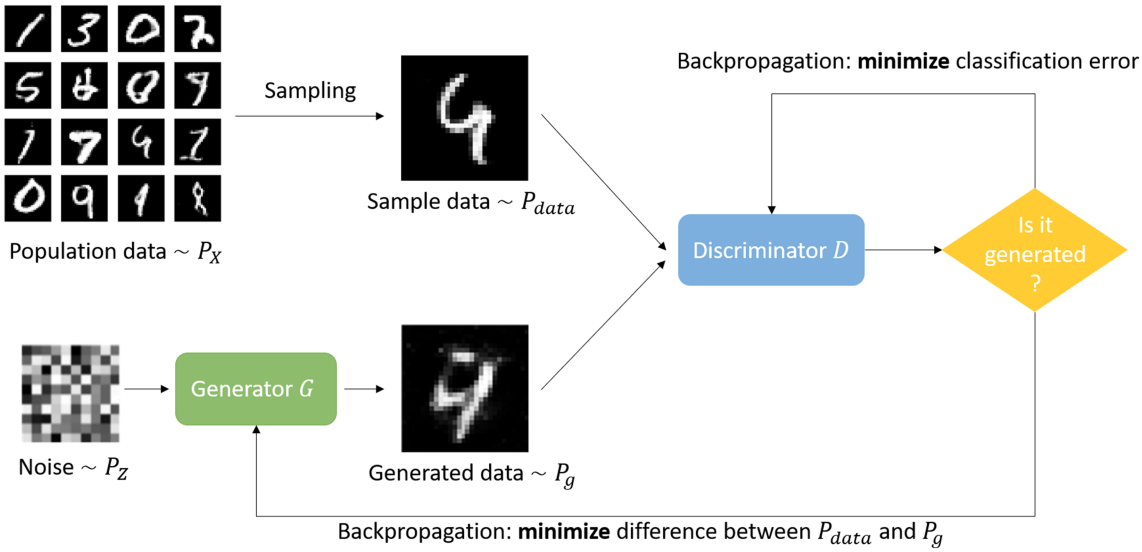

FIGURE 5 Architecture of generative adversarial network (GAN) to generate handwritten digits. The discriminator D determines whether the input data is sampled from the true population or generated by G. The generator G aims to minimize difference between P_X and P_g. Both D and G are multilayer networks

generated distribution P_g could approximate poorly at the modes with small density. To solve this, one can replace the Jensen-Shannon divergence with other divergence criteria (Nowozin, Cseke, & Tomioka, 2016), which generates multiple variants of GANs.

In addition, works have been done on approximation efficiency analysis regarding the convergence rate of the GAN towards the true distribution P_X (Liang, 2018). Because P_X is not observed directly, the GAN implicitly uses the empirical density of the data P_{data} based on n independent and identically distributed samples to approximate the unobserved P_X. Therefore, the discrepancy between the generated sampling distribution P_g and the target distribution P_X comes from two parts: the approximation error due to the iterative min-max process of the GAN and the statistical error due to the approximation of P_{data} instead of P_X. Under some smoothness assumptions, instead of using the empirical density function, one can use a kernel density approximation to improve the convergence rate

$$\frac{dP_{data}}{d\boldsymbol{x}} = \frac{1}{nh}\sum_{i=1}^{n} K\{(X_i - \boldsymbol{x})/h\}, \tag{2}$$

where K is a kernel function and h is the bandwidth. Another way to improve the convergence rate is to shrink the search space of D by modifying the evaluation metric. Note the target of the discriminator D can be generalized to

$$d_{\mathcal{F}_D}(P_g, P_X) := \max_{f \in \mathcal{F}_D} \left\{ E_{Y \sim P_g} f(Y) - E_{X \sim P_X} f(X) \right\}, \tag{3}$$

where $\mathcal{F}_D$ is the function class of the discriminator. One can utilize the prior information of P_X to reduce the size of $\mathcal{F}_D$ so that the generated distribution is closer to the target distribution. For example, if $\mathcal{F}_D$ contains only Lipschitz-1 function, then $d_{\mathcal{F}_D}$ can be chosen as the Wasserstein-1 metric, and it leads to the Wasserstein-GAN (Arjovsky, Chintala, & Bottou, 2017), which provides a lower approximation error to a smooth function compared with the original GAN.

One important advantage of the GAN is its high capability to learn nonparametric density functions, which are difficult to be fully captured using parametric generative models such as the Markov random field. Specifically, because the discriminator D is optimized over a function class $\mathcal{F}_D$, the GAN achieves the minimum loss from an family of divergence criteria, which is smaller than using a specific divergence such as negative log-likelihood. On the other hand, to make optimization feasible, conventional generative models use variational inference or impose additional assumptions on density functions, such as the factorial assumption in the Markov random field, which may introduce an additional approximation error in minimizing the divergence. In contrast, the GAN is able to decrease the approximation error because of its representation power of multilayer networks and therefore generates higher-quality samples.

In spite of high flexibility and accurate approximation power to the population distribution, GAN has drawbacks regarding high computational cost, the mode missing problem (Che et al. 2016; Metz, Poole, Pfau, & Sohl-Dickstein, 2016), and the difficulty to measure estimation uncertainty. The GAN also has limitations in generating discrete data distributions because training of GAN relies on gradient backpropagation.

4 | APPLICATION ON DYNAMIC SEQUENCE DATA

In this section, we elaborate the application of deep learning methods in the field of sequence data and introduce the RNN method and its variant LSTM (Hochreiter & Schmidhuber, 1997).

4.1 | Recurrent neural networks

The RNNs are first developed in the 1980s (Rumelhart et al. 1986; Werbos, 1988; Elman, 1990) and designed to process sequence data, for example, stock price prediction, speech recognition, machine translation, and genome sequencing, because they can utilize historical data to capture sequential patterns. Suppose we have historical sequence data $\boldsymbol{y} = \{y_1, y_2, ..., y_T\}$ and $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T\}$, where $y_t \in \mathbb{R}$ and $\boldsymbol{x}_t \in \mathbb{R}^p$. We can use RNNs to model dynamic systems based on observed sequence data for predicting $y_k(k>T)$. The structure of RNNs consists of an input layer, one or more hidden layers, and an output layer. The simplest RNN sequential model can be described as follows, respectively:

$$\boldsymbol{h}_t = \sigma_h(\mathbf{W}_{hh}\boldsymbol{h}_{t-1} + \mathbf{W}_{xh}\boldsymbol{x}_t + \boldsymbol{b}_h) \quad \text{and} \quad z_t = \sigma_z(\mathbf{W}_{hz}\boldsymbol{h}_t + \boldsymbol{b}_z),$$

where $\boldsymbol{h}_t \in \mathbb{R}^d$ is the hidden state of the recurrent network at time t, z_t is the output at the time t, $\boldsymbol{\sigma}_h$ and $\boldsymbol{\sigma}_z$ are given nonlinear activation functions, $\mathbf{W}_{hh}$, $\mathbf{W}_{xh}$ and $\mathbf{W}_{hz}$ are weight matrices, and $\boldsymbol{b}_h$ and $\boldsymbol{b}_z$ are bias vectors. Different from FNNs and CNNs, the hidden state $\boldsymbol{h}_t$ at time t depends not only on the current observation $\boldsymbol{x}_t$ but also on the previous hidden state $\boldsymbol{h}_{t-1}$. Therefore, $\boldsymbol{h}_t$ can capture historical information from a sequence via previous hidden states $\boldsymbol{h}_{t-1}, \boldsymbol{h}_{t-2}, ..., \boldsymbol{h}_0$ and incorporate update from the current observation $\boldsymbol{x}_t$.

Figure 6 shows the structure of a simple RNN with one input unit $\boldsymbol{x}_t$, one output unit z_t, and one hidden unit $\boldsymbol{h}_t$ at time t. It can be extended from the one hidden case to multiple layers by adding more hidden layers between $\boldsymbol{h}_t$s and $\boldsymbol{x}_t$s. We can minimize the loss function $L(\mathbf{X}, \boldsymbol{y}) = \sum_{t=1}^{T}(y_t - z_t)^2$ to estimate model parameters. During the training process, an RNN needs to calculate the gradients $\partial L(\mathbf{X}, \boldsymbol{y})/\partial \boldsymbol{h}_t$ in the reverse time order and uses a backpropagation algorithm. Computing $\partial L(\mathbf{X}, \boldsymbol{y})/\partial \boldsymbol{h}_t$ for a long sequence involves multiplication of many $\partial \boldsymbol{h}_{t+1}/\partial \boldsymbol{h}_t$, which usually results in singular values or divergent values. Therefore, the backpropagation algorithm may have gradient vanishing or exploding problems, which limits the use of RNNs. Hochreiter and Schmidhuber (1997) proposed LSTM networks to handle the gradient vanishing problem. LSTM has attracted great attention for sequence data application recently and yielded significant improvement over RNNs. We introduce LSTM in the following part.

