Deep Learning from a Statistical Perspective

Yubai Yuan
Department of Statistics, University of Illinois, Urbana-Champaign
Yujia Deng
Department of Statistics, University of Illinois, Urbana-Champaign
Yanqing Zhang
Department of Statistics, Yunnan University, China
and
Annie Qu*
Department of Statistics, University of California, Irvine

Abstract

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, the deep neural networks can be viewed as non-linear and non-parametric 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 to 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 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

*Corresponding author: aqu2@uci.edu.

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 [37]. The first stage is 1940s-1960s, where linear networks with a single neuron were proposed inspired by neural science [67, 81, 82]. However, it lacked non-linear features and thus the applications were limited.

The second stage is 1980s-1990s, gaining more successful developments due to successful backpropagation algorithms [83] in training neural networks with one or two layers. The backpropagation algorithm is still the dominant approach to train a neural network nowadays. Furthermore, the prototype of many popular deep models originates in late second stage, for example, the Boltzmann Machine [84], the long short-term memory (LSTM) [46] and the convolutional neural network (CNN) [57]. However, the applications of LSTM and CNN were limited at that time due to the high computational cost. For example, training a CNN with only two hidden layers could take three days [57]. Another issue is that the backpropagation algorithm is often trapped in a local optimum due to the nature of non-convexity of the multilayer networks optimization.

The computational issue was unresolved until Hinton proposed the deep belief network (DBN) [45] which is used as initialization strategy in deep neural network training. The initializing strategy is to weight a neural network with a configured DBN to achieve better performance than random initialization, and is more computationally efficient. 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 make deep learning more popular. For example, the deep learning model [21] achieved a lower error rate than humans in the traffic-sign recognition competition. AlexNet [54] won the 2012 ImageNet Recognition Challenge with more than 10% error rate than the best non-neural-network solution. In 2015, ResNet [43] 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 GPUs, which accelerate the training of neural networks for about 100 times [22] compared with using CPUs. Recently, the invention of TPU [49] furthers improve the speed by 20 times compared with GPU. The other key factor is the large available labeled datasets. The ImageNet dataset [25] is launched in 2009 and contains 3.2 million annotated images with 5,247 categories; while the WMT 2014 English to French dataset [14] consists of almost 1 billion sentence pairs for machine translation. These datasets make it feasible to generalize the deep neural networks by supplying a large amount of training data and therefore facilitate model prediction accuracy. On the other hand, the traditional parametric models fail to fully approximate the complex distribution from these large datasets, while 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: Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNNs are designed for image-related tasks and are applied in image classification, object detection, face recognition and medical imaging data analyses. RNNs including their variations long such as short-term memory (LSTM) and gated recurrent unit (GRU) [17] are designed for sequential data such as time series data and acoustic data. RNNs are also applied in dynamic tracking and monitoring sequential system including traffic forecasting system [7] and driver action prediction system [75].

The deep unsupervised learning aims at extracting features and learning underlying distributions of input data. Among this category, Autoencoders (AEs) [45] and Restricted Boltzmann Machines (RBMs) [97, 86] are widely applied in text mining and recommender system; and the generative adversarial networks (GANs) are effective for style transfer, video synthesis and so on. In addition, the deep unsupervised learning can be integrated into deep supervised learning as a preprocessing technique, which is commonly used in recommender systems [20] and natural language processing [26, 79].

The 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 [93] and Alpha-Zero [93]. Other applications include autonomous driving techniques [88] and game artificial intelligence [108] 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, the neural networks doubled in size every 2.4 years [37]. The increasing trend is even accelerated more since ResNet [43] has been proposed. Although few theoretical developments have been established so far, experimental results [94, 98] demonstrate that the performance of deep neural networks improves with increasing number of layers, or equivalently, the deeper networks provide lower generalization error.

In general, the deep learning deals with supervised or unsupervised learning problems under a framework similar to statistical methods. However, the deep learning goes beyond the 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 non-linear transformations over latent features from input data. This architecture is extremely effective in capturing highly non-linear relations between input data and output data. In addition, deep learning models bring new insights on 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, i.e., the general feed-forward network in Section 2. In Section 3, we present two popular models in imaging data analyses: the convolutional neural network and the generative adversarial network. 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 and Section 7 focus on optimization and implementations for training a deep learning model. We summarize the connections between deep learning and statistical models in Section 8. Section 9 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.

2 Basic Deep Architecture: Feed-forward Neural Network

In this section, we introduce the general structure of the Feed-forward Neural Network (FNN) [41]. We denote the training dataset as $\{(\boldsymbol{x}_i, y_i)\}_{1\leq i\leq n}$, 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 $\mathbb{R}$, while in classification problems, y_i belongs to a discrete set $[K] = \{1, 2, ..., K\}$, 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 a L-layer FNN, we denote the dimension of the ℓth layer as d_ℓ, and the ℓth layer can be defined recursively by

$$\begin{aligned}\boldsymbol{h}^{(\ell)} &= \boldsymbol{\sigma}(W^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}), \quad \ell = 1, 2, ..., L\\ \boldsymbol{h}^{(0)} &= \boldsymbol{x},\end{aligned}$$

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 $\boldsymbol{\sigma} : \mathbb{R}^{d_\ell} \to \mathbb{R}^{d_\ell}$ is a prespecified nonlinear transformation function, called activation function.

The choice of $\boldsymbol{\sigma}$ includes 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 [65] and ELU [23] 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 multi-category 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

$$\boldsymbol{f}(\boldsymbol{x}|\Theta) = \boldsymbol{g} \circ \boldsymbol{h}^{(L)} \circ \boldsymbol{h}^{(L-1)} \circ \cdots \circ \boldsymbol{h}^{(1)}(\boldsymbol{x}),$$

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 each layer. Figure 2 illustrates the structure of a FNN with 2-layers.

For model training, we can estimate Θ by minimizing the empirical loss function. E.g., for a regression problem, the quadratic loss is usually adopted:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{i=1}^n \left(\boldsymbol{f}(\boldsymbol{x}_i|\Theta) - y_i\right)^2,$$

and for a classification problem, the negative log likelihood loss is used:

$$\hat{\Theta} = \arg\min_{\Theta} - \sum_{i=1}^n \sum_{k=1}^K \mathbf{1}\{y_i = k\} \log[\boldsymbol{f}(\boldsymbol{x}|\Theta)]_k.$$

The training of parameter utilizes the back-propagation [83] which computes the derivatives of parameters in each layer via a chain rule. In addition, the depth L and the width of the FNN d_ℓ are the hyperparameters of the model, and are required to be tuned through cross-validation.

Compared with the parametric models, the advantage of FNN are mainly attributed from the nonlinearity induced by the activation function in each layer. Indeed, the universal approximation theorem [24, 47] states that any Borel measurable function can be approximated by a FNN with one hidden layer under the mild conditions on the activation function. Moreover, the derivatives of any Borel measurable function can be approximated by the derivatives of a FNN as well. The universal approximation 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 convolutional neural networks (CNNs) for supervised learning tasks and generative adversarial networks (GANs) for unsupervised learning tasks.

3.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) [57] are specialized feed-forward networks designed for imaging data. It has achieved great success in imaging data analysis including classification, object detection, segmentation, semantic description, etc. Different from the general FNN introduced in Section 2, CNNs use convolution operation instead of matrix multiplication to extract features, which enables CNN to utilize the grid structure of imaging data.

We first introduce the unique operators involved in CNN, i.e., convolution and pooling. Given a matrix $X \in \mathbb{R}^{d_1 \times d_2}$, we define the filter as $K \in \mathbb{R}^{p_1 \times p_2}$, $p_1 \leq d_1, p_2 \leq d_2$. Then the convolution between X and K can be defined elementwise as

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

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

In addition, the max pooling function is defined as

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

The max pooling function replaces the elements of a matrix by the maximum of elements in $p_1 \times p_2$ submatrices. There are other pooling functions such as average pooling and spatial pyramid pooling [42]. These pooling functions avoid overfitting by reducing the number of output parameters. Note that the stride s is not necessary to be equal in convolution and pooling.

The above definitions of convolution and max pooling can be generalized to three dimension, when the image data contains multiple colors or the network uses several filters to capture different features. Let $X \in \mathbb{R}^{c_{in} \times d_1 \times d_2}$ and $K \in \mathbb{R}^{c_{out} \times c_{in} \times p_1 \times p_2}$, then the convolution between X and K leads to a three-dimensional array defined by

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

where c_{in} is the number of matrices of input data and c_{out} is the number of matrices of output data. And the max pooling function can be generalized to

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

Next, we introduce the architecture of CNNs. In general, CNNs consist of a feature extractor and a classifier. The feature extractor is a combination of convolution layers and pooling layers. Denote the ℓ-th hidden layer as $\boldsymbol{h}^{(t)} \in \mathbb{R}^{c_{in}^{(\ell)} \times d_1^{(\ell)} \times d_2^{(\ell)}}$ and filters as $K^{(\ell)}$, then the feature extractor can be defined as

$$\boldsymbol{g}^{(\ell-1)} = \boldsymbol{\sigma}(\boldsymbol{h}^{(\ell-1)} * K^{(\ell-1)}), \quad \boldsymbol{h}^{(\ell)} = \boldsymbol{\pi}(\boldsymbol{g}^{(\ell-1)}), \tag{1}$$

where $\boldsymbol{\sigma}$ is an activation function, selected as ReLU by default. The outputs of the convolution layer $\boldsymbol{g}^{(\ell)}$ are also called features 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 fully-connected layer. Figure 4 illustrate the structure of the CNN with single convolution layer and single fully-connected layer.

The reason why convolution operator benefits the performance of CNN on image data is attributed to two parts. On one hand, the features extracted by CNNs are not location sensitive due to the translation invariance of convolution operator and thus CNNs are able to capture signals regardless of their relative position in the input image. Specifically, for any translation operation $\boldsymbol{\tau}(\cdot)$ and filter K, we have

$$\boldsymbol{\tau}(X * K) = \boldsymbol{\tau}(X) * K.$$

That is, convolution over the translated image is the same as translating the convoluted image. Therefore, the extracted features is insensitive to translation according to (1), and 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 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 CNN in computer vision field, there are also limitations. Although CNN is able to extract features over the entire image, it ignores 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. In addition, CNN performs poorly when the image data is highly heterogeneous across the subjects or the number of training samples are limited [99]. In practice, this problem can be alleviated by data augmentation, i.e., enlarging the training dataset manually through zooming, rotating and cropping. Nonetheless, CNNs are still not able to recognize the variations of images which are not presented in the augmented data. Finally, CNN is vulnerable to adversarial attacks which are specifically designed noise added to training data. Some of these attacks are imperceptible to human beings [39] or only minor changes of lighting and orientation settings [55], but could result in poor performance of a fined-trained CNN. How to defend CNNs from adversarial attacks remains an open problem.

3.2 Generative Adversarial Networks

Generative Adversarial Networks (GAN) is an unsupervised learning method that estimates the density of population distribution. It has shown great capability to sample from complex distributions and achieves great success in image-related tasks such as style transfer [51], image synthesis [104] and object detection [28]. The GAN is also applied to other fields such as information retrieval [109], network embedding [36] andmalware detection [48].

Given the observed data $\{\boldsymbol{x}_i\}_{1\leq i\leq n} \in \mathbb{R}^p$, 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; while 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, denote the parameters in D and G as θ_D and θ_G, respectively, then GAN solves the following min-max problem

$$\min_{\boldsymbol{\theta}_G} \max_{\boldsymbol{\theta}_D} E_{\boldsymbol{x}\sim P_X} \log\{D(\boldsymbol{x})\} + E_{\boldsymbol{z}\sim P_Z} \log\left(1 - D\{G(\boldsymbol{z})\}\right). \tag{2}$$

Figure 5 illustrates an example of GAN in generating images which resemble the handwritten digits.

