4 Deep Learning Models

This chapter addresses the background knowledge of deep learning theories and algorithms that is used as a foundation for developing speaker recognition solutions to meet different issues in implementation of real-world systems. We will start from an unsupervised learning machine called the restricted Boltzmann machine (RBM), which serves as a building block for deep belief networks and deep Boltzmann machines, which will be introduced in Section 4.3. Discriminative fine-tuning will be applied to build a supervised model with good performance for classification. RBM will be described in Section 4.1. Then, the algorithm for constructing and training a supervised deep neural network (DNN) will be addressed in Section 4.2. However, training a reliable DNN is challenging. Section 4.4 mentions the algorithm of stacking autoencoder, which is performed to build a deep model based on a two-layer structural module in a layer-by-layer fashion. Furthermore, in Section 4.3, the deep belief network is introduced to carry out a procedure of estimating a reliable deep model based on the building block of RBM. Nevertheless, DNN is not only feasible to build a supervised model for classification or regression problems in speaker recognition but also applicable to construct an unsupervised model that serves as generative model for data generation to deal with data sparseness problem. From this perspective, we introduce two deep learning paradigms. One is variational auto-encoder as mentioned in Section 4.5 while the other is the generative adversarial network as provided in Section 4.6. Such general neural network models can be merged in solving different issues in a speaker recognition system. At last, this chapter will end in Section 4.7, which addresses the fundamentals of transfer learning and the solutions based on deep learning. The deep transfer learning is developed to deal with domain mismatch problem between training and test data.

4.1 Restricted Boltzmann Machine

Restricted Boltzmann machine (RBM) [89, 90] plays a crucial role in building deep belief networks that are seen as the foundations in deep learning. Basically, RBM is seen as a bipartite graph that can be represented by an undirected graphical model or equivalently a bidirectional graphical model in a two-layer structure. One is the visible layer and the other is the hidden layer. Each connection must connect visible units $\mathbf{v} = \{v_i\}_{i=1}^{V}$ with hidden units $\mathbf{h} = \{h_j\}_{j=1}^{H}$ using weight parameters $\mathbf{W}$ where bias parameters $\{\mathbf{a}, \mathbf{b}\}$ are usually assumed to be zero for compact notation. Hidden variable

https://doi.org/10.1017/9781108552332.005 Published online by Cambridge University Press

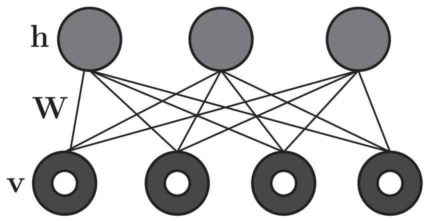


Figure 4.1 A two-layer structure of the restricted Boltzmann machine.

h extracts high-level information from observations **v**. Unlike the Boltzmann machines, the name "restricted" is added because there are no connections between two neurons in the same layer, namely visible-visible or hidden-hidden connections do not exist. Sparsity is controlled in RBM. RBM structure is depicted in Figure 4.1. There is no arrows in between-layer connections because these connections are bidirectional. RBM meets the property that visible units are conditionally independent given hidden units and vice versa. Or equivalently, the following probabilities are met

$$p(\mathbf{v}|\mathbf{h},\boldsymbol{\theta}) = \prod_i p(v_i|\mathbf{h},\boldsymbol{\theta}) \tag{4.1}$$

$$p(\mathbf{h}|\mathbf{v},\boldsymbol{\theta}) = \prod_j p(h_j|\mathbf{v},\boldsymbol{\theta}) \tag{4.2}$$

where model parameters $\boldsymbol{\theta} = \{\mathbf{a}, \mathbf{b}, \mathbf{W}\}$ are used.

4.1.1 Distribution Functions

In general, there are two different types of RBM that we usually use in DNN pretraining. The difference between two types depends on the input data values. First, the Bernoulli-Bernoulli RBM considers that both visible and hidden units are binary, i.e., either 0 or 1. Bernoulli distribution is used to represent the binary units. The conditional distribution in Bernoulli-Bernoulli RBM is expressed by

$$p(v_i = 1|\mathbf{h},\boldsymbol{\theta}) = \sigma\left(a_i + \sum_j w_{ij}h_j\right) \tag{4.3}$$

$$p(h_j = 1|\mathbf{v},\boldsymbol{\theta}) = \sigma\left(b_j + \sum_i w_{ij}v_i\right) \tag{4.4}$$

where w_{ij} denotes the weight between visible units v_i and between hidden units h_j, a_i denotes the visible unit bias, b_j denotes the hidden unit bias, $\boldsymbol{\theta} = \{a_i, b_j, w_{ij}\}$ denotes the model parameters and

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4.5}$$

denotes a logistic sigmoid function. The energy [91] of such a joint configuration of visible units and hidden units is given by

$$E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta}) = -\sum_{i=1}^{V} a_i v_i - \sum_{j=1}^{H} b_j h_j - \sum_{i=1}^{V}\sum_{j=1}^{H} w_{ij} v_i h_j, \tag{4.6}$$

which is accumulated from V units v_i in visible layer, H units h_j in hidden layer and $V \cdot H$ mutual connections between two layers by using the corresponding parameters a_i, b_j, and w_{ij}.

On the other hand, the Gaussian-Bernoulli RBM is developed for the case that hidden units are binary while input units follow a linear model with Gaussian noise. This RBM deals with the real-valued data. Speech features are treated as real-valued inputs to neural networks. Namely, for Gaussian-Bernoulli RBM, the conditional distribution is given by

$$p(h_j = 1|\mathbf{v},\boldsymbol{\theta}) = \sigma\left(b_j + \sum_i \frac{v_i}{\sigma_i} w_{ij}\right) \tag{4.7}$$

$$p(v_i = r|\mathbf{h},\boldsymbol{\theta}) = \mathcal{N}\left(a_i + \sigma_i \sum_j w_{ij} h_j, \sigma_i^2\right) \tag{4.8}$$

where r is a real value and σ_i^2 denotes the the variance of Gaussian noise for visible unit v_i. We usually set $\sigma_i^2 = 1$ to simplify the implementation. Using Gaussian-Bernoulli RBM, the energy of a joint configuration is calculated by

$$E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta}) = \sum_{i=1}^{V} \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_{j=1}^{H} b_j h_j - \sum_{i=1}^{V}\sum_{j=1}^{H} \frac{v_i}{\sigma_i} w_{ij} h_j. \tag{4.9}$$

Given the energy functions $E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})$ in Eqs. 4.6 and 4.9, the energy-based distribution of visible units $\mathbf{v}$ and latent units $\mathbf{h}$ is defined in the form

$$p(\mathbf{v},\mathbf{h}|\boldsymbol{\theta}) = \frac{e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})}}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})}}, \tag{4.10}$$

where the normalization constant or the *partition* function is defined as

$$Z(\boldsymbol{\theta}) = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})}. \tag{4.11}$$

This joint distribution of $\mathbf{v}$ and $\mathbf{h}$ is seen as a realization of exponential family.

The marginal likelihood and conditional likelihood are obtained by

$$p(\mathbf{v}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})} \tag{4.12}$$

$$p(\mathbf{h}|\mathbf{v},\boldsymbol{\theta}) = \frac{p(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})}{p(\mathbf{v}|\boldsymbol{\theta})} = \frac{e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})}}. \tag{4.13}$$

In general, $p(\mathbf{h}|\mathbf{v}, \boldsymbol{\theta})$ is an easy and exact calculation for an RBM and $p(\mathbf{v}|\boldsymbol{\theta})$ is functioned as an empirical likelihood. An RBM is seen as a generative model for observations $\mathbf{v}$, which is driven by the marginal distribution $p(\mathbf{v}|\boldsymbol{\theta})$. Correlations between nodes in $\mathbf{v}$ are present in the marginal distribution $p(\mathbf{v}|\boldsymbol{\theta})$ as shown in Eq. 4.6 or Eq. 4.9.

4.1.2 Learning Algorithm

According to the maximum likelihood (ML) principle, we maximize the logarithm of the likelihood function of visible data $\mathbf{v}$ to estimate the ML parameters by

$$\boldsymbol{\theta}_{\mathrm{ML}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\mathbf{v}|\boldsymbol{\theta}). \tag{4.14}$$

To solve this estimation problem, we differentiate the log-likelihood function with respect to individual parameters in $\boldsymbol{\theta}$ and derive the updating formulas. The derivative is derived as follows:

$$\begin{aligned}
\frac{\partial \log p(\mathbf{v}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &= \frac{\partial \log \left(\frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{Z} \right)}{\partial \boldsymbol{\theta}} \\
&= \frac{Z}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \frac{1}{Z^2} \left(\frac{Z \partial \left(\sum_{\mathbf{h}} Z e^{-E(\mathbf{v},\mathbf{h})} \right)}{\partial \boldsymbol{\theta}} - \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \frac{\partial Z}{\partial \boldsymbol{\theta}} \right) \\
&= \frac{1}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \frac{\partial \left(\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \right)}{\partial \boldsymbol{\theta}} - \frac{1}{Z} \frac{\partial Z}{\partial \boldsymbol{\theta}} \\
&= \frac{-\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} - \frac{-\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}}}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \\
&= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v},\boldsymbol{\theta}) \left(-\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \right) - \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}|\boldsymbol{\theta}) \left(-\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \right) \\
&= \mathbb{E}_{\mathbf{h} \sim p(\mathbf{h}|\mathbf{v},\boldsymbol{\theta})} \left\{ -\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \middle| \mathbf{v} \right\} - \mathbb{E}_{\mathbf{v},\mathbf{h} \sim p(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})} \left\{ -\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \right\}.
\end{aligned}$$

As a result, by referring to Eq. 4.6, the updating of the three parameters w_{ij}, a_i, and b_j in Bernoulli-Bernoulli RBMs can be formulated as

$$\frac{\partial \log p(\mathbf{v}|\boldsymbol{\theta})}{\partial w_{ij}} = \langle v_i h_j \rangle_0 - \langle v_i h_j \rangle_\infty \tag{4.15}$$

$$\frac{\partial \log p(\mathbf{v}|\boldsymbol{\theta})}{\partial a_i} = \langle v_i \rangle_0 - \langle v_i \rangle_\infty \tag{4.16}$$

$$\frac{\partial \log p(\mathbf{v}|\boldsymbol{\theta})}{\partial b_j} = \langle h_j \rangle_0 - \langle h_j \rangle_\infty \tag{4.17}$$

where the angle brackets $\langle\rangle$ denote the expectations under the distribution either $p(\mathbf{h}|\mathbf{v},\boldsymbol{\theta})$ in the first term or $p(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})$ in the second term. The subscript 0 means the step number from the original data and ∞ means the step number after doing the Gibbs sampling steps. The difference between two terms in Eqs. 4.15, 4.16, and 4.17 is seen as a contrastive divergence (CD) between the expectations of $v_i h_j$, v_i or h_j with initialization from visible data $\mathbf{v}$ and with calculation after infinite Gibbs sampling. A Gibbs chain is run continuously with initialization from the data.

The contrastive divergence algorithm [92, 93] is performed to train an RBM. Because the parameter updating in exact implementation is time-consuming, the so-called k-step contrastive divergence (CD-k) [92, 94] is applied to speed up the implementation procedure. In practice, the case $k = 1$ is used to carry out one step of Gibbs sampling or Markov-chain Monte Carlo sampling (CD-1), as addressed in Section 2.4, instead of $k = \infty$. More specifically, the blocked-Gibbs sampling is performed. This trick practically works well. Correspondingly, the formulas for parameter updating are modified as

$$\begin{aligned}\frac{\partial \log p(\mathbf{v}|\boldsymbol{\theta})}{\partial w_{ij}} &= \langle v_i h_j\rangle_0 - \langle v_i h_j\rangle_1 \\ &= \mathbb{E}_{p_{\text{data}}}[v_i h_j] - \mathbb{E}_{p_{\text{model}}}[v_i h_j] \triangleq \Delta w_{ij}\end{aligned} \tag{4.18}$$

$$\frac{\partial \log p(\mathbf{v}|\boldsymbol{\theta})}{\partial a_i} = \langle v_i\rangle_0 - \langle v_i\rangle_1 \tag{4.19}$$

$$\frac{\partial \log p(\mathbf{v}|\boldsymbol{\theta})}{\partial b_j} = \langle h_j\rangle_0 - \langle h_j\rangle_1, \tag{4.20}$$

where the subscript 1 means that one step of Gibbs sampling is performed. Figure 4.2 shows the training procedure of RBM based on CD-1 and CD-k. The arrows between visible layer and hidden layers are used to indicate the sampling steps and directions. Calculation of $\langle v_i h_j\rangle_\infty$ using each individual neuron i in visible layer and neuron j in hidden layer is performed step by step by starting from $\langle v_i h_j\rangle_0$. Step k is seen as a kind of state k in a Markov chain. A kind of Markov switching is realized in the CD algorithm.

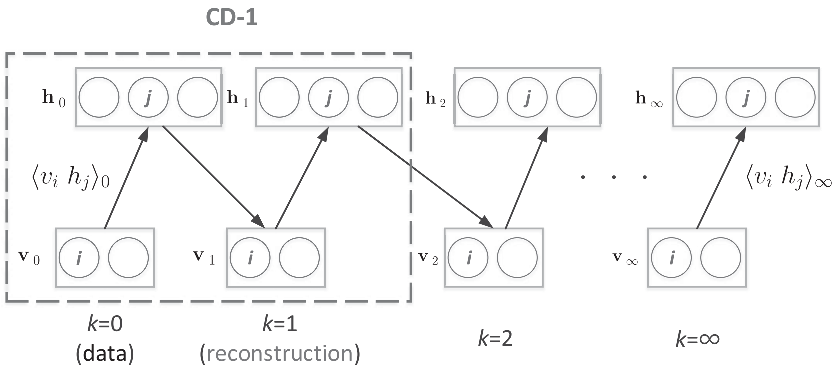


Figure 4.2 Training procedure for a restricted Boltzmann machine using the contrastive divergence algorithm.

After finding the partial derivatives of log likelihood with respect to three individual parameters $\{w_{ij}, a_i, b_j\}$ using the CD algorithm, the stochastic gradient descent (SGD) algorithm with momentum is implemented to construct the RBM model as shown in Algorithm 1. When running the SGD algorithm, the training data are split into minibatches. SGD algorithm proceeds with one minibatch at a time and runs a number of epochs. Meaningfully, the expectation from data $\langle v_i h_j \rangle_0$ using data distribution p_{data} is also denoted by $\langle v_i h_j \rangle_{\text{data}}$ or $\mathbb{E}_{p_{\text{data}}}[v_i h_j]$. The other expectations $\langle v_i h_j \rangle_1$ and $\langle v_i h_j \rangle_\infty$ are seen as those from model, i.e. $\langle v_i h_j \rangle_{\text{model}}$ or $\mathbb{E}_{p_{\text{model}}}[v_i h_j]$ using model distribution p_{model}. Calculation from model in step $k = 1$ is also viewed as a kind of reconstruction from data in step $k = 0$.

Algorithm 1 Training procedure for the restricted Boltzmann machine

```
$\Delta w_{ij} = \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}}$
Use CD-1 to approximate $\langle v_i h_j \rangle_{\text{model}}$
    Initialize $\mathbf{v}_0$ with observation data
    Sample $\mathbf{h}_0 \sim p(\mathbf{h}|\mathbf{v}_0, \boldsymbol{\theta})$
    Sample $\mathbf{v}_1 \sim p(\mathbf{v}|\mathbf{h}_0, \boldsymbol{\theta})$
    Sample $\mathbf{h}_1 \sim p(\mathbf{h}|\mathbf{v}_1, \boldsymbol{\theta})$
    Call $(\mathbf{v}_1, \mathbf{h}_1)$ a sample from the model
$(\mathbf{v}_\infty, \mathbf{h}_\infty)$ is a true sample from the model
$(\mathbf{v}_1, \mathbf{h}_1)$ is a very rough estimate but worked
```

An RBM acts as an unsupervised two-layer learning machine that is a shallow neural network with only one hidden layer. Nevertheless, a number of RBMs can act as building blocks to construct a deep belief network that will be addressed in Section 4.3. Deep model is equipped with strong modeling capability by increasing the depth of hidden layers so as to represent the high-level abstract meaning of data. In what follows, we address the principle of deep neural networks and their training procedure.

4.2 Deep Neural Networks

A deep neural network (DNN) has a hierarchy or deep architecture that is composed of a number of hidden layers. This architecture aims to learn an abstract representation or high-level model from observation data, which is used to find the highly nonlinear mapping between input data and their target values. DNNs have been widely developed to carry out different classification and regression applications, e.g., speaker recognition [95], image classification [96], speech recognition [97–99], natural language processing [100], and music or audio information retrieval, etc.

4.2.1 Structural Data Representation

Real-world applications involve different kinds of technical data that are inherent with structural features. It is crucial to capture the hierarchical information and conduct representation learning when building the information systems in presence of various observation data. System performance is highly affected by the modeling capability of a learning machine. To assure the learning capability, it is essential to increase the level of abstraction extracted from observation data and carry out a hierarchy of representations based on a deep learning machine. For example, image data can be learned via a representation process with multiple stages:

$$\text{pixel} \rightarrow \text{edge} \rightarrow \text{texton} \rightarrow \text{motif} \rightarrow \text{part} \rightarrow \text{object}.$$

Each stage is run as a kind of trainable feature transform that captures the hierarchical features corresponding to different levels of observed evidence including pixel, edge, texton, motif, part, and object. Such a trainable feature hierarchy can be also extended to represent other technical data, e.g., text and speech, in a hierarchical style of

$$\text{character} \rightarrow \text{word} \rightarrow \text{word group} \rightarrow \text{clause} \rightarrow \text{sentence} \rightarrow \text{story}$$

and

$$\text{sample} \rightarrow \text{spectral band} \rightarrow \text{sound} \rightarrow \text{phoneme} \rightarrow \text{phone} \rightarrow \text{word}$$

based on the hierarchies from character to story and from time sample to word sequence, respectively. A deep structural model is required to perform delicate representation so as to achieve desirable system performance in heterogeneous environments.

Figure 4.3 represents a bottom-up fully connected neural network that can characterize the multilevel abstraction in the hidden layers given the observation data in the bottom layer. The calculation starts from the input signals and then propagates with feedforward layers toward the abstraction outputs for regression or classification. In layer-wise calculation, we sum up the inputs in low-level layer and produce the activation outputs in high-level layer. The regression or classification error is determined at the top, and it is minimized and passed backward from the top to the bottom layer. Deep neural networks are structural models with three kinds of realizations as shown in Figure 4.4.

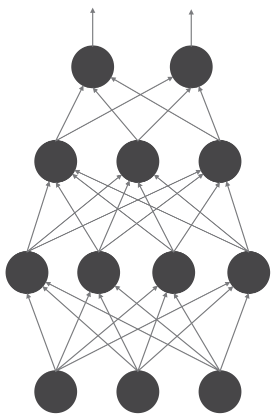


Figure 4.3 Bottom-up layer-wise representation for structural data.

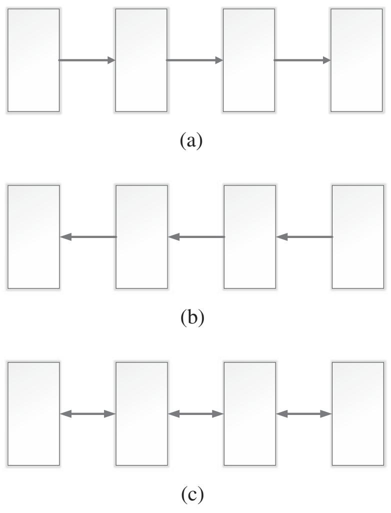


Figure 4.4 Three types of structural model (a) feedforward model, (b) feedback model, and (c) bidirectional model.

The multilayer neural networks and the convolutional neural networks correspond to the feedforward model where the inputs are summed, activated, and feedforwarded toward outputs. Without loss of generality, the bottom-up network has an alternative graphical representation using the left-to-right model topology. In addition to feedforward model, the layer-wise network can also be realized as a feedback model as well as a bidirectional model according to the direction of data flow in the model. Deconvolutional neural network [101] and stacked sparse coding [102] are seen as the feedback models. Deep Boltzmann machine [103], stacked autoencoder [104], and bidirectional recurrent neural network [105] are known as the bidirectional models. This book mainly addresses the feedforward model for deep learning based on multilayer perceptrons that will be introduced next.