4.2 | Long short-term memory

LSTM adds additional interactions per hidden unit for addressing the aforementioned drawbacks of the RNN. It is capable of learning long-term dependencies and remembering information for prolonged periods of time. A common LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. Specifically, given a sequence data $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_T\}$ for $\boldsymbol{x}_t \in \mathbb{R}^p$, an LSTM unit with a forget gate is composed as follows:

$$\begin{aligned} \boldsymbol{f}_t &= \sigma_g(\mathbf{W}_{xf}\boldsymbol{x}_t + \mathbf{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f), \quad \boldsymbol{i}_t = \sigma_g(\mathbf{W}_{xi}\boldsymbol{x}_t + \mathbf{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i), \\ \boldsymbol{g}_t &= \tanh(\mathbf{W}_{xc}\boldsymbol{x}_t + \mathbf{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c), \quad \boldsymbol{o}_t = \sigma_g(\mathbf{W}_{xo}\boldsymbol{x}_t + \mathbf{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o), \\ \boldsymbol{c}_t &= \boldsymbol{f}_t \circ \mathbf{c}_{t-1} + \boldsymbol{i}_t \circ \boldsymbol{g}_t, \quad \boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\mathbf{c}_t), z_t = \sigma_z(\mathbf{W}_{hz}\boldsymbol{h}_t + \boldsymbol{b}_z), \end{aligned} \tag{4}$$

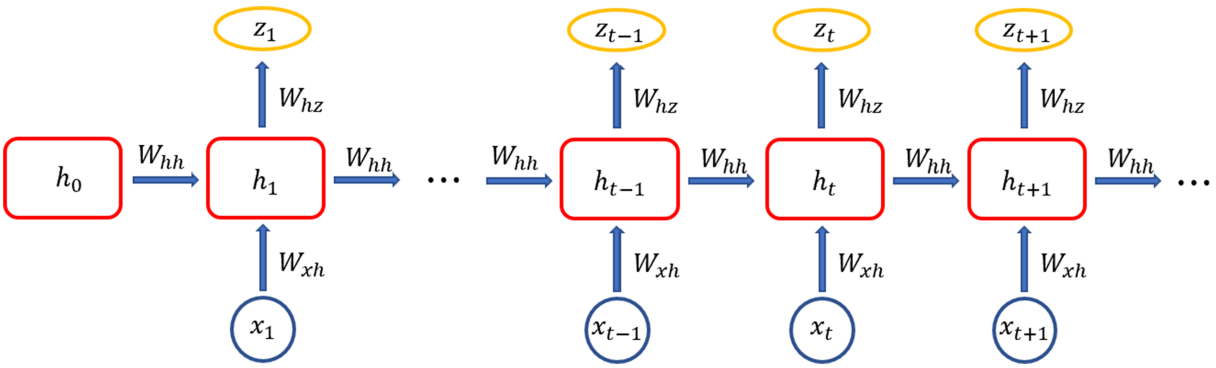

FIGURE 6 A simple recurrent neural network (RNN) with one input unit, one output unit, and one hidden unit

where $\boldsymbol{f}_t \in \mathbb{R}^d$ is a forget gate, $\boldsymbol{i}_t \in \mathbb{R}^d$ is an input gate, $\boldsymbol{g}_t \in \mathbb{R}^d$ is a cell input activation vector, $\boldsymbol{o}_t \in \mathbb{R}^d$ is an output gate, and $\boldsymbol{c}_t \in \mathbb{R}^d$ is a cell state vector. Here, z_t is an output, and $\mathbf{W}$, $\boldsymbol{b}$ with different subscripts are weight matrices and bias vectors, respectively. The $\boldsymbol{\sigma}_g$ and $\boldsymbol{\sigma}_z$ are the activation functions.

The core of LSTM is a memory unit or cell $\mathbf{c}_t$, which encodes historical information of the inputs. Compared with the hidden layer in normal RNNs, the memory cell $\mathbf{c}_t$ has more gating units, which control the historical information flow. The input gate and output gate respectively control the information input to the memory unit and the information output from the memory unit. Moreover, the forget gate $\boldsymbol{h}_t$ can be controlled by the output gate $\boldsymbol{o}_t$, and the cell $\mathbf{c}_{t-1}$ is also controlled by the forget gate $\boldsymbol{h}_{t-1}$ via resetting the gating unit with activation functions. Because these gates control the memorizing process, LSTM can avoid the long-term dependency problem. Figure 7 shows the structure of the LSTM network.

Deep learning methods learn the behaviour of sequence data using the compositional function class like (4). This is a sharp difference from traditional statistical methods such as linear models (e.g., autoregressive integrated moving average) and nonlinear models (e.g., autoregressive conditional heteroskedasticity). Although traditional model-based methods have proven to be quite effective in many circumstances, identifying a model that is broadly applicable has been difficult (Längkvist, Karlsson, & Loutfi, 2014). Deep learning methods do not involve human knowledge and make no claims about the generation process. Moreover, the performance of traditional statistical methods mainly depends on the properties of target functions and their related optimization are less complicated, whereas the statistical performance of deep learning methods depends heavily on optimization algorithms.

5 | DEEP MODELS FOR RECOMMENDER SYSTEM

Deep learning has been applied in recommendation architectures and brings more opportunities to improve the performance of recommender systems. A recommender system is used to recommend items to users based on users' personalized preference, represented by maximum rating scores in a utility matrix, denoted by $\boldsymbol{R} \in \mathbb{R}^{n \times m}$, where n and m are the numbers of users and items, respectively. In practice, there are a vast amount of items for users to choose from, but users can only choose and rate a few items, which leads to an extremely large and sparse utility matrix. Therefore, the key of recommender systems is to predict rating scores of a user on the basis of the observed scores and make recommendations based on the predicted rating scores. Among many existing recommender systems, deep learning-based recommender systems (Salakhutdinov, Mnih, & Hinton, 2007; Dong et al. 2017; Li, Kawale, & Fu, 2015; Bansal, Belanger, & McCallum, 2016) are promising. These include multilayer perception (MLP) (Liang, Zhan, & Ellis, 2015; Alashkar, Jiang, Wang, & Fu, 2017), CNN (Chu & Tsai, 2017; Elkahky, Song, & He, 2015), RNN (Bansal et al. 2016), autoencoder (Dong et al. 2017; Li et al. 2015), RBM-based recommender system (Salakhutdinov et al. 2007), and deep hybrid models. In the following, we mainly focus on RBM-based recommendation and autoencoder-based recommender systems.

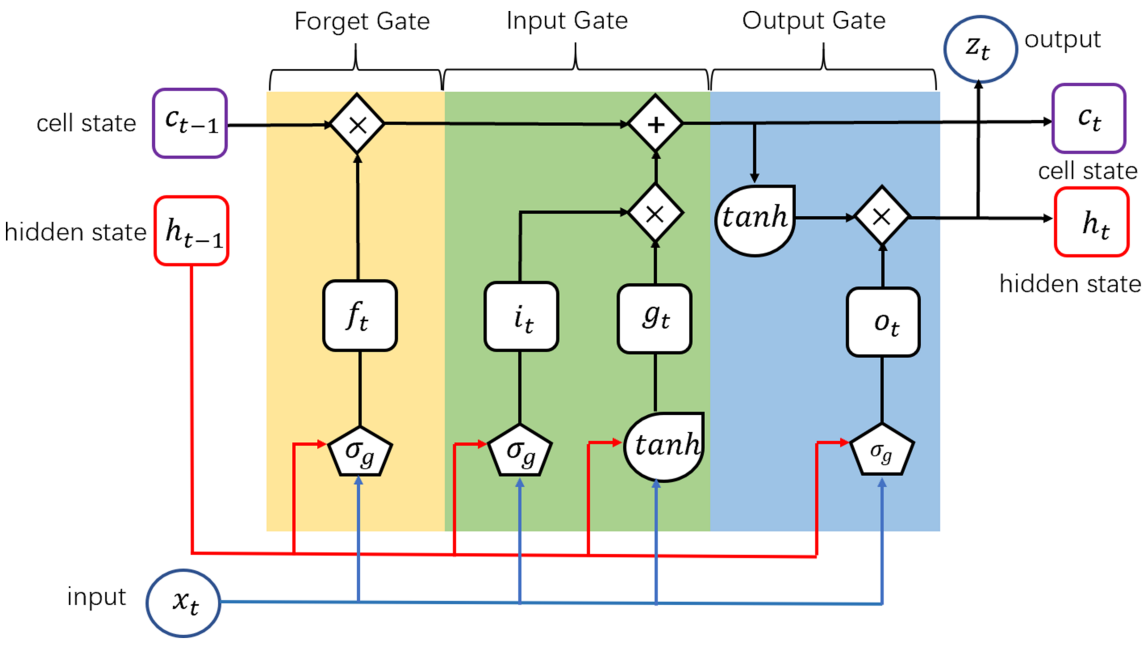

FIGURE 7 Typical structure of a cell of a single-layer long short-term memory (LSTM)

