Chapter 11 Introduction to Probabilistic Generative models

In supervised learning setting, we have a dataset $\mathcal{D} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \dots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})\}$, where one data sample $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ consists of an input $\mathbf{x}^{(i)}$ and a label $\mathbf{y}^{(i)}$. Discriminative models typically estimate the conditional distribution $p(\mathbf{y}|\mathbf{x})$ for a classification task. In this chapter, we will introduce a different type of models, called *generative models*, which often estimate the joint probability distribution $p(\mathbf{x}, \mathbf{y})$. Although generative models can be used for classification in a supervised setting, they demonstrate great advantages in understanding unlabeled datasets, and thus are widely used in unsupervised learning. As the name suggests, we can generate data samples from the distribution learned by the generative models on a particular dataset, and thus the generated samples have the approximately same distribution as the underlying distribution of the dataset.

We assume that a dataset $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\}$ are samples from the underlying distribution $p(\mathbf{x})$. The goal of a generative model is to approximate $p(\mathbf{x})$ by $\hat{p}(\mathbf{x}; \boldsymbol{\theta})$, given access to the dataset $\mathcal{D}$, where $\boldsymbol{\theta}$ denotes the learnable parameters of the model. With the learned generative model, we can perform the following inferences: 1) density estimation – find the probability of a new input $\mathbf{x}$, 2) sampling – generate a new sample from the learned distribution, and 3) unsupervised representation learning – learn meaningful feature representation of a new data sample.

In this chapter, we will introduce a visual tool to represent a generative model, which is called *probabilistic graphical model*, or simply called *graphical model*. The treatment of Gaussian mixture models (GMMs) and its solutions provides a foundation for the subsequent topics such as variational auto-encoders (VAEs) and generative adversarial networks (GANs). The *expectation-maximization* (EM) algorithm is derived to learn a GMM, and then is generalized to deal with the intractable scenarios. This generalization leads to the development of VAEs.

This chapter covers:

- Graph representation for generative models
- Gaussian mixture models with latent variables
- EM algorithm for GMMs
- General EM algorithms
- Evidence lower bound (ELBO) in EM algorithms
- Variational auto-encoder (VAE)
- VAE implementation on MNIST dataset in PyTorch

11.1. Generative Models with Latent Variables

11.1.1 Graph representation

It is advantageous to represent a joint probability distribution by a graph. In a probabilistic graphical model, a graph comprises nodes linked by arcs. In the graph, each node corresponds to a random variable, and the arcs specify the probabilistic relationships between the random variables. Let's start with directed graphical models, in which each arc has a particular direction indicated by an arrow. We restrict our discussions to directed acyclic graph (DAG), in which there is no closed path that starts and ends at the same node. Node *a* is said to be the *parent* of node *b* if there is an arc going from node *a* to node *b*, and node *b* is the *child* node of node *a*. A node without any incoming arc is called *root* node. Each node represents the conditional distribution of its corresponding random variable, conditioned on all its parent variables. A root node represents the distribution of the random variable without any condition. The joint distribution defined by the graph is the product of all conditional distributions of all nodes conditioned on the variables corresponding to their parents in the graph.

For instance, consider the joint distribution $p(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M)$ over M variables. By the product rule of probability, we can represent the joint distribution as the product of M conditional distributions,

$$p(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M) = p(\mathbf{x}_M|\mathbf{x}_1, \dots, \mathbf{x}_{M-1})p(\mathbf{x}_{M-1}|\mathbf{x}_1, \dots, \mathbf{x}_{M-2}) \dots p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_1) \tag{11.1}$$

The decomposition in (11.1) results in a graph shown in Fig.11.1. Note that we chose a particular number ordering for random variables for the decomposition in (11.1). It is usually convenient to choose an ordering such that the arcs go from lower numbered node to a higher numbered node. For a DAG, there exists an ordering of the nodes such that there are no arcs that go from any node to any lower numbered node.

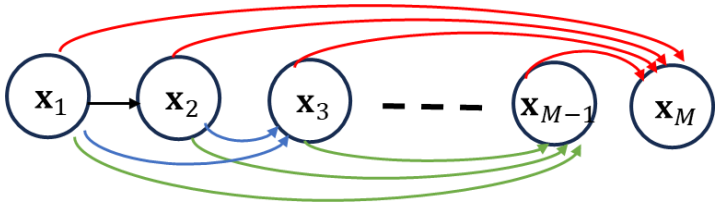


Fig.11.1 A fully connected graphical model

In general, a graphical mode is not necessarily fully connected, in other words the absence of some arcs is possible. Thus, for a graph with N nodes, the joint distribution is given by

$$p(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M) = \prod_{i=1}^{M} p(\mathbf{x}_i|pa_i) \tag{11.2}$$

where pa_i is the set of parent node variables of $\mathbf{x}_i$, as the condition for the conditional distribution of $\mathbf{x}_i$.

Given a graphical model corresponding to (11.2), we can draw one sample $\mathbf{x} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \dots, \hat{\mathbf{x}}_M]^T$ from the joint distribution as follows. We assume that each node is numbered higher than its parent nodes. First we draw a sample $\hat{\mathbf{x}}_1$ from the distribution associated with the lowest-numbered node, $p(\mathbf{x}_1)$. Then we sequentially draw samples $\hat{\mathbf{x}}_i, i = 2,3, \dots, M$, one at a time, from the conditional distribution $p(\mathbf{x}_i|\widehat{pa}_i)$ where the parents $\widehat{pa}_i$ have been set to the sampled values from $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \dots, \hat{\mathbf{x}}_{i-1}$.

In a generative model, the lower-numbered nodes typically correspond to *latent* variables, whereas the higher-numbered terminal nodes represent the observations. As we shall see shortly, the purpose of latent variables is to represent or sample a complicated distribution by simpler conditional distributions.

11.1.2 Gaussian mixture models

Gaussian mixture models (GMMs) are motivated by the fact that a linear combination of Gaussian distributions can provide a stronger expression for a complicated distribution than a single Gaussian distribution. In a general mixture model, the joint distribution $p(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^D$, can be represented by

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \, p(\mathbf{x}; \boldsymbol{\theta}_k) \tag{11.3}$$

where π_k are mixing coefficients, and $p(\mathbf{x}; \boldsymbol{\theta}_k)$ is the-k component specified by a pre-selected density function parameterized by $\boldsymbol{\theta}_k$. For a Gaussian mixture model, $p(\mathbf{x}; \boldsymbol{\theta}_k)$ is a Gaussian distribution given by

$$p(\mathbf{x}|\boldsymbol{\theta}_k) = \mathbb{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{11.4}$$

where $\boldsymbol{\mu}_k$ is the mean and $\boldsymbol{\Sigma}_k$ denotes the covariance matrix.

Fig.11.2 plots the density function of a Gaussian mixture model (D=1 and K=2) with $\pi_1 = 0.4, \pi_2 = 0.6$, $\boldsymbol{\mu}_1 = 5, \boldsymbol{\mu}_2 = 15, \boldsymbol{\Sigma}_1 = 4, \boldsymbol{\Sigma}_2 = 16$, assuming $\mathbf{x}$ is 1-dimensional random variable.

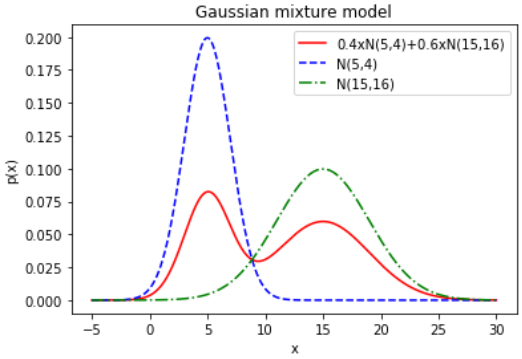


Fig.11.2 An example of GMM distribution

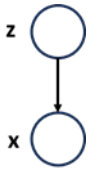


Fig.11.3 The graphic model of Gaussian mixture model

To associate π_k with a latent (or hidden or unobserved) random variable $\mathbf{z}$, we restrict π_k by

$$0 \le \pi_k \le 1 \text{ and } \sum_{k=1}^{K} \pi_k = 1 \tag{11.5}$$

Let $\mathbf{z}$ be a discrete random variable and $\mathbf{z} \in \{1,2,\dots,K\}$, and the discrete random variable $\mathbf{z}$ has a probability mass function given by $\{\pi_k, k = 1,2,\dots,K\}$. Although we assume the value of $\mathbf{z}$ is an integer here, we can map $\mathbf{z}$ to a vector space in general. Then, component k in (11.3), $p(\mathbf{x}; \boldsymbol{\theta}_k)$, can be interpreted as the conditional probability distribution of $\mathbf{x}$ given the value of $\mathbf{z}$, say k, denoted as $p(\mathbf{x}|\mathbf{z} = k)$. The joint distribution of $\mathbf{x}$ and $\mathbf{z}$ can be calculated by

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) \tag{11.6}$$

Thus, the mixture model (11.3) can be interpreted as the marginal distribution obtained by summing (11.6) over all $\mathbf{z}$ values.

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$$

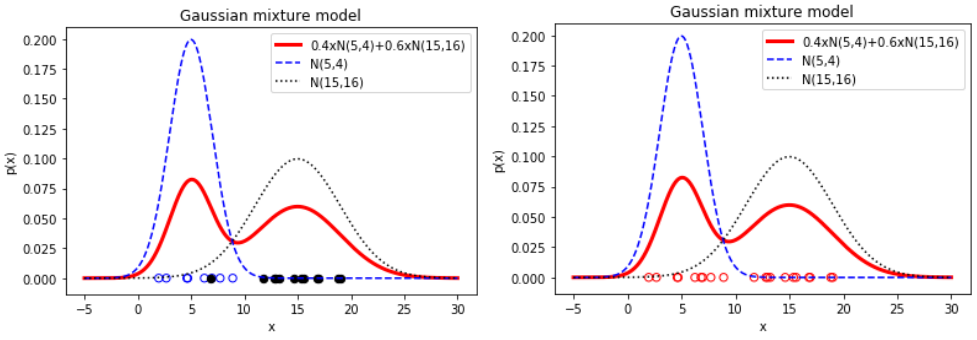
As a result, we have the Gaussian mixture model

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \, \mathbb{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{11.7}$$

Thus, the Gaussian mixture model specified by (11.3) can be represented by a graphic model in Fig.11.3. The parameters of the model include the distribution of $\mathbf{z}$, i.e., $\{\pi_k, k = 1,2,\dots,K\}$ and the Gaussian parameters $\{(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), k = 1,2,\dots,K\}$. To draw a sample from the Gaussian mixture model, we can first generate a sample $\mathbf{z}^{(i)}$ from the distribution $p(\mathbf{z})$, and then draw a sample $\mathbf{x}^{(i)}$ from the conditional distribution $p(\mathbf{x}|\mathbf{z})$ given $\mathbf{z} = \mathbf{z}^{(i)}$.