In the following, 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 (2) is nothing but minimizing the discrepancy between P_g and P_X. Specifically, by fixing G in (2), the optimal discriminator D is

$$D^*_G(\boldsymbol{x}) = \frac{P_X(\boldsymbol{x})}{P_X(\boldsymbol{x}) + P_g(\boldsymbol{x})}.$$

And (2) becomes:

$$\begin{aligned} &\min_{\theta_G} E_{x\sim P_X}\left(\log\frac{P_X(\boldsymbol{x})}{P_X(\boldsymbol{x}) + P_g(\boldsymbol{x})}\right) + E_{x\sim P_g}\left(\log\frac{P_g(\boldsymbol{x})}{P_X(\boldsymbol{x}) + P_g(\boldsymbol{x})}\right) \\ =&\min_{\theta_G} KL\left(P_X||\frac{P_X + P_g}{2}\right) + KL\left(P_g||\frac{P_X + P_g}{2}\right) - \log 4 \\ =&\min_{\theta_G} 2JSD(P_X||P_g) - \log 4, \end{aligned} \tag{3}$$

where $KL(p||q)$ is the Kullback-Leibler divergence, and JSD is the Jensen-Shannon divergence.

However, Jensen-Shannon divergence may not be suitable for multimodal distributions due to potential missing mode [15, 72], where the 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. Specifically, any f-divergence

$$\mathcal{D}_f(P||Q) = \int_x q(x) f\left(\frac{p(x)}{q(x)}\right) dx \tag{4}$$

can be formulated in training GANs [74]. Many well-known divergence functions fall into the f-divergence family such as KL divergence, squared Hellinger divergence and Pearson χ^2 divergence. The GAN can be applied to improve the sampling methods. Specifically, the GAN is useful in to sampling from a complex distribution since GAN has fewer convergence issues in practice [85], and is able to generate samples instantaneously [116] compared with the MCMC sampling strategies. Another advantage is the flexibility of GAN due to its nonparametric nature of G and D. This property can be used to enhance the statistical functions derived from the MCMC sampling to achieve a higher convergence rate, especially when the target distribution contains multiple modes [96].

In addition, works have been done on approximation efficiency analysis regarding the convergence rate of the GAN towards the true distribution P_X [61]. Since P_X is not observed directly, the GAN implicitly uses the empirical density of the data P_{data} based on n i.i.d. 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.

In the original framework [38], P_X is approximated by

$$\frac{dP_{data}}{d\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n}\delta_{X_i}(\boldsymbol{x}),$$

where δ_{X_i} is the delta function defined as $\delta_{X_i} = +\infty$ if $\boldsymbol{x} = X_i$, and 0 otherwise. However, when the P_X is known to be smooth, one can use a kernel density approximation instead:

$$\frac{dP_{data}}{d\boldsymbol{x}} = \frac{1}{nh}\sum_{i=1}^{n} K\left\{\left(X_i - \boldsymbol{x}\right)/h\right\}, \tag{5}$$

where K is a kernel function and h is the bandwidth. Analogous to the techniques in the nonparametric density estimation, (5) improves the convergence rate of GAN under certain smoothness assumptions.

Another way to improve the convergence rate is to modify the evaluation metric in D. Note the target of the discriminator D can be generalized to

$$d_{\mathcal{F}_D}(P_g, P_X) := \max_{f\in\mathcal{F}_D}\left\{E_{Y\sim P_g} f(Y) - E_{X\sim P_X} f(X)\right\}, \tag{6}$$

where $\mathcal{F}_D$ is the function class of the discriminator. The original GAN makes no assumptions about $\mathcal{F}_D$. However, one can specify $\mathcal{F}_D$ based on the prior information of P_X to shrink the function space, 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 Wasserstein-1 metric, and leads to Wasserstein-GAN [5], which provides lower approximation error to a smooth function compared with the original GAN.

One important advantage of the GAN is the high capability to learn nonparametric density functions which are difficult to be fully captured using parametric generative models such as Gaussian mixture model or Markov random field. Specifically, since 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, the conventional generative models use variational inference or impose additional assumptions on density functions, such as the factorial assumption in 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 due to its representation power of the 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 [15, 68] and the difficulty to measure estimation uncertainty. The GAN also has limitations in generating discrete data distribution since the 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 recurrent neural network (RNN) method and its variant Long-Short Term Memory (LSTM) [46].

RNNs are designed to process time series data and sequence data, for example stock price prediction, speech recognition, machine translation, genome sequencing. RNNs are used for sequence data because they utilize historical data to capture sequential patterns.

4.1 Recurrent Neural Networks

The RNN is first developed in the 1980s [83, 111, 30] and used to capture patterns developing through time. Suppose we have historical sequence data $\boldsymbol{y} = \{y_1, y_2, \ldots, y_T\}$ and $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}$, 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

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

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 functions called activation function, $\mathbf{W}_{hh}$, $\mathbf{W}_{xh}$ and $\mathbf{W}_{hz}$ are weight matrices, and $\boldsymbol{b}_h$ and $\boldsymbol{b}_z$ are bias vectors. Different from FNN and CNN, 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 $\boldsymbol{h}_{t-1}, \boldsymbol{h}_{t-2}, \ldots, \boldsymbol{h}_0$ and incorporate update from the current observation $\boldsymbol{x}_t$.

Figure 6 shows the structure of a simple RNN with one input unit, one output unit, and one hidden unit, where $\boldsymbol{x}_t$ is the input, layer z_t is the output and $\boldsymbol{h}_t$ is the hidden unit 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 $\{\mathbf{W}_{hh}, \mathbf{W}_{xh}, \mathbf{W}_{hz}, \boldsymbol{b}_h, \boldsymbol{b}_z\}$. During the training process, 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 the 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 [46] proposed long short term memory (LSTM) networks to handle the gradient vanishing problem. LSTM has attracted great attention for time series 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 $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$ for $\boldsymbol{x}_t \in \mathbb{R}^p$, an LSTM unit with a forget gate is composed as follows:

$$\begin{aligned} \boldsymbol{f}_t &= \boldsymbol{\sigma}_g(\mathbf{W}_{xf}\boldsymbol{x}_t + \mathbf{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f), & \boldsymbol{i}_t &= \boldsymbol{\sigma}_g(\mathbf{W}_{xi}\boldsymbol{x}_t + \mathbf{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i), \\ \boldsymbol{g}_t &= tanh(\mathbf{W}_{xc}\boldsymbol{x}_t + \mathbf{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c), & \boldsymbol{o}_t &= \boldsymbol{\sigma}_g(\mathbf{W}_{xo}\boldsymbol{x}_t + \mathbf{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o), \\ \boldsymbol{c}_t &= \boldsymbol{f}_t \circ \mathbf{c}_{t-1} + \boldsymbol{i}_t \circ \boldsymbol{g}_t, & \boldsymbol{h}_t &= \boldsymbol{o}_t \circ tanh(\mathbf{c}_t), z_t = \boldsymbol{\sigma}_z(\mathbf{W}_{hz}\boldsymbol{h}_t + \boldsymbol{b}_z), \end{aligned} \tag{7}$$

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, $\boldsymbol{c}_t \in \mathbb{R}^d$ is a cell state vector, $\mathbf{W}_{xf}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{xi}$, $\mathbf{W}_{hi}$, $\mathbf{W}_{xc}$, $\mathbf{W}_{hc}$, $\mathbf{W}_{xo}$, and $\mathbf{W}_{ho}$ are weight matrices, $\boldsymbol{b}_f$, $\boldsymbol{b}_i$, $\boldsymbol{b}_c$, and $\boldsymbol{b}_o$ are d-dimensional bias vectors. Here, z_t is a output, $\mathbf{W}_{hz}$ and $\boldsymbol{b}_z$ are weight and bias, respectively. The $\boldsymbol{\sigma}_g$ and $\boldsymbol{\sigma}_z$ are the activation functions, and $tanh(x)$ is a function as follows

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

The core of LSTM is a memory unit or cell $\mathbf{c}_t$, which encodes the historical information of the inputs. Compared with hidden layer in normal RNN, 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. Since 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 behavior of time series using the compositional function class like (7). This is a sharp difference from traditional statistical methods such as linear models (e.g., autoregressive integrated moving average) and non-linear model (e.g., autoregressive conditional heteroskedasticity). Many traditional statistical methods predefine a model based on some prior information. Although traditional methods have proven to be quite effective in many circumstances, identifying a model that is broadly applicable has been difficult [64]. Deep learning methods do not involve human knowledge and make no claims about the generation process. Moreover, the performance of traditional statistic methods mainly depends on the properties of target functions and their related optimization are less complicated, while the statistic performance of deep learning methods depends heavily on the optimization algorithms, which is very different from many classical statistical methods.

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 is the number of users, and m is the number of items. 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 a extremely large and sparse utility matrix. Therefore, the key of recommender systems is to predict the rating scores of a user based on observed scores, and make recommendation based on predicted rating scores. Among many existing recommender systems, deep learning-based recommender systems [87, 27, 59, 8] are promising.

Recent advances in deep learning-based recommender systems have gained significant attention by improving conventional models and achieving high recommendation accracy. These includes multilayer perception (MLP) [60, 2], autoencoder (AE) [27, 59], convolutional neural network (CNN) [20, 29], recurrent neural network (RNN) [8], restricted boltzmann machine (RBM)-based recommender system [87] and deep hybrid models. The advantages of these approaches can summarised as follows. The MLP is capable of modeling the nonlinear interactions between users and items. The CNN has advantages of extracting local and global representations from heterogeneous data sources. The RNNs model the temporal dynamics and sequential evolution of content information. In the following we mainly focus on restricted Boltzmann machine-based recommendation and autoencoder-based recommender systems.

5.1 Restricted Boltzmann Machine

A restricted Boltzmann machine (RBM, [95, 34, 44]) is to use unobserved hidden layers to explain observed visible layer. The RBM has wide applications in classification, feature learning, topic modelling and collaborative filtering. The standard type of RBM has binary-valued hidden layer $\boldsymbol{h} \in \mathbb{R}^d$ and visible layer $\boldsymbol{x} \in \mathbb{R}^p$, which are connected via an energy function of $(\boldsymbol{x}, \boldsymbol{h})$ as follows

$$E(\boldsymbol{x}, \boldsymbol{h}) = -\boldsymbol{a}^\top \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{h} - \boldsymbol{x}^\top \mathbf{W} \boldsymbol{h},$$

where $\mathbf{W} = (w_{ij}) \in \mathbb{R}^{p\times d}$ 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

$$P(\boldsymbol{x}, \boldsymbol{h}) = \exp\{-E(\boldsymbol{x}, \boldsymbol{h})\}/Z,$$

where Z is a partition function defined as the sum of $\exp\{-E(\boldsymbol{x}, \boldsymbol{h})\}$ over all possible pairs of vectors $(\boldsymbol{x}, \boldsymbol{h})$.