5.1 | Restricted Boltzmann machine

An RBM (Smolensky, 1986; Freund & Haussler, 1992; Hinton, 2002) is to use unobserved hidden layers to explain the observed visible layers. The RBM has wide applications in classification, feature learning, topic modelling, and collaborative filtering. The standard type of RBMs has a binary-valued hidden layer $\boldsymbol{h} \in \mathbb{R}^d$ and a visible layer $\boldsymbol{x} \in \mathbb{R}^p$, which are connected via an energy function of $(\boldsymbol{x}, \boldsymbol{h})$ as follows: $E(\boldsymbol{x}, \boldsymbol{h}) = -\boldsymbol{a}^\top \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{h} - \boldsymbol{x}^\top \boldsymbol{W}\boldsymbol{h}$, where $\boldsymbol{W} = (w_{ij}) \in \mathbb{R}^{p \times d}$ is a weighting matrix linking hidden units $\boldsymbol{h}$ and visible units $\boldsymbol{x}$ and $\boldsymbol{a}$ and $\boldsymbol{b}$ are bias weights for the visible units and the hidden units, respectively. In general Boltzmann machines, the probability distributions over hidden and/or visible vectors are defined in terms of the energy function as $P(\boldsymbol{x}, \boldsymbol{h}) = \exp\{-E(\boldsymbol{x}, \boldsymbol{h})\}/Z$, where Z is a partition function defined as the sum of $\exp\{-E(\boldsymbol{x}, \boldsymbol{h})\}$ over all possible pairs of vectors $(\boldsymbol{x}, \boldsymbol{h})$.

Because the RBM is a special bipartite graph with no intralayer connections, the hidden units are mutually independent given the visible units, and the visible units are mutually independent given the hidden units. Therefore, for p visible units and d hidden units, the conditional probabilities of the units $\boldsymbol{x}$ given $\boldsymbol{h}$ and of the units $\boldsymbol{h}$ given $\boldsymbol{x}$ are respectively

$$P(\boldsymbol{x}|\boldsymbol{h}) = \prod_{i=1}^{p} P(x_i|\boldsymbol{h}) \quad \text{and} \quad P(\boldsymbol{h}|\boldsymbol{x}) = \prod_{j=1}^{d} P(h_j|\boldsymbol{x}).$$

The individual activation probabilities are given by $P(h_j = 1|\boldsymbol{x}) = \sigma\left(b_j + \sum_{i=1}^{p} w_{ij} x_i\right)$ and $P(x_i = 1|\boldsymbol{h}) = \sigma\left(a_i + \sum_{j=1}^{d} w_{ij} h_j\right)$, respectively, where σ denotes the logistic sigmoid. The visible units of an RBM can be multinomial, although the hidden units are Bernoulli. In this case, the individual probability of the visible unit is given by $P(x_i^k = 1|\boldsymbol{h}) = \exp\left(a_i^k + \sum_{j=1}^{d} w_{ij}^k h_j\right) / \sum_{t=1}^{K} \exp\left(a_i^t + \sum_{j=1}^{d} w_{ij}^t h_j\right)$, where K is the number of categories the visible layers have, and $x_i^k = 1$ represents that the ith visible unit has the kth discrete value. Figure 8 provides a graphical representation of an RBM.

The RBM was first applied to recommender systems in 2007 (Salakhutdinov et al. 2007). Salakhutdinov et al. (2007) first proposed an RBM-based recommender systems for tackling the Netflix challenge. For the RBM-based recommender systems, the rating score is represented in a vector of dummy variables to represent binary-valued visible units. For example, in movie rating, (0,0,1,0,0) represents that the user gives a rating score 3 to a movie. The rating score vector of a user is denoted by $\boldsymbol{r} \in \{1, 2, \ldots, K\}^p$, where p is the number of items and K is the maximum rating score. Let $\boldsymbol{X}$ be a $K \times p$ matrix of the dummy variables of $\boldsymbol{r}$, where $x_i^k = 1$ if the user rated movie i as score k and $x_i^k = 0$ otherwise. We can estimate the parameters of the RBM via the contrastive divergence algorithm (Goodfellow et al., 2016). The above RBM-based recommender systems are user-based, where a given user's ratings are observed and input to the visible layer. Item-based RBM recommender systems can be similarly designed via inputting a given item's rating to the visible layer.

5.2 | Autoencoder

The autoencoder is an alternate unsupervised neural network and has been considered as a powerful dimension reduction tool for automatically extracting nonlinear features (Chen, Xu, Weinberger, & Sha, 2012). A basic autoencoder consists of an input layer, a hidden layer, and an output

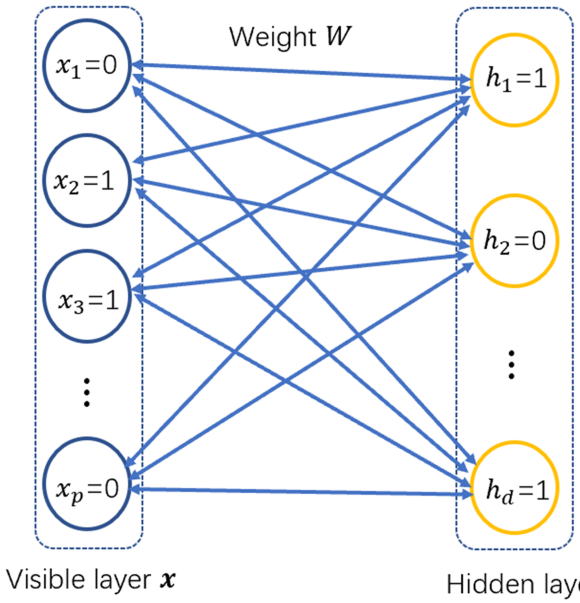

FIGURE 8 Graphical representation of a restricted Boltzmann machine

layer, and Figure 9 provides an illustration. An encoder part encodes the high-dimensional input data $\boldsymbol{x} = \{x_1, x_2, \ldots, x_p\}$ into a low-dimensional hidden representation $\boldsymbol{h} = \{h_1, h_2, \ldots, h_k\}$ via a function $\boldsymbol{f}$ in that $\boldsymbol{h} = \boldsymbol{f}(\boldsymbol{x}) = \sigma_f(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$, where σ_f is an activation function, $\mathbf{W}$ is a $k \times p$ weight matrix, and a bias vector $\boldsymbol{b} \in \mathbb{R}^k$. A decoder maps $\boldsymbol{h}$ back to a reconstructed data $\boldsymbol{x}' = \{x'_1, x'_2, \ldots, x'_p\}$ by a function $\boldsymbol{g}$ in that $\boldsymbol{x}' = \boldsymbol{g}(\boldsymbol{h}) = \sigma_g(\mathbf{W}'\boldsymbol{h} + \boldsymbol{b}')$, where σ_g is an activation function, $\mathbf{W}'$ is a $p \times k$ weight matrix, and a bias vector $\boldsymbol{b}' \in \mathbb{R}^p$. The functions σ_f and σ_g are usually nonlinear activation functions (Zhang, Yao, & Xu, 2017). Both of the encoder function $\boldsymbol{f}$ and the decoder function $\boldsymbol{g}$ are two key components of a basic autoencoder and can be multilayer neural networks.

An autoencoder is trained to minimize the reconstruction error between $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$ for $i = 1, 2, \ldots, n$. Let $L(\boldsymbol{x}_i, \boldsymbol{x}'_i)$ be a loss function that measures the difference between $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$. To find the $\boldsymbol{f}$ and $\boldsymbol{g}$ such that $L(\boldsymbol{x}_i, \boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x}_i)))$ is as small as possible, we can solve the following minimization problem: $\min_{f,g} \sum_{i=1}^{n} L(\boldsymbol{x}_i, \boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x}_i)))$. A regularized term can be added for constructing the loss function of anautoencoder. The loss function can be optimized by stochastic gradient descent (SGD) or alternative least squares. In recent years, there are many variants of autoencoders used in recommender systems such as denoising autoencoder (Vincent, Larochelle, Bengio, & Manzagol, 2008), stack denoising autoencoder (Vincent, Larochelle, Lajoie, Bengio, & Manzagol, 2010), and variational autoencoder (Kingma & Welling, 2013).