4.2.2 Multilayer Perceptron

Figure 4.5 depicts a multilayer perceptron (MLP) [106] where $\mathbf{x}_t$ denotes the input with D dimensions, $\mathbf{z}_t$ denotes the hidden feature with M dimensions, $\mathbf{y}_t$ denotes the output with K dimensions, and t denotes the time index. A hierarchy of hidden layers is constructed by stacking a number of hidden layers. The mapping or relation between an input vector $\mathbf{x} = \{x_{td}\}$ and an output vector $\mathbf{y} = \{y_{tk}\}$ is formulated as

$$\begin{aligned} y_{tk} &= y_k(\mathbf{x}_t, \mathbf{w}) \\ &= f\left(\sum_{m=0}^{M} w_{mk}^{(2)} f\left(\sum_{d=0}^{D} w_{dm}^{(1)} x_{td}\right)\right) \\ &= f\left(\sum_{m=0}^{M} w_{mk}^{(2)} z_{tm}\right) \triangleq f(a_{tk}). \end{aligned} \tag{4.21}$$

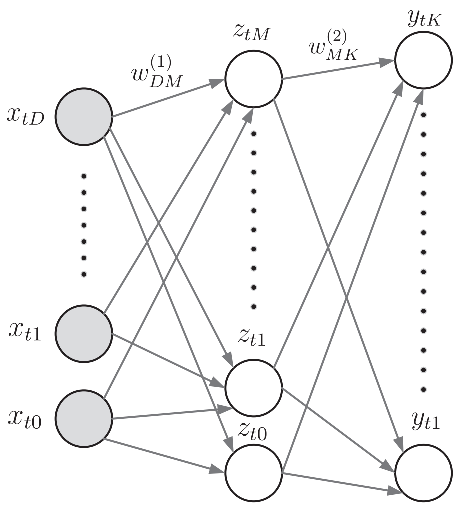


Figure 4.5 A multilayer perceptron with one input layer, one hidden layer, and one output layer. [Based on *Pattern Recognition and Machine Learning (Figure 5.1), by C. M. Bishop, 2006, Springer.*]

Here, we introduce the weight parameters of the first two layers, which are expressed by $\mathbf{w} = \{w^{(1)}_{dm}, w^{(2)}_{mk}\}$. We have $x_{t0} = z_{t0} = 1$ and the bias parameters that are denoted by $\{w^{(1)}_{0m}, w^{(2)}_{0k}\}$. This is a feedforward neural network (FNN) with two layer-wise calculations in forward pass. The first calculation is devoted to the layer-wise affine transformation with multiplication by using the layered parameters $\{w^{(1)}_{dm}\}$ and $\{w^{(2)}_{mk}\}$. The transformation is calculated to find the activation from input layer to output layer in an order of

$$a_{tm} = \sum_{d=0}^{D} w^{(1)}_{dm} x_{td} \quad \Longrightarrow \quad a_{tk} = \sum_{m=0}^{M} w^{(2)}_{mk} f(a_{tm}). \tag{4.22}$$

The second calculation is performed with the nonlinear activation function $f(\cdot)$. Figure 4.6 shows different activation functions, including the logistic sigmoid function in Eq. 4.5, the rectified linear unit (ReLU)

$$f(a) = \text{ReLU}(a) = \max\{0, a\}, \tag{4.23}$$

and the hyperbolic tangent function

$$f(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}. \tag{4.24}$$

The logic sigmoid function has value between 0 and 1. Differently, the hyperbolic tangent function has value between -1 and 1. However, the most popular activation in the implementation is ReLU, although others could be just as good for specific tasks [107]. For the case of classification network, the output vector $\mathbf{y}_t = \{y_{tk}\}$ is additionally calculated as a softmax function

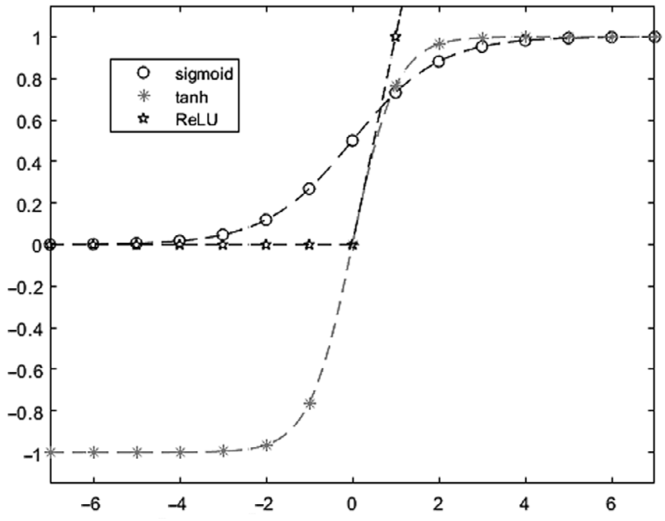


Figure 4.6 Three common activation functions in deep neural networks.

$$y_{tk} = \frac{\exp(a_{tk})}{\sum_m \exp(a_{tm})}. \tag{4.25}$$

In general, a DNN is constructed as a fully connected multilayer perceptron consisting of multiple hidden layers. Given an input vector, it is simple to compute the DNN's output in the forward pass, which includes the nonlinearity in the hidden layers and softmax activations in the output layer.

4.2.3 Error Backpropagation Algorithm

In implementation of DNN training, we collect a set of training samples $\{\mathbf{X}, \mathbf{R}\} = \{\mathbf{x}_t, \mathbf{r}_t\}_{t=1}^{T}$ in the form of input-output pairs. In the context of speech applications, $\mathbf{x}_t$ can be a speech vector or a number of consecutive speech vectors, whereas $\mathbf{r}_t$ is the desirable network output. In case $\mathbf{r}_t$'s are also speech vectors, a regression problem is formulated in an optimization procedure. We therefore estimate DNN parameters $\mathbf{w}$ by optimizing an objective based on the sum-of-squares error function, which is computed in DNN output layer in a form of

$$E(\mathbf{w}) = \frac{1}{2} \sum_{t=1}^{T} \|\mathbf{y}(\mathbf{x}_t, \mathbf{w}) - \mathbf{r}_t\|^2, \tag{4.26}$$

where $\mathbf{y}(\mathbf{x}_t, \mathbf{w}) = \{y_k(\mathbf{x}_t, \mathbf{w})\}$. However, the closed-form solution to solve this nonlinear regression problem is not analytical. The optimization is then implemented according to the stochastic gradient descent (SGD) algorithm

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}), \tag{4.27}$$

where τ denotes the iteration index, η denotes the learning rate and $E_n(\cdot)$ denotes the error function computed from the nth minibatch in training set $\{\mathbf{X}_n, \mathbf{R}_n\}$, which is sampled by using the entire training data $\{\mathbf{X}, \mathbf{R}\}$. A training epoch is run by scanning the whole set of minibatches denoted by $\{\mathbf{X}, \mathbf{R}\} = \{\mathbf{X}_n, \mathbf{R}_n\}_{n=1}^N$. In the implementation, we initialize the parameters from $\mathbf{w}^{(0)}$ and continuously perform SGD algorithm to reduce the value of error function E_n until the iteration procedure converges. Practically, the SGD algorithm is implemented by randomizing all minibatches in a learning epoch. Also, a large number of epochs are run to assure that the convergence condition is met after DNN training. In general, SGD training can achieve better regression or classification performance when compared with the batch training where all of the training data are put into a single batch.

The training of DNNs using the backpropagation algorithm involves two passes. In forward pass, an affine transformation followed by nonlinear activation is computed layer by layer until the output layer. In the background passes, the error (typically cross-entropy for classification and mean squared error for regression) between the desired output and the actual output is computed and the error gradient with respect to the network weights are computed from the output layer back to the input layer. That is, we find the gradients for updating the weights in different layers in the following order

$$\frac{\partial E_n(\mathbf{w}^{(\tau)})}{\partial w_{mk}^{(2)}} \rightarrow \frac{\partial E_n(\mathbf{w}^{(\tau)})}{\partial w_{dm}^{(1)}}, \tag{4.28}$$

where the minibatch samples $\{\mathbf{X}_n, \mathbf{R}_n\}_{n=1}^N$ are used. Figures 4.7(a) and (b) illustrate the computation in the forward pass and the backward pass, respectively. To fulfill the error backpropagation algorithm in a backward pass, an essential trick is to compute the local gradient of neuron m in a hidden layer by using the input vector at each time $\mathbf{x}_t$

$$\begin{aligned} \delta_{tm} &\triangleq \frac{\partial E_t}{\partial a_{tm}} \\ &= \sum_k \frac{\partial E_t}{\partial a_{tk}} \frac{\partial a_{tk}}{\partial a_{tm}} \\ &= \sum_k \delta_{tk} \frac{\partial a_{tk}}{\partial a_{tm}}, \end{aligned} \tag{4.29}$$

which is updated and recalculated by integrating local gradients δ_{tk} from all neurons k in output layer. The order of updating for local gradient is shown by $\delta_{tk} \rightarrow \delta_{tm}$. Given the local gradients in output layer and hidden layer, the SGD updating involves the calculation of differentiations in a way of

$$\frac{\partial E_n(\mathbf{w}^{(\tau)})}{\partial w_{dm}^{(1)}} = \sum_{t \in \{\mathbf{X}_n, \mathbf{R}_n\}} \delta_{tm} x_{td} \tag{4.30}$$

$$\frac{\partial E_n(\mathbf{w}^{(\tau)})}{\partial w_{mk}^{(2)}} = \sum_{t \in \{\mathbf{X}_n, \mathbf{R}_n\}} \delta_{tk} z_{tm}, \tag{4.31}$$

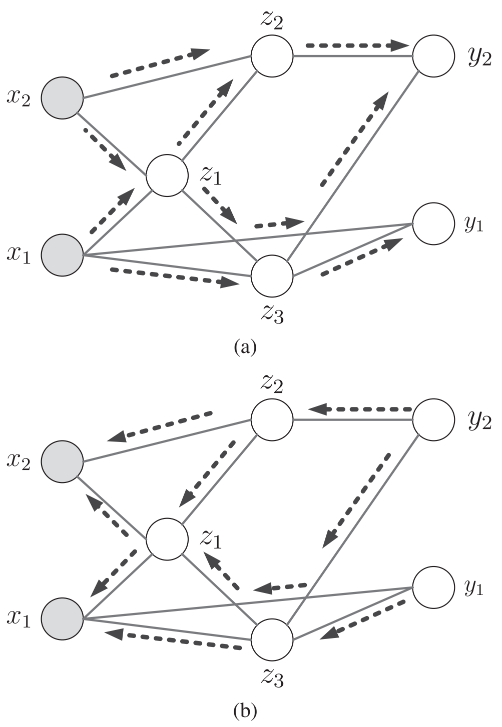


Figure 4.7 Illustration for computations in error backpropagation algorithm including those in (a) forward pass and (b) backward pass. The propagation directions are displayed by arrows. In a forward pass, each individual node involves the computation and propagation of activations a_t and outputs z_t. In a backward pass, each individual node involves the computation and propagation of local gradients δ_t. [Adapted from *Pattern Recognition and Machine Learning (Figure 5.2), by C. M. Bishop, 2006, Springer.*]

where the error function E_t is accumulated over $t \in \{\mathbf{X}_n, \mathbf{R}_n\}$, which is an calculation using the input-target pairs in a minibatch $E_n = \sum_{t\in\{\mathbf{X}_n,\mathbf{R}_n\}} E_t$. At each time t, we simply express the gradient for correcting a weight parameter $w_{mk}^{(2)}$ by using a multiplication of the output of neuron m, z_{tm}, in the hidden layer and the local gradient of neuron k, δ_{tk}, in the output layer. This type of computation is similarly employed in updating the weight parameter $w_{dm}^{(1)}$ for the neurons in input layer d and hidden layer m. In Figure 4.8, we illustrate a general procedure for error backpropagation algorithm that is used to train a l-layer multilayer perceptron. The forward calculation of error function and the backward calculations of local gradients or error gradients from layer l to layer $l-1$ until layer 1.

4.2.4 Interpretation and Implementation

The deep hierarchy in DNN is efficient in learning representation. The computation units based on linear transformation and nonlinear activation are modulated and adopted in

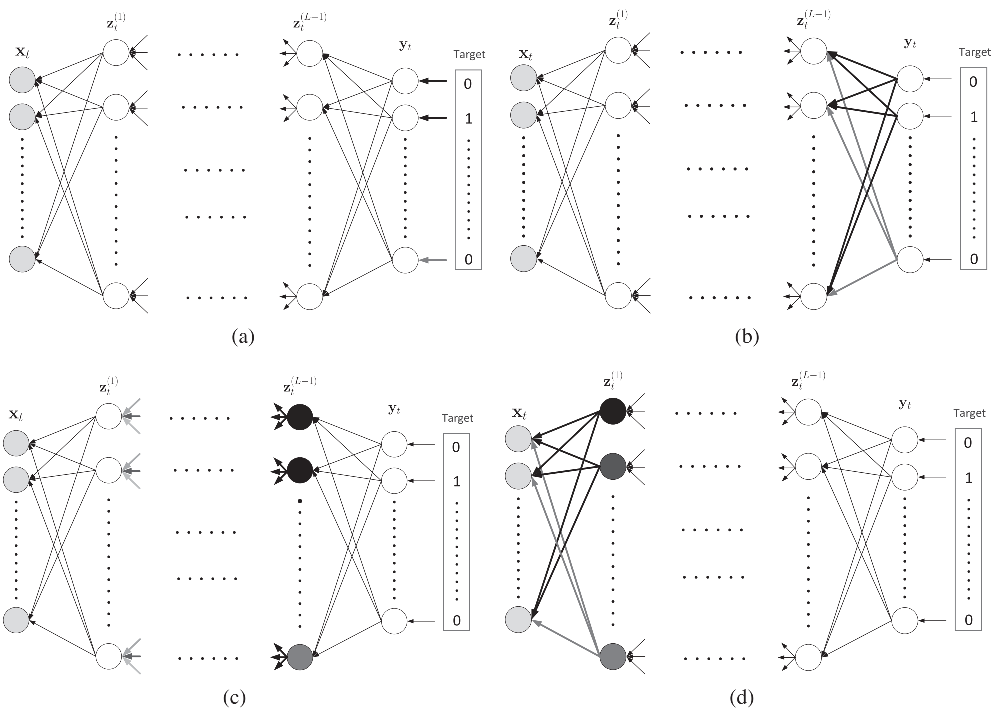


Figure 4.8 Procedure in the backward pass based on an error backpropagation algorithm. (a) Calculate the error function, (b) propagate the local gradient from output layer L to (c) hidden layer $L-1$ (d) until input layer 1.

different neurons and different layers. DNN is naturally seen as a discriminative model for hierarchical learning, which works for regression and classification tasks. It is crucial to explain why deep hierarchy does help when training a DNN for speaker recognition. Basically, the nonlinear and deep architecture provides a vehicle to deal with the weakness of the bounded performance due to a complicated regression mapping. Deep learning aims to conduct structural representation and pursue the comprehensibility in hierarchical learning. A hierarchical model with multiple layers of hidden neurons opens an avenue to conduct combinational sharing over the synapse of statistics. Structural learning in DNN provides a monitoring platform to analyze what has been learned and what subspace has been projected to obtain a better prediction for regression or classification. A hierarchical representation consisting of different levels of abstraction is trained for future prediction. Each level is trainable and corresponds to a specific feature transformation. A deep model is trained to find a high-level representation that is generalized for different tasks.

On the other hand, DNN is built with a huge parameter space and is spanned by a number of fully connected layers. The convergence condition in training procedure is usually hard to meet. Backpropagation does not work well if parameters are randomly initialized. The performance of deep model could not be theoretically guaranteed.

Basically, the weights of a DNN without unsupervised pretraining are randomly initialized. The resulting system performance is even worse than that of a shallow model [108]. It is crucial that DNN training procedure requires a reliable initialization and a rapid convergence to learn a desirable “deep” model for different regression or classification problems in a speaker recognition system.

The problems and solvers with the error backpropagation algorithm are discussed here. Basically, the gradient in backpropagation is progressively getting diluted. Below a top few layers, the correction signal for the connection weights is too weak. Backpropagation accordingly gets stuck in a local minimum, especially in case of random initialization where the starting point is far from good regions. On the other hand, in usual settings, we can use only labeled data to train a supervised neural network. But, almost all data are unlabeled in practical applications. Nevertheless, a human brain can efficiently learn from unlabeled data. To tackle this issue, we may introduce unsupervised learning via a greedy layer-wise training procedure. This procedure allows abstraction to be developed naturally from one layer to another and helps the network initialize with good parameters. After that, the supervised top-down training is performed as a final step to refine the features in intermediate layers that are directly relevant for the task. An elegant solution based on deep belief network is therefore constructed and addressed in what follows.

4.3 Deep Belief Networks

Deep belief network (DBN) [109] is a probabilistic generative model that provides a meaningful initialization or pretraining for constructing a neural network with deep structure consisting of multiple hidden layers. DBN conducts an unsupervised learning where the outputs are learned to reconstruct the original inputs. Different layers in DBN are viewed as the feature extractors. Such an unsupervised learning could be further merged with the supervised retraining for different regression or classification problems in speaker recognition. DBN is seen as a theoretical tool to pretrain or initialize different layers in DNN training. In the training procedure, the RBM, addressed in Section 4.1.2, is treated as a building component to construct multiple layers of hidden neurons for DBN. The building method is based on a stack-wise and bottom-up style. Each stack is composed of a pair of layers that is trained by RBM. After training each stack, the hidden layer of RBM is subsequently used as an observable layer to train the RMB in next stack for a deeper hidden layer. Following this style, we eventually train a bottom-up deep machine in accordance with a stack-wise and tandem-based training algorithm. DBN obtained great results in [109] due to good *initialization* and *deep* model structure. A lower bound of log likelihood of observation data $\mathbf{v}$ is maximized to estimate DBN parameters [94].

4.3.1 Training Procedure

As illustrated in Figure 4.9, a stack-wise and tandem-based training procedure is performed to learn DBN parameters. In this training procedure, the first RBM is estimated from a set of training samples $\{\mathbf{x}\}$ and then used to transform each individual speech token $\mathbf{x}$ into a latent variable $\mathbf{z}^{(1)}$ using the trained RBM parameters $\mathbf{w}^{(1)}$. The latent units $\{\mathbf{z}^{(1)}\}$ are subsequently treated as the observation data to learn the next RBM, which transforms each sample $\mathbf{z}^{(1)}$ into a deeper sample $\mathbf{z}^{(2)}$ in the next layer. RBM is used as a learning representation for a pair of layers. According to this procedure, a deep hierarchy using DBN is built to explore the observed and hidden variables from input layer to deep layers in a chain of

$$\mathbf{x} \to \mathbf{z}^{(1)} \to \mathbf{z}^{(2)} \to \mathbf{z}^{(3)} \to \cdots . \tag{4.32}$$

We develop the level of abstraction in a layer-wise manner. The progression of model structure from low level to high level represents the natural complexity. An unsupervised learning is performed to estimate a DBN by using a greedy and layer-wise training procedure. After training the DBN, we use DBN parameters as the initial parameters to train a DNN that performs much better than that based on the random initialization. The chance of going stuck in a local minimum point is much smaller. Finally, a supervised top-down fine-tuning procedure is executed to adjust the features in middle layers based on the labels of training data $\mathbf{r}$. Compared to the original features, the adapted features are better fitted to produce the target values $\mathbf{y}$ in output layer. The supervised learning using error backpropagation algorithm is employed in a fine-tuning process. The resulting method is also called DBN-DNN, which has been successfully developed for deep learning. This DBN-DNN is seen as *generative* model as well as *discriminative* model due to this two-step procedure, including one step of unsupervised stack-wise training and the other step of supervised fine-tuning and processing. Specifically, the unlabeled samples $\{\mathbf{x}\}$ are collected to build a generative model in a stack-wise and bottom-up manner. Then, a small set of labeled samples $\{\mathbf{x}, \mathbf{r}\}$ is used to adjust DBN parameters to final DNN parameters based on an error backpropagation algorithm.