Since the RBM is a special of a bipartite graph with no intra-layer 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 probability of the units $\boldsymbol{x}$ given the hidden units $\boldsymbol{h}$ and the conditional probability of $\boldsymbol{h}$ given $\boldsymbol{x}$ are

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

The individual activation probabilities are given by

$$P(h_j = 1|\boldsymbol{x}) = \sigma(b_j + \sum_{i=1}^{p} w_{ij}x_i) \text{ and } P(x_i = 1|\boldsymbol{h}) = \sigma(a_i + \sum_{j=1}^{d} w_{ij}h_j),$$

where σ denotes the logistic sigmoid. The visible units of RBM can be multinomial although the hidden units are Bernoulli. In this case, the individual probability of the visible unit is given by

$$P(x_i^k = 1|\boldsymbol{h}) = \frac{\exp(a_i^k + \sum_{j=1}^{d} w_{ij}^k h_j)}{\sum_{t=1}^{K} \exp(a_i^t + \sum_{j=1}^{d} w_{ij}^t h_j)}, \tag{8}$$

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 a RBM.

The RBM was first applied to recommender systems in 2007 [87]. Salakhutdinov [87] first proposed a restricted Boltzmann machine-based recommender systems for tackling the Netflix challenge. For the restricted Boltzmann machine-based recommender systems, the rating score is represented in a vector of dummy variables to represent binary-valued visible unit. 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 $\mathbf{X}$ be a $K \times p$ matrix of the dummy variables of the rating vector $\boldsymbol{r}$, where $x_i^k = 1$ if the user rated the movie i as score k and $x_i^k = 0$ otherwise. We can estimate the parameters of the RBM via the Contrastive Divergence algorithm [37]. The above restricted Boltzmann machine based-recommender systems is user-based where a given user's rating is observed, and are input to the visible layer. An 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 tool for automatically extracting nonlinear features [16]. The autoencoder can be viewed as a dimension reduction technique which reduces the dimensionality of the data and preserves the salient features of the data at the same time.

A basic autoencoder consists of input layer, hidden layer and output layer, where the numbers of neurons are p, k and p, respectively. Figure 9 provides an illustration. The 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 $\boldsymbol{f}$:

$$\boldsymbol{h} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{\sigma}_f(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}),$$

where $\boldsymbol{\sigma}_f$ is an activation function. Parameters in the encoder include a $k \times p$ weight matrix $\mathbf{W}$, and a bias vector $\boldsymbol{b} \in \mathbb{R}^k$. The input layer and the hidden layer construct an encoder. The decoder maps hidden representation $\boldsymbol{h}$ back to a reconstructed $\boldsymbol{x}' = \{x_1', x_2', \ldots, x_p'\}$ by a function $\boldsymbol{g}$:

$$\boldsymbol{x}' = \boldsymbol{g}(\boldsymbol{h}) = \boldsymbol{\sigma}_g(\mathbf{W}'\boldsymbol{h} + \boldsymbol{b}'),$$

where $\boldsymbol{\sigma}_g$ represents the decoder's activation function. The decoder's parameters consist of a $p \times k$ weight matrix $\mathbf{W}'$, and a bias vector $\boldsymbol{b}' \in \mathbb{R}^p$. The hidden layer and the output layer construct a decoder. The function $\boldsymbol{\sigma}_f$ and $\boldsymbol{\sigma}_g$ are usually non-linear activation function [120]. A basic autoencoder has two key components, namely, the encoder function $\boldsymbol{f}$, which maps the input to a hidden representation, and the

decoder function $\boldsymbol{g}$, which maps the hidden representation to a reconstruction. Both functions can be multilayer neural networks.

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 encoder $\boldsymbol{f}$ and decoder $\boldsymbol{g}$ 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 autoencoder. The loss function can be optimized by stochastic gradient descent or alternative least squares. In recent years, there are many variants of autoencoder used in recommender systems such as denoising autoencoder [106], stack denoising autoencoder [107], and variational autoencoder [53].

In the autoencoder-based recommender systems framework, autoencoder is used to extract the 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 $\mathbf{R} \in \mathbb{R}^{n \times m}$, each item can be represented as a vector $\boldsymbol{r}_i = (R_{1i}, R_{2i}, \ldots, R_{ni})^\top$ for $i = 1, 2, \ldots, m$. An item-based autoencoder can take each partially observed $\boldsymbol{r}_i$ 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

$$\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{r} \in \Omega} \|\boldsymbol{r} - \boldsymbol{g}(\boldsymbol{f}(\boldsymbol{r}, \boldsymbol{\theta}))\|^2 + \frac{\lambda}{2}(\|\mathbf{W}\|_F^2 + \|\mathbf{W}'\|_F^2),$$

where Ω is the set of observed ratings, the parameter $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{W}', \boldsymbol{b}, \boldsymbol{b}'\}$, λ is the regularization parameter, and $\|\cdot\|_F^2$ is Frobenius norm of a matrix.

The autoencoders and RBMs achieve similar goals, such as reducing dimensionality or extracting features from signals. Although their structures appear to be similar, they rely on different training theories. The RBM uses stochastic neural network with a particular distribution instead of deterministic distribution. The main task of training RBM is to obtain connectability between two sets of input variables and hidden variables. There are no intra-layer communications in visible layer or hidden layer. In contrast, for the autoencoders, the main task of training is to minimize an error in reconstruction, i.e. find the most efficient compact representation for 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 input data under the another space.

Obviously, the autoencoders resemble the 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 input data, and represents data using fewer dimensions than the initial data. However, there are many differences between the autoencoders and the PCA. First, while the PCA is a linear transformation, the autoencoders allow non-linear transformation, which makes the autoencoders more flexible and powerful. Unfortunately, the latent space found by the autoencoders lacks the interpretability [56]. 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 make the autoencoders 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 detail discussions regarding the relationship between the autoencoders and PCA, we refer the readers to [56, 3].

6 Optimization

In this section, we introduce some important optimization algorithms for deep hierarchical structures. Due to the highly non-convex nature of deep architectures, training deep learning models is challenging. We apply the gradient-descent type algorithms in deep learning optimization since 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 gradient descent is:

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

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

It is often common to utilize part of training data to update parameter in each step to achieve a balance between accuracy of update and computational cost. Based on the quantity of training data used for updating gradient, the gradient descent has two main variations. The first one is stochastic gradient descent (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 parameter 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 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 training dataset. This is much faster than the original gradient descent, and more stable than the SGD. However, it requires to determine a mini-batch size as an additional tuning parameter.

The main challenge for the original gradient descent is that it does not guarantee a good convergence. In addition, since the estimation can be easily trapped at the suboptimal local minima or saddle point, 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. In addition, the learning rate should not be 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) [70] and adaptive moment estimation (ADAM) [52].

The main idea of Nesterov Accelerated Gradient (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:

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

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 gradient at current step. Coefficients $\alpha \in [0, 1]$, $\beta \in [0, 1]$ are weights on the correction term and memory term respectively, and v_k is the velocity update. The NAG degenerates to vanilla gradient descent when $\beta = 0$. In general, the NAG speeds up 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 adaptive moment estimation (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:

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

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 $E(m_k) \approx (1-\beta_2^k)E(g_k^2)$:

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

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 current 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 different parameters updating, which is useful when data is sparse. In addition, it accommodates features with different frequencies and performs a large update for rarely occurring features, and a small update for the frequent ones. However, some cautions might be taken into account. Empirical study [112] 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 selecting a batch size, filter size, the number of layers and the number of nodes of each layer. While some of the hyperparameters can be adjusted adaptively during training such as the learning rate of gradient descent algorithms, most of these hyperparameters need to be predetermined. In contrast to the AIC or BIC criteria 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 based on the performance on 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 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 [11], which is more computational efficient than a grid search. Another method is the sequential model-based optimization [12] 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 which might not increase the model accuracy, and therefore is more efficient than the grid search or random search.

7 Implementation: Image Classification

In this section, we implement CNN in image classification to explain the rationales behind the CNN, and also demonstrate illustrate the robustness issue arising from the CNN

In the first implementation, we use the pre-trained VGG-16 [94] for demonstration. TheVGG-16 is a deep CNN trained on ImageNet and provides a 92.7% top-5 class test accuracy in classifying 1000 image categories. The network consists of 13 convolutional layers, 3 fully-connected layers and 5 max-pooling layers with 138 million parameters. This deep network receives colored images of size $224 \times 224 \times 3$ as input, and provides the classification probability of each category as output for each image. Figure 11 shows the architecture of the VGG-16 and the size of each layer.

In the first two examples, we input the images of a sorrel horse and a panda, respectively. The VGG-16 correctly classifies the two images with a probability almost equal to 1. In addition, we use the Grad-CAM [90] to highlight the most salient pixels associated with the classification to provide a visual explanation. The heatmaps in Figure 12 generated by the Grad-CAM indicate that VGG-16 captures the entire shape of the horse and a region of panda's face successfully. For another example, we input an image which contains a tiger and a goat in the same image. As a result, the neural network classifies the image as a "tiger" by correctly identifying the stripes from the image. Although the CNN misclassifies the goat as a lion with a small probability ($p = 0.03$), it correctly identifies the position of the goat. These two examples demonstrate the interpretability of the features extracted by CNN and illustrates the similar mechanism between the CNN and the human vision.

For the second implementation, we demonstrate the robustness issue of the CNN through two handwritten digit datasets. The translated MNIST dataset [58] randomly translates the 28×28 handwritten digits on a black background of 40×40 pixels. And the affNIST dataset [102] adjusts the digit images through more complex affine transformation other than translation through rotation, scaling and shearing. Figure 14 provides examples from the translated MNIST dataset and affNIST dataset. We first train a two-layer CNN with 50,000 samples from the translated MNIST dataset and then test on 10,000 samples from the translated MNIST dataset and affNIST dataset, respectively.

The trained model achieves an accuracy of 98.81% on the training set and 98.26% on the translated MNIST testing set. However, it only results 65.97% accuracy on the affNIST testing set, which implies that CNNs are not robust against affine transformation. This phenomenon suggests that the CNNs may fail when the input data is highly heterogeneous. To address this issue, one can apply data augmentation [91] to increase the training set by adding more images with small distortions to the original images. However, data augmentation increases significantly on training cost and could still fail when the types of image transformation are not included in the training set.

8 Connection Between Deep Learning Models and Statistical Models

In this section, we connect the principles of deep learning to statistical concepts and modeling. 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 its 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, the deep learning models are capable of representing the highly non-linear relations between input data and output data. This is achieved through multiple layers of composition functions for latent features from input data. Furthermore, the deep architecture brings some new insights about bias-variance trade-off principle under the overparameterization regime, which also motivates us to develop new learning theories and large-scale non-convex optimization algorithms.

8.1 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 $\boldsymbol{h}^{(\ell)} = \boldsymbol{\sigma}(W^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)})$ 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 FNN on feature selections for complex data, and enables approximation of complex non-linear functions.

For deep generative models, it also connects to the unsupervised learning models. For example, the autoencoder connects with the latent factor model in statistics. The autoencoder consists of an encoder and a decoder, where the encoder estimates the latent factor through a pre-specified 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 15 illustrates the encoding and decoding processes. Specifically, autoencoder aims to capture the hidden data generation process by optimizing

$$\mathcal{L}(\boldsymbol{Z_f}) = \log p(\boldsymbol{Y}|\boldsymbol{Z_f}) - \lambda R(\boldsymbol{Y}, \boldsymbol{Y'}),$$

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 principal component analysis (PCA), project pursuit regression, and kernel methods can be regarded as shallow models compared to the deep model as they follow a two-layer framework that consists of a data-transformation layer and a reconstruction layer [78]. Specifically, consider a training dataset (X, Y) and a two-layer model:

$$\boldsymbol{Z} = f_1(W^{(1)}X + b^{(1)}) \text{ and } E(Y) \approx f_2(W^{(2)}Z + b^{(2)}), \tag{9}$$

where $W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}$ are corresponding weights and biases defined in Section 2. For aforementioned shallow models, the information in 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 (9). For example, given that f_1 is the identity function, and f_2 is a non-linear smooth function, then (9) 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 (9) is equivalent to the SVM under the hinge loss. On the other hand, if f_1 and f_2 are non-linear functions such as logistic function, then (9) is a two-layer feed-forward neural network. Therefore, the deep learning model generalizes a two-layer shallow model to hierarchical multiple-layer model.