In the autoencoder-based recommender system framework, an autoencoder is used to extract features of the items, so we can predict the users' ratings on items. In collaborative filtering, there are n users, m items, and a partially observed user–item rating matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$. The rating of each item can be represented by a partially observed vector $\boldsymbol{r}_i = (R_{1i}, R_{2i}, \ldots, R_{ni})^\top$ for $i = 1, 2, \ldots, m$. An item-based autoencoder can take $\boldsymbol{r}_i$ as input, project it into a low-dimensional hidden representation, and then reconstruct $\boldsymbol{r}'_i$ to predict missing ratings for the purpose of recommendation. That is, the item-based autoencoder can be formulated as $\min_\theta \sum_{r \in \Omega} \|\boldsymbol{r} - \boldsymbol{g}(\boldsymbol{f}(\boldsymbol{r}, \theta))\|^2 + \lambda(\|\mathbf{W}\|_F^2 + \|\mathbf{W}'\|_F^2)/2$, where Ω is the set of ratings in $\mathbb{R}^n$, the parameter $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{W}', \boldsymbol{b}, \boldsymbol{b}'\}$, λ is the regularization parameter, and $\|\cdot\|_F^2$ is the Frobenius norm of a matrix. A user-based autoencoder can be similarly designed via inputting users' ratings.

Obviously, autoencoders resemble principal component analysis (PCA) for dimensional reduction. PCA extracts key information via searching a set of orthogonal eigenvectors corresponding to the larger eigenvalues of the empirical covariance matrix of theinput data and represents data using fewer dimensions than the initial data. However, there are many differences between autoencoders and the PCA. First, whereas PCA is a linear transformation, autoencoders allow nonlinear transformation, which makes autoencoders more flexible and powerful. Unfortunately, the latent space found by autoencoders lacks interpretability (Ladjal, Newson, & Pham, 2019). Second, the latent features extracted by PCA are orthogonal to each other and are ordered with respect to their eigenvalues. However, in the standard autoencoders, there is no such ordering and orthogonality, which makes it more difficult to guarantee that the bases of the latent spaces are independent and the size of the latent spaces needs to be predetermined. For more detailed discussions regarding the relationship between t autoencoders and PCA, we refer the readers to Ladjal et al. (2019) and Alkhayrat, Aljnidi, and Aljoumaa (2020).

RBMs and autoencoders achieve similar goals, such as reducing dimensionality or extracting features from signals, via different training theories. The RBM uses stochastic neural networks with a particular distribution instead of deterministic distributions. The main task of training an

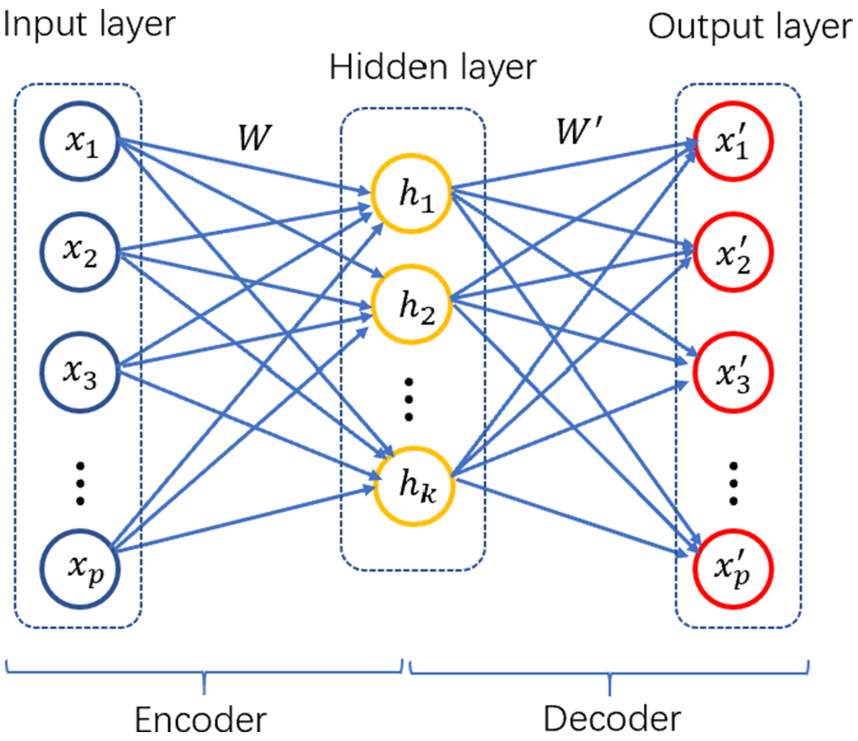

FIGURE 9 Graphical representation of a basic autoencoder

RBM is to obtain connectability between two sets of input variables and hidden variables. There are no intralayer communications in the visible layer or hidden layer. In contrast, for autoencoders, the main task of training is to minimize errors in reconstruction, that is, to find the most efficient compact representation for the input data. The hidden layer is used as important feature representations of the input data. The number of hidden units is much less than the number of visible ones, which extracts essential information of visible units and then represents the input data under another space.

6 | OPTIMIZATION

In this section, we introduce some important optimization algorithms for deep hierarchical structures. Because of the highly nonconvex nature of deep architectures, training deep learning models is challenging. We apply the gradient-descent type algorithms in deep learning optimization because most of the convex optimization algorithms are not applicable. In addition, we provide practical principles for tuning hyperparameters in deep learning.

6.1 | Parameter estimation and acceleration algorithms

We introduce the gradient descent optimization algorithms for parameter estimation as follows. Denote $L(\theta)$ as the loss function to be minimized; the general form of the gradient descent is

$$\theta = \theta - \eta * \nabla_\theta L(\theta),$$

where θ denotes the model parameter and η is the learning rate as a step size at each iteration.

It is common to utilize a part of training data to update the parameter in each step to achieve a balance between accuracy of update and computational cost. On the basis of the quantity of the training data used for updating thegradient, the gradient descent has two main variations. The first one is SGD: $\theta = \theta - \eta * \nabla_\theta L\left(\theta; x^{(i)}; y^{(i)}\right)$, where one random training sample $(x^{(i)}, y^{(i)})$ is used to update parameters in each step. It achieves a much faster computational speed than the original gradient descent. However, the estimation obtained by the SGD in general suffers from high variance and slow convergence. The second variation is the mini-batch gradient descent: $\theta = \theta - \eta * \nabla_\theta L\left(\theta; x^{(i:i+n)}; y^{(i:i+n)}\right)$, where the parameter is updated through averaging on a small subset of the training dataset. This is much faster than the original gradient descent and more stable than the SGD. However, it requires the determination of a mini-batch size as an additional tuning parameter.

The main challenge for the original gradient descent is that it lacks guarantee of a good convergence. In addition, because the estimation can be easily trapped at the suboptimal local minima or saddle points, many modifications focus on adaptively adjusting the learning rate. Specifically, some algorithms ensure that the learning rate is not too large to prevent fluctuated outcomes and not too small to be trapped in local minima, which leads to a slow convergence. Table 2 lists popular variants of gradient descent to accelerate the convergence. In the following, we provide details for Nesterov accelerated gradient (NAG) (Nesterov, 1983) and adaptive moment estimation (ADAM) (Kingma & Ba, 2014).

The main idea of NAG is to provide a short-term memory from the past gradients for the current gradient for achieving a fast acceleration and avoiding saddle points:

$$v_k = \beta v_{k-1} + \alpha \nabla_\theta L\left(\theta^{(k-1)} - \eta v_{k-1}\right), \quad \theta^{(k)} = \theta^{(k-1)} - v_k,$$

where v_{k-1} is the memory of the past gradient, and $\nabla_\theta L\left(\theta^{(k-1)} - \eta v_{k-1}\right)$ represents correction of the gradient at the current step. Coefficients $\alpha \in [0, 1]$, $\beta \in [0, 1]$ are weights on the correction term and the memory term, respectively, and v_k is the velocity update. The NAG degenerates to

TABLE 2 Summary of acceleration methods

Methods	Principle summary	Advantages
(mini-batch) SGD	Update gradient through random subset of samples	Reduce computational time and memory cost
		Suitable for online learning
NAG	Update gradient with adding previous gradients	Speed up convergence; avoid saddle points
AdaGrad/AdaDelta	Large update on infrequently updated directions	Handle sparse and unbalanced data
	Small update on frequently updated directions	Avoid overfitting
ADAM	Combine AdaDelta and NAG	Recommended optimizer for deep learning

Abbreviations: ADAM, adaptive moment estimation; NAG, Nesterov accelerated gradient; SGD, stochastic gradient descent.

the vanilla gradient descent when $\beta = 0$. In general, the NAG speeds up the convergence of parameter estimation up to a quadratic rate compared with the original gradient descent.