In general, the greedy and layer-wise training of DNN performs well from the perspectives of optimization and regularization due to twofold reasons. First, the pretraining step in each layer in a bottom-up way helps constraining the learning process around

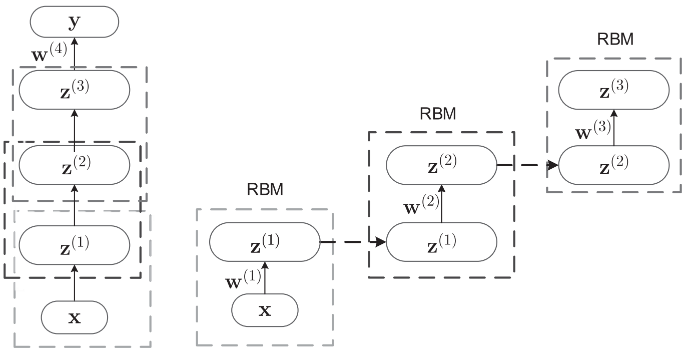


Figure 4.9 A stack-wise training procedure for deep belief network based on the restricted Boltzmann machine. [Based on *Source Separation and Machine Learning (Figure 2.16), by J.T. Chien, 2018, Academic Press.*]

the region of parameters, which is relevant to represent the unlabeled data. The learning representation is performed to describe the unlabeled data in a way of providing discriminative representation for labeled data. DNN is accordingly regularized to improve model generalization for prediction of unseen data. In addition, the second reason is caused by the fact that the unsupervised learning starts from lower-layer parameters that are close to the localities and near to the minimum in optimization. Random initialization is hard to attain these properties. Such a perspective sufficiently explains why the optimization for DNN based on DBN initialization works better than that based on random initialization.

There are three training strategies for deep neural networks or deep belief networks developed in different application domains under various training conditions. First, the deep model can be purely trained in supervised mode. The model parameters can be initialized randomly and estimated according to SGD using the backpropagation method to compute gradients. Most practical systems for speech, text, and image applications have been built by using this strategy. Second, the unsupervised learning is combined with supervised training for constructing a deep model. Using this strategy, each layer is trained in an unsupervised way using RBM. The layer-wise hierarchy is grown one layer after the other until the top layer. Then, a supervised classifier on top layers is trained while keeping the parameters of the other layers fixed. This strategy is suitable especially when very few labeled samples are available. A semi-supervised learning is carried out from a large set of unlabeled data and a small set of labeled data. Similar to the second strategy, the third strategy builds a deep structure in an unsupervised mode layer by layer based on RBMs. A classifier layer is added on the top for classification task. The key difference is to retrain the whole model in a supervised way. This strategy works well especially when the labels are poorly transcribed. Typically, the unsupervised pretraining in second and third strategies is often performed by applying the regularized stacked autoencoders that will be addressed later in Section 4.4. In what follows, we further detail the procedure and the meaning of greedy training for deep belief networks.

4.3.2 Greedy Training

In deep learning, it is essential to carry out the greedy training procedure for stack-wise construction of a bottom-up neural network model. Greedy training is performed in an unsupervised style. Figure 4.10 depicts a building block of an RBM and an approach to stack a number of RBMs toward a deep model. The undirectional two-layer RBM functions as described in Section 4.1. The hidden units in $\mathbf{h}$ are characterized from visible speech data $\mathbf{v}$ using the weight parameters $\mathbf{W}^1$ in the first RBM. The construction of DBN with one visible layer $\mathbf{v}$ and three hidden layers $\mathbf{h}^1$, $\mathbf{h}^2$, and $\mathbf{h}^3$ in this example can be decomposed into three steps as illustrated in Figures 4.10(b) and (c).

At the first step, a two-layer RBM is constructed with an input layer $\mathbf{v}$ and a hidden layer $\mathbf{h}^1$. RBM parameters $\mathbf{W}^1$ are trained by maximizing the log likelihood $\log p(\mathbf{v})$ or alternatively the lower bound (or variational bound) of log likelihood, which is derived by

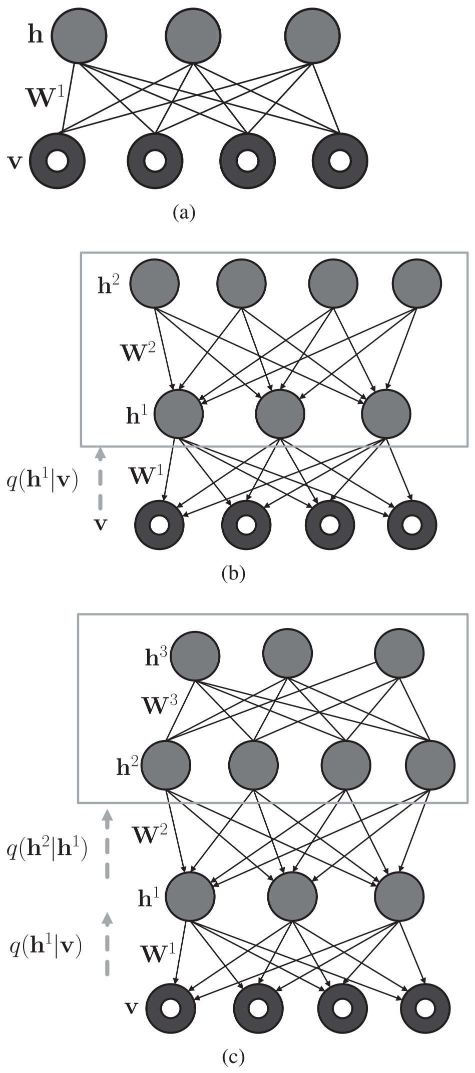


Figure 4.10 (a) A building block of two-layer RBM. Construction of deep belief network using (b) the second step of RBM and (c) the third step of RBM. The variational distributions $q(\mathbf{h}^1|\mathbf{v})$ and $q(\mathbf{h}^2|\mathbf{h}^1)$ are introduced to infer hidden variables $\mathbf{h}^1$ and $\mathbf{h}^2$ and model parameters $\mathbf{W}^1$ and $\mathbf{W}^2$ in different layers, respectively.

$$\log p(\mathbf{v}) \geq \sum_{\mathbf{h}^1} q(\mathbf{h}^1|\mathbf{v}) \Big(\log p(\mathbf{h}^1) + \log p(\mathbf{v}|\mathbf{h}^1)\Big) + \mathbb{H}_{q(\mathbf{h}^1|\mathbf{v})}[\mathbf{h}^1] \tag{4.33}$$

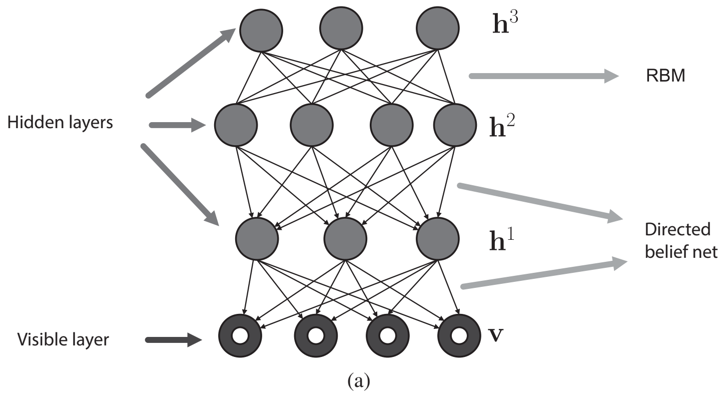
according to variational inference where the variational distribution $q(\mathbf{h}^1|\mathbf{v})$ for hidden variable $\mathbf{h}^1$ is used. $\mathbb{H}[\cdot]$ denotes an entropy function. The derivation is similar to Eq. 2.33 in Section 2.3 when finding the evidence lower bound. At the second step, another hidden layer is stacked on top of the RBM to form a new RBM. When training this second RBM, the parameters $\mathbf{W}^1$ of the first RBM are fixed and the units of the first hidden layer $\mathbf{h}^1$ are sampled by variational distribution $q(\mathbf{h}^1|\mathbf{v})$ and treated as visible data for training the second RBM with parameters $\mathbf{W}^2$. At the third step, this stack-wise process is continued in order to stack layers on top of the network and to train the third RBM parameter $\mathbf{W}^3$ as in the previous step, with the sample $\mathbf{h}^2$ drawn from $q(\mathbf{h}^2|\mathbf{h}^1)$ and treated as the visible data to explore the latent variable $\mathbf{h}^3$ in the third hidden layer. The joint distribution of visible data $\mathbf{v}$ and l hidden variables $\{\mathbf{h}^1, \ldots, \mathbf{h}^l\}$ is therefore expressed in accordance with the property of Markov switching where the probability of a layer k depends only on layer $k + 1$ as

$$p(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \ldots, \mathbf{h}^l) = p(\mathbf{v}|\mathbf{h}^1)p(\mathbf{h}^1|\mathbf{h}^2)\ldots p(\mathbf{h}^{l-2}|\mathbf{h}^{l-1})p(\mathbf{h}^{l-1}|\mathbf{h}^l). \qquad (4.34)$$

Figure 4.11(a) shows the construction of DBN with one visible layer and three hidden layers. RBM is on the top while the directed belief network is on the bottom. The arrows indicate the direction of data generation. We basically estimate the model parameters $\{\mathbf{W}^1, \ldots, \mathbf{W}^l\}$ by maximizing the variational lower bound of log likelihood $p(\mathbf{v})$ where all possible configurations of the higher variables are integrated to get the prior for lower variables toward the likelihood of visible variables. In addition, as illustrated in Figure 4.11(b), if the first RBM with parameter $\mathbf{W}^1$ is stacked by an inverse RBM (the second RBM) with parameters $(\mathbf{W}^1)^\top$, the hidden units $\mathbf{h}^2$ of the second RBM with input units $\mathbf{h}^2$ are seen as the reconstruction of visible data $\mathbf{v}$ in the first RBM.

It is important to explain why greedy training works for construction of a deep belief network. In general, DBN is constructed via RBM where the joint distribution of a two-layer model $p(\mathbf{v}, \mathbf{h})$ is calculated by using the conditional distributions $p(\mathbf{v}|\mathbf{h})$ and $p(\mathbf{h}|\mathbf{v})$ that implicitly reflects the marginal distributions $p(\mathbf{v})$ and $p(\mathbf{h})$. The key idea behind DBN is originated from the stacking of RBM, which preserves the conditional likelihood $p(\mathbf{v}|\mathbf{h}^1)$ from the first-level RBM and replaces the distribution of hidden units by using the distribution $p(\mathbf{h}^1|\mathbf{h}^2)$ generated by the second-level RBM. Furthermore, DBN performs an easy approximate inference where $p(\mathbf{h}^{k+1}|\mathbf{h}^k)$ is approximated by the associated RBM in layer k using $q(\mathbf{h}^{k+1}|\mathbf{h}^k)$. This is an approximation because $p(\mathbf{h}^{k+1})$ differs between RBM and DBN.

During training time [110], variational bound is maximized to justify greedy layer-wise training of RBMs. In the whole training procedure, we basically initialize from the stacked RBMs in a pretraining stage, which is seen as two-step fine-tuning. First, the *generative* fine-tuning is run to construct or fine-tune a deep model based on the variational or mean-field approximation where the persistent chain and the stochastic approximation are performed. Second, the *discriminative* fine-tuning is executed by minimizing the classification loss in the backpropagation procedure. In addition, greedy training meets the *regularization* hypothesis where the pretraining is performed to constrain the parameters in a region relevant to an unsupervised dataset. Greedy training also follows the *optimization* hypothesis where the unsupervised training initializes the lower-level parameters near the localities of better minima. Such a property is *not* held by using the scheme of random initialization. Nevertheless, the greedy procedure of stacking RBMs is suboptimal. A very approximate inference procedure is performed in



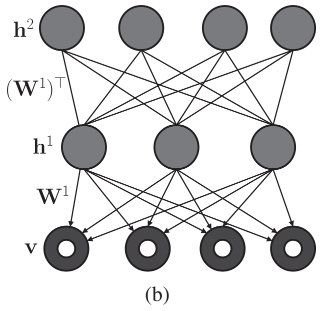


Figure 4.11 (a) Construction of deep belief network with three hidden layers. The top two layers form an undirectional bipartite graph using a RBM while the remaining layers form a sigmoid belief net with directed and top-down connections. (b) RBMs are stacked as an invertible learning machine for data reconstruction.

DBN. These weaknesses were tackled by using a different type of hierarchical probabilistic model, called the deep Boltzmann machine, which is addressed in what follows.

4.3.3 Deep Boltzmann Machine

Deep Boltzmann machine (DBM) [103, 111] is a type of Markov random field where all connections between layers are undirected as depicted in Figure 4.12, which differs from Figure 4.11 for deep belief network. The undirected connection between the layers make a complete Boltzmann machine. Similar to RBM and DBN, using this unsupervised DBM, there are no connections between the nodes in the same layer. High-level representations are built from the unlabeled inputs. Labeled data are only used to slightly fine-tune the model.

Considering the example of DBM with three hidden layers, the marginal likelihood function is given by

$$p(\mathbf{v}|\boldsymbol{\theta}) = \sum_{\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3} \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\mathbf{v}^\top \mathbf{W}^1 \mathbf{h}^1 + (\mathbf{h}^1)^\top \mathbf{W}^2 \mathbf{h}^2 + (\mathbf{h}^2)^\top \mathbf{W}^3 \mathbf{h}^3\right) \tag{4.35}$$

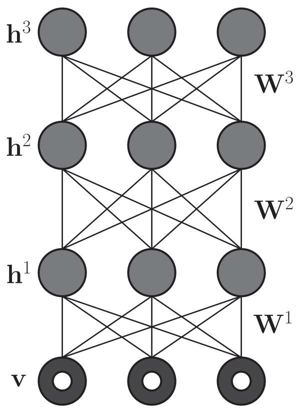


Figure 4.12 A deep Boltzmann machine with three hidden layers. All connections between layers are undirected but with no within-layer connections.

where $\{\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3\}$ are the hidden units, $\boldsymbol{\theta} = \{\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3\}$ are the model parameters, and $Z(\boldsymbol{\theta})$ is a normalization constant. Similar to RBM in Section 4.1, the energy-based likelihood function in DBM is defined by using the energy function

$$E(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta}) = -\mathbf{v}^\top \mathbf{W}^1 \mathbf{h}^1 - (\mathbf{h}^1)^\top \mathbf{W}^2 \mathbf{h}^2 - (\mathbf{h}^2)^\top \mathbf{W}^3 \mathbf{h}^3, \tag{4.36}$$

which is accumulated from three pairs of weight connections $\{\mathbf{v}, \mathbf{h}^1\}$, $\{\mathbf{h}^1, \mathbf{h}^2\}$, and $\{\mathbf{h}^2, \mathbf{h}^3\}$. For example, the updating of parameter vector $\mathbf{W}^1$ is performed according to the following derivative

$$\frac{\partial p(\mathbf{v}|\boldsymbol{\theta})}{\partial \mathbf{W}^1} = \mathbb{E}_{p_{\text{data}}}\left[\mathbf{v}(\mathbf{h}^1)^\top\right] - \mathbb{E}_{p_{\text{model}}}\left[\mathbf{v}(\mathbf{h}^1)^\top\right] \tag{4.37}$$

Similar updating can be also found for $\mathbf{W}^2$ and $\mathbf{W}^3$.

The mean-field theory is developed for variational inference of DBM parameters. A factorized variational distribution is introduced for a set of hidden variables $\mathbf{h} = \{\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3\}$ in a form of

$$q(\mathbf{h}|\mathbf{v}, \boldsymbol{\mu}) = \prod_j \prod_k \prod_m q(h_j^1) q(h_k^2) q(h_m^3) \tag{4.38}$$

where $\boldsymbol{\mu} = \{\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, \boldsymbol{\mu}^3\}$ denote the mean-field parameters with $q(h_i^k = 1) = \mu_i^k$ for $k = 1, 2, 3$. The lower bound of log likelihood in DBM is then derived by referring Eq. 4.33 using the variational distribution in Eq. 4.38. The bound is expressed by

$$\begin{aligned} \log p(\mathbf{v}|\boldsymbol{\theta}) \geq\ & \mathbf{v}^\top \mathbf{W}^1 \boldsymbol{\mu}^1 + (\boldsymbol{\mu}^1)^\top \mathbf{W}^2 \boldsymbol{\mu}^2 + (\boldsymbol{\mu}^2)^\top \mathbf{W}^3 \boldsymbol{\mu}^2 \\ & - \log Z(\boldsymbol{\theta}) + \mathbb{H}_{\mathbf{q}(\mathbf{h}|\mathbf{v})}[\mathbf{h}] \\ & \triangleq \mathcal{L}(q(\mathbf{h}|\mathbf{v}, \boldsymbol{\mu})). \end{aligned} \tag{4.39}$$

An approximate inference, by following Section 2.3, is performed to fulfill a learning procedure of RBM that corresponds to a type of VB-EM algorithm as addressed in

Section 2.3.3. For each training sample, we first estimate the mean-field parameters $\boldsymbol{\mu}$ by maximizing the variational lower bound $\mathcal{L}(q(\mathbf{h}|\mathbf{v},\boldsymbol{\mu}))$ given the current value of model parameters $\boldsymbol{\theta}$. This optimal variational parameters $\boldsymbol{\mu}$ are estimated to meet the mean-field conditions

$$\mu_j^1 = q(h_j^1 = 1) \leftarrow p(h_j^1 = 1|\mathbf{v},\boldsymbol{\mu}^2,\boldsymbol{\theta}) = \sigma\left(\sum_i W_{ij}^1 v_i + \sum_k W_{jk}^2 \mu_k^2\right) \tag{4.40}$$

$$\mu_k^2 = q(h_k^2 = 1) \leftarrow p(h_k^2 = 1|\boldsymbol{\mu}^1,\boldsymbol{\mu}^3,\boldsymbol{\theta}) = \sigma\left(\sum_j W_{jk}^2 \mu_j^1 + \sum_m W_{km}^3 \mu_m^3\right) \tag{4.41}$$

$$\mu_m^3 = q(h_m^3 = 1) \leftarrow p(h_m^3 = 1|\boldsymbol{\mu}^2,\boldsymbol{\theta}) = \sigma\left(\sum_k W_{km}^3 \mu_k^2\right) \tag{4.42}$$

where the biases in sigmoid function are considered and merged in the summation terms. The VB-E step is completed. Given the variational parameters $\boldsymbol{\mu}$, the lower bound $\mathcal{L}(q(\mathbf{h}|\mathbf{v},\boldsymbol{\mu}))$ is updated and then maximized to find model parameters $\boldsymbol{\theta}$ in the VB-M step. Let $\boldsymbol{\theta}_t$ and $\mathcal{H}_t = \{\mathbf{v}_t, \mathbf{h}_t^1, \mathbf{h}_t^2, \mathbf{h}_t^3\}$ denote the current parameters and the state, respectively. These two variables are sequentially updated and sampled. Continuous calculating $\mathcal{H}_{t+1} \leftarrow \mathcal{H}_t$ is similar to performing the Gibbs sampling. The new parameter $\boldsymbol{\theta}_{t+1}$ is updated by calculating a gradient step where the intractable model expectation $\mathbb{E}_{p_{\text{model}}}[\cdot]$ in the gradient is approximated by a point estimate at sample $\mathcal{H}_{t+1}$ or more precisely calculated from a set of L samples or particles $\{\mathbf{x}_t^{(1)}, \ldots, \mathbf{x}_t^{(L)}\}$. This MCMC-based optimization assures the asymptotic convergence in the learning process. Next, we further address the scheme of stacking autoencoder, which is employed to build a deep neural network model.

4.4 Stacked Autoencoder

It is crucial to explore a meaningful strategy and learn a useful representation for layer-wise structure in a deep neural network. Stacking layers of denoising autoencoder provides an effective approach to deep unsupervised learning [104]. In what follows, we will first address the principle of denoising autoencoder and then introduce the procedure of greedy layer-wise learning for a deep network with the stacked autoencoder.