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

$$p\left(x_1, \ldots x_T\right) = \prod_{t=1}^{T} \int p\left(x_t|h_t\right) p\left(h_t|h_{t-1}\right) dh_t. \tag{10}$$

The RNN is a special case of the probabilistic modeling in (10) in that the transition of hidden states is assumed to be deterministic, i.e., $p\left(h_t|h_{t-1}\right) = f_\theta(h_{t-1}, x_{t-1})$, where $f_\theta(\cdot)$ is the hidden unit in the RNN. Accordingly, training the RNN model is equivalent to maximizing the log-likelihood function $\log p\left(x_1, \ldots x_T\right)$ [37].

8.2 Beyond Statistical Model: Overparameterization

One significant difference between deep learning and statistical modeling is the understanding of overparameterization. The statistical principle suggests that increasing the complexity of model tends to cause overfitting and being less robust on future prediction. On the contrary, the deep learning is able to decrease the generalization error with increasing number of parameters. Specifically, Figure 16 illustrates the overparameterization 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 [118] 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 labeled, 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 [9, 10] explain that the vanishing of training error does not necessarily diminish generalization power on test data when the number of parameters exceeds the size of 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 rational behind the overparameterization in deep learning model from learning theory and optimization algorithm perspective.

Overparameterization via complexity measurement

Recent works [6, 62] 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 are established. For example, results in [71] indicate that the traditional model complexity measurements such as the number of parameters or Vapnik-Chervonenkis dimension are not applicable to the deep learning models, thus are not suitable for overparameterization. Instead, a complexity measurement for neural networks is introduced. Based on 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 output layer respectively. Let $L_0(f)$ and $\hat{L}_0(f)$ be the expected risk and the empirical risk. [71] shows that

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

where m is the size of 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.

Overparameterization via: implicit regularization in optimization

The power of generalization for a deep learning model is also determined by optimization algorithms. Empirical studies [119, 118] show that gradient descent algorithms improve generalization error. This is due to that the gradient descent algorithms introduces an implicit regularization controlled by the number of iterations [80]. In addition, an estimation from gradient descent asymptotically converges to a solution with a minimal norm with an appropriate initial value [118, 77].

We use a linear regression model to demonstrate the intuition. Suppose we consider a linear estimation problem on a training dataset $\{(x_i, y_i)\}_{i=1}^n$:

$$\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \text{loss}\left(w^T x_i, y_i\right), \tag{11}$$

where w is the weighting parameter. If we estimate w through the gradient descent algorithm $w_{t+1} = w_t - \eta e_t \frac{\sum x_i}{n}$ starting from an initial $w_0 = 0$, where η is the learning rate and e_t is the loss function error, then the solution satisfying (11) 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 $Xw = Y$. Under the classification setting, the minimum norm

solution in (11) is known to maximize a margin in classification [77]. 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 hard margin SVM [119, 77]. In summary, the implicit regularization property of gradient descent stands for a broad class of regression loss and classification loss functions.

9 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 which can be applied to train deep learning networks, for example, Tensorflow[1] by Google, Pytorch [76] by Facebook and Cognitive Toolkit (CNTK) [89] by Microsoft. Among these softwares, Tensorflow is widely implemented in industry since it facilitates an easy and fast deployment of the trained deep learning model for production. Tensorflow supports multiple programming languages such as Python, C and Java, under different platforms including mobile phones. 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 research community due to its simplicity to build a deep learning model. It is designed for flexible experimentation and customization, and can accommodate strong GPU acceleration. Another advantage of the Pytorch is its support for dynamic computing graphs, which greatly facilitates linguistic analyses. For R users, Keras [19] offers a user-friendly interface which enables quickly prototyping deep learning models. Keras supports convolutional networks, recurrent networks, and combinations of both 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 due to the complex multilayer structures and high volume of parameters. Nowadays, there are mainly three types of processing units: central processing unit (CPU), graphics processing unit (GPU) and tensor processing unit (TPU). In general, the GPUs can accelerate the training by 100 times compared with the CPUs due to its multi-core architect which facilitates parallel computing. The exact acceleration rate depends on the device, dataset, model structure and frameworks. The popular frameworks , such as Tensorflow, have integrated the GPU computing, yet most of these frameworks requires CUDA [73] library by Nvidia. On the other hand, the TPUs are highly optimized for large batches in deep learning. The TPUs can boost the computing speed by up to 20 times compared with the GPUs. One limitation of the TPU is that it currently only supports Tensorflow. Although the CPU has the slowest training speed among all processing units, it has the largest memory per core and thus is capable to implement extremely large models or datasets, while the GPU and the TPU may encounter out-of-memory issues. For more detailed comparison between TPU, GPU and CPUs in deep learning training, we refer the readers to [110].

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 [40], AWS Sagemaker [4] , FloydHub [33] and Azure [69]. These cloud services enable users to run deep learning models faster and cheaper by supplying GPU or TPU computing power remotely.

10 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 principle.

The deep learning models distinguish from standard statistical models in two aspects: extraction of task-relevant features from input data, and the target function for modeling the relations between input data and output data. In deep learning models, the 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 behavior.

In addition, the deep learning models are sequentially composed of multiple non-linear 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 based on its corresponding output value. Therefore, deep learning is able to approximate a 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, the deep learning has its own limitations. First, the deep learning lacks interpretability [63]. That is, the deep learning is mostly black-box function approximators. Nevertheless, there are some recent works on improving the deep learning regarding interpretability [31, 50, 66, 72]. For example, [31] developed visualizing the responses of individual units in any layer of a neural network. [117] and [50] extended the above idea to the CNN and the LSTM, respectively. [66, 115] and [72] investigated the latent features at different CNN levels for better understanding of layer representations in deep models. [92] provided a deep insight for analyzing deep networks using information theory, which applies the information bottleneck framework [103] to calculate information preserved on each layer's inputs and outputs.

Second, the deep learning has the consistency issue, including the consistency of a universal neural network classifier [32], the consistency of training deep neural networks [114], the consistency of using metrics to select the best network [105], and the consistency of neural network detections [113]. In addition, the uncertainties of neural networks such as model uncertainty [13, 35] and data uncertainty are evaluated using the tested algorithms on data with noise or disturbance. Nevertheless, these problems remain open and unsolved.

Acknowledgments

The authors would like to acknowledge support for this project from the National Science Foundation grants DMS-1821198 and DMS-1952406.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Alashkar, T., Jiang, S., Wang, S., and Fu, Y. (2017). Examples-rules guided deep neural network for makeup recommendation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, AAAI17, page 941947. AAAI Press.

[3] Alkhayrat, M., Aljnidi, M., and Aljoumaa, K. (2020). A comparative dimensionality reduction study in telecom customer segmentation using deep learning and PCA. *Journal of Big Data*, 7(9).

[4] Amazon Web Services Inc. Aws sagemaker. `https://aws.amazon.com/sagemaker/`.

[5] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223.

[6] Arora, S., Cohen, N., and Hazan, E. (2018). On the optimization of deep networks: Implicit acceleration by overparameterization. *arXiv preprint arXiv:1802.06509*.

[7] Azzouni, A. and Pujolle, G. (2017). A long short-term memory recurrent neural network framework for network traffic matrix prediction. *arXiv preprint arXiv:1705.05690*.

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

[9] Belkin, M., Hsu, D., Ma, S., and 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.

[10] Belkin, M., Ma, S., and Mandal, S. (2018). To understand deep learning we need to understand kernel learning. *arXiv preprint arXiv:1802.01396*.

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

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

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

[14] Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., Monz, C., Pecina, P., Post, M., Saint-Amand, H., Soricut, R., Specia, L., and Tamchyna, A. s. (2014). Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

[15] Che, T., Li, Y., Jacob, A. P., Bengio, Y., and Li, W. (2016). Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*.

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

[17] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[18] Chollet, F. et al. (2015). Keras. `https://keras.io`.

[19] Chollet, F. et al. (2017). R interface to keras. `https://github.com/rstudio/keras`.

[20] Chu, W.-T. and Tsai, Y.-L. (2017). A hybrid recommendation system considering visual information for predicting favorite restaurants. *World Wide Web*, 20(6):13131331.

[21] CireşAn, D., Meier, U., Masci, J., and Schmidhuber, J. (2012). Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338.

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

[23] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (ELUS). *arXiv preprint arXiv:1511.07289*.

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

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

[26] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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

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

[29] Elkahky, A. M., Song, Y., and 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*, pages 278–288.

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

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

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

[33] Floyd Labs Inc. Flyodhub. `https://www.floydhub.com`.

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

[35] Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge.

[36] Gao, H., Pei, J., and Huang, H. (2019). ProGAN: Network embedding via proximity generative adversarial network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1308–1316.

[37] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, Cambridge, MA, USA. `http://www.deeplearningbook.org`.

[38] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

[39] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

[40] Google LLC. Google Colaboratory. `https://colab.research.google.com`.

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

[42] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916.

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

[44] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):17711800.

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

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

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

[48] Hu, W. and Tan, Y. (2017). Generating adversarial malware examples for black-box attacks based on GAN. *arXiv preprint arXiv:1702.05983*.

[49] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., and Borchers, A. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12.

[50] Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

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

[52] Kingma, D. P. and Ba, J. (2014). ADAM: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[53] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.

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

[55] Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

[56] Ladjal, S., Newson, A., and Pham, C.-H. (2019). A PCA-like autoencoder. *arXiv preprint arXiv:1904.01277*.

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

[58] LeCun, Y., Cortes, C., and Burges, C. (2010). MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*.

[59] Li, S., Kawale, J., and Fu, Y. (2015). Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM 15, page 811820, New York, NY, USA. Association for Computing Machinery.

[60] Liang, D., Zhan, M., and Ellis, D. P. W. (2015). Content-aware collaborative music recommendation using pre-trained neural networks. In *ISMIR*, pages 295–301.

[61] Liang, T. (2018). How well can generative adversarial networks learn densities: A nonparametric view. *arXiv:1712.08244 [cs, math, stat]*. arXiv: 1712.08244.

[62] Liao, Q. and Poggio, T. (2017). Theory II: Landscape of the empirical risk in deep learning. *arXiv preprint arXiv:1703.09833*.

[63] Lipton, Z. C. (2018). The mythos of model interpretability. *Queue*, 16(3):3157.

[64] Lngkvist, M., Karlsson, L., and Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24.

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

[66] Mahendran, A. and Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *CVPR*, pages 5188–5196. IEEE Computer Society.

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

[68] Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2016). Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*.

[69] Microsoft Corporation. Microsoft azure. `https://azure.microsoft.com/en-in/services/machine-learning-studio/`.

[70] 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.

[71] Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., and Srebro, N. (2018). Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*.

[72] Nguyen, A., Yosinski, J., and Clune, J. (2016). Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616*.

[73] Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with CUDA. *Queue*, 6(2):4053.

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

[75] Olabiyi, O., Martinson, E., Chintalapudi, V., and Guo, R. (2017). Driver action prediction using deep (bidirectional) recurrent neural network. *arXiv preprint arXiv:1706.02257*.

[76] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[77] Poggio, T., Kawaguchi, K., Liao, Q., Miranda, B., Rosasco, L., Boix, X., Hidary, J., and Mhaskar, H. (2017). Theory of deep learning III: explaining the non-overfitting puzzle. *arXiv preprint arXiv:1801.00173*.