Given the Gaussian mixture model defined in (11.7) with unknown parameters and a dataset, our goal is to find the optimal values of the parameters to fit the dataset by maximizing the likelihood function. For instance, as illustrated in Fig.11.4, we want to estimate the probability density $p(\mathbf{x})$ (the red solid curve) using a data set. If the data set included both the data points $\mathbf{x}^{(n)}$ and the corresponding component sources $\mathbf{z}^{(n)}$, i.e., the data set is called a *complete dataset* denoted by $\{(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}), n = 1,2,\dots,N\}$, then the maximization would be reduced to estimate each individual Gaussian component, which is straightforward. Fig.11.4 (a) shows a complete data set consisting of 20 data points (8 points from one Gaussian component and the remaining 12 points from another Gaussian component, indicated by blue empty circles and black solid circles, respectively). We can simply estimate the mixture coefficients as $\pi_1 = \frac{8}{20} = 0.4$, and $\pi_2 = \frac{12}{20} = 0.6$, and estimate the first Gaussian distribution by the blue empty-circle data points and the second Gaussian distribution by the black solid-circle data points. With this complete dataset setting, we have a closed-form analytical solution.

However, in practice, the data set is typically *incomplete*, i.e., we are only given the data points $\mathbf{x}^{(n)}$ in the dataset, and the latent variable $\mathbf{z}$ is not observable, as illustrated in Fig.11.4 (b), where all data points are labeled as a red empty-circle. Since we don't know which Gaussian component shall be responsible for a particular data point in the dataset, the estimation of each Gaussian component should depend on all data points, and thus is much more complicated than the case with a complete dataset. The maximization of the likelihood function for an incomplete data set will be treated shortly.



(a) complete data set (b) incomplete data set

Fig.11.4 Complete data set versus incomplete data set

Now consider a question: given the value of **x** sampled from the underlying distribution $p(\mathbf{x})$, how likely does **x** originate from the k-th Gaussian component? Or how much responsibility that the k-th component takes for "representing" the sampled **x**? By Bayes' theorem, we have the posterior probability

$$p(\mathbf{z}=k|\mathbf{x}) = \frac{p(\mathbf{z}=k)\,p(\mathbf{x}|\mathbf{z}=k)}{p(\mathbf{x})} = \frac{\pi_k \mathbb{N}(\mathbf{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K}\pi_j\,\mathbb{N}\left(\mathbf{x};\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j\right)} \tag{11.8}$$

The quantity of this posterior probability plays an important role in the methods of learning the generative models, as we shall see shortly.

11.2. EM Algorithm

11.2.1 EM algorithm for GMMs

The *expectation maximization* algorithm (or EM algorithm) is a general technique for finding maximum likelihood solutions for probabilistic models with latent variables. First, we derive the EM algorithm for Gaussian mixture models as an instance of EM algorithm in this section, and then provide a treatment for the general EM algorithm applicable to more complex models in the next section. The concepts in EM algorithm provide a foundation for

Suppose we have a dataset $\mathcal{D} = \left\{\mathbf{x}^{(1)},\mathbf{x}^{(2)},\ldots,\mathbf{x}^{(N)}\right\}$, $\mathbf{x}^{(n)} \in \mathbb{R}^D$, and each sample $\mathbf{x}^{(n)}$ is drawn independently from the probabilistic model $p(\mathbf{x};\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma})$ in (11.7). Alternative, the dataset can be represented by an $N \times D$ matrix $\mathbf{X}$, in which the nth row is given by $\left(\mathbf{x}^{(n)}\right)^T$. The model parameters $\{\pi_k, k = 1,2,\ldots,K\}$ and $\{(\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k), k = 1,2,\ldots,K\}$ are denoted by $\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}$, respectively. Our goal is to maximize the log likelihood function, which is represented by

$$\ln p(\mathbf{X};\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) = \ln\prod_{n=1}^{N} p\left(\mathbf{x}^{(n)};\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}\right) = \sum_{n=1}^{N}\ln\left[\sum_{k=1}^{K}\pi_k\mathbb{N}\left(\mathbf{x}^{(n)};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right] \tag{11.9}$$

where a Gaussian component is given by

$$\mathbb{N}(\mathbf{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k{}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right\} \quad \text{where} \quad \mathbf{x}\in\mathbb{R}^D \tag{11.10}$$

First, let's calculate the derivative of the log likelihood function with respect to the parameter $\boldsymbol{\mu}_k$ and set it to zero.

$$\frac{\partial \ln p(\mathbf{X};\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma})}{\partial\boldsymbol{\mu}_k} = \sum_{n=1}^{N}\left\{\frac{\pi_k\mathbb{N}\left(\mathbf{x}^{(n)};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)}{\sum_{j=1}^{K}\pi_j\mathbb{N}\left(\mathbf{x}^{(n)};\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j\right)}\boldsymbol{\Sigma}_k{}^{-1}\left(\mathbf{x}^{(n)}-\boldsymbol{\mu}_k\right)\right\} = 0 \tag{11.11}$$

Using the posterior probability (11.8), we rewrite (11.11) as

$$\sum_{n=1}^{N}\left\{p\left(\mathbf{z}=k|\mathbf{x}^{(n)}\right)\boldsymbol{\Sigma}_k{}^{-1}\left(\mathbf{x}^{(n)}-\boldsymbol{\mu}_k\right)\right\} = 0 \tag{11.12}$$

Assuming $\boldsymbol{\Sigma_k}$ is nonsingular, by solving for $\boldsymbol{\mu_k}$, we have

$$\boldsymbol{\mu_k} = \frac{\mathbf{1}}{\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} p(\mathbf{z} = k|\mathbf{x}^{(n)})} \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} p\big(\mathbf{z} = k|\mathbf{x}^{(n)}\big)\mathbf{x}^{(n)} = \frac{\mathbf{1}}{\boldsymbol{N_k}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} p\big(\mathbf{z} = k|\mathbf{x}^{(n)}\big)\mathbf{x}^{(n)} \tag{11.13}$$

where

$$\boldsymbol{N_k} = \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} p\big(\mathbf{z} = k|\mathbf{x}^{(n)}\big) \tag{11.14}$$

$\boldsymbol{N_k}$ can be interpreted as the effective number of data points belonging to the k-th Gaussian component. $p\big(\mathbf{z} = k|\mathbf{x}^{(n)}\big)$ is the weight for $\mathbf{x}^{(n)}$ to be counted for the *k*-th Gaussian component. Note that (11.14) is not a closed-form solution for $\boldsymbol{\mu_k}$ because the terms at the right-hand side, $\boldsymbol{N_k}$ and $p\big(\mathbf{z} = k|\mathbf{x}^{(n)}\big)$, depends on parameters $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$.

Similarly, by setting the derivative of the log likelihood function, with respect to $\boldsymbol{\Sigma_k}$, to zero, we have

$$\boldsymbol{\Sigma_k} = \frac{\mathbf{1}}{\boldsymbol{N_k}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} p\big(\mathbf{z} = k|\mathbf{x}^{(n)}\big)\big(\mathbf{x}^{(n)} - \boldsymbol{\mu_k}\big)\big(\mathbf{x}^{(n)} - \boldsymbol{\mu_k}\big)^{\boldsymbol{T}} \tag{11.15}$$

To maximize the log likelihood function (11.9) with respect to π_k, we need to consider a constraint defined by (11.5). Thus, maximizing (11.9) is equivalent to maximize the Lagrangian function

$$L = \ln p(\mathbf{X}; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \lambda\left(\sum_{k=1}^{K} \pi_k - 1\right) \tag{11.16}$$

By setting $\frac{\partial L}{\partial \pi_k} = 0$ and $\frac{\partial L}{\partial \lambda} = 0$, we have the solution

$$\lambda = -N \tag{11.17}$$

$$\pi_k = \frac{\boldsymbol{N_k}}{N} \tag{11.18}$$

Again, neither (11.15) nor (11.18) is a closed-form solution because some terms at their right-hand side depend on the model parameters. However, (11.13), (11.15) and (11.18) suggest a simple iterative scheme, known as the *expectation-maximization* (EM) algorithm. In the EM algorithm, each iteration consists of two steps: E-step and M-step. In the E-step, we evaluate the posterior probability $p\big(\mathbf{z} = z_k|\mathbf{x}^{(n)}\big)$, i.e., the responsibility of component *k* for sample $\mathbf{x}^{(n)}$, based on the current model parameters. In the M-step, the model parameters are updated in the order of (11.13), (11.15) and (11.18), based on the newest values. For instance, we use the result from (11.13) for $\boldsymbol{\mu_k}$ in (11.15)**.**

The EM algorithm for GMMs is summarized as follows. Suppose we have a GMM $p(\mathbf{X}; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ defined in (11.7), and a dataset of observations $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\}$ independently drawn from the GMM. We want to estimate the parameters $(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ based on the dataset, so that the log likelihood function is maximized. The steps in the EM algorithm are described as:

Step 1: initialize the parameters $\{\pi_k, (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), k = 1,2,\dots,K\}$ and calculate the initial value of the log likelihood function.

Step 2 (E-step): compute the posterior probabilities of $\mathbf{z}$ (i.e., responsibilities of z_k for $\mathbf{x}^{(n)}$) using the current parameters. Note that each sample results in a posterior distribution.

$$p(\mathbf{z} = k|\mathbf{x}^{(n)}) = \frac{\pi_k \mathbb{N}(\mathbf{x}^{(n)}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathbb{N}(\mathbf{x}^{(n)}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \qquad n = 1,2,..,N, k = 1,2,\dots,K \tag{11.19}$$

Step 3 (M-step): update the parameters using the current posterior distributions (or responsibilities) in the following order.

$$\boldsymbol{N_k} = \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} p(\mathbf{z} = k|\mathbf{x}^{(n)}) \tag{11.20}$$

$$\boldsymbol{\mu_k}^{\boldsymbol{new}} = \frac{\mathbf{1}}{\boldsymbol{N_k}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} p(\mathbf{z} = k|\mathbf{x}^{(n)}) \mathbf{x}^{(n)} \tag{11.21}$$

$$\boldsymbol{\Sigma_k}^{\boldsymbol{new}} = \frac{\mathbf{1}}{\boldsymbol{N_k}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} p(\mathbf{z} = k|\mathbf{x}^{(n)}) \left(\mathbf{x}^{(n)} - \boldsymbol{\mu_k}^{\boldsymbol{new}}\right)\left(\mathbf{x}^{(n)} - \boldsymbol{\mu_k}^{\boldsymbol{new}}\right)^{\boldsymbol{T}} \tag{11.22}$$

$$\pi_k{}^{new} = \frac{\boldsymbol{N_k}}{N} \tag{11.23}$$

$$\boldsymbol{\mu_k} \leftarrow \boldsymbol{\mu_k}^{\boldsymbol{new}},\ \boldsymbol{\Sigma_k} \leftarrow \boldsymbol{\Sigma_k}^{\boldsymbol{new}}, \pi_k \leftarrow \pi_k{}^{new}$$

Step 4: compute the log likelihood using (11.9) and check whether either the parameters or the log likelihood is converged. If not**,** return to step 2.