4.4.1 Denoising Autoencoder

Denoising autoencoder (DAE) [107, 112] is an autoencoder that is trained to produce the original and uncorrupted data point as output when given a corrupted data point as its input. Figure 4.13 illustrates a procedure of finding a reconstructed data $\widehat{\mathbf{x}}$ from a corrupted sample $\widetilde{\mathbf{x}}$ given its original data $\mathbf{x}$ via a deterministic hidden code $\mathbf{h}$. The corrupted version $\widetilde{\mathbf{x}}$ is obtained by means of a stochastic mapping function for data corruption under a predefined probability distribution $\widetilde{\mathbf{x}} \sim q_c(\widetilde{\mathbf{x}}|\mathbf{x})$. DAE aims to learn an autoencoder that maps the corrupted sample $\widetilde{\mathbf{x}}$ to a hidden representation $\mathbf{h}$ via an

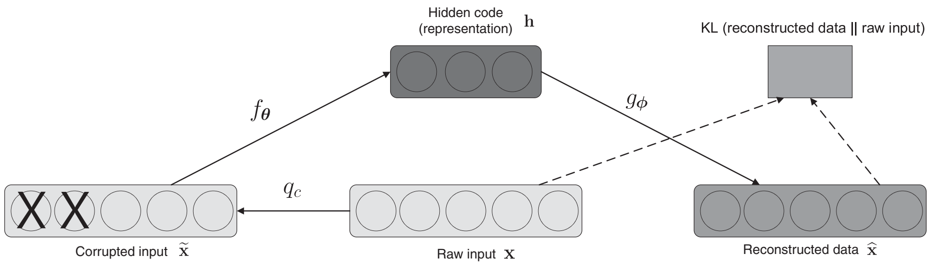

Hidden code (representation) h
KL (reconstructed data || raw input)
f_θ
g_ϕ
q_c
Corrupted input $\widetilde{\mathbf{x}}$
Raw input $\mathbf{x}$
Reconstructed data $\widehat{\mathbf{x}}$

Figure 4.13 An illustration of denosing autoencoder for finding a hidden representation $\mathbf{h}$ from a corrupted input $\widetilde{\mathbf{x}}$. [Based on *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion (Figure 1), by P. Vincent et al., J. of Machine Learning Research, vol. 11, 2010, pp. 3371–3408, MIT Press.*]

encoder $f_{\boldsymbol{\theta}}(\widetilde{\mathbf{x}})$ with parameters $\boldsymbol{\theta}$ and attempts to estimate the reconstructed sample $\widehat{\mathbf{x}}$ via a decoder $\widehat{\mathbf{x}} = g_{\boldsymbol{\phi}}(\mathbf{h})$ with parameters $\boldsymbol{\phi}$. DAE is constructed by learning the parameters of encoder $\boldsymbol{\theta}$ and decoder $\boldsymbol{\phi}$ by minimizing the reconstruction error or the negative log likelihood of decoder output or reconstructed data, i.e., $-\log p(\widehat{\mathbf{x}}|\mathbf{h})$, from the minibatches of training pairs $\{\mathbf{x}, \widetilde{\mathbf{x}}\}$. More specifically, we perform the stochastic gradient descent and minimize the following expectation function

$$\mathcal{L}_{\text{DAE}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x}), \widetilde{\mathbf{x}}\sim q_c(\widetilde{\mathbf{x}}|\mathbf{x})} \left[\log p(\widehat{\mathbf{x}} = g_{\boldsymbol{\phi}}(\mathbf{h})|\mathbf{h} = f_{\boldsymbol{\theta}}(\widetilde{\mathbf{x}}))\right], \tag{4.43}$$

where $p_{\text{data}}(\mathbf{x})$ denotes the distribution of training data. The fulfillment of DAE learning procedure is done by running three steps.

(1) Sample a training sample $\mathbf{x}$ from a collection of training data.
(2) Sample a corrupted version $\widetilde{\mathbf{x}}$ from the corruption model $q_c(\widetilde{\mathbf{x}}|\mathbf{x})$.
(3) Use this data pair $\{\mathbf{x}, \widetilde{\mathbf{x}}\}$ as a training example for estimating the distribution of decoder for reconstruction $p(\widehat{\mathbf{x}}|\mathbf{h})$ or equivalently finding the encoder and decoder parameters $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ by minimizing $\mathcal{L}_{\text{DAE}}(\boldsymbol{\theta}, \boldsymbol{\phi})$.

In the implementation, the encoder and decoder are expressed by a feedforward network that is trained by the standard error backpropagation algorithm as mentioned in Section 4.2.3. The corruption model is represented by an isotropic Gaussian distribution

$$q_c(\widetilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\widetilde{\mathbf{x}}|\mathbf{x}, \sigma^2\mathbf{I}), \tag{4.44}$$

where the corrupted sample $\widetilde{\mathbf{x}}$ is around the the original data $\mathbf{x}$ with a shared variance σ^2 across different dimensions of $\mathbf{x}$.

Figure 4.14 depicts the concept of DAE from a manifold learning perspective. The corrupted sample $\widetilde{\mathbf{x}}$ is shown with a circle of equiprobable corruption. Basically, DAE is trained to learn a *mean vector field*, i.e., $g_{\boldsymbol{\phi}}(f_{\boldsymbol{\theta}}(\widetilde{\mathbf{x}})) - \mathbf{x}$, toward the regions with high probability or low reconstruction error. Different corrupted samples $\widetilde{\mathbf{x}}$ are constructed with vector fields shown by the dashed arrows. A generative model based on DAE is accordingly trained by minimizing the loss function in Eq. 4.43. This objective function corresponds to the negative variational lower bound by using the variational autoencoder, which will be addressed in Section 4.5. In fact, DAE loss function is minimized

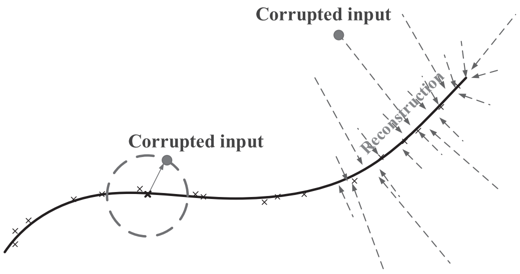


Figure 4.14 A denoising autoencoder that maps corrupted data $\widetilde{\mathbf{x}}$'s to their raw data $\mathbf{x}$'s along a data manifold. [Based on *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion (Figure 2), by P. Vincent et al., J. of Machine Learning Research, vol. 11, 2010, pp. 3371–3408, MIT Press.*]

to realize a maximum likelihood (ML) estimate of DAE parameters $\{\boldsymbol{\theta}, \boldsymbol{\phi}, \sigma^2\}$. Such an estimation in DAE is similar to fulfill the regularized score matching [113] or the denoising score matching [107] on an RBM with Gaussian visible unites. The score matching aims to find the model distribution p_{model}, which is encouraged to produce the same likelihood score as the data distribution p_{data} at every training point $\mathbf{x}$. The robustness to noise corruption is assured in this specialized and regularized RBM. In the next section, the denoising autoencoder is treated as a building block to perform unsupervised stacking for a deep neural network.

4.4.2 Greedy Layer-Wise Learning

This section addresses how the autoencoder or RBM is stacked to carry out the greedy layer-wise learning for a deep structural model. Uncorrupted encoding is acted as the inputs for next level of stacking. Uncorrupted inputs are reconstructed in each stacking. Briefly speaking, we first start with the lowest level and stack upwards. Second, each layer of autoencoder is trained by using the intermediate codes or features from the layer below. Third, top layer can have a different output, e.g., softmax nonlinearity, to provide an output for classification. Figures 4.15(a), (b), and (c) demonstrate three steps toward constructing a deep classification network.

(1) Autoencoder or RBM is trained from visible data $\mathbf{x}$ to explore the hidden units $\mathbf{h}^1$ in the first hidden layer by using the first-level parameter $\mathbf{W}^1$. The transpose of parameters $(\mathbf{W}^1)^\top$ is used to implement an inverse of model or equivalently to find the reconstructed data $\widehat{\mathbf{x}}$.

(2) Reconstructed data $\widehat{\mathbf{x}}$ are masked by fixing the trained parameter $\mathbf{W}^1$ or $(\mathbf{W}^1)^\top$. The second hidden layer with hidden units $\mathbf{h}^2$ is explored by training the second-level parameter $\mathbf{W}^2$ or its transpose $(\mathbf{W}^2)^\top$ to reconstruct the hidden units $\widehat{\mathbf{h}}^1$ in the second hidden layer.

(3) Labels $\mathbf{y}$ of the visible data $\mathbf{x}$ are then used to train the third-level parameter $\mathbf{U}$ where the reconstructions $\widehat{\mathbf{x}}$ and $\widehat{\mathbf{h}}^1$ are masked by fixing the inverse parameters $\{(\mathbf{W}^1)^\top, (\mathbf{W}^2)^\top\}$. Three levels of parameters $\{\mathbf{W}^1, \mathbf{W}^2, \mathbf{U}\}$ are further fine-tuned by using error backpropagation algorithm via minimization of classification loss.

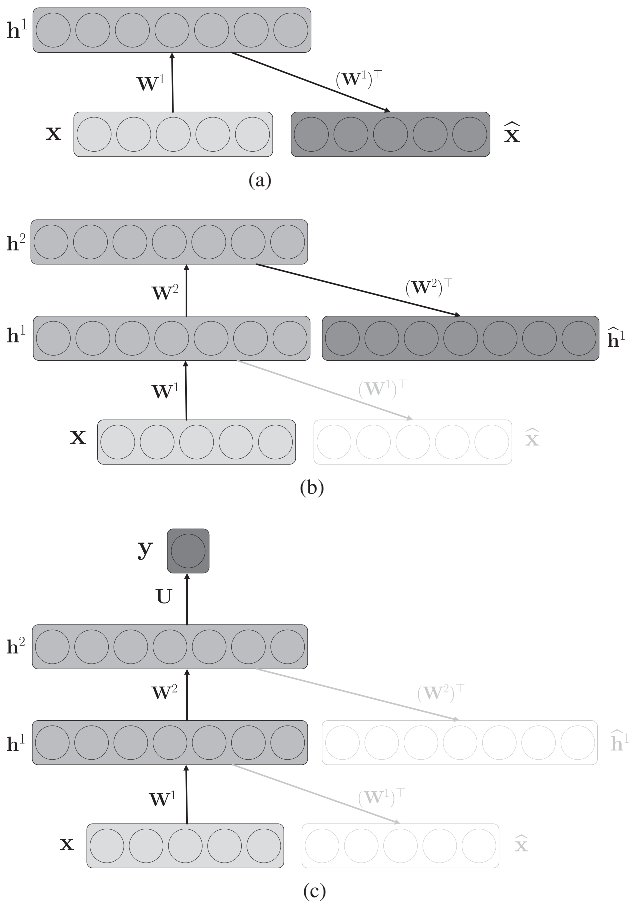


Figure 4.15 Procedure of stacking autoencoders for the construction of a deep classification network with input $\mathbf{x}$ and classification output $\mathbf{y}$.

Alternatively, Figures 4.16(a), (b), and (c) show a different greedy procedure to construct a deep classification network where the output is predicted directly for class label $\widehat{\mathbf{y}}$ rather than indirectly for training sample $\widehat{\mathbf{x}}$. In the first step, the hidden units $\mathbf{h}^1$ are discovered from visible data $\mathbf{v}$ by using the first-level parameter $\mathbf{W}^1$. The reconstructed class label $\widehat{\mathbf{y}}$ is then calculated by using transformation parameter $\mathbf{U}^1$. However, the two-layer model is insufficient to estimate a good class label $\widehat{\mathbf{y}}$. The second step is presented to explore a deeper model to $\mathbf{h}^2$ and use this higher abstraction to predict class label $\widehat{\mathbf{y}}$ via an updated higher-level parameter $\mathbf{U}^2$ while the estimated class label $\widehat{\mathbf{y}}$ using lower-level parameter $\mathbf{U}^1$ is masked. The whole model with three layers is fine-tuned by using error backpropagation.

In general, using DAE, there is no partition function in the training criterion. The encoder and decoder can be represented by any parametric functions although feedforward neural networks have been widely adopted. The experiments on classification tasks in [104] showed that the unsupervised pretraining based on greedy layer-wise learning using RBM and DAE worked well for the construction of deep classification networks.

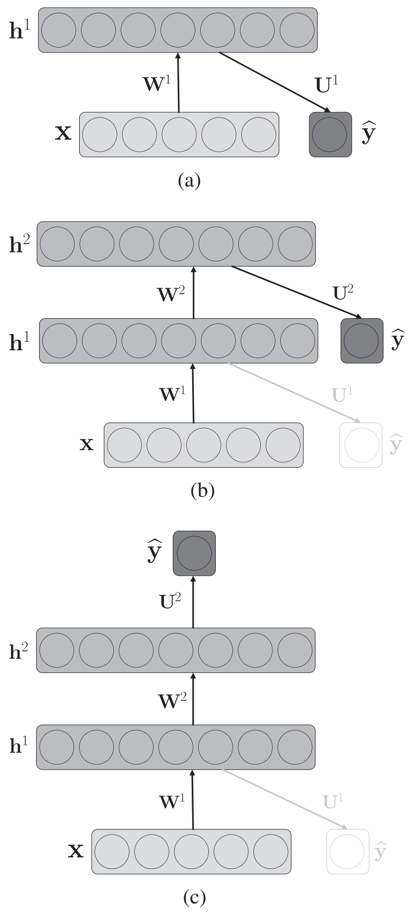


Figure 4.16 An alternative procedure for greedy layer-wise learning in a deep classification network where the output layer produces the reconstructed class label $\widehat{\mathbf{y}}$.

When the number of training examples is increased, the stacking DAE performs better than stacking RBM.

Although the restricted Boltzmann machine and denoising autoencoder are developed as the building block for greedy layer-wise learning of a deep neural network, the hidden units in different layers $\{\mathbf{h}^k\}_{k=1}^{l}$ are seen as the deterministic variables without uncertainty modeling. The interpretation of model capacity is therefore constrained. For example, in real-world speaker recognition systems, the utterances or feature vectors of a target speaker are usually collected in adverse environments with different contents and lengths in presence of various noise types and noise levels. It is challenging to build a robust deep model that can accommodate the variations of speech patterns from abundant as well as sparse training utterances. Uncertainty modeling or Bayesian learning (as described in Section 2.5) becomes crucial in the era of deep learning. In the next section, we will introduce the variational autoencoder where the hidden units are considered random.

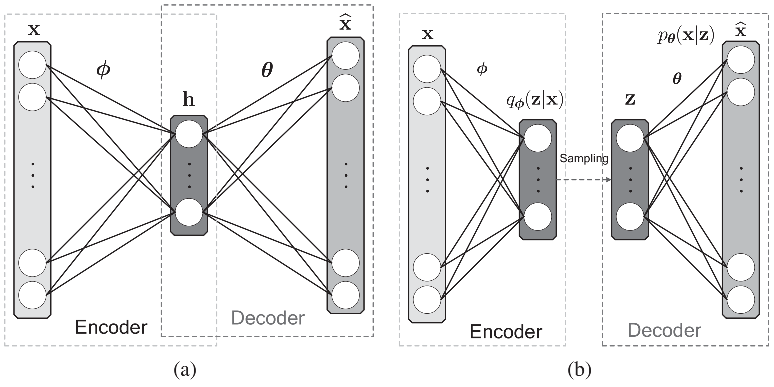


Figure 4.17 Network structures for (a) autoencoder and (b) variational autoencoder. [Adapted from *Variational Recurrent Neural Networks for Speech Separation (Figure 2), by J.T. Chien and K.T. Kuo, Proceedings Interspeech, 2017, pp. 1193–1197, ISCA.*]

4.5 Variational Autoencoder

Figures 4.17(a) and (b) show the model structures for an autoencoder and a variational autoencoder, respectively, which are designed to find the reconstructed data $\widehat{\mathbf{x}}$ corresponding to its original data $\mathbf{x}$ based on an encoder and decoder. A traditional autoencoder is seen as a deterministic model to extract the deterministic latent features $\mathbf{h}$. In [114], the variational autoencoder (VAE) was developed by incorporating the variational Bayesian learning into the construction of an autoencoder that is used as a building block for different stochastic neural networks [115, 116]. Uncertainty modeling is performed to build a latent variable model for data generation through stochastic hidden features $\mathbf{z}$. As addressed in Section 2.3, the variational inference involves the optimization of an approximation to an intractable posterior distribution. The standard mean-field approach requires analytical derivations to calculate the expectations with respect to the approximate posterior. Such a calculation is obviously intractable in the case of neural networks. A VAE provides an approach to illustrate how the reparameterization of variational lower bound is manipulated to yield a simple differentiable and unbiased estimator, also called the stochastic gradient variational Bayes method. This estimator is used to calculate the posterior distribution and carry out the approximate inference in presence of continuous latent variables. The optimization is implemented according to a standard stochastic gradient ascent technique as detailed in what follows.

4.5.1 Model Construction

Given a data collection $\mathbf{X} = \{\mathbf{x}_t\}_{t=1}^T$ consisting of T i.i.d. (identically, independently distributed) samples of continuous observation variable $\mathbf{x}$, we aim to train a generative model to synthesize a new value of $\mathbf{x}$ from its latent variable $\mathbf{z}$. The latent variable $\mathbf{z}$ and observation data $\mathbf{x}$ are generated by prior distribution $p_{\boldsymbol{\theta}}(\mathbf{z})$ and likelihood function $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, respectively, but the true parameters $\boldsymbol{\theta}$, as well as the values of $\mathbf{z}$, are unknown.

Notably, there is no assumption or simplification on the marginal distribution $p(\mathbf{x})$ or posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. To deal with the issue that marginal likelihood or posterior distribution is intractable, the variational Bayes algorithm is introduced to approximate the logarithm of marginal likelihood of individual data points $\log p(\mathbf{x})$ or

$$\log p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \sum_{t=1}^{T} \log p(\mathbf{x}_t) \tag{4.45}$$

through maximizing the variational lower bound $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$ derived by

$$\begin{aligned}
\log p(\mathbf{x}) &= \log \sum_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z}) \\
&= \log \sum_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} p_{\boldsymbol{\theta}}(\mathbf{z}) \\
&= \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right] \\
&\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \Bigg[\underbrace{\log \left(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \frac{p_{\boldsymbol{\theta}}(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right)}_{f_{\boldsymbol{\Theta}}(\mathbf{x},\mathbf{z})} \Bigg] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \right] - \mathcal{D}_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})) \triangleq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})
\end{aligned} \tag{4.46}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ denote the whole parameter set of a VAE. By subtracting this lower bound from the log-marginal distribution, as referred to Eqs. 2.32 and 2.33, we obtain the relation

$$\log p(\mathbf{x}) = \mathcal{D}_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}). \tag{4.47}$$

Maximizing the likelihood function $p(\mathbf{x})$ turns out to maximizing lower bound $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$ while the KL divergence term is minimized, or equivalently, the variational distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is estimated to be maximally close to the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$.

Figure 4.18 shows the graphical representation of generative model based on a VAE. The variational autoencoder is composed of two parts. One is the recognition model or probabilistic encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, which is an approximation to the intractable true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The other part is the generative model or probabilistic decoder $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, which generates the data sample $\mathbf{x}$ from latent code $\mathbf{z}$. Note that in contrast with the approximate posterior in mean-field variational inference, the factorized inference is not required in a VAE. Variational parameter $\boldsymbol{\phi}$ and model parameter $\boldsymbol{\theta}$ are not computed via a closed-form expectation. All parameters are jointly learned with this network structure by maximizing the variational lower bound $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$, as shown in Eq. 4.46, which consists of two terms: the KL divergence term corresponds to the optimization of the parameters for recognition model $\boldsymbol{\phi}$, and the other term corresponds to maximizing the log-likelihood of training samples given the generative model $\boldsymbol{\theta}$.

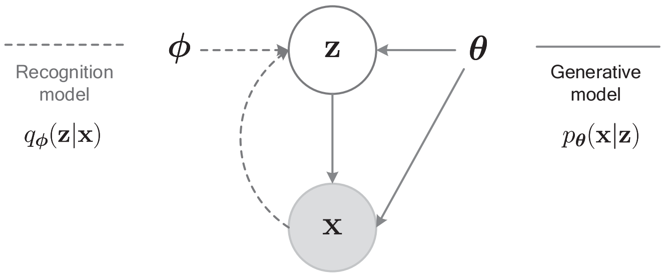


Figure 4.18 Graphical model for variational autoencoder. Solid lines denote the generative model $p_\theta(\mathbf{x}|\mathbf{z})$ and dash lines denote the variational approximation $q_\phi(\mathbf{z}|\mathbf{x})$ to the intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$. [Adapted from *Variational Recurrent Neural Networks for Speech Separation (Figure 2), by J.T. Chien and K.T. Kuo, Proceedings Interspeech, 2017, pp. 1193–1197, ISCA.*]

4.5.2 Model Optimization

Different from traditional error backpropagation algorithm developed for training neural networks with deterministic hidden variables $\mathbf{h}$, the stochastic neural network, i.e., the variational autoencoder in this section, is trained according to the stochastic backpropagation [117] where the random hidden variable $\mathbf{z}$ is considered in the optimization of the variational lower bound.

Stochastic Backpropagation

There is no closed-form solution for the expectation in the objective function of a VAE in Eq 4.46. The approximate inference based on sampling method, as introduced in Section 2.4, was employed in model inference for a VAE [114]. To implement the stochastic backpropagation, the expectation with respect to variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ in a VAE is simply approximated by sampling a hidden variable $\mathbf{z}^{(l)}$ and using this random sample to calculate the gradient for parameter updating. For example, the Gaussian distribution with mean vector $\boldsymbol{\mu}_z$ and diagonal covariance matrix $\boldsymbol{\sigma}_z^2\mathbf{I}$

$$\mathbf{z}^{(l)} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z(\mathbf{x}), \boldsymbol{\sigma}_z^2(\mathbf{x})\mathbf{I}) \tag{4.48}$$

is popularly assumed and used to find the M-dimensional random sample of continuous-density latent variable $\mathbf{z}^{(l)}$. Given the sample $\mathbf{z}^{(l)}$, the lower bound can be approximated by a *single sample* using

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[f_\Theta(\mathbf{x}, \mathbf{z})\right] \\ &\simeq f_\Theta(\mathbf{x}, \mathbf{z}^{(l)}).\end{aligned} \tag{4.49}$$

Figure 4.19(a) shows three steps toward fulfilling the stochastic gradient estimator for model parameter and variational parameter $\boldsymbol{\Theta} = \{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ where the single sample $\mathbf{z}^{(l)}$ is used in the expectation.