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

[79] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

[80] Rosasco, L. and Villa, S. (2015). Learning with incremental iterative regularization. In *Advances in Neural Information Processing Systems*, pages 1630–1638.

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

[82] Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc., Buffalo, NY.

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

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

[85] Salakhutdinov, R. (2015). Learning deep generative models. *Annual Review of Statistics and Its Application*, 2(1):361–385.

[86] Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455.

[87] Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML'07, pages 791–798, New York, NY, USA. Association for Computing Machinery.

[88] Sallab, A. E., Abdou, M., Perot, E., and Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76.

[89] Seide, F. and Agarwal, A. (2016). CNTK: Microsofts open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 16, page 2135, New York, NY, USA. Association for Computing Machinery.

[90] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626.

[91] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60.

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

[93] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., and Graepel, T. (2017). Mastering chess and Shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

[94] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[95] Smolensky, P. (1986). *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, page 194281. MIT Press, Cambridge, MA, USA.

[96] Song, J., Zhao, S., and Ermon, S. (2017). A-NICE-MC: Adversarial training for MCMC. In *Advances in Neural Information Processing Systems*, pages 5140–5150.

[97] Srivastava, N. and Salakhutdinov, R. R. (2012). Multimodal learning with deep Boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2222–2230.

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

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

[100] The MathWorks, Inc. (2019). *Matlab deep learning toolbox R2019b*. Natick, Massachusetts, United State.

[101] The Theano Development Team, Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., et al. (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*.

[102] Tieleman, T. The affNIST dataset. http://www.cs.toronto.edu/ tijmen/affNIST.

[103] Tishby, N., Pereira, F. C., and Bialek, W. (1999). The information bottleneck method. In *Proceedings of the 37-th Annual Allerton Conference on Communication, Control and Computing*, pages 368–377.

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

[105] Twomey, J. M. and Smith, A. E. (1995). Performance measures, consistency, and power for artificial neural network models. *Mathematical and Computer modelling*, 21(1):243258.

[106] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML'08, pages 1096–1103, New York, NY, USA. Association for Computing Machinery.

[107] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and 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.

[108] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., and Schrittwieser, J. (2017). Starcraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*.

[109] Wang, J., Yu, L., Zhang, W., Gong, Y., Xu, Y., Wang, B., Zhang, P., and Zhang, D. (2017). IRGAN: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 515–524.

[110] Wang, Y. E., Wei, G.-Y., and Brooks, D. (2019). Benchmarking TPU, GPU, and CPU platforms for deep learning. *arXiv:1907.10701 [cs, stat]*. arXiv: 1907.10701.

[111] Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356.

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

[113] Yang, T., Gregg, R. E., and Babaeizadeh, S. (2020). Detection of strict left bundle branch block by neural network and a method to test detection consistency. *Physiological Measurement*, 41(2):025005.

[114] Ye, C., Yang, Y., Fermuller, C., and Aloimonos, Y. (2017). On the importance of consistency in training deep neural networks. *arXiv preprint arXiv:1708.00631*.

[115] Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*.

[116] Yuan, G., Yuling, J., Yang, W., Yao, W., Can, Y., and Shunkang, Z. (2019). Deep generative learning via variational gradient flow. *arXiv preprint arXiv:1901.08469*.

[117] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014*, pages 818–833, Cham. Springer International Publishing.

[118] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

[119] Zhang, C., Liao, Q., Rakhlin, A., Miranda, B., Golowich, N., and Poggio, T. (2018). Theory of deep learning IIb: Optimization properties of SGD. *arXiv preprint arXiv:1801.02254*.

[120] Zhang, S., Yao, L., and 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, pages 957–960, New York, NY, USA. Association for Computing Machinery.

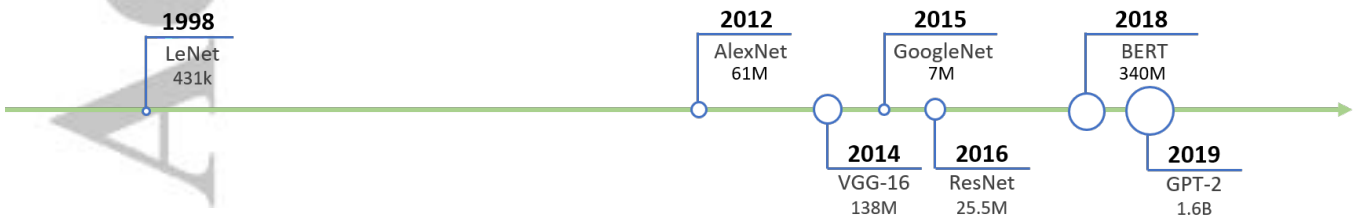

Figure 1: Size evolution of deep neural networks, showing the names of deep neural networks along with the year and the total number of total parameters.

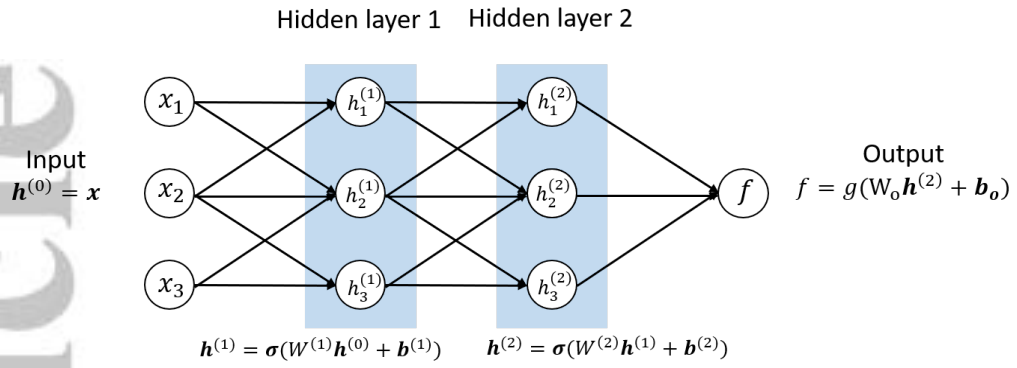

Figure 2: General structure of a 2-layer FNN with a scalar output. Each circle represents an element of vectors of input $\boldsymbol{x}$ and hidden units $\boldsymbol{h}^{(\ell)}$, where $\ell = 1$ and 2.

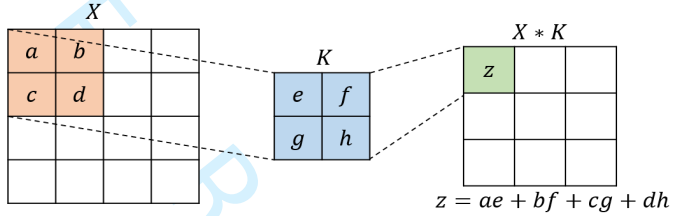

Figure 3: Illustration of convolution operator between 4×4 input data and a 2×2 filter with stride equal 1. The output is of size 3×3.

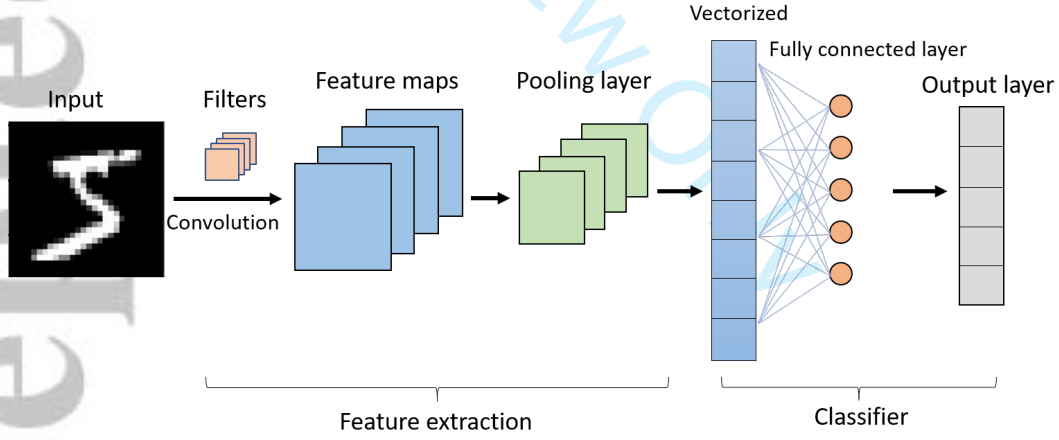

Figure 4: Illustration of a CNN with single convolution layer, single pooling layer and single fully connected layer.

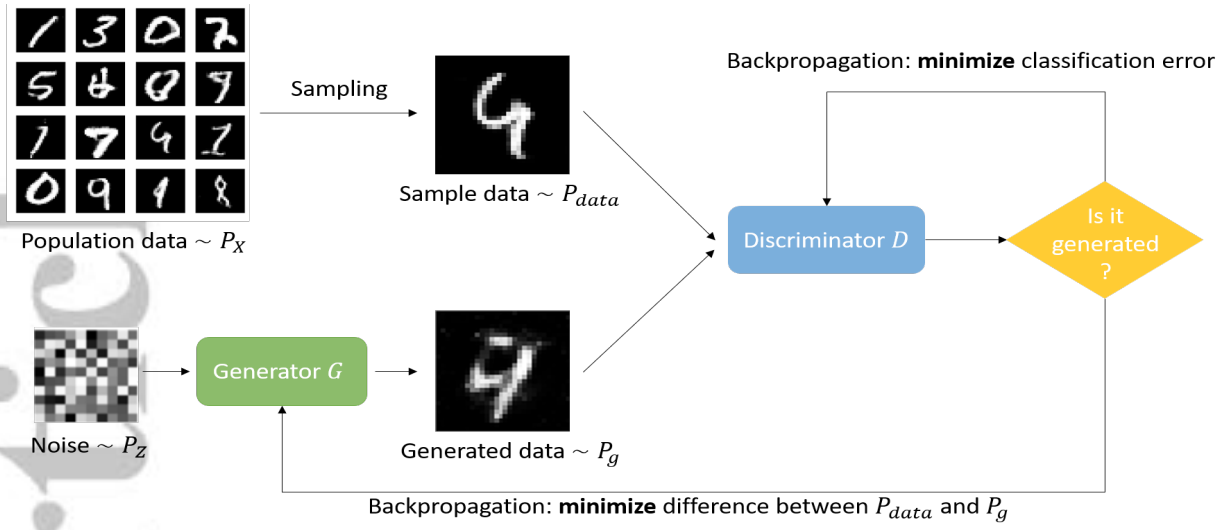

Figure 5: Architecture of GAN to generate handwritten digits. The discriminator D determines whether the input data is sampled from population or generated by G. The generator G aims to minimize difference between P_X and P_g. Both D and G are multilayer networks.