To get more insight into the EM algorithm of Gaussian mixture model, specified by (11.13), (11.14), (11.15), and (11.18), we revisit at the maximum likelihood (ML) solution to the Gaussian mixture model for a complete data set, and then show how we can generalize the ML result to the EM algorithm for incomplete data set scenarios. Given a complete data set $\{(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}), n = 1,2,\dots,N\}$, we can partition the data set into K subsets according to the value of $\mathbf{z}^{(n)}$ so that each subset is associated with one particular value of the latent variable $\mathbf{z}$, then we estimate the parameters for each Gaussian component using the corresponding subset. Specifically, the result can be summarized by

- Count the number of data points in each subset *k,* k=1,2,…,K.

$$\boldsymbol{N_k} = \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} (\mathbf{z}^{(n)} == k) = \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} z_{nk} \tag{11.24}$$

where

$$(\mathbf{z}_n == k) = \begin{cases} 1 & if\ \mathbf{z}^{(n)} = k \\ 0 & else \end{cases} \tag{11.25}$$

For a purpose of comparison later, we map $\mathbf{z}^{(n)} \in \{1,2,\dots,K\}$ to a K-dimensional binary one-hot vector denoted by

$$\overrightarrow{\mathbf{z}^{(n)}} = \begin{bmatrix} z_{n1} \\ z_{n2} \\ \vdots \\ z_{nK} \end{bmatrix} \quad \text{where} \quad z_{nk} = \begin{cases} 1 & if\ \mathbf{z}^{(n)} = k \\ 0 & else \end{cases} \tag{11.26}$$

Thus, $\overrightarrow{\mathbf{z}_n}$ represents an assignment of Gaussian component for sample $\mathbf{x}^{(n)}$ by its "1" element.

- Estimate the mixture coefficient.

$$\pi_k = \frac{\boldsymbol{N_k}}{N} \tag{11.27}$$

- Estimate the mean and covariance for each subset.

$$\boldsymbol{\mu_k} = \frac{\mathbf{1}}{\boldsymbol{N_k}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} z_{nk}\, \mathbf{x}^{(n)} \tag{11.28}$$

$$\boldsymbol{\Sigma_k} = \frac{\mathbf{1}}{\boldsymbol{N_k}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} z_{nk} \left(\mathbf{x}^{(n)} - \boldsymbol{\mu_k}\right)\left(\mathbf{x}^{(n)} - \boldsymbol{\mu_k}\right)^{\boldsymbol{T}} \tag{11.29}$$

When the value of $\mathbf{z}^{(n)}$ is available, i.e., given by a complete dataset, then $\overrightarrow{\mathbf{z}^{(n)}}$ is a binary one-hot vector with the k-element $z_{nk} = 1$ for $\mathbf{z}^{(n)} = k$.

Since the value of $\mathbf{z}^{(n)}$ is not available in an incomplete dataset $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\}$, we don't have exact binary correspondence (0 or 1) between a data point $\mathbf{x}^{(n)}$ and Gaussian components. Instead, each data point $\mathbf{x}^{(n)}$ is associated with Gaussian component k in some degree specified by the posterior probability $\hat{z}_{nk} = p(\mathbf{z} = k|\mathbf{x}^{(n)})$, i.e., the responsibility of Gaussian component k for $\mathbf{x}^{(n)}$. Thus, $\overrightarrow{\mathbf{z}^{(n)}}$ can be generalized from a binary one-hot vector to the posterior probability distribution of $\mathbf{z}^{(n)}$ over k, for a given $\mathbf{x}^{(n)}$. As a result, the one-hot vector in (11.26) (i.e., *hard assignment*) will blur to a probability distribution $\{\hat{z}_{nk}, k = 1,2,\dots,K\}$, i.e., a *soft assignment*. To generalize the ML solution (11.24) - (11.29) for a complete dataset to the EM algorithm for an incomplete data set, we just simply replace z_{nk} in (11.24-11.29) with the estimated responsibility $\hat{z}_{nk}$, and obtain (11.13), (11.14), (11.15), and (11.18). In the EM algorithm, the E-step evaluates the responsibilities of Gaussian components for datapoints using the current model parameters, and then the M-step maximizes the log likelihood function based on the estimated responsibilities. The iterations of (E-step, M-step) result in the convergence of the solution.

11.2.2 EM Algorithm for Latent Variable Models in General

In this section, we provide a formal treatment of EM algorithm for a general latent variable model. A latent variable model is a probabilistic model for which certain variables are never observed. For example, random variable **z** in GMMs is a latent variable while **x** is an observed variable. In our subsequent context, the latent variable and the observed variable are denoted by **z** and **x**, respectively. All parameters of the model are

denoted by $\theta \in \Theta$. In GMMs, for instance, θ includes all elements in $\{\pi_k, (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), k = 1,2, \dots, K\}$. The joint probability distribution of the model is $p(\mathbf{x}.\mathbf{z}|\theta)$.

Problem formulation

We denote a complete dataset by $\{(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}), n = 1,2, \dots, N\}$, and an incomplete data set as $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\}$. In a learning problem, we are given the incomplete data set $\mathbf{X}$, and the goal is to find the maximum likelihood estimate (MLE) of the model parameters,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, p(\mathbf{X}|\theta) \tag{11.30}$$

In an inference problem, given a new datapoint (or sample) $\mathbf{x}$, we want to find the conditional distribution over z,

$$p(\mathbf{z}|\mathbf{x}, \theta) \tag{11.31}$$

In practice, it is usually convenient to maximize the log likelihood,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \ln\big(p(\mathbf{X}|\theta)\big) \tag{11.31}$$

Note that the gradient descent method is typically not a good option to solve (11.31) (see exercise11.1).

Since the complete data log-likelihood is typically easy to optimize by $\max_{\theta} \ln p(\mathbf{X}, \mathbf{z}|\theta)$ (e.g., GMM for complete dataset), if we had a distribution $q(\mathbf{z})$ for the latent variable $\mathbf{z}$, then we would maximize the expected complete data log-likelihood, i.e.,

$$\hat{\theta} = arg\max_{\theta} \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}[\ln p(\mathbf{X}, \mathbf{z}|\theta)] = arg\max_{\theta} \textstyle\sum_z q(\mathbf{z}) \ln p(\mathbf{X}, \mathbf{z}|\theta) \tag{11.32}$$

Of course, the effectiveness of the result depends on how close $q(\mathbf{z})$ is to the true underlying distribution $p(\mathbf{z})$.

Jesen's inequality and DL divergence

The derivation of EM algorithm involves two math concepts: Jensen's inequality and Kullback-Leibler (KL) divergence. The Jensen's inequality is stated as: if function f is a *convex* function, and $\mathbf{x}$ is a random variable, then

$$\mathbb{E}[f(\mathbf{x})] \geq f(\mathbb{E}[\mathbf{x}]) \tag{11.33}$$

If the function *f* is a *concave* function, then the direction of inequality in (11.33) is reversed. Furthermore, if the function is strictly convex or concave, the equality holds if only if $\mathbf{x}$ is a constant. (proof, see exercise, and Bishop pp.56)

KL divergence is defined to measure how different two distribution functions are, say $p(\mathbf{z})$ and $q(\mathbf{z})$, which is given by

$$KL(p||q) = \sum_z p(\mathbf{z}) \ln \frac{p(\mathbf{z})}{q(\mathbf{z})} \tag{11.34}$$

Using Jensen's inequality, we can prove that 1) $KL(p||q) \geq 0, \forall\, p, q$; and 2) $KL(p||q) = 0$, if and only if $p(\mathbf{z}) = q(\mathbf{z}), \forall \mathbf{z}$. (see exercise)

Evidence lower bound (ELBO)

Now let's find the lower bound of the marginal log-likelihood $\ln\big(p(\mathbf{X}|\theta)\big)$ using Jensen's inequality. Since the marginal log-likelihood on a data set is equal to the sum of the marginal log-likelihoods over samples

$$\ln\big(p(\mathbf{X}|\theta)\big) = \sum_{n=1}^{N} \ln\Big(p\big(\mathbf{x}^{(n)}|\theta\big)\Big) \tag{11.35}$$

Since the number of samples in the dataset is not essential for our exposition of EM algorithm, we consider a single sample **x** for simplicity, and then obtain the final results by adding the operation of summation over the data samples.

Given a data sample **x**, by Jesen's inequality, for any choice of a distribution $q(\mathbf{z})$, we have

$$\ln\big(p(\mathbf{x}|\theta)\big) = \ln\left\{\sum_{\mathbf{z}} p(\mathbf{x},\mathbf{z}|\theta)\right\} = \ln\left\{\sum_{\mathbf{z}} q(\mathbf{z})\frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z})}\right\} \geq \sum_{\mathbf{z}} q(\mathbf{z})\ln\left[\frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z})}\right] = l_q(\theta) \tag{11.36}$$

The quantity $l_q(\theta)$ is called the *evidence lower bound (ELBO)* given a single sample **x**. Any choice of $q(\mathbf{z})$ results in a different lower bound curve $l_q(\theta)$ which is a function of θ. Furthermore, $\ln\big(p(\mathbf{x}|\theta)\big)$ can be decomposed into two components: the lower bound and the KL divergence for a given $q(\mathbf{z})$,

$$\begin{aligned}\ln\big(p(\mathbf{x}|\theta)\big) &= \left[\sum_{\mathbf{z}} q(\mathbf{z})\right]\ln\big(p(\mathbf{x}|\theta)\big) = \sum_{\mathbf{z}}\left[q(\mathbf{z})\ln\left(\frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z})}\frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x},\theta)}\right)\right] \\ &= \sum_{\mathbf{z}} q(\mathbf{z})\ln\left[\frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z})}\right] + \sum_{\mathbf{z}} q(\mathbf{z})\ln\left(\frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x},\theta)}\right) \\ &= l_q(\theta) + KL\big(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x},\theta)\big)\end{aligned} \tag{11.37}$$

The decomposition (11.37) implies that the gap between the log-likelihood and the ELBO is equal to the divergence between the "guessed" distribution $q(\mathbf{z})$ and the true posterior probability distribution over **z** given **x**. To make the lower bound tight (i.e., $\ln\big(p(\mathbf{x}|\theta)\big) = l_q(\theta)$, or equivalently $KL\big(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x},\theta)\big) = 0$), it is sufficient and necessary to set the distribution $q(\mathbf{z})$ to be the true posterior distribution $p(\mathbf{z}|\mathbf{x},\theta)$

$$q(\mathbf{z}) = p(\mathbf{z}|\mathbf{x},\theta) \tag{11.38}$$

Adding the summation operation on (11.37) over all samples in the dataset, we have the log-likelihood on the data set **X**,

$$\begin{aligned}\ln\big(p(\mathbf{X}|\theta)\big) &= \sum_{n=1}^{N} l_{q_n}(\theta) + \sum_{n=1}^{N} KL\left(q_n(\mathbf{z})||p\big(\mathbf{z}|\mathbf{x}^{(n)},\theta\big)\right) \\ &= \mathcal{L}_q(\theta) + KL\big(q(\mathbf{z})||p(\mathbf{z}|\mathbf{X},\theta)\big)\end{aligned} \tag{11.39}$$

where

$$\mathcal{L}_q(\theta) = \sum_{n=1}^{N} l_{q_n}(\theta) \qquad \textit{is ELBO}$$

$$l_{q_n}(\theta) = \sum_{\mathbf{z}} q_n(\mathbf{z}) \ln\left[\frac{p\left(\mathbf{x}^{(n)}, \mathbf{z}|\theta\right)}{q_n(\mathbf{z})}\right] \qquad \textit{is ELBO component for sample } \mathbf{x}^{(n)}$$