(1) Sample the latent variable $\mathbf{z}^{(l)}$ from $q_\phi(\mathbf{z}|\mathbf{x})$.
(2) Replace the variable $\mathbf{z}$ by $\mathbf{z}^{(l)}$ to obtain the objective $\mathcal{L} \simeq f_\Theta(\mathbf{x}, \mathbf{z}^{(l)})$.

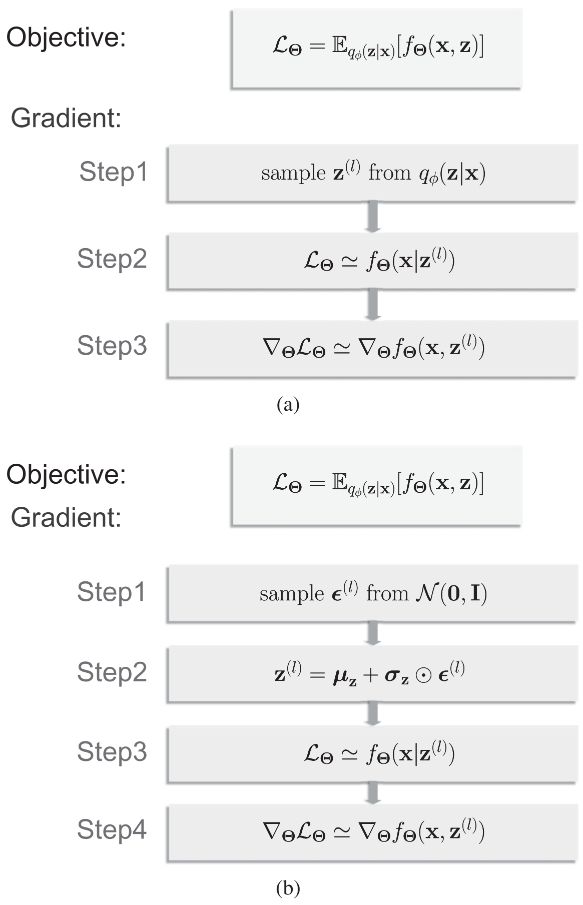

Objective:
$\mathcal{L}_\Theta = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f_\Theta(\mathbf{x},\mathbf{z})]$
Gradient:
Step1
sample $\mathbf{z}^{(l)}$ from $q_\phi(\mathbf{z}|\mathbf{x})$
Step2
$\mathcal{L}_\Theta \simeq f_\Theta(\mathbf{x}|\mathbf{z}^{(l)})$
Step3
$\nabla_\Theta \mathcal{L}_\Theta \simeq \nabla_\Theta f_\Theta(\mathbf{x},\mathbf{z}^{(l)})$
(a)
Objective:
$\mathcal{L}_\Theta = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f_\Theta(\mathbf{x},\mathbf{z})]$
Gradient:
Step1
sample $\boldsymbol{\epsilon}^{(l)}$ from $\mathcal{N}(\mathbf{0},\mathbf{I})$
Step2
$\mathbf{z}^{(l)} = \boldsymbol{\mu}_\mathbf{z} + \boldsymbol{\sigma}_\mathbf{z} \odot \boldsymbol{\epsilon}^{(l)}$
Step3
$\mathcal{L}_\Theta \simeq f_\Theta(\mathbf{x}|\mathbf{z}^{(l)})$
Step4
$\nabla_\Theta \mathcal{L}_\Theta \simeq \nabla_\Theta f_\Theta(\mathbf{x},\mathbf{z}^{(l)})$
(b)

Figure 4.19 Objectives and implementation steps for (a) stochastic backpropagation and (b) stochastic gradient variational Bayes.

(3) Individually take gradient of $\mathcal{L}$ with respect to parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to obtain $\nabla_{\boldsymbol{\Theta}} \mathcal{L} \simeq \nabla_{\boldsymbol{\Theta}} f_{\boldsymbol{\Theta}}(\mathbf{x}, \mathbf{z}^{(l)})$.

As shown in Figure 4.17(b), the encoder of a VAE is configured as a two-layer neural network with variational parameter $\boldsymbol{\phi}$ where the D-dimensional input data $\mathbf{x}$ is used to estimate the variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. Practically, the hidden layer is arranged with M units for mean vector $\boldsymbol{\mu}_z(\mathbf{x})$ and M units for diagonal covariance matrix $\boldsymbol{\sigma}_z^2(\mathbf{x})\mathbf{I}$, which are both functions of input data $\mathbf{x}$. Hidden variable $\mathbf{z}$ corresponding to input $\mathbf{x}$ is then sampled from this Gaussian distribution.

Although the standard gradient estimator is valid by using variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, the problem of *high variance* will happen [117] if latent variable $\mathbf{z}$ is *directly* sampled from a Gaussian distribution $\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z(\mathbf{x}), \boldsymbol{\sigma}_z^2(\mathbf{x})\mathbf{I})$. To deal with this problem, a reparameterization trick [114] is introduced and applied as the variance reduction method, which will be detailed in the following section.

Stochastic Gradient Variational Bayes

Stochastic gradient variational Bayes (SGVB) is a method to address the high variance issue in a VAE. In this implementation, the sampling based on a chosen distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is executed via a reparameterization trick. This trick is performed by reparameterizing the random variable $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ using a differentiable transformation function $g_\phi(\mathbf{x}, \boldsymbol{\epsilon})$ with an auxiliary random noise $\boldsymbol{\epsilon}$

$$\widetilde{\mathbf{z}} = g_\phi(\mathbf{x}, \boldsymbol{\epsilon}) \tag{4.50}$$

with $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ where $p(\boldsymbol{\epsilon})$ is chosen for an appropriate distribution. Basically, the M-dimensional standard Gaussian distribution

$$p(\boldsymbol{\epsilon}) = \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I}) \tag{4.51}$$

is adopted. In this way, the latent variable $\mathbf{z}$ is viewed as a deterministic mapping function with deterministic parameter $\boldsymbol{\phi}$ but driven by a standard Gaussian variable $\boldsymbol{\epsilon}$. A stable *random* variable sampled by the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ is then approximated by the mapping function $g_\phi(\mathbf{x}, \boldsymbol{\epsilon})$. The reduction of variance in sampling can be assured. More specially, we can calculate the Monte Carlo estimate of the expectation of a target function $f(\mathbf{z})$ with respect to $q_\phi(\mathbf{z}|\mathbf{x})$ by using L samples of $\mathbf{z}$ drawn from $q_\phi(\mathbf{z}|\mathbf{x})$, by referring to Eqs. 2.50 and 2.51, via the reparameterization trick

$$\begin{aligned}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(g_\phi(\mathbf{x}, \boldsymbol{\epsilon}))] \\ &\simeq \frac{1}{L}\sum_{l=1}^{L} f(g_\phi(\mathbf{x}, \boldsymbol{\epsilon}^{(l)}))\end{aligned} \tag{4.52}$$

where $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$. An SGVB estimator is then developed by maximizing the variational lower bound in Eq. 4.46, which is written in a form of

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\mathbf{x}|\mathbf{z}^{(l)}) - \mathcal{D}_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \tag{4.53}$$

where $\mathbf{z}^{(l)} = g_\phi(\mathbf{x}, \boldsymbol{\epsilon}^{(l)})$ and $\boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})$. In this way, the variance of the gradient simply depends on the noise term $\boldsymbol{\epsilon}$, which is sampled from distribution $p(\boldsymbol{\epsilon})$. When compared with directly sampling of $\mathbf{z}$, the variance of sampling $p(\boldsymbol{\epsilon})$ is considerably reduced.

Algorithm 2 Autoencoding variational Bayes algorithm

```
Initialize hidden state parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$
For parameters ($\boldsymbol{\theta}$, $\boldsymbol{\phi}$) until convergence
    $\mathbf{X}_n \leftarrow$ Random minibatch with $T_n$ data points from full dataset $\mathbf{X}$
    $\boldsymbol{\epsilon} \leftarrow$ Random samples from noise distribution $p(\boldsymbol{\epsilon})$
    $\mathbf{g} \leftarrow \nabla_{\theta,\phi}\widetilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}_n, \boldsymbol{\epsilon})$ (Gradient of Eq. (4.54))
    $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Update parameters using $\mathbf{g}$
End For
Return $\boldsymbol{\theta}, \boldsymbol{\phi}$
```

4.5.3 Autoencoding Variational Bayes

Based on the model construction in Section 4.5.1 and the model optimization in Section 4.5.2, the autoencoding variational Bayes (AEVB) algorithm is developed to fulfill the building block of variational autoencoder for deep learning. AEVB learning for variational autoencoder is shown in Algorithm 2. From the full data set $\mathbf{X} = \{\mathbf{x}_t\}_{t=1}^{T}$ with T data points, we first randomly sample a minibatch $\mathbf{X}_n = \{\mathbf{x}_t\}_{t=1}^{T_n}$ with T_n data points and calculate an estimator of variational lower bound of log marginal likelihood using a VAE as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}) \simeq \frac{T}{T_n} \widetilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}_n, \boldsymbol{\epsilon}) = \frac{T}{T_n} \sum_{t=1}^{T_n} \widetilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}_t, \boldsymbol{\epsilon}), \tag{4.54}$$

where

$$T = \sum_n T_n \tag{4.55}$$

and a random noise $\boldsymbol{\epsilon}$ is used by sampling from a standard Gaussian distribution ϵ. Then, the gradients of variational lower bound with respect to a VAE parameters $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ are calculated to perform the stochastic optimization based on the stochastic gradient ascent or the adaptive gradient (AdaGrad) algorithm [107] via

$$\{\boldsymbol{\theta}, \boldsymbol{\phi}\} \leftarrow \{\boldsymbol{\theta}, \boldsymbol{\phi}\} + \eta \underbrace{\nabla_{\boldsymbol{\theta}, \boldsymbol{\phi}} \widetilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}_n, \boldsymbol{\epsilon})}_{\triangleq \mathbf{g}}. \tag{4.56}$$

In the implementation, an isotropic Gaussian distribution is adopted as the prior density of latent variable

$$p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}). \tag{4.57}$$

Likelihood function $p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z})$ for the reconstructed data $\widehat{\mathbf{x}}_t$ is either modeled by a multivariate Gaussian distribution for real-valued data or a Bernoulli distribution for binary data where the distribution parameters are computed from $\mathbf{z}$ by using a decoder neural network with parameter $\boldsymbol{\theta}$. More specifically, AEVB algorithm is performed to maximize the variational lower bound jointly for the variational parameter $\boldsymbol{\theta}$ in the encoder and the model parameter $\boldsymbol{\phi}$ in the decoder. The variational posterior distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_t)$ is approximated by a multivariate Gaussian with parameters calculated by an encoder neural network $[\boldsymbol{\mu}_z(\mathbf{x}_t), \boldsymbol{\sigma}_z^2(\mathbf{x}_t)] = f_{\boldsymbol{\phi}}^{\text{enc}}(\mathbf{x}_t)$. As explained in Section 4.5.2, we sample the M-dimensional posterior latent variable by

$$\mathbf{z}^{(l)} = g_{\boldsymbol{\phi}}(\mathbf{x}_t, \boldsymbol{\epsilon}^{(l)}) = \boldsymbol{\mu}_z(\mathbf{x}_t) + \boldsymbol{\sigma}_z(\mathbf{x}_t) \odot \boldsymbol{\epsilon}^{(l)}, \tag{4.58}$$

where $\odot$ denotes the element-wise product of two vectors and $\boldsymbol{\epsilon}^{(l)}$ is a standard Gaussian variable with $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The resulting estimator for the variational lower bound in Eq. 4.54 using a speech observation $\mathbf{x}_t$ is expressed by

$$\widetilde{\mathcal{L}}(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x}_t,\epsilon) \simeq \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{z}^{(l)}) + \underbrace{\frac{1}{2}\sum_{j=1}^{M}\Big(1+\log(\sigma_{z,j}^2(\mathbf{x}_t)) - \mu_{z,j}^2(\mathbf{x}_t) - \sigma_{z,j}^2(\mathbf{x}_t)\Big)}_{-\mathcal{D}_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}))}, \tag{4.59}$$

where the second term on the right-hand side is calculated as a negative KL divergence between the variational posterior distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and the standard Gaussian distribution $p_{\boldsymbol{\theta}}(\mathbf{z})$. The KL term can be derived as

$$-\mathcal{D}_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z})) = \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\big(\log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\big)\,d\mathbf{z} \tag{4.60}$$

where

$$\begin{aligned}\int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\log p_{\boldsymbol{\theta}}(\mathbf{z})d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z(\mathbf{x}_t),\boldsymbol{\sigma}_z^2(\mathbf{x}_t))\log\mathcal{N}(\mathbf{z}|\mathbf{0},\mathbf{I})d\mathbf{z} \\ &= -\frac{M}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{M}\Big(\mu_{z,j}^2(\mathbf{x}_t) + \sigma_{z,j}^2(\mathbf{x}_t)\Big)\end{aligned} \tag{4.61}$$

and

$$\begin{aligned}\int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z(\mathbf{x}_t),\boldsymbol{\sigma}_z^2(\mathbf{x}_t))\log\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z(\mathbf{x}_t),\boldsymbol{\sigma}_z^2(\mathbf{x}_t))d\mathbf{z} \\ &= -\frac{M}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{M}(1+\log\sigma_j^2).\end{aligned} \tag{4.62}$$

The AEVB algorithm is accordingly implemented to construct the variational autoencoder with parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. This type of autoencoder can be stacked in a way as addressed in Section 4.4 to build the deep regression or classification network for speaker recognition. In general, a VAE is sufficiently supported by exploring the randomness of latent code $\mathbf{z}$. The capability as a generative model is assured to synthesize an artificial sample $\widehat{\mathbf{x}}$ from a latent variable $\mathbf{z}$ where the global structure of observation data are preserved to minimize the reconstruction error. In what follows, we introduce another paradigm of generative model, called the generative adversarial network, which produces artificial samples based on adversarial learning.

4.6 Generative Adversarial Networks

This section addresses the pros and cons of the generative models based on the generative adversarial network (GAN) [118] that has been extensively studied and developed for different applications including adversarial learning for speaker recognition [119, 120]. There are two competing neural networks that are estimated mutually through adversarial learning. Several generative models based on deep learning and maximum likelihood estimation have been discussed in Sections 4.1 and 4.5. Before explaining the details of GANs, we would like to point out the key idea and property where GANs perform differently when compared with the other generative models.

4.6.1 Generative Models

Figure 4.20 depicts the taxonomy of generative models. In general, the parameter of a generative model is estimated according to the maximum likelihood (ML) principle where a probabilistic model or distribution $p_{\text{model}}(\mathbf{x}|\boldsymbol{\theta})$ with parameter $\boldsymbol{\theta}$ characterize how a speech observation $\mathbf{x}$ is obtained, namely $\mathbf{x} \sim p_{\text{model}}(\mathbf{x}|\boldsymbol{\theta})$. Training samples are collected or drawn from an unknown data distribution by $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$. The ML method assumes that there exists a parameter $\boldsymbol{\theta}$ that make the model distribution exactly the same as the true data distribution

$$p_{\text{model}}(\mathbf{x}|\boldsymbol{\theta}) = p_{\text{data}}(\mathbf{x}). \tag{4.63}$$

The ML parameter $\boldsymbol{\theta}$ of a model distribution p_{model} is then estimated by maximizing the log likelihood function of training data collected from data distribution p_{data}, i.e., solving the optimization problem

$$\boldsymbol{\theta}_{\text{ML}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log p_{\text{model}}(\mathbf{x}|\boldsymbol{\theta})]. \tag{4.64}$$

Traditionally, the ML solution is derived through an *explicit* model distribution where a specific distribution is assumed. Such a model distribution is possibly intractable. An approximate distribution is accordingly required. The approximation can be either based on the variational inference or driven by the Markov chain. The variational autoencoder in Section 4.5 adopts the variational distribution, which is a Gaussian, to approximate a true posterior distribution in variational inference. The restricted Boltzmann machine in Section 4.1 employs the explicit distributions of Bernoulli and Gaussian in a style of distribution approximation based on a Markov chain using the contrastive divergence

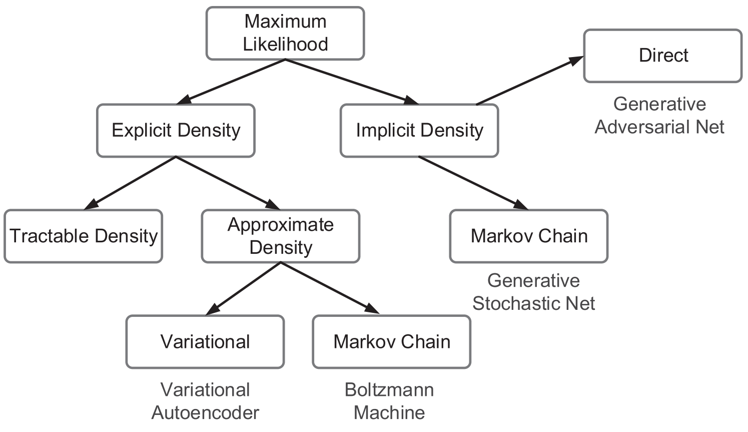


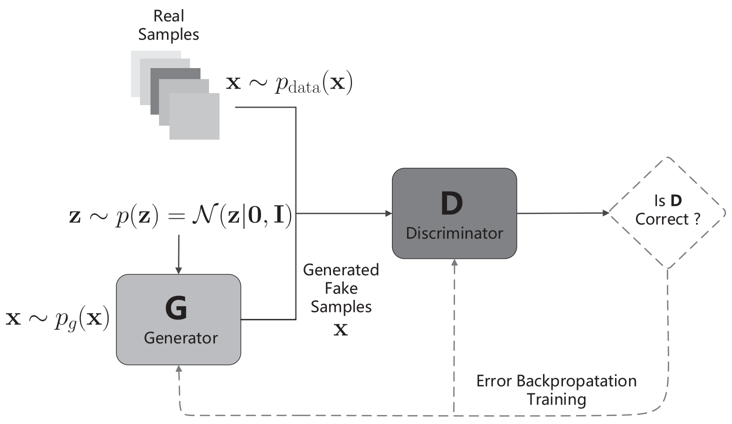
Figure 4.20 Taxonomy of generative models based on maximum likelihood estimation.

algorithm. On the other hand, the ML model can also be constructed without assuming any explicit distribution. In [121], the generative stochastic network was proposed to carry out a Markov chain to find state transitions for updating a generative model. An implicit distribution was actually adopted to drive the construction of the generative model.

In what follows, we are introducing the generative model based on the generative adversarial network. GAN presents a general idea to learn a sampling mechanism for data generation. The generation process is based on the sampling of a latent variable $\mathbf{z}$ by using a generator network G. The generated sample is distinguished from the true data sample by using a discriminator network D. There is no explicit density assumed in the optimization procedure. There is no Markov chain involved. The issue of *mixing* between the updated models of two states is mitigated. The property of asymptotically consistent performance is assured. GAN is likely to produce the best samples through the adversarial learning. GAN is seen as a *direct* model that directly estimates the data distribution without assuming distribution function. Some theoretical justification is provided to illustrate why GAN is powerful to estimate true data distribution as addressed in what follows.