The second popular acceleration algorithm for many deep learning applications is called the ADAM. The main idea of ADAM is to use small updating steps in a direction where parameters have large uncertainty. Considering an update in a given direction $g_k := \nabla_\theta L(\theta^{(k)})$, the ADAM consists of three steps:

1. Incorporate memory from previous gradients:

$$\begin{aligned} &\text{(i) update velocity}: v_k = \beta_1 v_{k-1} + (1-\beta_1) g_k, \\ &\text{(ii) update uncertainty}: m_k = \beta_2 m_{k-1} + (1-\beta_2) g_k^2, \end{aligned}$$

   where the default parameters recommended are $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

2. Correct bias via calculating the high-order moments given $E(v_k) \approx (1-\beta_1^k)E(g_k)$ and $E(m_k) \approx (1-\beta_2^k)E(g_k^2)$:

$$\hat{v}_k = \frac{v_k}{1-\beta_1^k} \text{ and } \hat{m}_k = \frac{m_k}{1-\beta_2^k}.$$

3. Update parameter: $\theta_{k+1} = \theta_k - \eta \frac{\hat{v}_k}{\sqrt{\hat{m}_k}}$, where $\frac{\hat{v}_k}{\sqrt{\hat{m}_k}}$ is a signal-to-noise ratio and determines the learning rate.

The acceleration algorithms enable an adaptive adjustment for the learning rate. For example, the acceleration algorithms such as ADAM apply different learning rates for updating different parameters , which is useful when data are sparse. In addition, it accommodates features with different frequencies and performs a large update for rarely occurring features and a small update for frequent ones. However, some cautions might be taken into account. The empirical study (Wilson, Roelofs, Stern, Srebro, & Recht, 2017) shows that adaptive gradient methods tend to overfit the training data and select spurious features, which might lead to a poor generalization on test data.

6.2 | Hyperparameter tuning

The performance of deep neural networks is sensitive to hyperparameters such as the batch size, filter size, the number of layers, and the number of nodes of each layer, and most of these hyperparameters need to be predetermined. In contrast to the Akaike information criterion (AIC) or Bayesian information criterion (BIC) used in the statistical model selection, there are no explicit criteria for tuning hyperparameters in deep neural networks. Instead, deep learning selects the best hyperparameters on the basis of the performance on a validation set. The entire tuning procedure can be viewed as a black-box optimization where only the input values and the output values are known. The most widely adopted method is the grid search, which traverses all the combinations of hyperparameters. However, the grid search is intractable and computational costly when the range of hyperparameters is large or the parameter space is continuous.

Alternatively, we can choose hyperparameters without exhaustive search. One method is to randomly select a subset of the hyperparameters (Bergstra & Bengio, 2012). Another method is the sequential model-based optimization (Bergstra, Bardenet, Bengio, & Kégl, 2011), which applies an explicit surrogate model to approximate the unknown target function. Specifically, the sequential model-based method views the hyperparameters as independent variables and the accuracy on the validation set as dependent variables, respectively, then estimates the posterior distribution of hyperparameters using a Gaussian process or random forest. The optimization is proceeded in a sequential fashion, where the hyperparameters in the next step are selected to maximize the expected improvement of the surrogate model estimated in the current step. Consequently, this method avoids sampling the parameters that might not increase the model accuracy and therefore is more efficient than the grid search or random search.

7 | CONNECTION BETWEEN DEEP LEARNING MODELS AND STATISTICAL MODELS

In this section, we connect the principles of deep learning to statistical concepts and modelling. Most of statistical models can be viewed as shallow models, in contrast to deep learning models, which can be considered as generalization of statistical models due to their increased model depth.

Specifically, the principle of deep learning models is to perform feature engineering and seek a class of functions to achieve good performance on the learning task. In addition, deep learning models are capable of representing highly nonlinear relations between the input data and the output data. This is achieved through multiple layers of composition functions for latent features from the input data. Furthermore, the deep

architecture brings some new insights about the bias-variance trade-off principle under the overparameterization regime, which also motivates us to develop new learning theories and large-scale nonconvex optimization algorithms.

7.1 | Deep learning as a statistical model

Deep neural networks are connected with many existing statistical or machine learning approaches. For example, the FNN with hidden layers is equivalent to a recursive generalized linear model with a hierarchical structure. If there is no hidden layer, the FNN is equivalent to a generalized linear model. However, if there are multiple hidden layers, then the hidden unit in each layer $\boldsymbol{h}^{(\ell)} = \sigma(W^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)})$ is just a generalized linear model, where the activation function $\sigma(\cdot)$ corresponds to a link function. The composition of sequential simple functions provides the flexibility of FNNs on feature selections for complex data and enables approximation of complex nonlinear functions.

For deep generative models, it also connects to unsupervised learning models. For example, the autoencoder connects with the latent factor model in statistics (Qiu & Wang, 2020; Qiu, Zhang, & Wang, 2020). The autoencoder consists of an encoder and a decoder, where the encoder estimates the latent factor through a prespecified model $\boldsymbol{Z}_f = f(Y', \Theta)$, and Y' is a sample of the observed dataset Y containing random noise and Θ is the model parameter. The latent factor $\boldsymbol{Z}_f$ serves as a compressed representation for Y', and the encoding procedure is equivalent to inferring a parametric distribution from the sample data. In contrast, decoding is a sampling process that generates data from a specific distribution $p(\boldsymbol{Y}|\boldsymbol{Z}_f)$ based on the learned latent factor. Figure 10 illustrates the encoding and decoding processes. Specifically, the autoencoder aims to capture the hidden data generation process by optimizing $\mathcal{L}(\boldsymbol{Z}_f) = \log p(\boldsymbol{Y}|\boldsymbol{Z}_f) - \lambda R(\boldsymbol{Y}, Y')$, where the penalty term $R(\boldsymbol{Y},\boldsymbol{Y}')$ measures the discrepancy between $\boldsymbol{Y}$ and $\boldsymbol{Y}'$.

Most of the supervised or unsupervised algorithms such as PCA, project pursuit regression, and kernel methods can be regarded as shallow models compared with the deep model as they follow a two-layer framework that consists of a data-transformation layer and a reconstruction layer (Polson & Sokolov, 2017). Specifically, consider a training dataset (X, Y) and a two-layer model:

$$\boldsymbol{Z} = f_1(W^{(1)}X + b^{(1)}) \text{ and } E(Y) \approx f_2(W^{(2)}Z + b^{(2)}), \tag{5}$$

where $W^{(1)}$, $W^{(2)}$, $b^{(1)}$, $b^{(2)}$ are corresponding weights and biases defined in Section 2. For the aforementioned shallow models, the information in the original data X is projected into the latent space $\boldsymbol{Z}$ through a data transformation $f_1(\cdot|W^{(1)}, b^{(1)})$. Then, the output Y is reconstructed through a composition procedure $f_2(\cdot|W^{(2)}, b^{(2)})$ on the latent space $\boldsymbol{Z}$. The main challenge here is to determine appropriate f_1 and f_2 to uncover relevant features in predicting the output Y. The traditional statistical learning methods differ at the specification of output Y, f_1 and f_2 in (5). For example, given that f_1 is the identity function and f_2 is a nonlinear smooth function, then (5) is equivalent to the project pursuit regression. For another example, given that $Y = \{1, -1\}$, f_2 is the sign function, and f_1 denotes a class of eigen-functions expanded by a kernel, then (5) is equivalent to the support vector machine (SVM) under the hinge loss. On the other hand, if f_1 and f_2 are nonlinear functions such as the logistic function, then (5) is a two-layer FNN. Therefore, the deep learning model generalizes a two-layer shallow model to a hierarchical multiple-layer model.

The RNN also has its counterpart in statistics, which is the probabilistic recurrent state-space model. A sequence data $(x_1, x_2, \ldots, x_T)$ is governed by the joint distribution of the observed sequence given the corresponding hidden states $h_t, 1 \le t \le T$:

$$p(x_1, \ldots x_T) = \prod_{t=1}^{T} \int p(x_t|h_t)p(h_t|h_{t-1})dh_t. \tag{6}$$

The RNN is a special case of the probabilistic modelling in (6) in that the transition of hidden states is assumed to be deterministic, that is, $p(h_t|h_{t-1}) = f_\theta(h_{t-1}, x_{t-1})$, where $f_\theta(\cdot)$ is the hidden unit in the RNN. Accordingly, training the RNN model is equivalent to maximizing the log-likelihood function $\log p(x_1, \ldots x_T)$ (Goodfellow et al., 2016).