$q_n(\mathbf{z})$ *is the selected distribution of* z *for sample* $\mathbf{x}^{(n)}$

$$KL\big(q(\mathbf{z})||p(\mathbf{z}|\mathbf{X},\theta)\big) = \sum_{n=1}^{N} KL\left(q_n(\mathbf{z})||p\big(\mathbf{z}|\mathbf{x}^{(n)},\theta\big)\right) \qquad \textit{is the total KL divergence between}$$

$q_n(\mathbf{z})$ *and* $p\big(\mathbf{z}|\mathbf{x}^{(n)},\theta\big), n = 1,2,\dots,N$

Since the GMM is an example of latent variable models, we can visualize the ELBO using a particular GMM. Consider the Gaussian mixture model in Fig.11.2. The dataset consists of 20 data samples, $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(20)}\}$, as shown in Fig.11.4(b). The parameters of the underlying model are $\pi_1 = 0.4, \pi_2 = 0.6, \mu_1 = 5, \Sigma_1 = 4, \mu_2 = 15, \Sigma_2 = 16$. For simplicity, we only learn one parameter μ_1 denoted by θ, assuming all remaining model parameters, i.e., $\pi_1, \pi_2, \Sigma_1, \mu_2, \Sigma_2$, are known. The red solid curve in Fig.11.5 shows the log-likelihood on the dataset. The black dotted curve is the ELBO for an arbitrarily particular set of distributions $\{q_n(z),\ z \in \{1,2\}, n = 1,2,\dots,N\}$. There is always a gap between the log-likelihood and this ELBO instance, which implies that the KL in (11.39) is more than zero, i.e., $q(\mathbf{z})$ is not the same as $p(\mathbf{z}|\mathbf{X},\theta)$, for any θ.

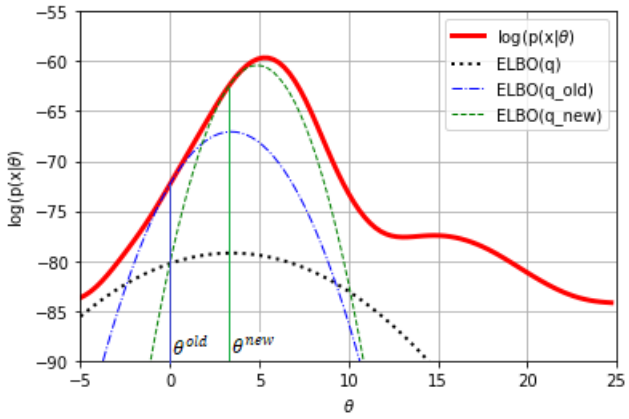


Fig.11.5 The log-likelihood and a few lower bounds

To find an ELBO curve that touches the log-likelihood curve at some point, say θ^{old}, i.e., KL is zero at θ^{old}, we need to choose the posterior probabilities for the distributions $q(\mathbf{z})$, denoted as $q^{old}(\mathbf{z})$, given by

$$q_n^{old}(\mathbf{z}) = p\big(\mathbf{z}|\mathbf{x}^{(n)}, \theta^{old}\big) \tag{11.40}$$

Note that for each sample there is a posterior probability distribution.

Thus, the corresponding ELBO on the data set is a function of θ,

$$\mathcal{L}_{q^{old}}(\theta) = \sum_{n=1}^{N} \sum_{\mathbf{z}} q_n^{old}(\mathbf{z}) \ln\left[\frac{p\left(\mathbf{x}^{(n)}, \mathbf{z}|\theta\right)}{q_n^{old}(\mathbf{z})}\right] \tag{11.41}$$

and

$$\ln\left(p\left(\mathbf{X}|\theta^{old}\right)\right) = \mathcal{L}_{q^{old}}\left(\theta^{old}\right) \tag{11.42}$$

As a result, the blue dash-dot curve in Fig.11.5 corresponds to the ELBO $\mathcal{L}_{q^{old}}(\theta)$, with $q(\mathbf{z})$ specified by (11.40) and $\theta^{old} = 0$. However, as shown in Fig.11.5, $\theta^{old} = 0$ does not get the maximum of ELBO $\mathcal{L}_{q^{old}}(\theta)$. If we can find

$$\theta^{new} = arg\max_{\theta} \mathcal{L}_{q^{old}}(\theta) \tag{11.43}$$

then we have

$$\begin{aligned} \ln\left(p(\mathbf{X}|\theta^{new})\right) &> \mathcal{L}_{q^{old}}(\theta^{new}) && Lower\ bound \\ &> \mathcal{L}_{q^{old}}\left(\theta^{old}\right) && because\ \theta^{new} = arg\max_{\theta} \mathcal{L}_{q^{old}}(\theta) \\ &= \ln\left(p\left(\mathbf{X}|\theta^{old}\right)\right) && see\ (11.42) \end{aligned}$$

Thus, we have

$$\ln\left(p(\mathbf{X}|\theta^{new})\right) > \ln\left(p\left(\mathbf{X}|\theta^{old}\right)\right) \tag{11.44}$$

(11.44) implies that the update of θ in (11.43) always makes the log-likelihood monotonically increase. Therefore, by replacing the value of θ^{old} in (11.40) with the value of θ^{new} calculated in (11.43), and repeat (11.40), (11.41) and (11.43), we can achieve the convergence of θ to a local maximum. Since this approach is susceptible to local optima, it is common practice to reinitialize θ at different points.

Back to Fig.11.5, we can obtain the maximum point of $\mathcal{L}_{q^{old}}(\theta)$ at $\theta = \theta^{new} = 3.446$ by a closed form (11.21) for the GMM. Then we plot a new ELBO $\mathcal{L}_{q^{new}}(\theta)$ curve (the dashed green curve) based on a new distribution $q(\mathbf{z})$ specified by (11.40) using $\theta^{old} = 3.446$. The maximum point of the new ELBO is closer to the maximum point of the log-likelihood, compared to the old ELBO.

[**Lemma**] Suppose we have found a global maximum of $\mathcal{L}_q(\theta)$,

$$\mathcal{L}_{q*}(\theta^*) \geq \mathcal{L}_q(\theta), \forall q, \theta \tag{11.45}$$

Then θ^* is a global maximum of $\ln\left(p(\mathbf{X}|\theta)\right)$, i.e.,

$$\ln\left(p(\mathbf{X}|\theta^*)\right) \geq \ln\left(p(\mathbf{X}|\theta)\right), \forall\theta \tag{11.46}$$

Proof: let $q^*(\mathbf{z}) = p(\mathbf{z}|\mathbf{X}, \theta^*)$, we have

$$\ln\left(p(\mathbf{X}|\theta^*)\right) = \mathcal{L}_q(\theta^*) + KL\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{X}, \theta^*)\right) = \mathcal{L}_{q*}(\theta^*) + KL\left(q^*(\mathbf{z})||p(\mathbf{z}|\mathbf{X}, \theta^*)\right) = \mathcal{L}_{q*}(\theta^*)$$

for any θ', let $q'(\mathbf{z}) = p(\mathbf{z}|\mathbf{X}, \theta')$, we have

$$\begin{aligned} &\ln\left(p(\mathbf{X}|\theta')\right) = \mathcal{L}_q(\theta') + KL\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{X}, \theta')\right) \\ &= \mathcal{L}_{q'}(\theta') + KL\left(q'(\mathbf{z})||p(\mathbf{z}|\mathbf{X}, \theta')\right) = \mathcal{L}_{q'}(\theta') \leq \mathcal{L}_{q*}(\theta^*) \end{aligned}$$

Thus, $\ln\left(p(\mathbf{X}|\theta^*)\right) \geq \ln\left(p(\mathbf{X}|\theta')\right)$, for any θ'.

EM algorithm

The analysis of ELBO above suggests an iteration algorithm, called the EM algorithm, to find the model parameters maximizing the log-likelihood for a general latent variable model. The EM algorithm for GMMs in Section 11.2.1 is a special case of the EM algorithm. In a general EM algorithm, each iteration consists of two consecutive steps: E-step and M-step, as illustrated by Fig.11.6.

Suppose the current parameters is denoted by θ^{old}. In E-step, our goal is to maximize the lower bound over $q(\boldsymbol{z})$ with fixed parameters θ^{old}. Since the log-likelihood function $\ln p(\mathbf{X}|\theta)$ does not depend on $q(\boldsymbol{z})$, the maximum of the lower bound occurs when $KL(q||p)$ is equal to zero, i.e., $q(\boldsymbol{z})$ is equal to the posterior distribution of the latent variable at current parameters. Thus, we calculate the posterior distribution, and compute the expectation of the log-likelihood under the posterior distribution, i.e., the ELBO.

In M-step, holding q fixed, we update the model parameters by maximizing $\mathcal{L}_q(\theta)$ over the parameters, where q is the one computed in the E-step, i.e., the posterior distribution at current parameters θ^{old}, $p(\mathbf{z}|\mathbf{X},\theta^{old})$. Substituting $q(\boldsymbol{z}) = p(\mathbf{z}|\mathbf{X},\theta^{old})$ into (11.36), we have

$$\begin{aligned}\mathcal{L}_q(\theta) &= \sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{X},\theta^{old}) \ln \frac{p(\boldsymbol{X},\boldsymbol{z}|\theta)}{p(\boldsymbol{z}|\boldsymbol{X},\theta^{old})} \\ &= \sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{X},\theta^{old}) \ln p(\boldsymbol{X},\boldsymbol{z}|\theta) - \sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{X},\theta^{old}) \ln p(\boldsymbol{z}|\boldsymbol{X},\theta^{old}) \end{aligned} \quad (11.47)$$

Since the second term in (11.47) is independent of θ, the quantity to be maximized in M-step is the first term in (11.47), which is the expectation of the complete-data log likelihood function with respect to the posterior distribution of $\mathbf{z}$. Thus, we have

$$\theta^{new} = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}_q(\theta) = \underset{\theta}{\operatorname{argmax}} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{X},\theta^{old}) \ln p(\boldsymbol{X},\boldsymbol{z}|\theta) \quad (11.48)$$

The updated parameters θ^{new} leads to a larger lower bound value $\mathcal{L}_q(\theta^{new})$ and a non-zero KL divergence, and thus resulting in an increased log likelihood value $\ln p(\mathbf{X}|\theta^{new})$.