4.6.2 Adversarial Learning

Generative adversarial network is a generative model consisting of two competing neural networks as illustrated in Figure 4.21. One of the competing neural networks is the generator G that takes an random hidden variable $\mathbf{z}$ from $p(\mathbf{z})$ where $\mathbf{z}$ is a sample from probability distribution $p(\mathbf{z})$ and is used to generate an artificial or fake sample. The other model is the discriminator D that receives samples from both the generator and the real samples. The discriminator is to take the input either from the real data or from the generator and tries to predict whether the input is real or generated. It takes an input $\mathbf{x}$



Real Samples
$\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$
$\mathbf{z} \sim p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0},\mathbf{I})$
D
Discriminator
Is D Correct ?
$\mathbf{x} \sim p_g(\mathbf{x})$
G
Generator
Generated Fake Samples $\mathbf{x}$
Error Backpropatation Training

Figure 4.21 Illustration for generative adversarial network. [Adapted from *Adversarial Learning and Augmentation for Speaker Recognition (Figure 1), by J.T. Chien and K.T. Peng, Proceedings Odyssey 2018 The Speaker and Language Recognition Workshop, pp. 342–348, ISCA.*]

from real data distribution $p_{\text{data}}(\mathbf{x})$. The discriminator then solves a binary classification problem using a sigmoid function giving output in the range 0 to 1. More specifically, GAN is formulated as a two-player game between a generator with mapping function

$$G(\mathbf{z}) : \mathbf{z} \to \mathbf{x} \tag{4.65}$$

and a binary discriminator with the mapping

$$D(\mathbf{x}) : \mathbf{x} \to [0, 1] \tag{4.66}$$

via a *minimax* optimization over a value function $V(G, D)$ or an adversarial loss $\mathcal{L}_{\text{adv}}$. This minimax game is expressed by a joint optimization over G and D according to

$$\min_{G} \max_{D} V(G, D). \tag{4.67}$$

GAN aims to pursue a discriminator D, which optimally classifies if the input comes from true sample $\mathbf{x}$ (with discriminator output $D(\mathbf{x})$) or the fake sample $G(\mathbf{z})$ using a random code $\mathbf{z}$ for generator G (with discriminator output $D(G(\mathbf{z}))$). Meaningfully, the value function is formed as a *negative cross-entropy error* function for binary classification in a two-player game

$$\begin{aligned} V(G, D) = {} & \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] \\ & + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \triangleq \mathcal{L}_{\text{adv}}, \end{aligned} \tag{4.68}$$

where $p_{\text{data}}(\mathbf{x})$ is data distribution and $p(\mathbf{z})$ is a distribution that draws a noise sample or latent code $\mathbf{z}$. Similar to a VAE in Section 4.5, GAN adopts the standard Gaussian distribution as the prior density of $\mathbf{z}$

$$\mathbf{z} \sim p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}). \tag{4.69}$$

Using GAN, $G(\mathbf{z})$ captures the data distribution and generates a sample $\mathbf{x}$ from $\mathbf{z}$. The discriminator produces a probability $0 \leq D(\mathbf{x}) \leq 1$, which measures how likely a data point $\mathbf{x}$ is sampled from the data distribution $p_{\text{data}}(\mathbf{x})$ as a real sample or from the generator $G(\mathbf{z})$ (with a distribution of generator $p_g(\mathbf{x})$ or equivalently a model distribution $p_{\text{model}}(\mathbf{x})$) as a fake sample. Correspondingly, discriminator D classifies a sample coming from the training data rather than G.

The generator G is trained via maximizing the probability of discriminator D to make a mistake. Generator G minimizes the classification accuracy of D. Generator $G(\mathbf{z})$ and discriminator $D(\mathbf{x})$ are both realized as the fully connected neural network models. This minimax optimization assures the *worst* performance in which a fake sample is classified as a real sample. A powerful generator model $G(\mathbf{z})$ is therefore implemented.

4.6.3 Optimization Procedure

There are two steps in the optimization procedure for training the discriminator and the generator.

First Step: Finding the Optimal Discriminator

To derive the formula for an optimal discriminator, we first rewrite the value function in Eq. 4.67 by

$$\begin{aligned} V(G, D) &= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log\left(1 - D(G(\mathbf{z}))\right)] \\ &= \int p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x}) d\mathbf{x} + \int p(\mathbf{z}) \log\left(1 - D(G(\mathbf{z}))\right) d\mathbf{z} \\ &= \int \left\{ p_{\text{data}}(\mathbf{x}) \log(D(\mathbf{x})) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) \right\} d\mathbf{x}, \end{aligned} \tag{4.70}$$

where the variable $\mathbf{z}$ in the second term in the right-hand side is manipulated and expressed by using the variable $\mathbf{x}$ so that $p(\mathbf{z})$ is expressed by $p_g(\mathbf{x})$ and $D(G(\mathbf{z})) = D(\mathbf{x})$. Taking the derivative of Eq. 4.70 with respect to the discriminator $D(\mathbf{x})$, the optimal discriminator is derived for any given generator G with a distribution $p_g(\mathbf{x})$ in a form of

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \tag{4.71}$$

A global optimum happens in the condition that the generator or model distribution is exactly the same as data distribution

$$p_g(\mathbf{x}) \text{ (or } p_{\text{model}}(\mathbf{x})) = p_{\text{data}}(\mathbf{x}), \tag{4.72}$$

which results in the worst performance in binary classification using the discriminator. Namely, no matter the input $\mathbf{x}$ to the discriminator is either drawn from the data distribution $p_{\text{data}}(\mathbf{x})$ or from the model distribution $p_g(\mathbf{x})$, the classification accuracy is always 50 percent because

$$D^*(\mathbf{x})\Big|_{p_g(\mathbf{x})=p_{\text{data}}(\mathbf{x})} = 0.5. \tag{4.73}$$

Second Step: Finding the Optimal Generator

By substituting this discriminator into value function, the second step of GAN is to find the optimal generator G by solving a minimization problem

$$\begin{aligned} \min_G V(G, D^*) &= \min_G C(G) \\ &= \min_G \ \{2\mathcal{D}_{\text{JS}}(p_{\text{data}}(\mathbf{x}) \| p_g(\mathbf{x})) - \log 4\}, \end{aligned} \tag{4.74}$$

where $\mathcal{D}_{\text{JS}}(\cdot\|\cdot)$ denotes the Jensen–Shannon (JS) divergence. JS divergence quantifies how distinguishable two distributions p and q are from each other. It is related to the KL divergence by

$$\mathcal{D}_{\text{JS}}(p\|q) = \frac{1}{2}\mathcal{D}_{\text{KL}}\left(p \middle\| \frac{p+q}{2}\right) + \frac{1}{2}\mathcal{D}_{\text{KL}}\left(q \middle\| \frac{p+q}{2}\right). \tag{4.75}$$

In Eq. 4.74, the detailed derivation of the cost function $C(G)$ for finding optimal generator is shown as

$$\begin{aligned}
C(G) &= \max_D V(G,D) \\
&= \max_D \int \left\{p_{\text{data}}(\mathbf{x})\log(D(\mathbf{x})) + p_g(\mathbf{x})\log(1-D(\mathbf{x}))\right\} d\mathbf{x} \\
&= \int \left\{p_{\text{data}}(\mathbf{x})\log(D^*(\mathbf{x})) + p_g(\mathbf{x})\log(1-D^*(\mathbf{x}))\right\} d\mathbf{x} \\
&= \int \left\{p_{\text{data}}(\mathbf{x})\log\left(\frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}\right) + p_g(\mathbf{x})\log\left(\frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}\right)\right\} d\mathbf{x} \\
&= \int \left\{p_{\text{data}}(\mathbf{x})\log\left(\frac{2p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}\right) + p_g(\mathbf{x})\log\left(\frac{2p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}\right)\right\} d\mathbf{x} \\
&\quad - \log 4 \\
&= \mathcal{D}_{\text{KL}}\left(p_{\text{data}}(\mathbf{x}) \middle\| \frac{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}{2}\right) + \mathcal{D}_{\text{KL}}\left(p_g(\mathbf{x}) \middle\| \frac{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}{2}\right) - \log 4.
\end{aligned} \tag{4.76}$$

Eq. 4.74 is therefore obtained by substituting Eq. 4.75 into Eq. 4.76. According to Eq. 4.74, the optimal generator G^* is calculated to reflect the generator distribution $p_g(\mathbf{x})$, which is closest to real data distribution $p_{\text{data}}(\mathbf{x})$. GAN encourages G with $p_g(\mathbf{x})$ to fit $p_{\text{data}}(\mathbf{x})$ so as to fool D with its generated samples. The global minimum or the lower bound of $C(G)$

$$C^*(G) = -\log 4 \tag{4.77}$$

is achieved if and only if $p_g(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$. G and D are trained to update the parameters of both models by the *error backpropagation* algorithm.

Training Algorithm

Algorithm 3 presents the stochastic training algorithm for estimating the parameters $\{\boldsymbol{\theta}_d, \boldsymbol{\theta}_g\}$ of discriminator D and generator G for the construction of generative adversarial network. Starting from an initial set of parameters, GAN is implemented by running K updating steps for discriminator parameter $\boldsymbol{\theta}_d$ before one updating step for generator parameter $\boldsymbol{\theta}_g$. The minibatches of noise samples $\mathbf{Z}_n = \{\mathbf{z}_t\}_{t=1}^{T_n}$ and training samples $\mathbf{X}_n = \{\mathbf{x}_t\}_{t=1}^{T_n}$ are drawn from $p(\mathbf{z})$ and $p_{\text{data}}(\mathbf{x})$, respectively, and then employed in different updating steps for $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_g$. Gradient-based updates with momentum can be used [118]. Minimax optimization is performed because the value function is maximized to run the ascending of stochastic gradient for discriminator parameter $\boldsymbol{\theta}_d$ in Eq. 4.78 and is minimized to run the descending of stochastic gradient for generator $\boldsymbol{\theta}_g$ in Eq. 4.79. Notably, Eq. 4.79 does not include the first term of Eq. 4.78 because this term is independent of generator parameter $\boldsymbol{\theta}_g$.

Algorithm 3 SGD training for generative adversarial net

```
**Initialize** discriminator and generator parameters $\boldsymbol{\theta}_d$, $\boldsymbol{\theta}_g$
**For** number of training iterations
    **For** $K$ steps
        Sample a minibatch of $T_n$ noise samples $\{\mathbf{z}_1, \ldots, \mathbf{z}_{T_n}\}$ from prior $p(\mathbf{z})$
        Sample a minibatch of $T_n$ training examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_{T_n}\}$ from $p_{\text{data}}(\mathbf{x})$
        Update the discriminator by *ascending* its stochastic gradient
            $$\boldsymbol{\theta}_d \leftarrow \nabla_{\boldsymbol{\theta}_d} \frac{1}{T_n} \sum_{t=1}^{T_n} \big[ \log D(\mathbf{x}_t) + \log\big(1 - D(G(\mathbf{z}_t))\big) \big] \quad (4.78)$$
    **End For**
    Sample a minibatch of $T_n$ noise samples $\{\mathbf{z}_1, \ldots, \mathbf{z}_{T_n}\}$ from prior $p(\mathbf{z})$
    Update the generator by *descending* its stochastic gradient
        $$\boldsymbol{\theta}_g \leftarrow \nabla_{\boldsymbol{\theta}_g} \frac{1}{T_n} \sum_{t=1}^{T_n} \log\big(1 - D(G(\mathbf{z}_t))\big) \quad (4.79)$$
**End For**
**Return** $\boldsymbol{\theta}_d$, $\boldsymbol{\theta}_g$
```

Interpretation for Learning Procedure

Figure 4.22(a) interprets how the discriminator $D(\mathbf{x})$ and generator $G(\mathbf{x})$ are run to estimate the distribution of generated data $p_g(\mathbf{x})$ relative to the unknown distribution of real data $p_{\text{data}}(\mathbf{x})$. Actually, the distribution of generated data $p_g(\mathbf{x})$ is obtained from the samples $\mathbf{x}$ generated by the generator $G(\mathbf{z})$ using the samples $\mathbf{z}$ drawn by prior density

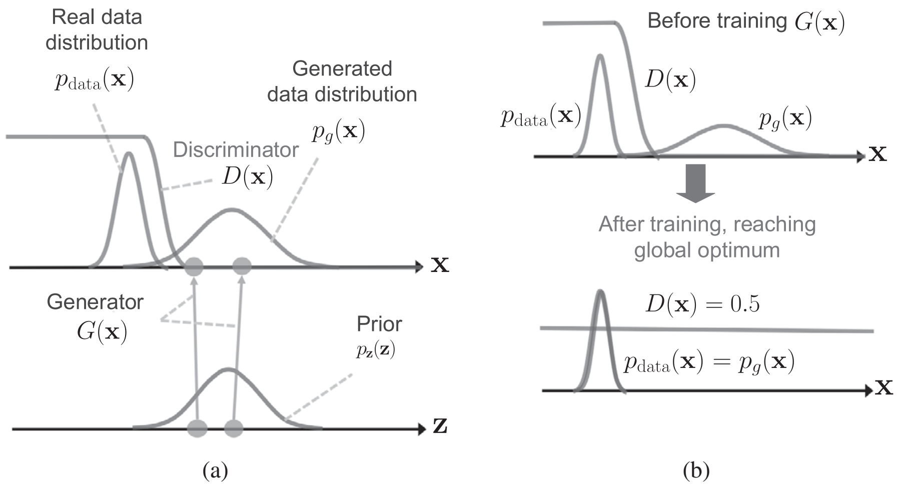

Real data distribution
$p_{\text{data}}(\mathbf{x})$
Generated data distribution
$p_g(\mathbf{x})$
Discriminator
$D(\mathbf{x})$
$\mathbf{x}$
Generator
$G(\mathbf{x})$
Prior
$p_{\mathbf{z}}(\mathbf{z})$
$\mathbf{z}$
(a)
Before training $G(\mathbf{x})$
$D(\mathbf{x})$
$p_{\text{data}}(\mathbf{x})$
$p_g(\mathbf{x})$
$\mathbf{x}$
After training, reaching global optimum
$D(\mathbf{x}) = 0.5$
$p_{\text{data}}(\mathbf{x}) = p_g(\mathbf{x})$
$\mathbf{x}$
(b)

Figure 4.22 Interpretation of discriminator, generator, data distribution, and model distribution for generative adversarial networks.

$p(\mathbf{z})$ with a standard Gaussian distribution. The discriminator D will correctly determine whether the input data $\mathbf{x}$ comes from the real data distribution or from the generator distribution. The output of D is likely to be one for true data in low-value range or to be zero for fake data in high-value range.

Figure 4.22(b) shows the situation of data distribution $p_{\text{data}}(\mathbf{x})$ and model distribution $p_g(\mathbf{x})$ before and after GAN training. In the beginning of GAN training, the estimated model distribution is different from the original data distribution. The resulting discriminator classifies the data in region of $p_{\text{data}}(\mathbf{x})$ as $D(\mathbf{x}) = 1$ and the data in region of $p_g(\mathbf{x})$ as $D(\mathbf{x}) = 0$. After a number of updating steps and learning epochs, we may achieve the global optimum at $p_{\text{data}} = p_g$ where the output of discriminator becomes $D(\mathbf{x}) = 0.5$ in various data region. Nevertheless, the joint training of discriminator and generator may be difficult and easily stuck in local optimum.

The training procedure for GAN can be further demonstrated by Figures 4.23(a) and (b) where the optimal discriminator and generator are trained and searched in the spaces of D and G, respectively. In estimating the discriminator, the maximization problem

$$D^* = \underset{D}{\operatorname{argmax}}\, V(G, D) \tag{4.80}$$

is solved to find the optimum D^*. Using the estimated discriminator D^*, the resulting value function is minimized to estimate the optimal generator G^* by

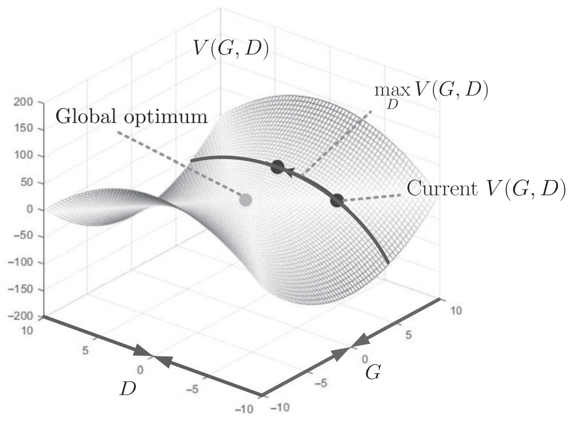
$$G^* = \underset{G}{\operatorname{argmax}}\, V(G, D^*). \tag{4.81}$$

The global optimum happens in an *equilibrium* condition for minimax optimization where the *saddle point* of the value function $V(G, D)$ is reached in a hybrid space of the discriminator and the generator. However, the value manifold in real-world applications is much more complicated than this example. The training procedure becomes difficult for GAN that uses neural networks as the discriminator and the generator in presence of heterogeneous training data. Although GAN is theoretically meaningful from the optimization perspective, in the next section, we discuss a number of challenges in the training procedure that will affect the performance of data generation.

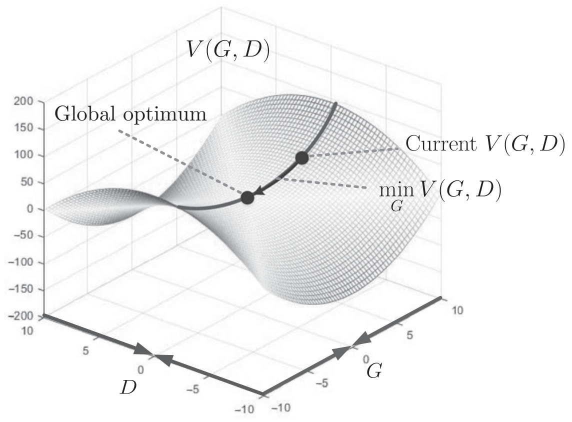
4.6.4 Gradient Vanishing and Mode Collapse

Generative adversarial networks are generally difficult to optimize. Their training procedure is prone to be unstable. A careful setting and design of network architecture is required to balance the model capacities between discriminator and generator so that a successful training with convergence can be achieved. This section addresses two main issues in the training procedure that are caused by the unbalance in model capacity. One is the gradient vanishing and the other is the mode collapse.

First of all, the *gradient vanishing* problem may easily happen in the optimization procedure in accordance with a minimax game where the discriminator minimizes a cross-entropy error function, but the generator maximizes the same cross-entropy error function. This is unfortunate for the generator because when the discriminator



(a)



(b)

Figure 4.23 Optimization steps for finding (a) optimal discriminator D and then (b) optimal generator G. The D- and G-axis represent the parameter space of D and G, respectively.

successfully rejects the generated data with high confidence, the gradient of cross-entropy error or value function with respect to the generator will vanish. No updating or slow updating is performed for the generator. Only the discriminator is updated continuously. Therefore, the generator is too weak and the discriminator is too strong. The discriminator is completely optimal and achieves 100 percent accuracy in binary classification, which means that the classification probability is $D^*(\mathbf{x})$ for real data $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$ and $(1 - D^*(\mathbf{x}))$ for generated data $\mathbf{x} \sim p_g(\mathbf{x})$ are surely 1 and 0, respectively. This shows that $\nabla_{\boldsymbol{\theta}_g} D^*(G(\mathbf{z})) = 0$ because $\log(1 - D^*(G(\mathbf{z}))) \approx \log 1$. Such a problem usually happens when the generator is poorly designed or built with weak capacity. To deal with this problem, a common practice is to reformulate the optimization for the generator from Eq. 4.79 to a minimization of the generator loss

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[-\log D^*(G(\mathbf{z}))] \quad \text{or} \quad \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[-\log D^*(\mathbf{x})], \tag{4.82}$$

which yields the same solution but exhibits much stronger gradient for updating.

However, the *mode collapse* problem is further exacerbated in the training procedure if the generator loss in Eq. 4.82 is minimized. The key issue in the implementation of GAN is the instability when calculating the gradients of the value function with respect to $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_g$. This issue results in mode collapse in the trained model where the generator or the model could not produce a variety of samples with sufficient randomness. To address this issue, we first express the loss function of a standard GAN given by an optimal discriminator $D^*(G(\mathbf{z}))$ or $D^*(\mathbf{x})$ as

$$\begin{aligned}&\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\big[\log D^*(\mathbf{x})\big] + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}\big[\log(1-D^*(\mathbf{x}))\big]\\ &\quad = 2\mathcal{D}_{\text{JS}}(p_{\text{data}}(\mathbf{x})\|p_g(\mathbf{x})) - \log 4,\end{aligned} \tag{4.83}$$

which is minimized to estimate the optimal generator G or equivalently the optimal generator distribution $p_g(\mathbf{x})$. From a representation of Kullback–Leibler divergence between $p_g(\mathbf{x})$ and $p_{\text{data}}(\mathbf{x})$

$$\begin{aligned}\mathcal{D}_{\text{KL}}(p_g(\mathbf{x})\|p_{\text{data}}(\mathbf{x})) &= \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}\left[\log \frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x})}\right]\\ &= \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}\left[\log \frac{\frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}}{\frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}}\right]\\ &= \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}\left[\log \frac{1-D^*(\mathbf{x})}{D^*(\mathbf{x})}\right]\\ &= \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[\log(1-D^*(\mathbf{x}))] - \mathbb{E}_{\mathbf{x}\sim p_g}[\log D^*(\mathbf{x})].\end{aligned} \tag{4.84}$$

We derive an equation for the modified generator loss in Eq. 4.82 by

$$\begin{aligned}&\mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[-\log D^*(\mathbf{x})]\\ &\quad = \mathcal{D}_{\text{KL}}(p_g(\mathbf{x})\|p_{\text{data}}(\mathbf{x})) - \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[\log(1-D^*(\mathbf{x}))]\\ &\quad = \mathcal{D}_{\text{KL}}(p_g(\mathbf{x})\|p_{\text{data}}(\mathbf{x})) - 2\mathcal{D}_{\text{JS}}(p_{\text{data}}(\mathbf{x})\|p_g(\mathbf{x}))\\ &\qquad + \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D^*(\mathbf{x})] + \log 4,\end{aligned} \tag{4.85}$$

which is obtained by substituting Eqs. 4.84 and 4.83 into Eq. 4.82.