7.2 | Beyond statistical model: Overparameterization

One significant difference between deep learning and statistical modelling is the understanding of overparameterization. The statistical principle suggests that increasing the complexity of models tends to cause overfitting and being less robust on future prediction. On the contrary, deep

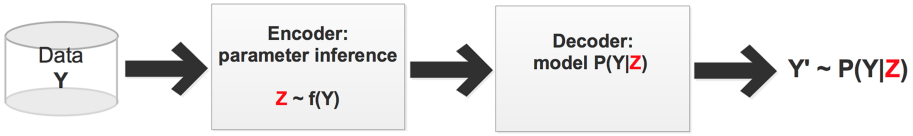

FIGURE 10 A deep generative model as a latent factor model

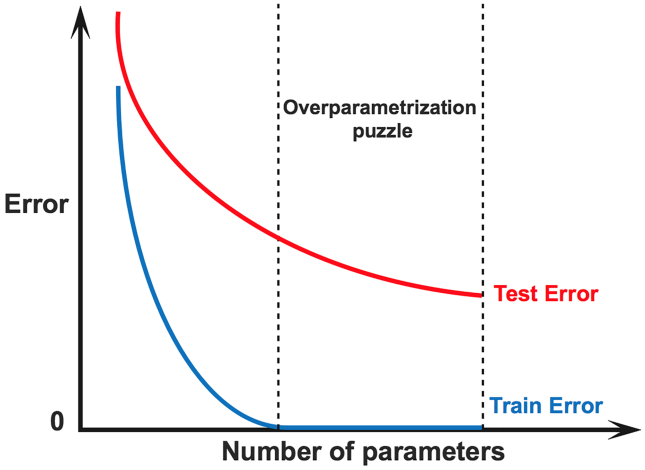

FIGURE 11 Overparameterization on deep learning: test error keeps decreasing after train error vanishes

learning is able to decrease the generalization error with increasing the number of parameters. Specifically, Figure 11 illustrates the over-parameterization phenomenon where the test error could further decrease through increasing the number of hidden units while the training error vanishes.

This phenomenon has been confirmed by empirical studies. For example, numerical results in Zhang, Bengio, Hardt, Recht, and Vinyals (2016) demonstrate the benefits of overparameterization on an image classification task. They show that the deep neural network can achieve zero training error even when the images are randomly labelled, and the trained neural network also fits the test dataset well. These results also imply that the overparameterization enables the deep learning model to memorize the entire dataset. Although it seems to contradict the statistical principle of bias-variance trade-off, some recent works (Belkin, Hsu, Ma, & Mandal, 2019; Belkin, Ma, & Mandal, 2018) explain that the vanishing of the training error does not necessarily diminish the generalization power on test data when the number of parameters exceeds the size of the training dataset.

For statistical models, it is conventional to control the model complexity via regularization. In contrast, the model complexity of deep learning is not directly controlled by the explicit regularizations. Instead, the regularizations serve as tuning parameters for decreasing the test error. The absence of explicit regularizations does not necessarily lead to insufficient power of generalization on test data. In the following, we explain the rationale behind overparameterization in deep learning models from the learning theory and optimization algorithm perspective.

7.2.1 | Overparameterization via complexity measurement

Recent works (Arora, Cohen, & Hazan, 2018; Chen, Mo, Yang, & Wang, 2019; Liao & Poggio, 2017; Xu & Wang, 2018; Xu, Zheng, Yang, & Wang, 2020) provide theoretical justification on why overparameterization could benefit decreasing generalization errors. Specifically, a relation between the complexity of neural networks and the corresponding generalization errors is established. For example, results in Neyshabur, Li, Bhojanapalli, LeCun, and Srebro (2018) indicate that the traditional model complexity measurements such as the number of parameters or Vapnik-Chervonenkis dimension are not applicable to deep learning models and thus are not suitable for overparameterization. Instead, a complexity measurement for neural networks is introduced. On the basis of the new measurement, it can be shown that the generalization error decreases as the number of hidden units increases. Specifically, consider a two-layer and fully connected ReLU network $f_{W^{(1)},W^{(2)}}(x) = W^{(2)}[W^{(1)}x]_+$, where $[x]_+ = \max(0, x)$, $x \in R^d$ is the d-dimension input, and $W^{(1)} \in R^{h\times d}$, $W^{(2)} \in R^{c\times h}$ are weighting matrices for the hidden layer and the output layer, respectively. Let $L_0(f)$ and $\hat{L}_0(f)$ be the expected risk and the empirical risk. Neyshabur et al. (2018) shows that

$$L_0(f) \le \hat{L}_0(f) + O\left(\frac{\|W^{(2)}\|_F(\|W^{(1)} - W_0^{(1)}\|_F + \|W_0^{(1)}\|_2)}{\sqrt{m}}\right),$$

where m is the size of the training dataset, $W_0^{(1)}$ stands for the initial weight of $W^{(1)}$, $\|W_i^{(1)} - W_{0i}^{(1)}\|_F$ determines the complexity of the ith hidden unit, and $\|W_i^{(2)}\|_F$ stands for the weight of the ith hidden unit in the output. Empirical studies show that both $\|W_i^{(1)} - W_{0i}^{(1)}\|_F$ and $\|W_i^{(2)}\|_F$ decrease at a rate faster than $1/\sqrt{n}$, where n is the number of hidden units. In other words, the complexity of hidden units decreases faster than the increasing number of hidden units. Consequently, the generalization error of the neural network decreases as the number of parameters increases. In other words, although the function space expanded by the neural network with all possible weights $(W^{(1)}, W^{(2)})$ become larger, the function space to fit the observed dataset becomes smaller.

7.2.2 | Overparameterization via implicit regularization in optimization

The power of generalization for a deep learning model is also determined by optimization algorithms. Empirical studies (Zhang et al. 2018; Zhang et al. 2016) show that gradient descent algorithms improve the generalization error. This is because the gradient descent algorithms introduce an implicit regularization (Rosasco & Villa, 2015; Ali, Dobriban, & Tibshirani, 2020; Wu et al. 2019; Shuxiao, Dobriban, & Jane, 2020). In addition, an estimation from gradient descent asymptotically converges to a solution with a minimal norm with an appropriate initial value (Poggio et al. 2017; Zhang et al. 2016).

We use a linear regression model to demonstrate the intuition. Suppose we consider a linear estimation problem on a training dataset $\{(x_i, y_i)\}_{i=1}^n$: $\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \text{loss}(w^T x_i, y_i)$, where w is the weighting parameter. If we estimate w through the gradient descent algorithm $w_{t+1} = w_t - \eta e_t \frac{1}{n}\sum x_i$ starting from an initial $w_0 = 0$, where η is the learning rate and e_t is the loss function error, then the solution based on the gradient descent has the form $w = X^T\alpha$ with a coefficient α. Given a perfect interpolation such that $Xw = Y$, we have $XX^T\alpha = Y$. Note that this equation only relies on the dot-product between data points x_i, and XX^T is a full rank under the overparameterization regime. Under the regression loss setting, α converges to a unique solution $\alpha^* = (XX^T)^{-1}Y$. Therefore, $w^* = X^T\alpha^* = X^T(XX^T)^{-1}Y$ has the smallest L^2 norm among all w satisfying $Xw = Y$. In classification, the minimum norm solution is known to maximize a margin (Poggio et al. 2017). Specifically, under the logistic or cross-entropy loss function, the solution from the gradient descent converges asymptotically to the max-margin solution, which is similar to the solution of the hard-margin SVM (Zhang et al. 2018; Poggio et al. 2017). In summary, the implicit regularization property of the gradient descent stands for a broad class of regression loss and classification loss functions.

8 | SOFTWARE AND COMPUTING RESOURCE

In this section, we introduce some development tools including softwares, hardwares, and cloud computing resources. There are many open-source softwares that can be applied to train deep learning networks, for example, Tensorflow(Abadi et al. 2015), Pytorch (Paszke et al. 2019), and Cognitive Toolkit (CNTK) (Seide & Agarwal, 2016). Among these softwares, Tensorflow is widely implemented in industry because it facilitates an easy and fast deployment of the trained deep learning model for production. In addition, the integrated tool Tensorboard provides visualization of the training process, which is useful for debugging and hyperparameter tuning. Meanwhile, Pytorch is more favored in the research community because of its simplicity to build a deep learning model. Another advantage of Pytorch is its support for dynamic computing graphs, which greatly facilitates linguistic analyses. For R users, Keras (Chollet, 2017) offers a user-friendly interface that enables quickly prototyping deep learning models. Moreover, Keras allows the same coding to run on CPU or on GPU without significant changes. More softwares and their platforms as well as supporting languages are also listed in Table 3.

Training deep learning models has high demand on hardwares because of high volume of parameters. Nowadays, there are mainly three types of processing units: CPU, GPU, and TPU. In general, GPUs can accelerate the training by 100 times compared with CPUs because of its multicore architecture, which facilitates parallel computing. On the other hand, TPUs are highly optimized for large batches in deep learning and can boost the computing speed by up to 20 times compared with GPUs. Although CPUs have the slowest training speed among all processing units, they have the largest memory per core and thus are capable of implementing extremely large models or datasets, whereas GPUs and TPUs may encounter out-of-memory issues. For a more detailed comparison between TPUs, GPUs, and CPUs in deep learning training, we refer the readers to Wang et al. (2019).

Instead of training deep learning models locally, which may require high investment on hardware and extra effort to set up the environment, there are cloud computing resources available online, such as Google Colaboratory (Google LLC. 2020), AWS Sagemaker (Amazon Web Services Inc. 2020), FloydHub (Floyd Labs Inc. 2020) and Azure (Microsoft Corporation, 2020). These cloud services enable users to run deep learning models faster and cheaper by supplying GPU or TPU computing power remotely.