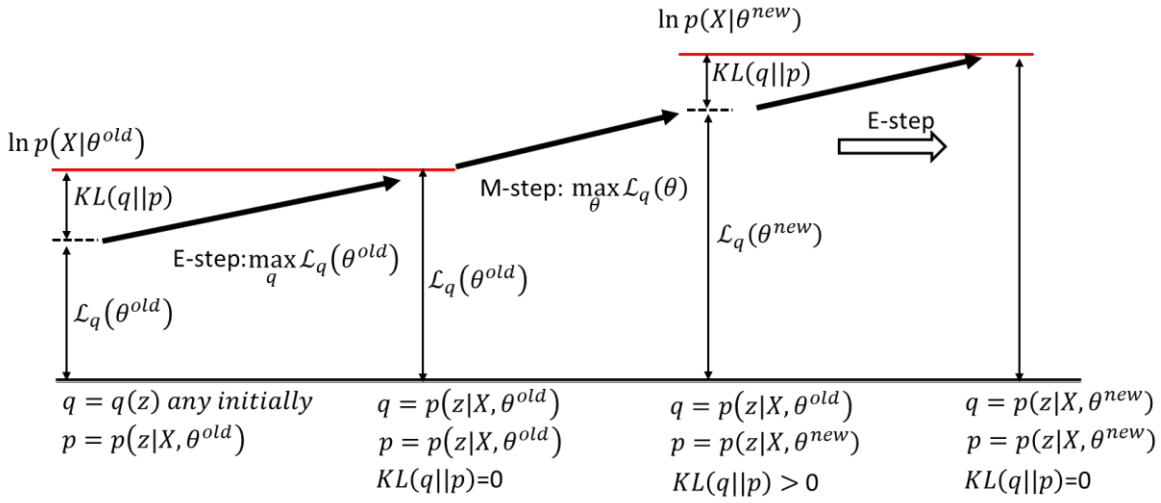


Fig.11.6 Illustration of EM algorithm

Suppose we are given a latent variable model (but the model parameters are unknown) and a dataset $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\}$. The goal is to find the values of model parameters that maximize the likelihood function on the dataset. The steps in the EM algorithm are described as:

Step 1: choose the initial parameters θ^{old}

Step 2 (E-step):

- Choose an optimal distribution, which is the posterior probability, based on the dataset and current parameters θ^{old}

$$q_n(\mathbf{z}) := arg\max_q \mathcal{L}_q(\theta^{old}) = p(\mathbf{z}|\mathbf{x}^{(n)}, \theta^{old}), n = 1,2, \dots, N \tag{11.49}$$

- compute the expectation of the log-likelihood, i.e., the current ELBO

$$\mathcal{L}_q(\theta) := \sum_{n=1}^{N} \sum_{\mathbf{z}} q_n(\mathbf{z}) \ln \left[\frac{p(\mathbf{x}^{(n)}, \mathbf{z}|\theta)}{q_n(\mathbf{z})} \right] \tag{11.50}$$

Step 3 (M-step): update the parameters by maximizing the current ELBO.

$$\theta^{new} := arg\max_{\theta} \mathcal{L}_q(\theta) \tag{11.51}$$

Step 4: check whether either the parameters or the log likelihood function is converged. If not, $\theta^{old} \leftarrow \theta^{new}$, and return to step 2.

Note that if the latent variable is continuous, our discussion will apply equally well by replacing the sum over $\mathbf{z}$ with the corresponding integral.

Convergence of EM algorithm can be described as follows. Let θ^i be the value of EM algorithm after i steps. We define the transition function M( ) such that $\theta^{i+1} = M(\theta^i)$. Suppose the log-likelihood function $\ln(p(\mathbf{X}|\theta))$ is differentiable. Let S be the set of stationary points of $\ln(p(\mathbf{X}|\theta))$. For any starting point θ^0, we have

$$\lim_{i \to \infty} \theta^i = \theta^* \in S$$

$$\theta^* = M(\theta^*)$$

$\ln\left(p(\mathbf{X}|\theta^i)\right)$ monotonically increases to $\ln(p(\mathbf{X}|\theta^*))$ as $i \to \infty$.

For GMMs, both E-step (11.49, 11.50) and M-step (11.51) have a closed-form analytical solution. In practice, it may be difficult to compute (11.50) and (11.51) if the model is such that working with the true posterior distribution is intractable. Our strategy is to make the computation of E-step and M-step easy by reasonable approximation. For example, in E-step we restrict the $q_n(\mathbf{z})$ in (11.50) in a subset of distributions $\mathbb{Q}$ (not posterior distribution) so that it is easy (tractable) to work with (11.50) and we select $q_n(\mathbf{z})$ from $\mathbb{Q}$ by

$$q_n(\mathbf{z}) = arg\min_{q \in \mathbb{Q}} KL\left(q||p(\mathbf{z}|\mathbf{x}^{(n)}, \theta^{old})\right) \tag{11.52}$$

For M-step, instead of finding the maximum point of the current ELBO in (11.51), we just need to find a better new point such that

$$\mathcal{L}_q(\theta^{new}) \geq \mathcal{L}_q\left(\theta^{old}\right) \tag{11.53}$$

11.3. Variational Auto-encoder (VAE)

Variational auto-encoder (VAE) (Kingma and Welling, 2014) refers to a framework that extends the EM algorithm to more complex models by neural networks. In the framework of VAE, we approximate the intractable underlying posterior distribution $p\left(\mathbf{z}|\mathbf{x}^{(n)}\right)$ by a family of distributions that are easy to work with (e.g., parameterized Gaussian distributions). Since neural networks are used to approximate the relevant distributions, the gradient-based optimization can be easily applied using a standard framework such as TensorFlow or PyTorch.

11.3.1 Variational Lower Bound

In general, our latent variable model can be represented by a joint distribution $p_\theta(\mathbf{x}, \mathbf{z})$ over both the observed variables $\mathbf{x}$ and the latent variables $\mathbf{z}$. The marginal distribution over the observed variables $p_\theta(\mathbf{x})$ is given by

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \tag{11.54}$$

For a datapoint $\mathbf{x}$, $p_\theta(\mathbf{x})$ is called the *marginal likelihood* or the *model evidence*. If $\mathbf{z}$ is discrete and $p_\theta(\mathbf{x}|\mathbf{z})$ is a Gaussian distribution, then $p_\theta(\mathbf{x})$ is a mixture Gaussian distribution. For continuous latent variables $\mathbf{z}$, $p_\theta(\mathbf{x})$ can be viewed as an infinite mixture.

The simplest and most common latent variable model is the one with the following factorization

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \tag{11.55}$$

The goal of maximum likelihood learning is to look for the parameter values that maximize the likelihood or (equivalently the log-likelihood) on the dataset $\mathbf{X} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}\right\}$,

$$\theta^{ML} = arg\max_\theta \sum_{n=1}^{N} \ln p_\theta\left(\mathbf{x}^{(n)}\right)$$

Like the EM algorithm, the optimization objective of the VAE is equivalently the ELBO. For any choice of distribution $q(\mathbf{z})$, we have the log-likelihood on a single datapoint

$$\begin{aligned} \ln p_\theta(\mathbf{x}) &= \mathbb{E}_{q(\mathbf{z})}[\ln p_\theta(\mathbf{x})] \\ &= \mathbb{E}_{q(\mathbf{z})}\left[\ln \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})}\right] + \mathbb{E}_{q(\mathbf{z})}\left[\ln \frac{q(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \\ &= \mathcal{L}_\mathbf{q}(\theta; \mathbf{x}) + KL\left(q(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})\right) \end{aligned} \tag{11.56a}$$

The best choice of $q(\mathbf{z})$ should be the true posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ of $\mathbf{z}$ for a given datapoint $\mathbf{x}$, such that the KL divergence is zero, i.e., the log-likelihood is equal to the ELBO. However, the true posterior distribution is intractable in general.

In the VAE framework, we use a neural network to generate $q_\phi(\mathbf{z}|\mathbf{x})$, which approximates $p_\theta(\mathbf{z}|\mathbf{x})$. This approximation makes the problem tractable. Furthermore, we don't need to computer a posterior distribution per datapoint. Instead, the neural network generates different posterior distribution approximation per input x, with the shared variational inference parameters ϕ across the datapoints. This variational inference is called *amortized variational inference* that can improve the computing efficiency.

For any choice of inference model $q_\phi(\mathbf{z}|\mathbf{x})$, including the form of $q_\phi(\mathbf{z}|\mathbf{x})$ and the variational parameters ϕ, we have

$$\begin{aligned}\ln p_\theta(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\ln p_\theta(\mathbf{x})] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \\ &= \mathcal{L}(\theta,\phi;\mathbf{x}) + KL\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})\right) \qquad (11.56b)\end{aligned}$$

The first term in (11.56b) is the variational lower bound or (ELBO). The ELBO can be written as

$$\mathcal{L}(\theta,\phi;\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x},\mathbf{z}) - \ln q_\phi(\mathbf{z}|\mathbf{x})\right] \qquad (11.57a)$$

Or

$$\mathcal{L}(\theta,\phi;\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\ln p_\theta(\mathbf{x}|\mathbf{z})\,] - KL\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right) \qquad (11.57b)$$

The second term in (11.56b) is the KL divergence between $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z}|\mathbf{x})$, which is non-negative. From our previous analysis, we conclude that 1) for a fixed θ, maximizing $\mathcal{L}(\theta,\phi;\mathbf{x})$ over ϕ leads to minimizing the KL divergence in (11.56b); 2) for a fixed ϕ, maximizing $\mathcal{L}(\theta,\phi;\mathbf{x})$ over θ will increase the likelihood $p_\theta(\mathbf{x})$. The global maximum of $\mathcal{L}(\theta,\phi;\mathbf{x})$ corresponds to the global maximum of $p_\theta(\mathbf{x})$ (see Lemma). Therefore, maximization of the ELBO $\mathcal{L}(\theta,\phi;\mathbf{x})$, w.r.t. the parameters θ and ϕ, will simultaneously and approximately maximize the likelihood $p_\theta(\mathbf{x})$ and minimize the KL divergence between the approximation $q_\phi(\mathbf{z}|\mathbf{x})$ and the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

The ELBO on the dataset is given by the sum of ELBOs over datapoints,

$$\mathcal{L}(\theta,\phi;\mathbf{X}) = \sum_{n=1}^{N} \mathcal{L}\left(\theta,\phi;\mathbf{x}^{(n)}\right) \qquad (11.58)$$

11.3.2 Gradients of Variational Lower Bound

An important property of the ELBO is that it can be jointly optimized w.r.t. all parameters θ and ϕ using stochastic gradient ascent. The gradients of the individual datapoint ELBO w.r.t. the generative model parameters θ can be estimated unbiasedly by

$$\nabla_\theta \mathcal{L}(\theta,\phi;\mathbf{x}) = \nabla_\theta \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x},\mathbf{z}) - \ln q_\phi(\mathbf{z}|\mathbf{x})\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\nabla_\theta(\ln p_\theta(\mathbf{x},\mathbf{z}))] \qquad (11.59)$$

However, computing the gradient of the ELBO w.r.t. the inference model parameters ϕ is tricky because the expectation $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}$ in (11.57) is taken w.r.t. the distribution $q_\phi(\mathbf{z}|\mathbf{x})$, which depends on ϕ. In general,

$$\nabla_{\phi}\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[f(z)] \neq \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\nabla_{\phi} f(z)] \tag{11.60}$$

Reparameterization trick

To compute the gradients of the ELBO straightforwardly, a technique called *reparameterization trick* is introduced to represent the ELBO in an expectation taken w.r.t. another distribution $p(\boldsymbol{\epsilon})$, which is independent of ϕ and $\mathbf{x}$. Suppose there exists an invertible transformation on $\boldsymbol{\epsilon}$

$$\boldsymbol{z} = g(\boldsymbol{\epsilon}, \phi, \mathbf{x}) \tag{11.61}$$

where random variable $\boldsymbol{\epsilon}\sim p(\boldsymbol{\epsilon})$, such that $\boldsymbol{z}\sim q_{\phi}(\mathbf{z}|\mathbf{x})$. For example, if $\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0}, \mathbf{I})$, then $\mathbf{z}\sim\mathcal{N}\left(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)\right)$ with the transformation

$$\boldsymbol{z} = g(\boldsymbol{\epsilon}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{11.62}$$

where $\boldsymbol{\mu}, \boldsymbol{\sigma}$ are the mean vector and standard deviation vector, respectively. $\odot$ denotes the element-wise product. The covariance matrix $\text{diag}(\boldsymbol{\sigma}^2)$ is a full rank diagonal matrix whose diagonal entries are $\sigma_i{}^2$.