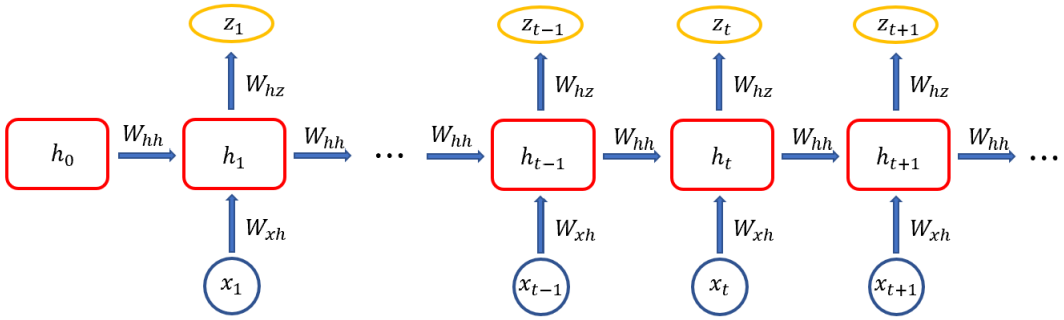

Figure 6: A simple RNN with one input unit, one output unit, and one hidden unit

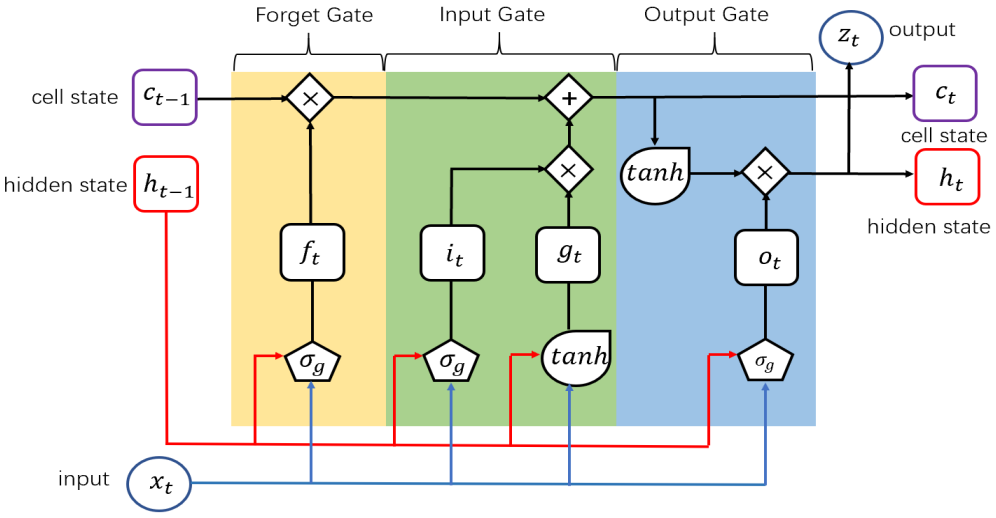

Figure 7: Typical structure of a cell of a single layer LSTM

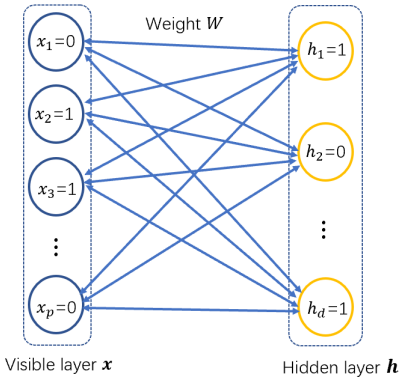

Figure 8: Graphical representation of a restricted Boltzmann machine.

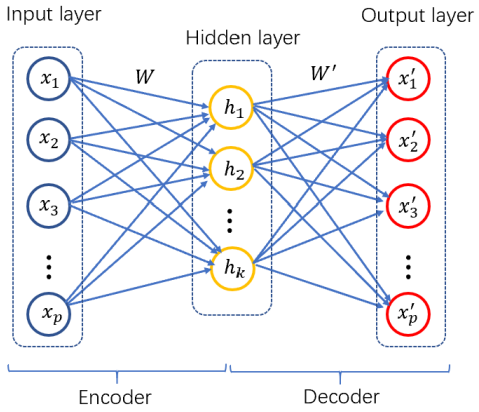

Figure 9: Graphical representation of a basic autoencoder

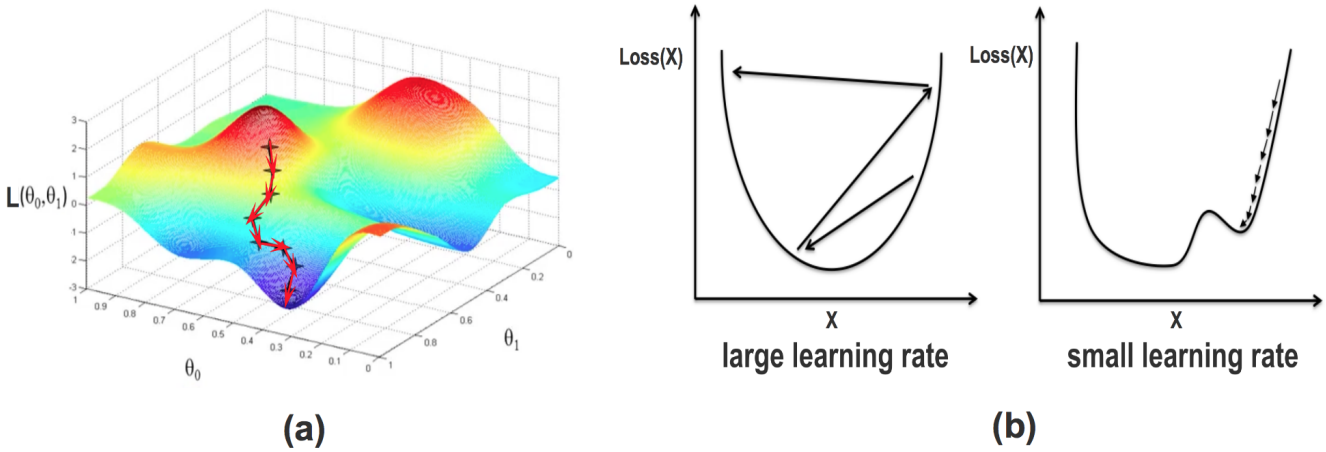

Figure 10: (a): Find minimization through gradient descent. (b): Illustration showing that too large learning rate leads to missing optimization while too small learning rate could lead to trapping at local minimum.

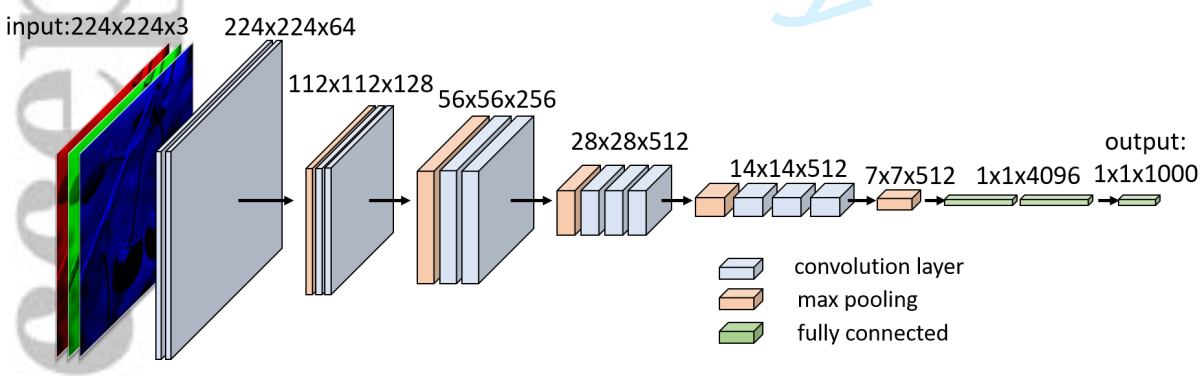

Figure 11: Architecture of the VGG-16. The numbers denote the size of each layer.

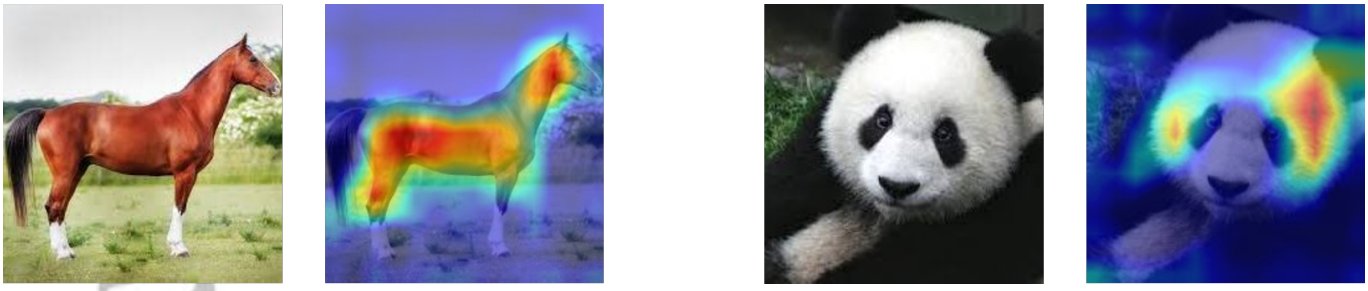

Figure 12: Classification results of a horse and a panda, each pair of images includes input image (*left*) and the corresponding heatmap (*right*) indicating the location of important pixels for classification. The warmer color in the heatmaps indicates higher pixel importance.

Figure 13: An image of a tiger and a goat is classified as a "tiger"($p = 0.71$) and a "lion"($p = 0.047$), showing the input image (*left*), the heatmap for "tiger" (*middle*) and the heatmap for "lion" (*right*)

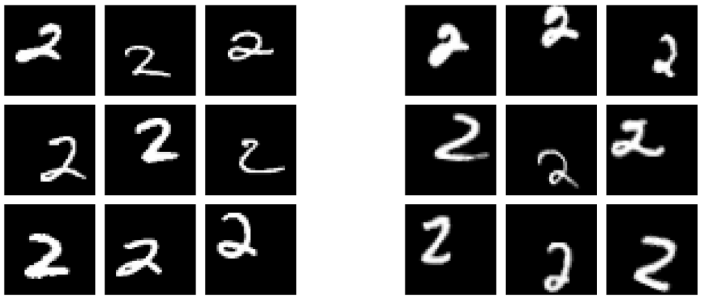

Figure 14: Samples of digit 2 from the translated MNIST dataset (*left*) and affNIST dataset (*right*). The translated MNIST applies only translation on MNIST images, and affNIST applies more complex affine transformation, including rotation, scaling and shearing.

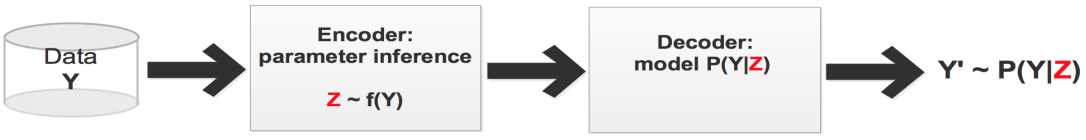

Figure 15: A deep generative model as a latent factor model.

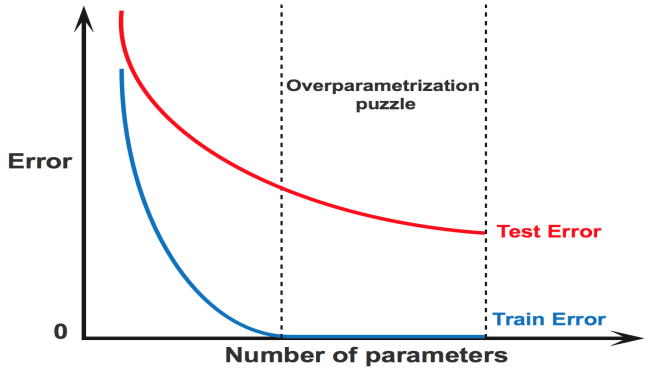

Figure 16: Overparameterization on deep learning: test error keeps decreasing after train error vanishes.