There are two problems when minimizing Eq. 4.82 or equivalently Eq. 4.85 for the estimation of optimal generator G^*. First, during the minimization of Eq. 4.85, we simultaneously minimize the KL divergence $\mathcal{D}_{\text{KL}}(p_g(\mathbf{x})\|p_{\text{data}}(\mathbf{x}))$ and maximizing the JS divergence $\mathcal{D}_{\text{JS}}(p_{\text{data}}(\mathbf{x})\|p_g(\mathbf{x}))$. This circumstance is very strange and inconsistent and makes the calculation of gradients unstable. Second, KL divergence is *not* symmetric. In addition to the condition of global optimum $p_g(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$, there are two possibilities to obtain the lowest divergence

$$\mathcal{D}_{\text{KL}}(p_g(\mathbf{x})\|p_{\text{data}}(\mathbf{x})) \to 0, \quad \text{if } p_g(\mathbf{x}) \to 0 \text{ and } p_{\text{data}}(\mathbf{x}) \to 1 \tag{4.86}$$

and

$$\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}}(\mathbf{x}) \| p_g(\mathbf{x})) \to 0, \quad \text{if } p_{\mathrm{data}}(\mathbf{x}) \to 0 \text{ and } p_g(\mathbf{x}) \to 1. \tag{4.87}$$

Eq. 4.86 corresponds to the case that the generator is too weak to produce the samples with sufficient variety. Eq. 4.87 is seen as the case that the generator is too arbitrary to generate realistic and meaningful samples. Because of these two different possibilities, the generative model based on GAN may repeatedly produce similar samples or weakly synthesized samples with lack of variety. Both penalties considerably cause the issue of mode collapse in the training procedure of the generative model.

In what next, the generative adversarial network is employed to conduct the probabilistic autoencoder as a new type of deep generative model. The variational inference in Section 2.3 is again introduced in the implementation.

4.6.5 Adversarial Autoencoder

An AAE [122] was proposed by incorporating the generative adversarial network (GAN), as addressed in Section 4.6, into the construction of a VAE, as described in Section 4.5. An AAE is seen as a variant of a VAE where the latent variable $\mathbf{z}$ is learned via adversarial learning through a discriminator. Alternatively, an AAE is also viewed as a variant of GAN driven by variational inference where the discriminator in adversarial learning is trained to tell if the latent variable $\mathbf{z}$ is either inferred from real data $\mathbf{x}$ with a variational distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ or artificially generated from a prior density $p(\mathbf{z})$. The neural network parameter $\boldsymbol{\phi}$ is estimated to calculate the data-dependent mean and variance vectors $\{\boldsymbol{\mu}_{\phi}(\mathbf{x}), \boldsymbol{\sigma}^2_{\phi}(\mathbf{x})\}$ of a Gaussian distribution of $\mathbf{z}$ in the encoder output of a neural network where

$$\mathbf{z} \sim \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\phi}(\mathbf{x}), \boldsymbol{\sigma}^2_{\phi}(\mathbf{x})\mathbf{I}). \tag{4.88}$$

Figure 4.24 depicts how the adversarial learning is merged in an adversarial autoencoder. An AAE is constructed with three components, which are encoder, discriminator, and decoder. The building components of an AAE are different from those of a VAE consisting of an encoder and a decoder and those of GAN, containing a generator and a discriminator. Similar to a VAE the latent code $\mathbf{z}$ in an AAE is sampled from a variational distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ through an inference process. Different from a VAE the decoder in an AAE is learned as a deep generative model that maps the imposed prior $p(\mathbf{z})$ to the reconstructed data distribution. Such a mapping is estimated by adversarial learning through a discriminator that is optimally trained to judge $\mathbf{z}$ is drawn from a variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ or came from an imposed prior $p(\mathbf{z})$.

A VAE and AAE are further compared as follows. First of all, a VAE directly reconstructs an input data through an encoder and a decoder while an AAE indirectly generates data with an auxiliary discriminator that never sees the data $\mathbf{x}$ but classifies the latent variable $\mathbf{z}$. Second, a VAE optimizes the encoder and decoder by maximizing the evidence lower bound $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$, which consists of a log likelihood term for the lowest reconstruction error and a regularization term for the best variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. The derivative of the learning objective is calculated with respect to the decoder

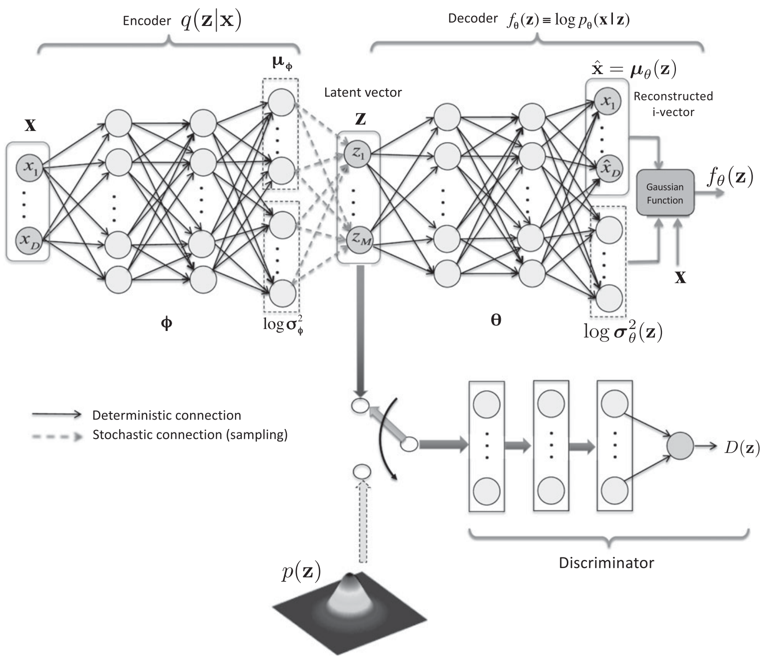

Encoder $q(\mathbf{z}|\mathbf{x})$
Decoder $f_\theta(\mathbf{z}) \equiv \log p_\theta(\mathbf{x}|\mathbf{z})$
$\boldsymbol{\mu}_\phi$
Latent vector
$\mathbf{z}$
$\hat{\mathbf{x}} = \boldsymbol{\mu}_\theta(\mathbf{z})$
Reconstructed i-vector
$\mathbf{x}$
x_1
x_D
z_1
z_M
$\hat{x}_1$
$\hat{x}_D$
Gaussian Function
$f_\theta(\mathbf{z})$
$\boldsymbol{\phi}$
$\log \boldsymbol{\sigma}_\phi^2$
$\boldsymbol{\theta}$
$\log \boldsymbol{\sigma}_\theta^2(\mathbf{z})$
Deterministic connection
Stochastic connection (sampling)
$D(\mathbf{z})$
Discriminator
$p(\mathbf{z})$

Figure 4.24 Adversarial autoencoder for data reconstruction, which consists of an encoder, a decoder, and a discriminator.

parameter $\boldsymbol{\theta}$ and encoder parameter $\boldsymbol{\phi}$ through the continuous hidden variable $\mathbf{z}$. Error backpropagation could not directly run with discrete latent variable $\mathbf{z}$. On the other hand, an AAE optimizes the encoder, decoder, and discriminator by maximizing the evidence lower bound $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$ as well as minimizing the cross-entropy error function of binary classification of latent variable $\mathbf{z}$ either from the variational posterior $q(\mathbf{z}|\mathbf{x})$ or from the imposed prior $p(\mathbf{z})$. The derivative of the learning objective over three components via minimax procedure is likely unstable. Third, a VAE and an AAE may be both underfitting but caused by different reasons. The capacity of a VAE is degraded due to the Bayesian approximation in variational inference while the capacity of an AAE is considerably affected by the non-convergence in the learning procedure due to minimax optimization. From the perspective of data generation, a VAE basically synthesizes the data where the global composition is assured but with blurred details. On the contrary, an AAE is trained for data generation where the local features of training data are captured but the global structure of data manifold is disregarded.

To balance the tradeoff between a VAE and an AAE, a new combination of a VAE and GAN was proposed in [123] as a new type of autoencoder. The construction of a hybrid VAE and GAN is illustrated in Figure 4.25. Using this hybrid VAE and GAN, the feature representation learned in GAN discriminator is used as the basis for a VAE reconstruction objective. The hybrid model is trained to estimate the optimal discriminator with parameter $\boldsymbol{\theta}_d$, which almost could not tell the difference between true data $\mathbf{x}$ or fake data $\widehat{\mathbf{x}}$, which is reconstructed by a decoder or a generator with parameter $\boldsymbol{\theta}_g$ from

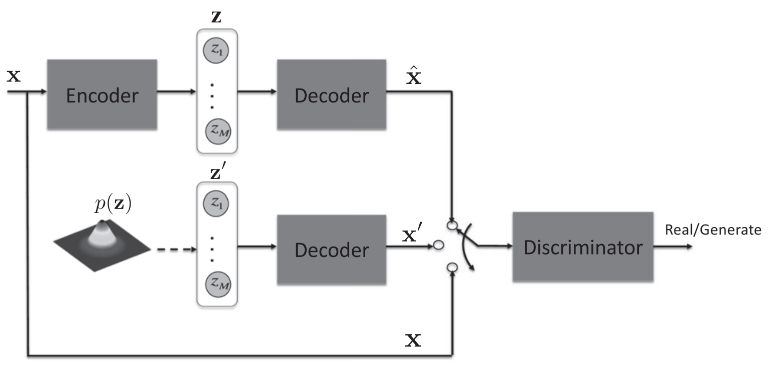


Figure 4.25 A hybrid VAE and GAN model where the decoder and generator are unified.

an inferred latent variable $\mathbf{z}$ through an encoder with parameter $\boldsymbol{\theta}_e$. The parameters of the encoder, generator, and discriminator $\{\boldsymbol{\theta}_e, \boldsymbol{\theta}_g, \boldsymbol{\theta}_d\}$ are jointly learned by a minimax optimization procedure over a learning objective consisting of an evidence lower bound and a cross-entropy error function. Different from an AAE, this hybrid model adopts the cross-entropy function measured in data domain $\mathbf{x}$ rather than in feature domain $\mathbf{z}$. Using a hybrid VAE and GAN, the evidence lower bound is maximized to pursue the learned similarity in feature space $\mathbf{z}$, which is different from the learned similarity of a VAE in data space $\mathbf{x}$. Abstract or high-level reconstruction error is minimized. Global structure is captured with the preserved local features.

4.7 Deep Transfer Learning

Traditional machine learning algorithms in speaker recognition work well under a common assumption that training and test data are in the same feature space and follow the same distribution. However, the real-world speech signals of different speakers may not follow this assumption due to the varying feature space or the mismatch between training and test conditions. In practical circumstances, we may train a speaker recognition system in a *target domain*, but there are only sufficient training data in the other *source domain*. The training data in different domains basically follow by different distributions or locate in different feature spaces. As shown in Figure 4.26(a), the traditional learning models are separately trained from scratch by using newly collected data from individual domains in presence of various distributions and feature spaces. Learning tasks in existing domains are assumed to be independent. However, the collection and labeling of training data in new domain are expensive. Solely relying on supervised learning is impractical. It is crucial to conduct the knowledge transfer with semi-supervised learning so as to reduce the labeling cost and improve the system utility.

Knowledge is transferred across different domains through learning a good feature representation or a desirable model adaptation method. More specifically, it is beneficial to cotrain the feature representation and the speaker recognition model to build a domain

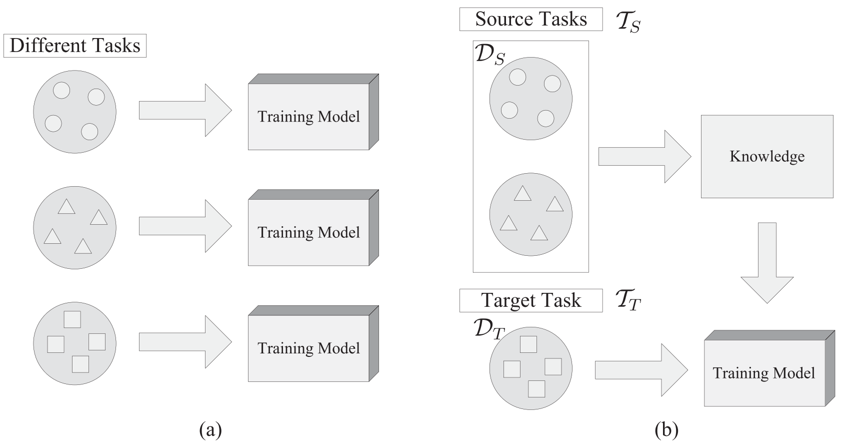


Figure 4.26 Learning processes based on (a) traditional machine learning and (b) transfer learning.

invariant system without the labeling in the target domain. Such an emerging research topic on transfer learning or in particular domain adaptation has been recently attracting the attention of many researchers in speaker recognition areas. Accordingly, this section will first address the general definition of transfer learning and discuss the relationships among different transfer learning settings such as multi-task learning and domain adaptation. Then, domain adaptation is emphasized and described in different types of solutions. Finally, a variety of deep learning solutions to transfer learning or domain adaptation are detailed for the development of modern speaker recognition systems.

4.7.1 Transfer Learning

Figure 4.26(b) conceptually depicts how knowledge transfer is performed for domains from different source tasks to a target task [124]. The domains and tasks are defined as follows. A domain $\mathcal{D} = \{\mathcal{X}, p(X)\}$ is composed of feature space $\mathcal{X}$ and a marginal probability distribution $p(X)$, where $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathcal{X}$. Here, $\mathbf{x}_i$ means the ith training sample and $\mathcal{X}$ is the space of all samples. Fundamentally, if two domains are different, their observation data may follow different marginal probability distributions in different feature spaces. In addition, the task is denoted as $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$. Basically, a task is composed of a label space $\mathcal{Y}$ and an objective predictive function $f(\cdot)$, which is expressed as $p(Y|X)$ from a probabilistic perspective and can be learned from the training data. Let $\mathcal{D}_S = \{(\mathbf{x}_1^s, \mathbf{y}_1^s), \ldots, (\mathbf{x}_m^s, \mathbf{y}_m^s)\}$ denotes the training data in a source domain, where $\mathbf{x}_i^s \in \mathcal{X}_s$ means the observation input and $\mathbf{y}_i^s \in \mathcal{Y}_s$ corresponds to its label information. Similarly, the training data in the target domain are denoted as $\mathcal{D}_T = \{(\mathbf{x}_1^t, \mathbf{y}_1^t), \ldots, (\mathbf{x}_n^t, \mathbf{y}_n^t)\}$, where $\mathbf{x}_i^t \in \mathcal{X}_t$ and $\mathbf{y}_i^t \in \mathcal{Y}_t$.

Traditional machine learning methods strongly assume that the source domain and the target domain are the same, i.e., $\mathcal{D}_S = \mathcal{D}_T$, and that the source task and the target task are identical, i.e., $\mathcal{T}_S = \mathcal{T}_T$. When such an assumption does not exist in learning scenario,

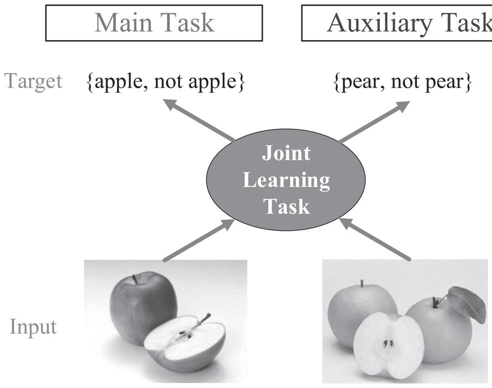


Figure 4.27 Illustration for multi-task learning with a main task and an auxiliary task.

the learning problem turns out to be a transfer learning problem. In general, transfer learning commonly assumes that the source and target domains are different $\mathcal{D}_S \neq \mathcal{D}_T$ or the source task and target are different $\mathcal{T}_S \neq \mathcal{T}_T$. Transfer learning aims to improve the target predictive function $f_T(\cdot)$ of task $\mathcal{T}_T$ in target domain $\mathcal{D}_T$ by transferring the knowledge from source domain $\mathcal{D}_S$ under source task $\mathcal{T}_S$. Here, the condition $\mathcal{D}_S \neq \mathcal{D}_T$ means that either $\mathcal{X}_S \neq \mathcal{X}_T$ or $p(X^s) \neq p(X^t)$, whereas the condition $\mathcal{T}_S \neq \mathcal{T}_T$ means that either $\mathcal{T}_S \neq \mathcal{T}_T$ or $p(Y^s|X^s) \neq p(Y^t|X^t)$.

There are two popular styles of transfer learning. One is multi-task learning, while the other is domain adaptation.

Multi-Task Learning

Multi-task learning is seen as an inductive transfer learning mechanism. Rather than only concerning about the target task, multi-task learning aims to learn all of the source and target tasks simultaneously. The goal of multi-task learning is to improve the generalization performance by using the domain information contained in the training data across different related tasks. For example, as shown in Figure 4.27, we may find that learning to recognize the pears in an auxiliary task might help to build a classifier to recognize the apples in a main task. More specifically, multi-task learning is seen as a learning strategy where the main task is performed together with the other related tasks at the same time so as to discover the shared feature representations that help in improving the learning fidelity of each individual task.

Previous works on multi-task learning were proposed to learn a common feature representation shared by different tasks [125–127]. To encourage a model to also work well on other tasks that are different but related is a better regularization than uninformed regularization such as weight decay. A standard learning objective with model regularization is expressed by

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}, \boldsymbol{\theta}) + \lambda \Omega(\boldsymbol{\theta}) \tag{4.89}$$

where $\boldsymbol{\theta}$ denotes the model parameter and λ denotes the regularization parameter. Multi-task learning adopts the regularization $\Omega(\grave{})$ that reflects the shared information among different tasks. Thus, we can obtain an improved model for the main task by learning an integrated model for multiple tasks jointly.

4.7.2 Domain Adaptation

Domain adaptation is another popular case of transfer learning that has been recently developed for speaker recognition [128, 129]; see also Chapter 6. Assume that the collected speech samples are available in the source domain and also in the target domain. Let $\{X^s, Y^s\} = \{(\mathbf{x}_1^s, \mathbf{y}_1^s), \ldots, (\mathbf{x}_m^s, \mathbf{y}_m^s)\}$, which denotes the labeled training samples in the source domain. Here, $\mathbf{x}_i^s$ denotes the training token and $\mathbf{y}_i^s$ denotes the corresponding label information. On the other hand, we have the unlabeled data from the target domain $\{X^t, Y^t\} = \{(\mathbf{x}_1^t, \mathbf{y}_1^t), \ldots, (\mathbf{x}_n^t, \mathbf{y}_n^t)\}$ where the label information $\mathbf{y}_i^t$ is unseen. One important assumption in domain adaptation is that two domains $\{\mathcal{D}_S, \mathcal{D}_T\}$ are related but different, namely the joint distributions of samples and labels in two domains $p(X^s, Y^s)$ and $p(X^t, Y^t)$ are different. Domain adaptation is a special category under transfer learning where the marginal distributions in two domains $p(X^s)$ and $p(X^t)$ are different while making the assumption that two conditional distributions $p(Y^s|X^s)$ and $p(Y^t|X^t)$ are identical. Typically, the condition of distinct marginal distributions in training data and test data is also known as the sample selection bias or the covariate shift [130, 131]. There are two types of domain adaptation methods that are instance-based domain adaptation and feature-based domain adaptation.