With the transformation (11.61), for a function $f(z)$ we have

$$\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[f(z)] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g(\boldsymbol{\epsilon}, \phi, \mathbf{x}))] \tag{11.63}$$

The gradient of $\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[f(z)]$ w.r.t. ϕ can be computed by

$$\nabla_{\phi}\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[f(z)] = \nabla_{\phi}\mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g(\boldsymbol{\epsilon}, \phi, \mathbf{x}))] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\phi} f(g(\boldsymbol{\epsilon}, \phi, \mathbf{x}))] \tag{11.64}$$

Fig.11.7 illustrates the backpropagation of gradients after the reparameterization trick is applied.

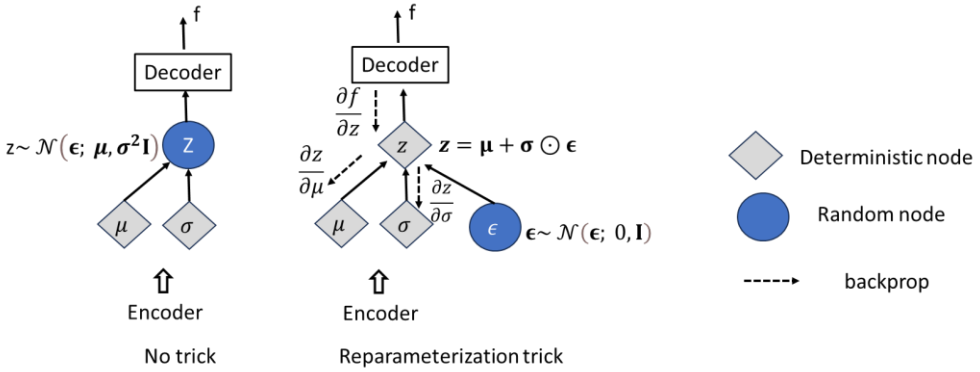


Fig.11.7 Backpropagation after a reparameterization trick

Then we can perform a simple Monte Carlo (MC) estimator on the RHS of (11.64) where we draw samples $\boldsymbol{\epsilon}\sim p(\boldsymbol{\epsilon})$. As a result, $\nabla_{\phi}\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[f(z)]$ can be estimated by

$$\nabla_{\phi}\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[f(z)] \simeq \nabla_{\phi}\tilde{f}(g(\boldsymbol{\epsilon}, \phi, \mathbf{x})) \tag{11.65}$$

where $\tilde{f}(g(\boldsymbol{\epsilon}, \phi, \mathbf{x}))$ is a Monte Carlo estimation of $f(g(\boldsymbol{\epsilon}, \phi, \mathbf{x}))$

$$\tilde{f}(g(\boldsymbol{\epsilon}, \phi, \mathbf{x})) = \frac{1}{L}\sum_{l=1}^{L} f\left(g(\boldsymbol{\epsilon}^{(l)}, \phi, \mathbf{x})\right) \quad \textit{where L is the length of MC, } \boldsymbol{\epsilon}^{(l)}\sim p(\boldsymbol{\epsilon}). \tag{11.66}$$

Stochastic Gradient Variational Bound (SGVB) Estimator

With the reparameterization trick, the single datapoint ELBO in (11.57) can be rewritten as

$$\mathcal{L}(\theta,\phi;\mathbf{x}) = \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[\ln p_\theta(\mathbf{x},\mathbf{z}) - \ln q_\phi(\mathbf{z}|\mathbf{x})\big] \tag{11.67a}$$

$$\mathcal{L}(\theta,\phi;\mathbf{x}) = \mathbb{E}_{p(\boldsymbol{\epsilon})}[\ln p_\theta(\mathbf{x}|\mathbf{z})\,] - KL\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right) \tag{11.67b}$$

where $\boldsymbol{\epsilon}\sim p(\boldsymbol{\epsilon})$, $\boldsymbol{z} = g(\boldsymbol{\epsilon},\phi,\mathbf{x})$, so that $\boldsymbol{z}\sim q_\phi(\mathbf{z}|\mathbf{x})$.

We perform an MC estimation on the ELBO in (11.67a) on datapoint $\mathbf{x}^{(\boldsymbol{n})}$

$$\mathcal{L}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big) \simeq \widetilde{\mathcal{L}^A}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big) = \frac{1}{L}\sum_{l=1}^{L}\big[\ln p_\theta\big(\mathbf{x}^{(\boldsymbol{n})},\mathbf{z}^{(\boldsymbol{n},\boldsymbol{l})}\big) - \ln q_\phi\big(\mathbf{z}^{(\boldsymbol{n},\boldsymbol{l})}|\mathbf{x}^{(\boldsymbol{n})}\big)\big] \tag{11.68}$$

where $\mathbf{z}^{(\boldsymbol{n},\boldsymbol{l})} = g\big(\boldsymbol{\epsilon}^{(\boldsymbol{n},\boldsymbol{l})},\phi,\mathbf{x}\big)$, and $\boldsymbol{\epsilon}^{(\boldsymbol{n},\boldsymbol{l})}\sim p(\boldsymbol{\epsilon})$.

Since the KL divergence in (11.67b) can be integrated into a closed form in some typical situations (we will see later), we only need to sample the first term at the RHS of (11.67b) for the ELBO estimate. This gives us a second way to the estimate of ELBO, according to (11.67b),

$$\mathcal{L}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big) \simeq \widetilde{\mathcal{L}^B}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big) = -KL\left(q_\phi\big(\mathbf{z}|\mathbf{x}^{(\boldsymbol{n})}\big)||p_\theta(\mathbf{z})\right) + \frac{1}{L}\sum_{l=1}^{L}\ln p_\theta\big(\mathbf{x}^{(\boldsymbol{n})}|\mathbf{z}^{(\boldsymbol{n},\boldsymbol{l})}\big) \tag{11.69}$$

where $\mathbf{z}^{(\boldsymbol{n},\boldsymbol{l})} = g\big(\boldsymbol{\epsilon}^{(\boldsymbol{n},\boldsymbol{l})},\phi,\mathbf{x}\big)$, and $\boldsymbol{\epsilon}^{(\boldsymbol{n},\boldsymbol{l})}\sim p(\boldsymbol{\epsilon})$. Please note that the estimators in (11.68) and (11.69) are unbiased.

Given a minibatch $\mathcal{M} = \big\{\mathbf{x}^{(1)},\mathbf{x}^{(2)},\ldots,\mathbf{x}^{(M)}\big\}$, which is randomly drawn from the full dataset $\mathbf{X}$, we can construct an estimator of the full dataset ELBO, based on the minibatch datapoints,

$$\mathcal{L}(\theta,\phi;\mathbf{X}) \simeq \tilde{\mathcal{L}}^{\mathcal{M}}(\theta,\phi;\boldsymbol{\mathcal{M}}) = \frac{N}{M}\sum_{n=1}^{M}\tilde{\mathcal{L}}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big) \tag{11.70}$$

where $\tilde{\mathcal{L}}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big)$ is equal to either $\widetilde{\mathcal{L}^A}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big)$ in (11.68) or $\widetilde{\mathcal{L}^B}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big)$ (11.69).

Taking the gradients of $\tilde{\mathcal{L}}^{\mathcal{M}}(\theta,\phi;\boldsymbol{\mathcal{M}})$ in (11.70), we can use gradient-based optimization techniques to maximize the ELBO over parameters θ,ϕ.

The minibatch version of the variational auto-encoding algorithm can be described below.

Initialize parameters θ,ϕ

Repeat

  $\mathcal{M}\leftarrow$ Random minibatch of M datapoints drawn from the full dataset

  Random samples from noise distribution $\boldsymbol{\epsilon}^{(\boldsymbol{n},\boldsymbol{l})}\sim p(\boldsymbol{\epsilon})$

  Estimate the ELBO $\tilde{\mathcal{L}}^{\mathcal{M}}(\theta,\phi;\boldsymbol{\mathcal{M}}) = \frac{N}{M}\sum_{n=1}^{M}\tilde{\mathcal{L}}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big)$ (11.70)

  $g\leftarrow\nabla_{\theta,\phi}\tilde{\mathcal{L}}^{\mathcal{M}}(\theta,\phi;\boldsymbol{\mathcal{M}})$ (gradients of the ELBO)

  Update parameters θ,ϕ using the gradients g.

Until convergence of parameters θ,ϕ

Return θ,ϕ

11.3.3 Variational Auto-Encoder

Posterior Encoder

Since the true posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ is intractable in general, we use a neural network called *encoder* to learn $q_\phi(\mathbf{z}|\mathbf{x})$, which is the approximation to $p_\theta(\mathbf{z}|\mathbf{x})$, but tractable. A common choice of $q_\phi(\mathbf{z}|\mathbf{x})$ is a multivariate Gaussian with a diagonal covariance matrix,

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\mathbf{z}, diag\left((\boldsymbol{\sigma}_\mathbf{z})^2\right)\right) = \prod_{k=1}^{K} \mathcal{N}(z_k; \mu_k, {\sigma_k}^2) \tag{11.71}$$

where $\mathbf{z} \in \mathbb{R}^K, \mathbf{x} \in \mathbb{R}^D$. $\boldsymbol{\mu}_\mathbf{z} \in \mathbb{R}^K$ and $\boldsymbol{\sigma}_\mathbf{z} \in \mathbb{R}^K$ are the mean and standard deviation vectors, respectively, and they are output of the encoder, i.e.,

$$\left(\boldsymbol{\mu}_\mathbf{z}, \ln {\boldsymbol{\sigma}_\mathbf{z}}^2\right) = EncoderNeuralNet(\mathbf{x}; \phi) \tag{11.72}$$

Note the neural network delivers $\ln {\boldsymbol{\sigma}_\mathbf{z}}^2$, instead of $\boldsymbol{\sigma}_\mathbf{z}$, because we allow the neural network to output both negative and positive values.