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)	$\max(cx,x)(c>0)$	$\begin{cases} x & x \geq 0 \\ \alpha\left(e^{x}-1\right) & x<0 \end{cases}$

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
Nesterov Accelerated Gradient (NAG)	update gradient with adding previous gradients	speed up convergence; avoid saddle points
AdaGrad/AdaDelta	large update on infrequently updated directions small update on frequently updated directions	handle sparse and unbalanced data avoid overfitting
ADAM	combine AdaDelta and NAG	recommended optimizer for deep learning

Table 3: Summary of deep learning softwares

Software	Platforms	Language	Author
Tensorflow	Linux, macOS, Windows, Android	Python, C/C++, R, Java	[1]
Pytorch	Linux, macOS, Windows, Android (beta)	Python, C++	[76]
CNTK	Linux, Windows	Python, C++	[89]
Theano	Linux, macOS, Windows	Python	[101]
Keras	Linux, macOS, Windows	Python, R	[18]
Matlab Deep learning toolbox	Linux, macOS, Windows	Matlab	[100]

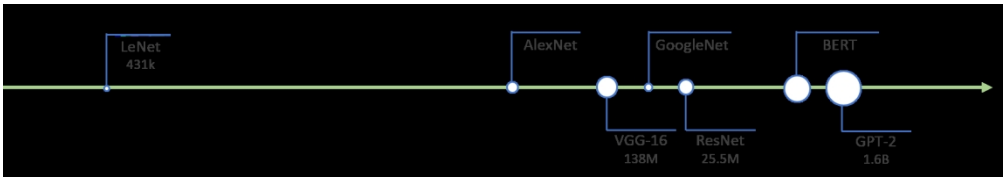

Figure 1: Size evolution of deep neural networks, showing the names of deep neural networks along with the year and the total number of total parameters.

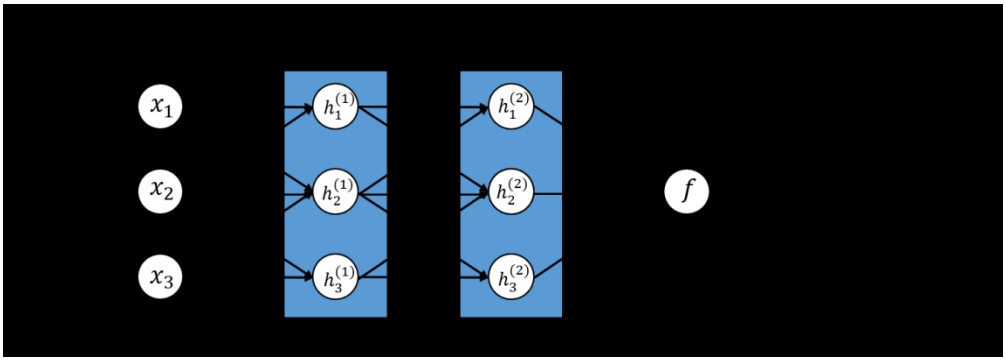

Figure 2: General structure of a 2-layer FNN with a scalar output. Each circle represents an element of vectors of input x and hidden units h (l) , where l = 1 and 2.

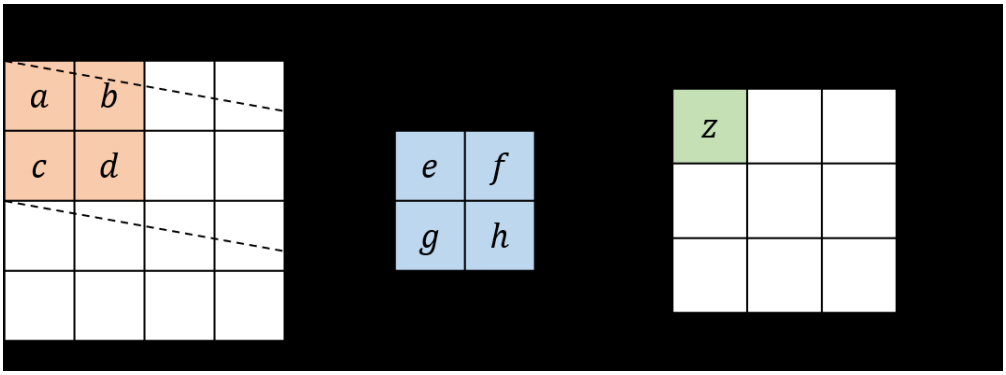

Figure 3: Illustration of convolution operator between 4 × 4 input data and a 2 × 2 filter with stride equal 1. The output is of size 3 × 3.

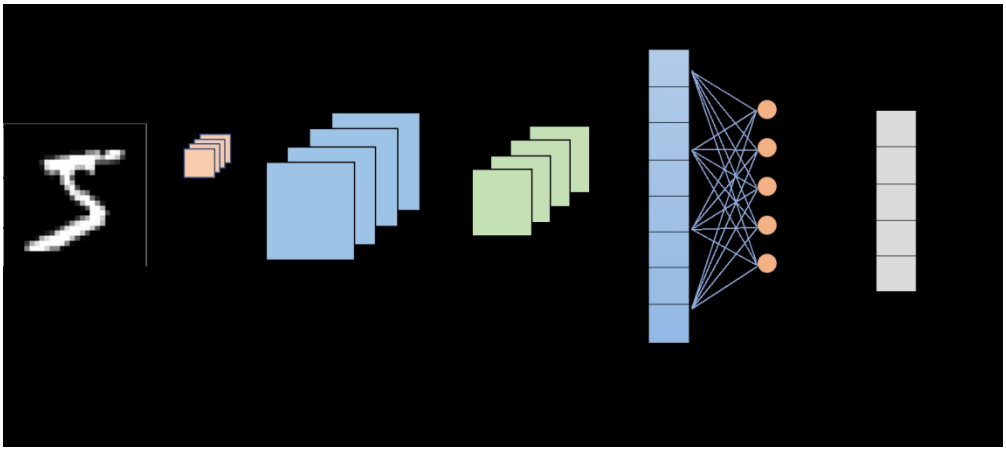

Figure 4: Illustration of a CNN with single convolution layer, single pooling layer and single fully connected layer.

Figure 5: Architecture of GAN to generate handwritten digits. The discriminator D determines whether the input data is sampled from 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.

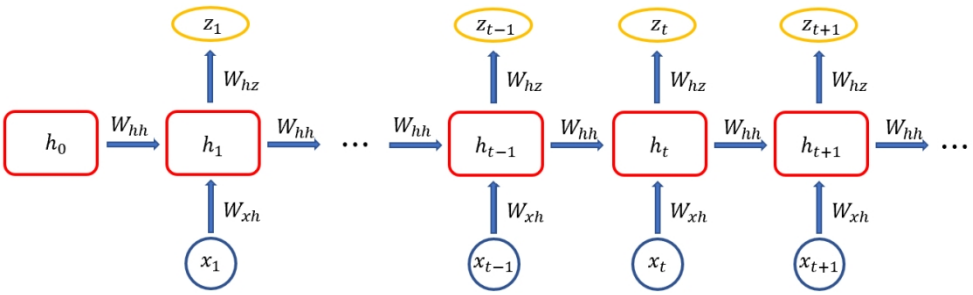

Figure 6: A simple RNN with one input unit, one output unit, and one hidden unit

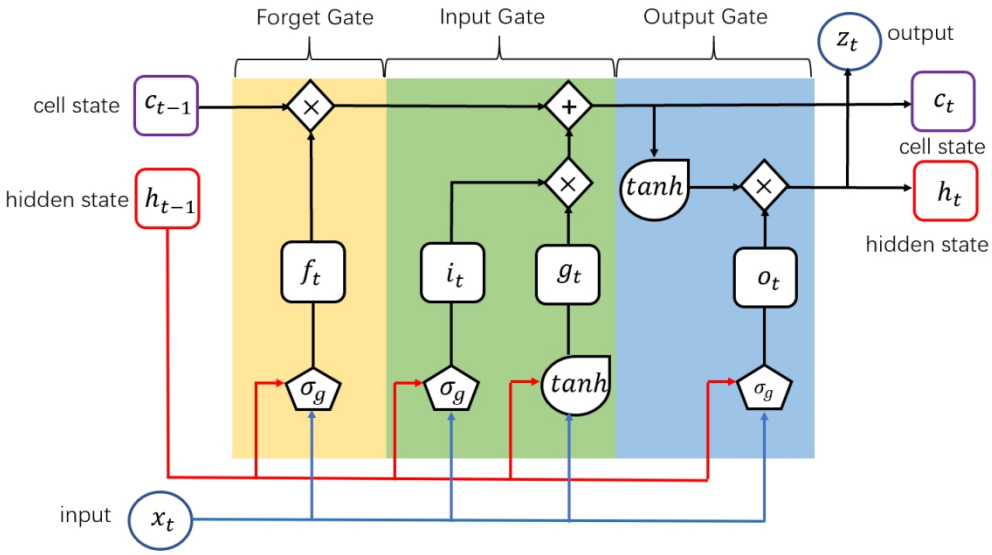

Figure 7: Typical structure of a cell of a single layer LSTM

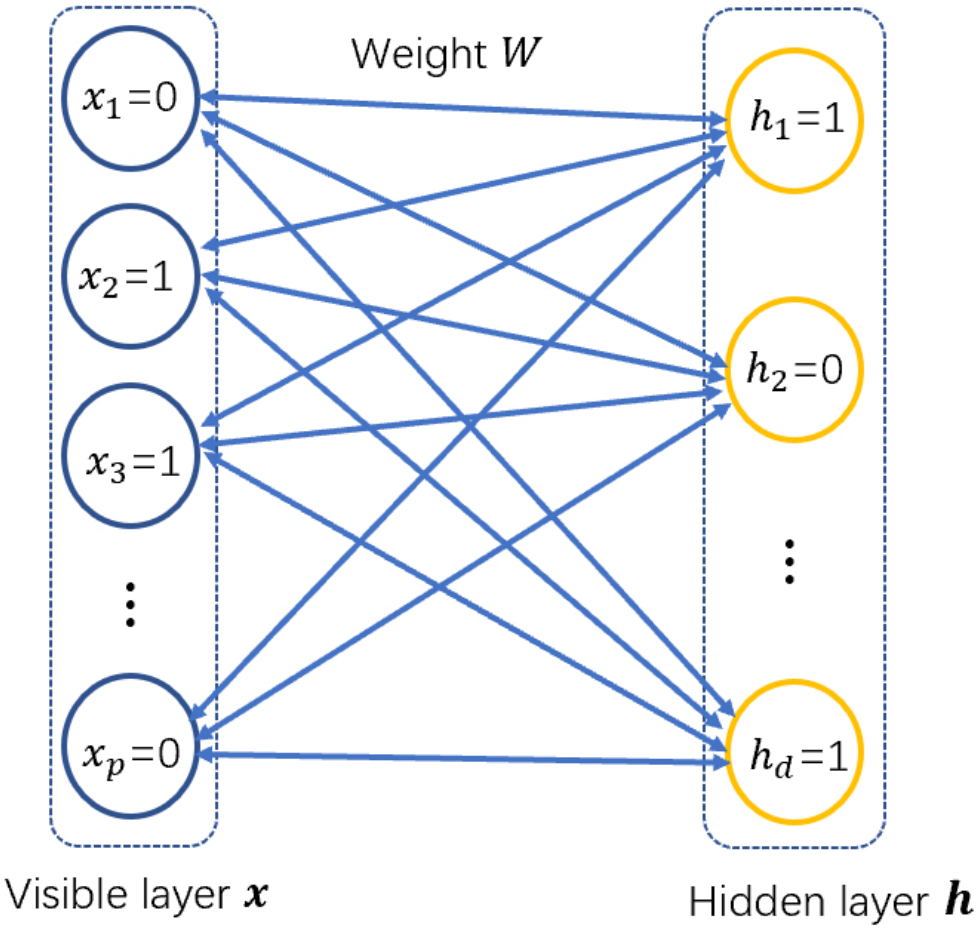

Figure 8: Graphical representation of a restricted Boltzmann machine.