TABLE 3 Summary of deep learning softwares

Software	Platforms	Language	Author
Tensorflow	Linux, macOS, Windows, Android	Python, C/C++, R, Java	Abadi et al. (2015)
Pytorch	Linux, macOS, Windows, Android (beta)	Python, C++	Paszke et al. (2019)
CNTK	Linux, Windows	Python, C++	Seide and Agarwal (2016)
Theano	Linux, macOS, Windows	Python	The Theano Development Team et al. (2016)
Keras	Linux, macOS, Windows	Python, R	Chollet (2015)
Matlab Deep learning toolbox	Linux, macOS, Windows	Matlab	The MathWorks, Inc. (2019)

9 | DISCUSSION AND CONCLUSION

In this review, we introduce the general structure of deep neural networks and its applications in imaging analyses, sequence data and recommender systems. We also review important optimization algorithms, software, and platforms for implementing deep learning models. In addition, we provide the general connections between deep learning models and statistical principles.

Deep learning models distinguish from standard statistical models in two aspects: extraction of task-relevant features from input data and the target function for modelling the relations between input data and output data. In deep learning models, latent features are hierarchically extracted from input data in that high-level features are composite of low-level features. The hierarchical structure is able to digest input data at multiple levels, which is important for many artificial intelligence applications that mimic human behaviour. In addition, deep learning models are sequentially composed of multiple nonlinear and affine transformations. This architecture is equivalent to a procedure that splits the input space into a large number of irregular regions and discriminates each region on the basis of its corresponding output value. Therefore, deep learning is able to approximate complex relations between input data and output data if the model is sufficient deep. In contrast to statistical models, the number of parameters in deep learning models is much larger than the size of training data. However, the overparameterization of deep learning indeed decreases the generalization error and enhances the performance in many applications.

However, deep learning has its own limitations. First, deep learning lacks interpretability (Lipton, 2018). That is, deep learning is mostly black-box function approximators. Nevertheless, there are some recent works on improving interpretability (Erhan, Bengio, Courville, & Vincent, 2009; Karpathy, Johnson, & Fei-Fei, 2015; Mahendran & Vedaldi, 2015; Nguyen et al. 2016). For example, Erhan et al. (2009) developed visualizing the responses of individual units in any layer of a neural network. Zeiler and Fergus (2014) and Karpathy et al. (2015) extended the above idea to the CNN and the LSTM, respectively. Mahendran and Vedaldi (2015), Yosinski, Clune, Nguyen, Fuchs, and Lipson (2015), and Nguyen et al. (2016) investigated the latent features at different CNN levels for better understanding of layer representations in deep models. Shwartz-Ziv and Tishby (2017) provided a deep insight for analysing deep networks using information theory, which applies the information bottleneck framework (Tishby, Pereira, & Bialek, 1999) to calculate information preserved on each layer's inputs and outputs. Second, deep learning has consistency issues, including the consistency of a universal neural network classifier (Faragó & Lugosi, 1993), the consistency of training deep neural networks (Ye, Yang, Fermuller, & Aloimonos, 2017), the consistency of using metrics to select the best network (Twomey & Smith, 1995) and the consistency of neural network detections (Yang, Gregg, & Babaeizadeh, 2020). In addition, the uncertainties of neural networks such as model uncertainty (Blundell, Cornebise, Kavukcuoglu, & Wierstra, 2015; Gal, 2016) and data uncertainty are evaluated using the tested algorithms on data with noise or disturbance. Nevertheless, these problems remain open and unsolved.

ACKNOWLEDGEMENTS

The authors would like to acknowledge support for this project from the National Science Foundation Grants DMS-1821198 and DMS-1952406. The authors thank the Chief Editor, Guest Editor and anonymous reviewer for their suggestions and helpful feedback, which improved the paper significantly.

ORCID

Annie Qu https://orcid.org/0000-0002-8396-7828

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ..., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, Software available from tensorflow.org.

Alashkar, T., Jiang, S., Wang, S., & Fu, Y. (2017). Examples-rules guided deep neural network for makeup recommendation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, AAAI 17, AAAI Press, pp. 941–947.

Ali, A., Dobriban, E., & Tibshirani, R. J. (2020). The implicit regularization of stochastic gradient flow for least squares. arXiv preprint arXiv:200307802.

Alkhayrat, M., Aljnidi, M., & Aljoumaa, K. (2020). A comparative dimensionality reduction study in telecom customer segmentation using deep learning and PCA. Journal of Big Data, 7(9). https://doi.org/10.1186/s40537-020-0286-0

Amazon Web Services Inc. Aws sagemaker. Retrieved Aug 20, 2020 from https://aws.amazon.com/sagemaker/

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pp. 214–223.

Arora, S., Cohen, N., & Hazan, E. (2018). On the optimization of deep networks: Implicit acceleration by overparameterization. arXiv preprint arXiv: 180206509.

Azzouni, A., & Pujolle, G. (2017). A long short-term memory recurrent neural network framework for network traffic matrix prediction. arXiv preprint arXiv: 170505690.

Bansal, T., Belanger, D., & McCallum, A. (2016). Ask the GRU: Multi-task learning for deep text recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 107–114.

Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. Proceedings of the National Academy of Sciences, 116(32), 15849–15854.

Belkin, M., Ma, S., & Mandal, S. (2018). To understand deep learning we need to understand kernel learning. arXiv preprint arXiv:180201396.

Bergstra, J. S., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pp. 2546–2554.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb), 281–305.

Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015). Weight uncertainty in neural network. In *International Conference on Machine Learning*, pp. 1613–1622.

Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., ..., & Tamchyna, A. (2014). Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, Association for Computational Linguistics, Baltimore, Maryland, USA, pp. 12–58. http://www.aclweb.org/anthology/W/W14/W14-3302

Che, T., Li, Y., Jacob, A. P., Bengio, Y., & Li, W. (2016). Mode regularized generative adversarial networks. arXiv preprint arXiv:161202136.

Chen, H., Mo, Z., Yang, Z., & Wang, X. (2019). Theoretical investigation of generalization bound for residual networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, AAAI Press, pp. 2081–2087.

Chen, M., Xu, Z., Weinberger, K. Q., & Sha, F. (2012). Marginalized denoising autoencoders for domain adaptation. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, Omnipress, Madison, WI, USA, pp. 1627–1634.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:14061078.

Chollet, F. (2015). Keras. https://keras.io

Chollet, F. (2017). R interface to Keras. https://github.com/rstudio/keras

Chu, W.-T., & Tsai, Y.-L. (2017). A hybrid recommendation system considering visual information for predicting favorite restaurants. *World Wide Web*, 20 (6), 1313–1331. https://doi.org/10.1007/s11280-017-0437-1

CireşAn, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12), 3207–3220.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (ELUS). arXiv preprint arXiv: 151107289.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4), 303–314.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, pp. 248–255.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:181004805.

Dong, X., Yu, L., Wu, Z., Sun, Y., Yuan, L., & Zhang, F. (2017). A hybrid collaborative filtering model with deep structure for recommender systems. In *The 31st AAAI Conference on Artificial Intelligence*, pp. 1309–1315.

Ehsani, K., Mottaghi, R., & Farhadi, A. (2018). SeGAN: Segmenting and generating the invisible. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6144–6153.

Elkahky, A. M., Song, Y., & He, X. (2015). A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 278–288.

Elman, J. L. (1990). Finding structure in time. Cognitive Science, 14(2), 179–211. http://dblp.uni-trier.de/db/journals/cogsci/cogsci14.html/Elman90

Erhan, D., Bengio, Y., Courville, A., & Vincent, P. (2009). Visualizing higher-layer features of a deep network. (*1341*). Canada: University of Montreal Also presented at the ICML 2009 Workshop on Learning Feature Hierarchies, Montréal.

Faragó, A., & Lugosi, G. (1993). Strong universal consistency of neural network classifiers. *IEEE Transactions on Information Theory*, 39(4), 1146–1151.

Floyd Labs Inc. FloydHub. Retrieved Aug 20, 2020 from https://www.floydhub.com

Freund, Y., & Haussler, D. (1992). Unsupervised learning of distributions on binary vectors using two layer networks. In *Advances in Neural Information Processing Systems*, pp. 912–919.

Gal, Y. (2016). Uncertainty in deep learning. (Ph.D. Thesis).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge, MA, USA: MIT Press. http://www.deeplearningbook.org

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:14126572.

Google LLC. Google Colaboratory. Retrieved Aug 20, 2020 from https://colab.research.google.com

Haykin, S. (1994). *Neural networks: A comprehensive foundation*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8), 1771–1800. https://doi.org/10.1162/089976602760128018

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

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

Hornik, K., Stinchcombe, M., & White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5), 551–560.

Karpathy, A., Johnson, J., & Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. arXiv preprint arXiv:150602078.

Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410.