An example of EncoderNeuralNet is given by (Kingma 2014) as below,

$$\mathbf{h} = tanh(W_1\mathbf{x} + b_1) \tag{11.73a}$$

$$\boldsymbol{\mu}_\mathbf{z} = W_2\mathbf{h} + b_2 \tag{11.73b}$$

$$\ln {\boldsymbol{\sigma}_\mathbf{z}}^2 = W_3\mathbf{h} + b_3 \tag{11.73c}$$

where the weights and biases $\{w_1, w_2, w_3, b_1, b_2, b_3\}$ are the variational parameters ϕ.

Sampling Latent Variables

Based on the prediction ($\boldsymbol{\mu}_\mathbf{z}, \boldsymbol{\sigma}_\mathbf{z}$) from the encoder on the posterior distribution, we sample the laten variables $\mathbf{z}^{(l)} \sim q_\phi(\mathbf{z}|\mathbf{x})$ using the reparameterization trick,

$$\mathbf{z}^{(l)} = g_\phi\left(\mathbf{x}, \boldsymbol{\epsilon}^{(l)}\right) = \boldsymbol{\mu}_\mathbf{z} + \boldsymbol{\sigma}_\mathbf{z} \odot \boldsymbol{\epsilon}^{(l)} \text{ where } \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\boldsymbol{\epsilon};\ \mathbf{0}, \mathbf{I}) \tag{11.74}$$

Decoder

The lower bound is our objective to be maximized. One way to approximate the ELBO is (11.69), copied below,

$$\mathcal{L}\left(\theta, \phi; \mathbf{x}^{(n)}\right) \simeq -KL\left(q_\phi\left(\mathbf{z}|\mathbf{x}^{(n)}\right)||p_\theta(\mathbf{z})\right) + \frac{1}{L}\sum_{l=1}^{L} \ln p_\theta\left(\mathbf{x}^{(n)}|\mathbf{z}^{(n,l)}\right) \tag{11.75}$$

The first term in (11.75) is a KL divergence that can be computed analytically. We assume that both the prior $p_\theta(\mathbf{z})$ and the posterior $q_\phi(\mathbf{z}|\mathbf{x})$ are Gaussian such that $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ and $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\mathbf{z}, diag(\boldsymbol{\sigma}_\mathbf{z})\right)$, where $\boldsymbol{\mu}_\mathbf{z}$, $\boldsymbol{\sigma}_\mathbf{z}$ are the outputs of the encoder neural network. Let K be the dimension of $\mathbf{z}$, then we have (see exercise for proof),

$$\int q_\phi(\mathbf{z}|\mathbf{x}) \ln p_\theta(\mathbf{z})\, dz = \int \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\mathbf{z}, diag({\boldsymbol{\sigma}_\mathbf{z}}^2)\right) \ln \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})\, d\mathbf{z}$$

$$= -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\sum_{k=1}^{K}\left(\sigma_{zk}^2 + \mu_{zk}^2\right) \tag{11.76}$$

$$\int q_\phi(\mathbf{z}|\mathbf{x}) \ln q_\phi(\mathbf{z}|\mathbf{x})\, dz = \int \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\mathbf{z}, diag(\boldsymbol{\sigma}_\mathbf{z}{}^2)\right) \ln \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\mathbf{z}, diag(\boldsymbol{\sigma}_\mathbf{z}{}^2)\right) d\mathbf{z}$$

$$= -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\sum_{k=1}^{K}\left(1 + \ln \sigma_{zk}^2\right) \tag{11.77}$$

Substituting (11.76) and (11.77) into the KL divergence in (11.75), we have

$$-KL\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right) = \int q_\phi(\mathbf{z}|\mathbf{x}) \ln p_\theta(\mathbf{z})\, dz - \int q_\phi(\mathbf{z}|\mathbf{x}) \ln q_\phi(\mathbf{z}|\mathbf{x})\, dz$$

$$= \frac{1}{2}\sum_{k=1}^{K}\left(1 - \sigma_{zk}^2 - \mu_{zk}^2 + \ln \sigma_{zk}^2\right) \tag{11.78}$$

The second term in (11.75) is the Monte Carlo estimate of the reconstruction likelihood $\ln p_\theta\left(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}\right)$. We can use another neural network, denoted by $DecoderNeuralNet(\mathbf{z};\theta)$, to approximate $p_\theta(\mathbf{x}|\mathbf{z})$. Similar to the EncoderNeuralNet, we can have the following structure for the DecoderNeuralNet, assuming $p_\theta(\mathbf{x}|\mathbf{z})$ is multivariate Gaussian, for example,

$$\mathbf{h} = tanh(W_4\mathbf{z} + b_4) \tag{11.79a}$$

$$\boldsymbol{\mu}_\boldsymbol{x} = W_5\mathbf{h} + b_5 \tag{11.79b}$$

$$\ln \boldsymbol{\sigma}_\boldsymbol{x}{}^\mathbf{2} = W_6\mathbf{h} + b_6 \tag{11.79c}$$

$$\ln p_\theta(\mathbf{x}|\mathbf{z}) = \ln \mathcal{N}\left(\mathbf{x}; \boldsymbol{\mu}_\boldsymbol{x}, diag\left(\boldsymbol{\sigma}_\boldsymbol{x}{}^\mathbf{2}\right)\right) \tag{11.79d}$$

where the weights and biases $\{w_4, w_5, w_6, b_4, b_5, b_6\}$ are the decoder parameters θ.

If $\mathbf{x}$ is a binary random vector $\mathbf{x} \in \{0,1\}^\boldsymbol{D}$, i.e., $p_\theta(\mathbf{x}|\mathbf{z})$ is a multivariate Bernoulli distribution, a fully connected neural network with a single hidden layer is suggested for $p_\theta(\mathbf{x}|\mathbf{z})$ as

$$\mathbf{y} = sigmoid[W_8 tanh(W_7\mathbf{z} + b_7) + b_8] \tag{11.80a}$$

$$\ln p_\theta(\mathbf{x}|\mathbf{z}) = \sum_{i=1}^{D}[x_i \ln y_i + (1 - x_i)\ln(1 - y_i)] \tag{11.80b}$$

Overall Architecture of VAE

We want to use the observations $\mathbf{x}$ to understand the latent variables $\mathbf{z}$. The goal of VAE is to learn two distributions $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{x}|\mathbf{z})$, so that we can sample the latent variable samples $\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})$ to generate new samples $x'\sim p_\theta(\mathbf{x}|\mathbf{z})$. The architecture of VAE is illustrated by Fig. 11.7. The encoder $q_\phi(\mathbf{z}|\mathbf{x})$, also called inference network or recognition model, maps an observation to the latent space by modelling the approximate posterior distribution. The decoder aims to reconstruct the observation through sampling its estimated distribution $p_\theta(\mathbf{x}|\mathbf{z})$. Note that if $\boldsymbol{\mu}_\boldsymbol{x}$ and $\boldsymbol{\sigma}_\boldsymbol{x}{}^\mathbf{2}$ are estimated by the decoder if $\mathbf{x}$ is a continues multivariate random variable. If $\mathbf{x}$ is a binary-valued vector random variable, then (11.80) with Bernoulli distribution is applied, and thus the output of the decoder is $\mathbf{y}$ (defined by (11.80a)), instead of $\boldsymbol{\mu}_\boldsymbol{x}$ and $\boldsymbol{\sigma}_\boldsymbol{x}{}^\mathbf{2}$**.**

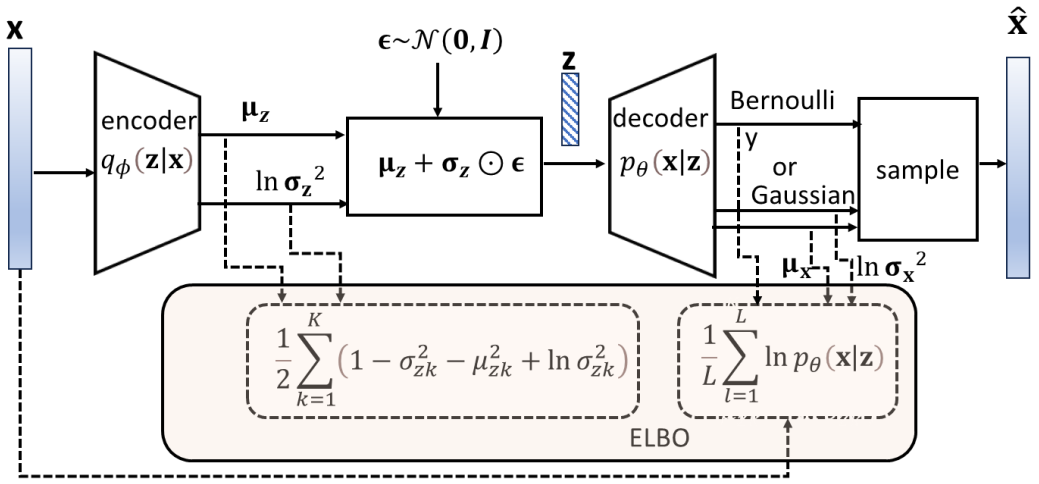


Fig.11.8 Architecture of variational auto-encoder (VAE)

The learning process is to maximize the ELBO or equivalently minimize the loss which is defined as the negative of the ELBO.

$$(\phi^*,\theta^*)=arg\max_{\phi,\theta}ELBO(\theta,\phi;\mathcal{D})=arg\min_{\phi,\theta}\big(-ELBO(\theta,\phi;\mathcal{D})\big) \tag{11.81}$$

In our VAE, the ELBO for a single sample $\mathbf{x}$ is approximated by $\tilde{\mathcal{L}}(\theta,\phi;\mathbf{x})$ given below.

$$ELBO(\theta,\phi;\mathbf{x})=\ln p_\theta(\mathbf{x})-KL\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})\right) \tag{11.82a}$$

$$=\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\ln p_\theta(\mathbf{x}|\mathbf{z})\,]-KL\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right) \tag{11.82b}$$

$$\simeq\frac{1}{L}\sum_{l=1}^{L}\ln p_\theta\big(\mathbf{x}|\boldsymbol{\mu}_\mathbf{z}+\boldsymbol{\sigma}_\mathbf{z}\odot\boldsymbol{\epsilon}^{(l)}\big)+\frac{1}{2}\sum_{k=1}^{K}\big(1-\sigma_{zk}^2-\mu_{zk}^2+\ln\sigma_{zk}^2\big)\equiv\tilde{\mathcal{L}}(\theta,\phi;\mathbf{x}) \tag{11.82c}$$

where $\boldsymbol{\mu}_\mathbf{z},\boldsymbol{\sigma}_\mathbf{z}$ are the outputs of the encoder $q_\phi(\mathbf{z}|\mathbf{x})$, $p_\theta(\mathbf{x}|\mathbf{z})$ is the decoder defined by (11.79) or (11.80), and L is the Monte Carlo length (a hyperparameter).The first term in (11.82c) is the reconstruction term, which calculates the ability of the VAE to recover the input data correctly. It can be proved that maximizing $\frac{1}{L}\sum_{l=1}^{L}\ln p_\theta\big(\mathbf{x}|\boldsymbol{\mu}_\mathbf{z}+\boldsymbol{\sigma}_\mathbf{z}\odot\boldsymbol{\epsilon}^{(l)}\big)$ is equivalent to minimizing the regular auto-encoder reconstruction MSE loss $\|\mathbf{x}-\hat{\mathbf{x}}\|^2$. The second term, the KL divergence, defines the difference between the estimated posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and the prior distribution $p_\theta(\mathbf{z})$.