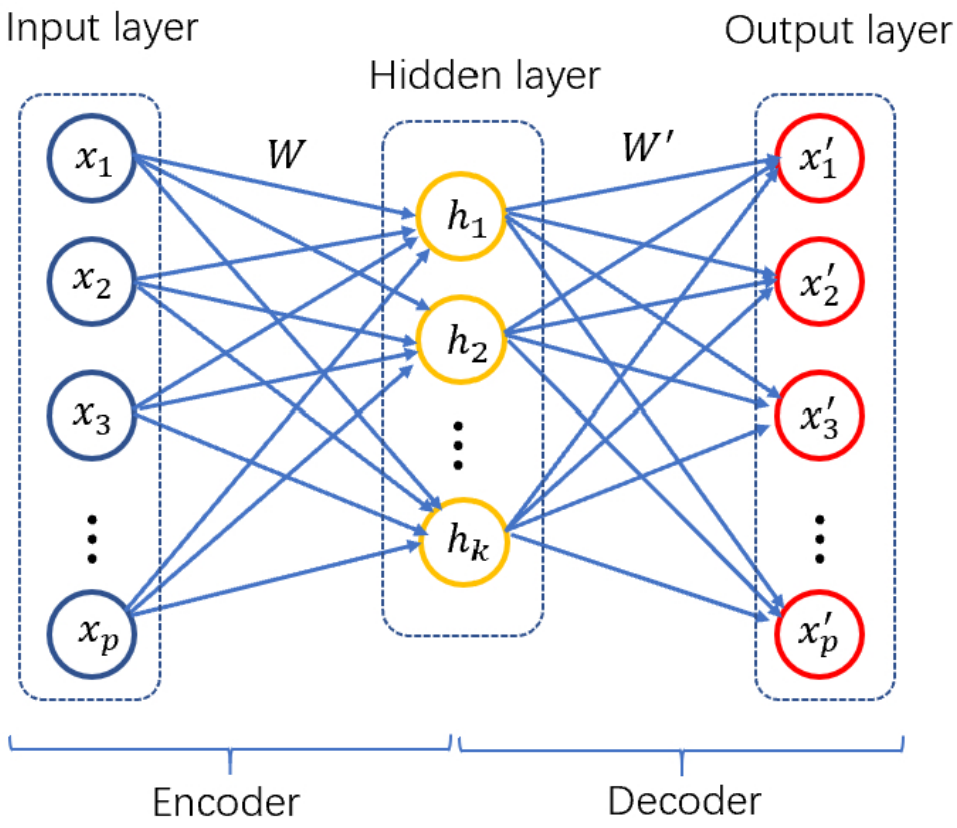

Figure 9: Graphical representation of a basic autoencoder

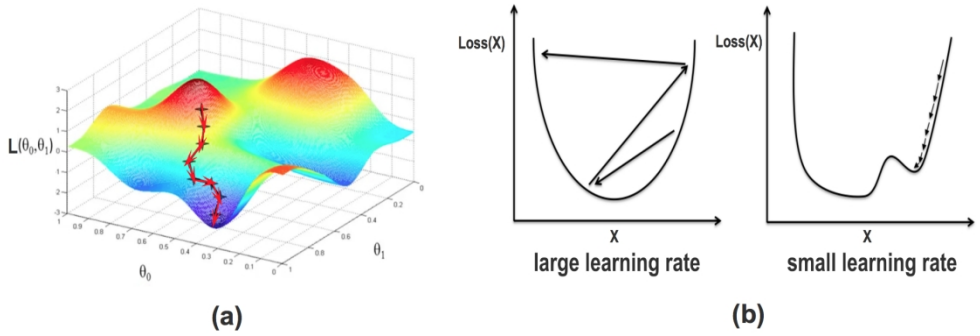

Figure 10: (a): Find minimization through gradient descent. (b): Illustration showing that too large learning rate leads to missing optimization while too small learning rate could lead to trapping at local minimum.

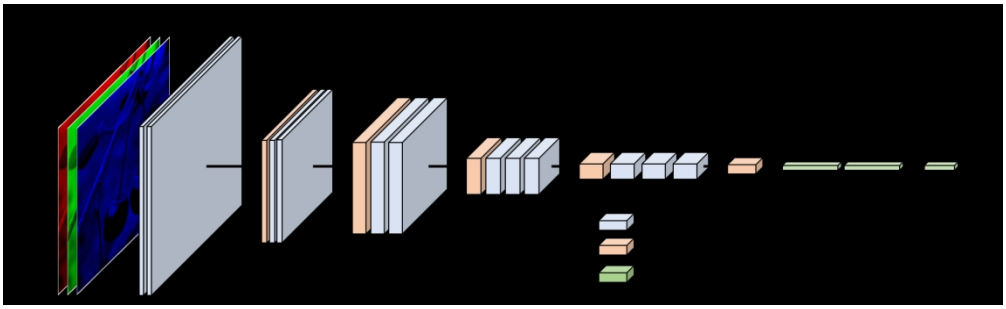

Figure 11: Architecture of the VGG-16. The numbers denote the size of each layer.

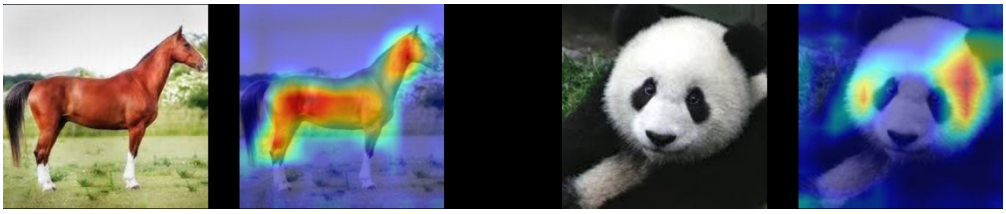

Figure 12: Classification results of a horse and a panda, each pair of images includes input image (left) and the corresponding heatmap (right) indicating the location of important pixels for classification. The warmer color in the heatmaps indicates higher pixel importance.

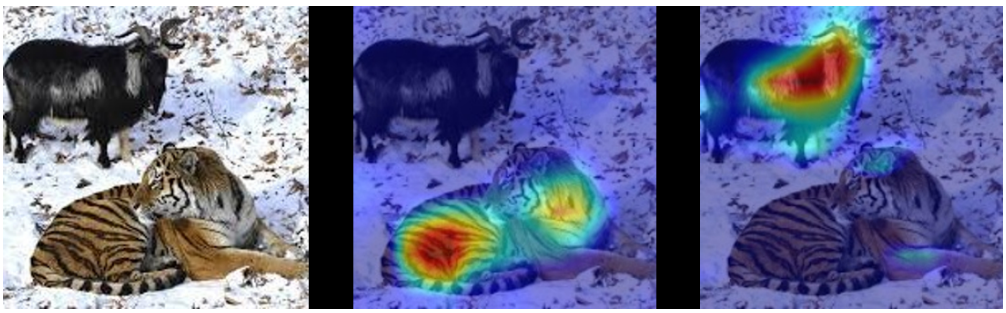

Figure 13: An image of a tiger and a goat is classified as a "tiger"(p = 0.71) and a "lion"(p = 0.047), showing the input image (left), the heatmap for "tiger" (middle) and the heatmap for "lion" (right)

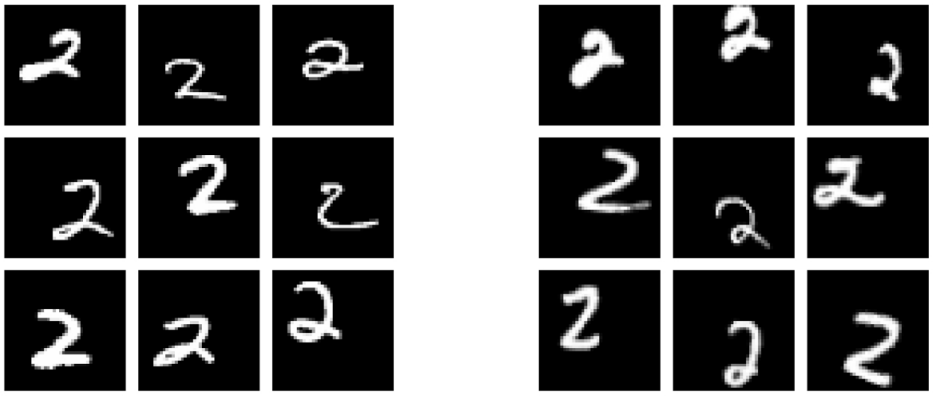

Figure 14: Samples of digit 2 from the translated MNIST dataset (left) and affNIST dataset (right). The translated MNIST applies only translation on MNIST images, and affNIST applies more complex affine transformation, including rotation, scaling and shearing.

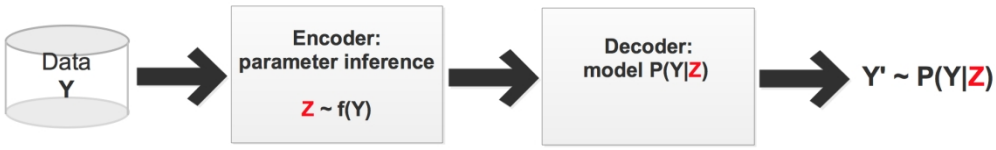

Figure 15: A deep generative model as a latent factor model.

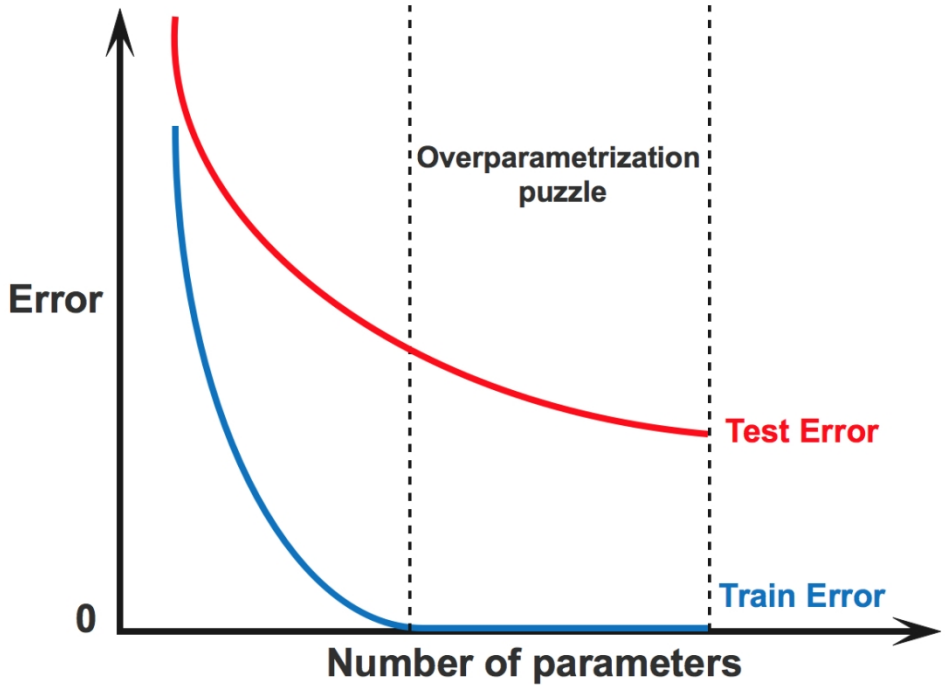

Figure 16: Overparameterization on deep learning: test error keeps decreasing after train error vanishes.