Kingma, D. P., & Ba, J. (2014). ADAM: A method for stochastic optimization. arXiv preprint arXiv:14126980.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational Bayes. arXiv preprint arXiv:13126114.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial examples in the physical world. arXiv preprint arXiv:160702533.

Ladjal, S., Newson, A., & Pham, C.-H. (2019). A PCA-like autoencoder. arXiv preprint arXiv:190401277.

Längkvist, M., Karlsson, L., & Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. Pattern Recognition Letters, 42, 11–24. http://www.sciencedirect.com/science/article/pii/S0167865514000221

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.

Li, S., Kawale, J., & Fu, Y. (2015). Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, CIKM 15, Association for Computing Machinery, New York, NY, USA, pp. 811–820. https://doi.org/10.1145/2806416.2806527

Liang, T. (2018). How well can generative adversarial networks learn densities: A nonparametric view. arXiv:171208244 [cs, math, stat].

Liang, D., Zhan, M., & Ellis, D. P. W. (2015). Content-aware collaborative music recommendation using pre-trained neural networks. In *Ismir*, pp. 295–301. http://dblp.uni-trier.de/db/conf/ismir/ismir2015.html/LiangZE15

Liao, Q., & Poggio, T. (2017). Theory II: Landscape of the empirical risk in deep learning. arXiv preprint arXiv:170309833.

Lipton, Z. C. (2018). The mythos of model interpretability. Queue, 16(3), 31–57. https://doi.org/10.1145/3236386.3241340

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, Citeseer, pp. 3.

Mahendran, A., & Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *CVPR*, IEEE Computer Society, pp. 5188–5196. http://dblp.uni-trier.de/db/conf/cvpr/cvpr2015.html/MahendranV15

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 5(4), 115–133.

Metz, L., Poole, B., Pfau, D., & Sohl-Dickstein, J. (2016). Unrolled generative adversarial networks. arXiv preprint arXiv:161102163.

Microsoft Corporation. Microsoft azure. Retrieved Aug 20, 2020 from https://azure.microsoft.com/en-in/services/machine-learning-studio%/

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Dokl Akad Nauk SSSR*, 269, 543–547.

Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., & Srebro, N. (2018). Towards understanding the role of over-parametrization in generalization of neural networks. arXiv preprint arXiv:180512076.

Nguyen, A., Yosinski, J., & Clune, J. (2016). Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. arXiv preprint arXiv:160203616.

Nowozin, S., Cseke, B., & Tomioka, R. (2016). f-GAN: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, pp. 271–279.

Olabiyi, O., Martinson, E., Chintalapudi, V., & Guo, R. (2017). Driver action prediction using deep (bidirectional) recurrent neural network. arXiv preprint arXiv:170602257.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ..., & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems 32, pp. 8024–8035.

Poggio, T., Kawaguchi, K., Liao, Q., Miranda, B., Rosasco, L., Boix, X., ..., & Mhaskar, H. (2017). Theory of deep learning III: Explaining the non-overfitting puzzle. arXiv preprint arXiv:180100173.

Polson, N. G., & Sokolov, V. (2017). Deep learning: A Bayesian perspective. *Bayesian Analysis*, 12(4), 1275–1304.

Qiu, Y., & Wang, X. (2020). Almond: Adaptive latent modeling and optimization via neural networks and Langevin diffusion. Journal of the American Statistical Association, 1–13. https://doi.org/10.1080/01621459.2019.1691563

Qiu, Y., Zhang, L., & Wang, X. (2020). Unbiased contrastive divergence algorithm for training energy-based latent variable models. In *International Conference on Learning Representations*, pp. 1-17. https://openreview.net/forum?id=r1eyceSYPr

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI Blog, 1(8), 9.

Rosasco, L., & Villa, S. (2015). Learning with incremental iterative regularization. In *Advances in Neural Information Processing Systems*, pp. 1630–1638.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386.

Rosenblatt, F. (1961). *Principles of neurodynamics. Perceptrons and the theory of brain mechanisms*. Buffalo, NY: Cornell Aeronautical Lab Inc.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533–536.

Sabour, S., Frosst, N., & Hinton, G. (2018). Matrix capsules with EM routing. In *6th International Conference on Learning Representations, ICLR*, pp. 1–15.

Salakhutdinov, R., & Hinton, G. (2009). Deep Boltzmann machines. In *Artificial intelligence and statistics*, pp. 448–455.

Salakhutdinov, R., Mnih, A., & Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML'07, Association for Computing Machinery, New York, NY, USA, pp. 791–798. https://doi.org/10.1145/1273496.1273596

Sallab, A. E. L., Abdou, M., Perot, E., & Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19), 70–76.

Seide, F., & Agarwal, A. (2016). CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD. 16, Association for Computing Machinery, New York, NY, USA, pp. 2135. https://doi.org/10.1145/2939672.2945397

Shuxiao, C., Dobriban, E., & Jane, H. L. (2020). A group-theoretic framework for data augmentation. arXiv preprint arXiv:190710905.

Shwartz-Ziv, R., & Tishby, N. (2017). Opening the black box of deep neural networks via information.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ..., & Graepel, T. (2017). Mastering chess and Shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:171201815.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory, *Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1: Foundations*, pp. 194–281.

Srivastava, N., & Salakhutdinov, R. R. (2012). Multimodal learning with deep Boltzmann machines. In *Advances in Neural Information Processing Systems*, pp. 2222–2230.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ..., & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.

Tang, X., Bi, X., & Qu, A. (2019). Individualized multilayer tensor learning with an application in imaging analysis. *Journal of the American Statistical Association*, 1–26.

The MathWorks, Inc. (2019). Matlab deep learning toolbox R2019b, The MathWorks, Natick, Massachusetts, United States. https://www.mathworks.com/help/deeplearning/

The Theano Development Team, Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., ..., & Zhang, Y. (2016). Theano: A Python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688.

Tishby, N., Pereira, F. C., & Bialek, W. (1999). The information bottleneck method. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing*, pp. 368–377. https://arxiv.org/abs/physics/0004057

Tran, L., Yin, X., & Liu, X. (2018). Representation learning by rotating your faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(12), 3007–3021.

Twomey, J. M., & Smith, A. E. (1995). Performance measures, consistency, and power for artificial neural network models. *Mathematical and Computer Modelling*, 21(1), 243–258.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML'08, Association for Computing Machinery, New York, NY, USA, pp. 1096–1103.https://doi.org/10.1145/1390156.1390294

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11, 3371–3408.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., ..., & Schrittwieser, J. (2017). Starcraft II: A new challenge for reinforcement learning. arXiv preprint arXiv:170804782.

Wang, Y. E., Wei, G.-Y., & Brooks, D. (2019). Benchmarking TPU, GPU, and CPU platforms for deep learning. http://arxiv.org/abs/1907.10701

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. Neural Networks, 1(4), 339–356. http://www.sciencedirect.com/science/article/pii/089360808890007X

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., & Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pp. 4148–4158.

Wu, X., Dobriban, E., Ren, T., Wu, S., Li, Z., Gunasekar, S., ..., & Liu, Q. (2019). Implicit regularization of normalization methods. arXiv preprint arXiv:191107956.

Xu, Y., & Wang, X. (2018). Understanding weight normalized deep neural networks with rectified linear units. In *Advances in Neural Information Processing Systems*, pp. 130–139.

Xu, Y., Zheng, Y., Yang, Z., & Wang, X. (2020) Statistica sinica preprint no: Ss-2018-0468.

Yang, T., Gregg, R. E., & Babaeizadeh, S. (2020). Detection of strict left bundle branch block by neural network and a method to test detection consistency. *Physiological Measurement*, 41(2), 25005.

Ye, C., Yang, Y., Fermuller, C., & Aloimonos, Y. (2017). On the importance of consistency in training deep neural networks. arXiv preprint arXiv:170800631.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). Understanding neural networks through deep visualization. arXiv preprint arXiv:150606579.

Yuan, Yubai, Deng, Yujia, Zhang, Yanqing, & Qu, Annie (2020). Supplement to "Deep learning from a statistical perspective". http://publish.illinois.edu/yujiadeng/files/2020/05/supplementary_material.pdf

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision—ECCV 2014*, Springer International Publishing, Cham, pp. 818–833.

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:161103530.

Zhang, C., Liao, Q., Rakhlin, A., Miranda, B., Golowich, N., & Poggio, T. (2018). Theory of deep learning IIb: Optimization properties of SGD. arXiv preprint arXiv:180102254.

Zhang, S., Yao, L., & Xu, X. (2017). AutoSVD++: An efficient hybrid collaborative filtering model via contractive auto-encoders. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'17, Association for Computing Machinery, New York, NY, USA, pp. 957–960. https://doi.org/10.1145/3077136.3080689

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.

How to cite this article: Yuan Y, Deng Y, Zhang Y, Qu A. Deep learning from a statistical perspective. *Stat*. 2020;9:e294. https://doi.org/10.1002/sta4.294