Instance-Based Domain Adaptation

Using the instance-based method, importance reweighting or importance sampling is performed to reweight the labeled instances from the source domain. The divergence between marginal distributions in two domains can be compensated. In general, we minimize the expected risk or the expectation of a loss function $\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$ to learn the optimal model parameter $\boldsymbol{\theta}^*$ via empirical risk minimization:

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p(\mathbf{x},\mathbf{y})}[\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})\end{aligned} \tag{4.90}$$

where n is the number of training samples. Domain adaptation or covariate shift aims at learning the optimal model parameters $\boldsymbol{\theta}^*$ for the *target* domain by minimizing the expected risk

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}_T} p(\mathcal{D}_T)[\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})]. \tag{4.91}$$

In traditional machine learning, the optimal model parameter $\boldsymbol{\theta}^*$ is learned by assuming $p(\mathcal{D}_S) = p(\mathcal{D}_T)$ and accordingly using the source domain data $\mathcal{D}_s$. However, the distributions are different $p(\mathcal{D}_S) \neq p(\mathcal{D}_T)$ in the real-world speaker recognition where the domain adaptation is required. Furthermore, no labeled data in the target domain are available while a lot of labeled data in source domain are available in this situation. Accordingly, we modify the optimization problem to fit the target domain using a source domain data as follows [124]:

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}_S} \frac{p(\mathcal{D}_T)}{p(\mathcal{D}_S)} p(\mathcal{D}_S)\mathcal{L}(\mathbf{x},\mathbf{y},\boldsymbol{\theta}) \\ &\approx \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{m} \left(\frac{p_T(\mathbf{x}_i^t,\mathbf{y}_i^t)}{p_S(\mathbf{x}_i^s,\mathbf{y}_i^s)}\right) \mathcal{L}(\mathbf{x}_i^s,\mathbf{y}_i^s,\boldsymbol{\theta}),\end{aligned} \tag{4.92}$$

where m is the number of training samples in the source domain. The optimization problem in Eq. 4.92 is solved to learn a model for the target domain by giving different weight $\frac{p_T(\mathbf{x}_i^t,\mathbf{y}_i^t)}{p_S(\mathbf{x}_i^s,y_i^s)}$ to each instance $(\mathbf{x}_i^s,\mathbf{y}_i^s)$ in the source domain. Under the assumption that the conditional distributions $p(Y^s|X^s)$ and $p(Y^t|X^t)$ are the same, the difference between $p(D_S)$ and $p(D_T)$ is caused by $p(X^s)$ and $p(X^t)$. It is because that the assumption of domain adaptation yields

$$\frac{p_T(\mathbf{x}_i^t,\mathbf{y}_i^t)}{p_S(\mathbf{x}_i^s,\mathbf{y}_i^s)} = \frac{p(\mathbf{x}_i^t)}{p(\mathbf{x}_i^s)}, \tag{4.93}$$

where $\frac{p(\mathbf{x}_i^t)}{p(\mathbf{x}_i^s)}$ is a reweighting factor for the loss function of individual training sample. Or equivalently, the reweighting factor can be used for adjusting the difference between the source and target distributions. Finally, the domain adaptation problem is simplified as a problem of estimating the factor $\frac{p(\mathbf{x}_i^t)}{p(\mathbf{x}_i^s)}$ for calculating the learning objective using each sample $\mathbf{x}_i^s$ in the training procedure. Figure 4.28 illustrates a regression example before and after the instance-based domain adaptation where the data distributions and the learned models are shown.

Feature-Based Domain Adaptation

Feature-based domain adaptation is a common approach. Instead of reweighting the training instances, the feature-based method is developed with the assumption that there exists a domain-invariant feature space. The goal of feature-based approach is to span a feature space by minimizing the divergence between two domains and simultaneously preserving the discriminative information for classification in test session. In [132], the structural correspondence learning was developed to match the correspondence between the source domain and the target domain. This method provides an important technique that makes use of the unlabeled data in the target domain to discover relevant features that could reduce the divergence between the source and target domains. The key idea was to evaluate the feature correspondence between the source and target domains, which are based on the correlation governed by the pivot features. Pivot features behaved similarly for discriminative learning in both domains. Non-pivot features are correlated with some of the pivot features. A shared low-dimensional real-valued feature space was learned by using the pivot features. However, these previous studies do not minimize the divergence between different domains directly.

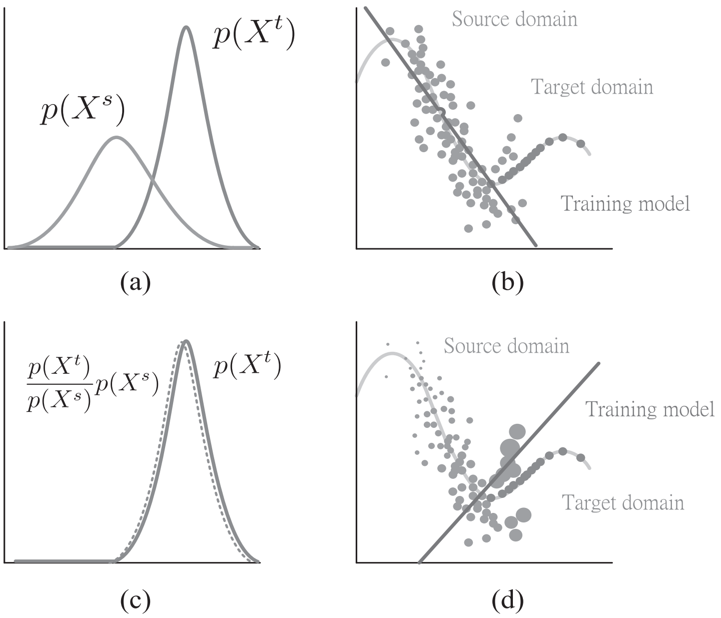


Figure 4.28 Illustration of instance-based domain adaptation. (a) Data distributions of source and target domains. (b) Straight line is a linear model learned from the source domain. (c) Distribution of the source domain is adjusted by the reweighting factor. (d) After reweighting the data in source domain, the learned model fits the target domain.

In [133], a stationary subspace analysis was proposed to match distributions in a low-dimensional space. In [134], a method based on dimensionality reduction was presented to simultaneously minimize the divergence between source distribution and target distribution and minimize the information loss in data. Divergence reduction can be also performed via the maximum mean discrepancy, which will be addressed in the next section.

4.7.3 Maximum Mean Discrepancy

Maximum mean discrepancy (MMD) [135, 136] provides an effective approach to domain adaptation based on the distribution matching, which is a major work in domain adaptation by compensating the mismatch between the marginal distributions in the source and target domains. There exist many criteria that can be used to estimate the distance between the source and target distributions. However, many of them, such as the Kullback–Leibler (KL) divergence, require estimating the densities before estimating the divergence. Traditionally, it was popular to carry out the distribution matching based on the estimation of parametric distributions.

Compared with the parametric distributions, it is more attractive to implement the nonparametric solution to the distance between distributions without the need of density estimation. Maximum mean discrepancy is a well-known nonparametric method that is referred as a divergence between distributions of two data sets in the reproducing kernel Hilbert space (RKHS) denoted by $\mathcal{H}$. MMD can directly evaluate whether two distributions are different on the basis of data samples from each of them. A key superiority of MMD to the other methods is that MMD does not require the density estimation.

The MMD criterion has been employed in many domain adaptation methods for reweighting data or building domain-invariant feature space. Let the kernel-induced feature map be denoted by ϕ. The training observations in the source domain and the target domain are denoted by $X^s = \{\mathbf{x}_1^s, \ldots, \mathbf{x}_m^s\}$ and $X^t = \{\mathbf{x}_1^t, \ldots, \mathbf{x}_n^t\}$, which are drawn independently and identically distributed (i.i.d.) from $p(X^s)$ and $p(X^t)$ (p_S and p_T in short), respectively. The MMD between $\{\mathbf{x}_1^s, \ldots, \mathbf{x}_m^s\}$ and $\{\mathbf{x}_1^t, \ldots, \mathbf{x}_n^t\}$ is defined as follows

$$\text{MMD}(p_S, p_T) = \sup_{\|f\|_{\mathcal{H}} \leqslant 1} (\mathbb{E}_{\mathbf{x}^s}[f(\mathbf{x}^s)] - \mathbb{E}_{\mathbf{x}^t}[f(\mathbf{x}^t)]), \tag{4.94}$$

where $f \in \mathcal{F}$ are functions belonging to the unit ball in a reproducing kernel Hilbert space $\mathcal{H}$. Then, we obtain a biased empirical estimate of MMD by replacing the population expectations with empirical expectations

$$\text{MMD}(X^s, X^t) = \sup_{\|f\|_{\mathcal{H}} \leqslant 1} \left(\frac{1}{m} \sum_{i=1}^{m} f(\mathbf{x}_i^s) - \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{x}_i^t) \right), \tag{4.95}$$

where $\|\cdot\|_{\mathcal{H}}$ denotes the RKHS norm. Due to the property of RKHS, the function evaluation can be rewritten as $f(\mathbf{x}) = \langle \phi(\mathbf{x}), f \rangle$, where $\phi(\mathbf{x}) \colon \mathcal{X} \to \mathcal{H}$, the empirical estimate of MMD is expressed as

$$\text{MMD}(X^s, X^t) = \left\| \frac{1}{m} \sum_{i=1}^{m} \phi(\mathbf{x}_i^s) - \frac{1}{n} \sum_{i=1}^{n} \phi(\mathbf{x}_i^t) \right\|_{\mathcal{H}}. \tag{4.96}$$

Rewriting the norm in Eq. 4.96 as an inner product in RKHS and using the reproducing property, we have

$$\begin{aligned} &\text{MMD}(X^s, X^t) \\ &= \left[\frac{1}{m^2} \sum_{i,j=1}^{m} k(\mathbf{x}_i^s, \mathbf{x}_j^s) - \frac{2}{mn} \sum_{i,j=1}^{m,n} k(\mathbf{x}_i^s, \mathbf{x}_j^t) + \frac{1}{n^2} \sum_{i,j=1}^{n} k(\mathbf{x}_i^t, \mathbf{x}_j^t) \right]^{1/2}, \end{aligned} \tag{4.97}$$

where $k(\cdot, \cdot)$ denotes the characteristic function based on the positive semidefinite kernel. One commonly used kernel is the Gaussian kernel in the form

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2 \right) \tag{4.98}$$

with a variance parameter σ^2.

In summary, the distance between two distributions is equivalent to the distance between the means of two samples in the RKHS. The value of MMD is nonnegative, and it vanishes if and only if two distributions are the same. In Chapter 6, the domain adaptation techniques are further specialized for the application in speaker recognition. We will also address how MMD is developed in a type of deep learning solution to speaker recognition in Section 6.4. In what follows, we introduce the realization of deep neural networks for transfer learning.

4.7.4 Neural Transfer Learning

In accordance with the fundamentals of transfer learning, hereafter, we address the implementation of deep neural networks for the fulfillment of generalizable learning as well as for the realization of classification learning with training data in source domain and test data in target domain. Neural transfer learning is carried out as a new type of machine learning in a form of transfer learning where deep neural networks are configured.

Generalizable Learning

Neural transfer learning is basically feasible to conduct the so-called generalizable learning with two types of learning strategies in presence of multiple learning tasks. First, as depicts in Figure 4.29(a), deep transfer learning can be realized for multi-task learning from raw data through learning the shared representation in intermediate layers of a deep neural network. Such a shared intermediate representation is learned in an unsupervised manner that conveys deep latent information generalizable across different tasks. As we can see, the outputs of the shared intermediate layer are forwarded to carry out different tasks. On the other hand, neural transfer learning can also be implemented as a kind of generalizable learning via the partial feature sharing as shown in Figure 4.29(b). In this case, the low-level features and high-level features are partially connected and shared between different layers toward the outputs $\{y_1, \ldots, y_N\}$ of N tasks. The mixed mode learning is performed as another type of generalizable learning where different compositions of functions are calculated in different nodes and different layers.

Typically, transfer learning is implemented as a learning algorithm that can discover the relations across different tasks as well as share and transfer knowledge across multiple domains. It is because that the representation of deep models has the capability of capturing the underlying factors from training data in different domains that are virtually associated with individual tasks. Correspondingly, the feature representation using deep models with multi-task learning is advantageous and meaningful for system improvement due to the shared factors across tasks.

In [137], a hierarchical feed-forward model was trained by leveraging the cross-domain knowledge via transfer learning from some pseudo tasks without supervision. In [138], the multi-task learning was applied to elevate the performance of deep models by transferring the shared domain-specific information that was contained in the related tasks. In particular, the multi-task learning problem can be formulated as a special realization based on the neural network architecture as displayed in Figure 4.30(a) where a main task and N auxiliary tasks are considered. Let $\mathcal{D}_m = \{\mathbf{x}_n, y_{mn}\}$ denote N input samples of main task, which is indexed by m. The optimization problem is formulated as

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}_m, \boldsymbol{\theta}) + \lambda \Omega(\boldsymbol{\theta}) \tag{4.99}$$

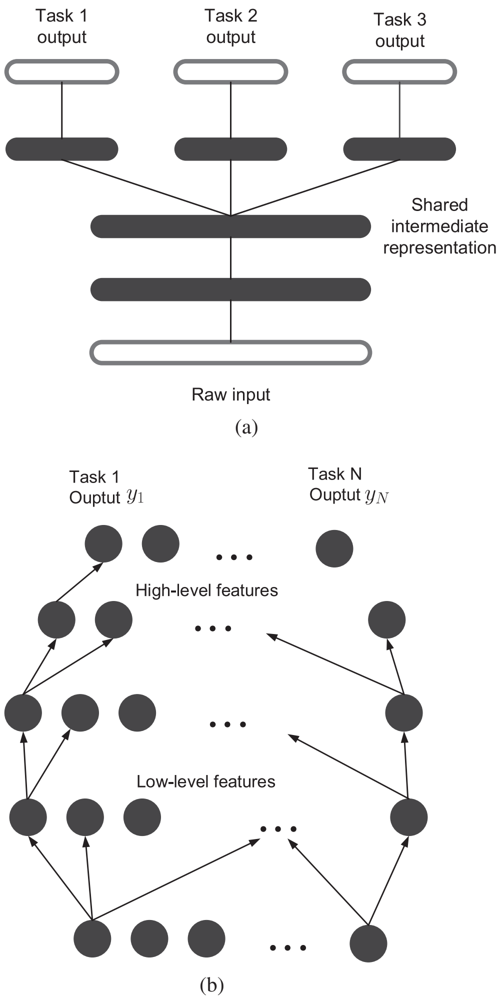


Figure 4.29 Multi-task neural network learning with (a) the shared representation and (b) the partial feature sharing.

where $\boldsymbol{\theta}$ is the model parameter, $\Omega(\boldsymbol{\theta})$ is a regularization term and $\mathcal{L}$ amounts to an empirical loss. Eq. 4.99 is therefore expanded for the main task in a form of

$$\min_{\mathbf{w}_m} \left[\sum_n \mathcal{L}\left(y_{mn}, \mathbf{w}_m^\top f(\mathbf{x}_n; \boldsymbol{\theta}) \right) + \lambda \| \mathbf{w}_m^\top \mathbf{w}_m \| \right] \tag{4.100}$$

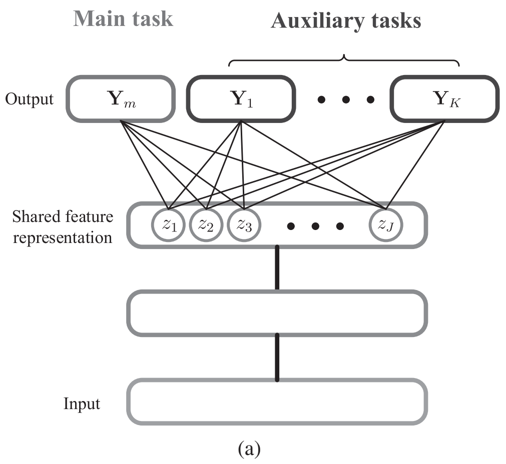
where $f(\mathbf{x}_n; \boldsymbol{\theta}) = \mathbf{z} = [z_1 \cdots z_K]^\top$ denotes the hidden units of a network, $\mathbf{w}_m$ denotes the model parameter and $\mathcal{L}(\cdot)$ denotes the error function for the main task or the target task. In multi-task learning, $\Omega(\boldsymbol{\theta})$ is implemented by introducing N auxiliary tasks. Each auxiliary task is learned by using $\mathcal{D}_k = \{\mathbf{x}_n, y_{kn}\}$, where $y_{kn} = g_k(\mathbf{x}_n)$ denotes the output of an auxiliary function. The regularization term $\Omega(\boldsymbol{\theta})$ is also incorporated in the learning objective:

$$\min_{\{\mathbf{w}_k\}} \sum_k \left[\sum_n \mathcal{L}\left(y_{kn}, \mathbf{w}_k^\top f(\mathbf{x}_n; \boldsymbol{\theta})\right) + \lambda \|\mathbf{w}_k^\top \mathbf{w}_k\| \right]. \tag{4.101}$$

As we can see, a model is encouraged to work well in the other tasks. This provides a better regularization than the noninformative regularization. Thus, we are able to achieve a better feature representation for the main task through the help of deep models jointly trained from multiple tasks. Next, neural transfer learning is further implemented as a general solution to build a classification system based on semi-supervised learning.

Semi-Supervised Domain Adaptation

Figure 4.30(b) depicts the learning strategy based on a deep semi-supervised model adaptation. The learning objective is to train a classification system for the target domain



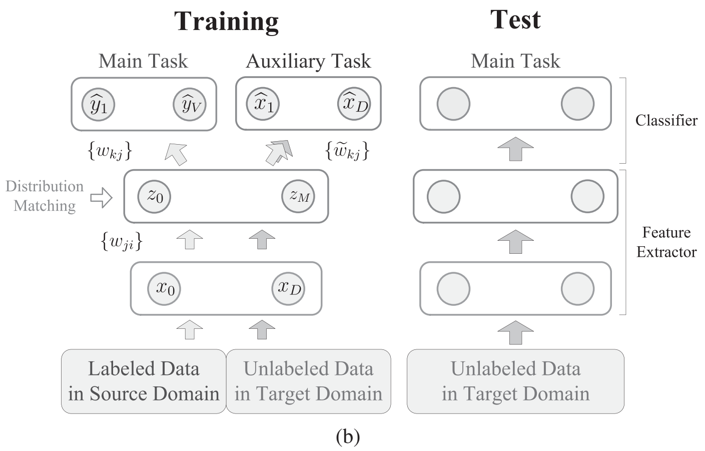


Figure 4.30 (a) Deep transfer learning via a main task and an auxiliary task. (b) A classification system is built with training data in source domain and test data in target domain. [Reprinted from *Deep Semi-Supervised Learning for Domain Adaptation (Figure 1), by H.Y. Chen and J.T. Chien, Proceedings IEEE Workshop on Machine Learning for Signal Processing, Boston, 2015, with permission of IEEE.*]

where a set of labeled samples in source domain and a set of unlabeled samples or test samples in target domain are available. This domain adaptation problem requires assigning a group of output neurons for the main classification task and a group of neurons for the auxiliary regression tasks, where training is jointly performed to acquire the shared information between two domains. Such information is conveyed through different hidden layers consisting of different levels of hidden features.

Basically, Figure 4.30(b) shows a hybrid model of deep neural network (DNN) for the application of semi-supervised domain adaptation. The DNN classifier is trained and applied for prediction of class labels for test samples. For the case of D dimensional input data $\mathbf{x} = [x_1 \ldots x_D]^\top$, the auxiliary task for regression can be designed as the reconstruction of input data as $\widehat{\mathbf{x}} = [\widehat{x}_1 \ldots \widehat{x}_D]^\top$. We therefore develop a semi-supervised domain adaptation by matching the distributions between two domains based on the shared information in hidden layers. There are two learning tasks in an integrated objective function, which is formulated to find the solution to semi-supervised domain adaptation. One is the cross-entropy error function, which is minimized for optimal classification while the other is the reconstruction error function, which is minimized for optimal regression. Based on this learning strategy, the semi-supervised learning is conducted under multiple objectives. Notably, the main and auxiliary tasks are separated in the output layer by using separate weights, $\{w_{kj}\}$ and $\{\widetilde{w}_{kj}\}$. The layer next to the output layer is known as the layer for distribution matching. Those low-level features were shared for both classification and regression using weight parameters $\{w_{ji}\}$.

In this chapter, we have addressed the fundamentals of deep learning ranging from the traditional models including restricted Boltzmann machines, deep neural networks, and the deep belief networks to the advanced models including variational autoencoders, generative adversarial networks, and the neural transfer learning machines. These models will be employed in two categories of speaker recognition systems. One is for robust speaker recognition while the other is for domain adaptive speaker recognition, which will be detailed in Chapter 5 and Chapter 6, respectively.