The full dataset ELBO can be estimated, based on a minibatch $\mathcal{M}=\{\mathbf{x}^{(1)},\mathbf{x}^{(2)},\dots,\mathbf{x}^{(M)}\}$

$$\tilde{\mathcal{L}}^{\mathcal{M}}(\theta,\phi;\boldsymbol{\mathcal{M}})=\frac{N}{M}\sum_{n=1}^{M}\tilde{\mathcal{L}}\big(\theta,\phi;\mathbf{x}^{(\boldsymbol{n})}\big) \tag{11.83}$$

where N is the total number of samples, and M is the number of samples in each minibatch.

11.4. VAE on MNIST Dataset in PyTorch

In this section, we will show an example of VAE on the MNIST Dataset. The VAE learns from the MNIST Dataset without using the labels, and the trained decoder in the VAE can generate a handwritten digit given a random value of the latent variable.

11.4.1 Architecture of VAE

The architecture of our variational auto-encoder is shown in Fig.11.9. Since the size of images in MNIST is $28 \times 28 = 784$, the input $\mathbf{x}$ is a vector $\mathbb{R}^{784}$. There are three fully connected (FC) layers in both the encoder and the decoder. The last FC layer in the encoder, without any activation function, generates a mean vector and a log-variance vector. The last FC layer in the decoder uses the sigmoid activation to generate the reconstructed input. All other FC layers use ReLU activation. The "*sampling*" box performs the reparameterization trick. We select 2 as the dimension of the latent variable $\mathbf{z}$.

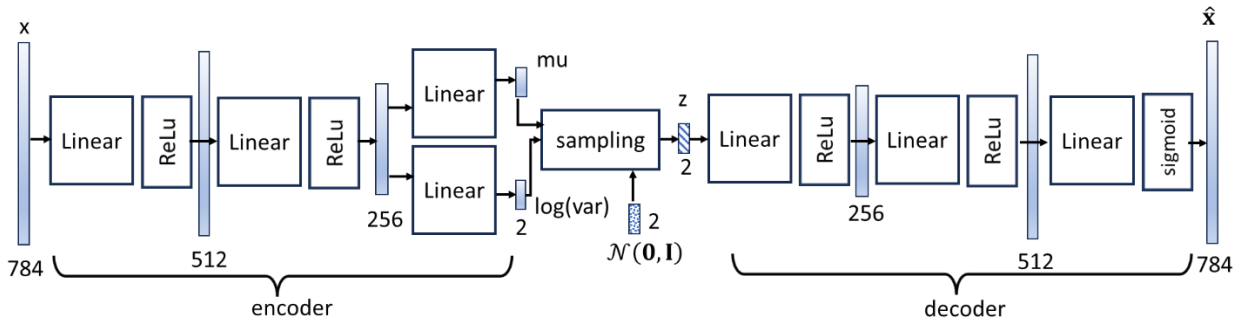


Fig.11.9 Architecture of the VAE

11.4.2 Implementation in PyTorch

First, we import required packages.

```
# Package imports
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable
from torchvision.utils import save_image
```

Prepare the data loaders.

```
# MNIST Dataset
train_dataset = datasets.MNIST(root='../torch_tutorial/data', train=True, transform=tr
ansforms.ToTensor(), download=True)
test_dataset = datasets.MNIST(root='../torch_tutorial/data/', train=False, transform=t
ransforms.ToTensor(), download=True)
```

```
bs=128
# Data Loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=bs, shuff
le=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=bs, shuffle
=False)
```

Construct model VAE.

```
class VAE(nn.Module):
    def __init__(self, x_dim, h_dim1, h_dim2, z_dim):
        super(VAE, self).__init__()

        # encoder
        self.encoder = nn.Sequential(
            nn.Linear(x_dim, h_dim1),
            nn.LeakyReLU(0.2),
            nn.Linear(h_dim1, h_dim2),
            nn.LeakyReLU(0.2)
            )

        # latent mean and variance
        self.mean_layer = nn.Linear(h_dim2, z_dim)
        self.logvar_layer = nn.Linear(h_dim2, z_dim)

        # decoder
        self.decoder = nn.Sequential(
            nn.Linear(z_dim, h_dim2),
            nn.LeakyReLU(0.2),
            nn.Linear(h_dim2, h_dim1),
            nn.LeakyReLU(0.2),
            nn.Linear(h_dim1, x_dim),
            nn.Sigmoid()
            )

    def encode(self, x):
        x = self.encoder(x)
        mu, log_var = self.mean_layer(x), self.logvar_layer(x)
        return mu, log_var

    # sample z from q(z/x) by reparameterization trick
    def reparameterization(self, mu, log_var):
        std = torch.exp(0.5*log_var)
        eps = torch.randn_like(std)
        z = eps.mul(std).add_(mu)
        return z

    def decode(self, x):
        return self.decoder(x)

    def forward(self, x):
        mu, log_var = self.encode(x.view(-1, 784))
        z = self.reparameterization(mu, log_var)
        y = self.decode(z)
        return y, mu, log_var
```

Set the architecture parameters and instantiate the VAE.

```
h_dim1 = 512
h_dim2 = 256
```

```
z_dim = 2
vae = VAE(x_dim=784, h_dim1=h_dim1, h_dim2=h_dim2, z_dim=z_dim)
if torch.cuda.is_available():
    vae.cuda()
```

Define the optimizer and loss function. Although MNIST is real-valued, it is constrained between 0 and 1. We can view the output of the decoder $p_\theta(\mathbf{x}|\mathbf{z})$ as the reconstructed datapoint $\hat{\mathbf{x}}$ obtained by independently sampling each dimension as $\hat{x}_i \sim Bernoulli(x_i)$, where x_i is the MNIST input pixel. Thus, we use the Binary Cross Entropy (BCE) loss for the decoder $p_\theta(\mathbf{x}|\mathbf{z})$. This is not quite what the VAE prescribes but works well in practice.

```
optimizer = optim.Adam(vae.parameters(), lr=0.001)
# loss_function returns reconstruction error + KL divergence losses
def loss_function(recon_x, x, mu, log_var):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
    KLD = 0.5 * torch.sum(1 + log_var - mu.pow(2) - log_var.exp())
    return BCE - KLD
```

Define the train loop.

```
def train(epoch):
    vae.train()
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):

        optimizer.zero_grad()

        recon_batch, mu, log_var = vae(data)
        loss = loss_function(recon_batch, data, mu, log_var)

        loss.backward()
        #train_loss += loss.item()
        optimizer.step()

        if batch_idx % 100 == 0:
            print('Epoch {} , Batch {}   \tLoss: {:.6f}'.format(
                epoch, (batch_idx), loss.item() / len(data)))
```

Compute the loss on the test dataset for a monitoring purpose.

```
def test():
    vae.eval()
    test_loss= 0
    with torch.no_grad():
        for data, _ in test_loader:
            #data = data.cuda()
            recon, mu, log_var = vae(data)

            # sum up batch loss
            test_loss += loss_function(recon, data, mu, log_var).item()

    test_loss /= len(test_loader.dataset)
    print('=> Test set average loss per batch: {:.4f}'.format(test_loss))
```

Run training loops and test for 9 epochs.

```
for epoch in range(1, 10):
```

```
    train(epoch)
    test()
```

Generate random samples. The result is shown in Fig.11.10.

```
with torch.no_grad():
    z = torch.randn(64, z_dim)
    sample = vae.decode(z)
    # generate 64 samples
    save_image(1-sample.view(64, 1, 28, 28), './samples/sample_weidong_z2' + '.png')
```

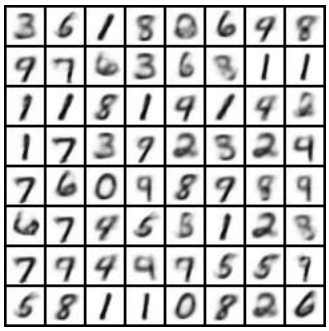
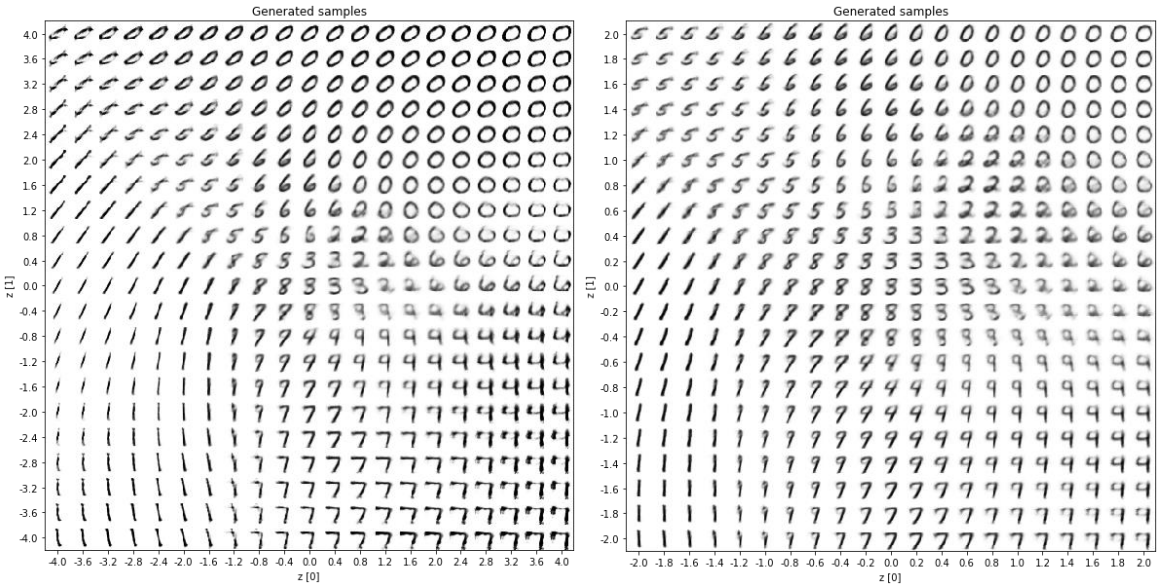


Fig.11.10 Generated samples (sample_weidong_z2.png)

Plot the generated samples in the Z space (2D). The result is shown in Fig.11.11.

```
def plot_reconstructed(model, scale=1.0, n=21, digit_size=28, figsize=10):
    # display a n*n handwritten digits in the 2D latent space
    figure = np.zeros((digit_size * n, digit_size * n))

    # construct a grid in the 2D Z space
    grid_x = np.linspace(-scale, scale, n)
    grid_y = np.linspace(-scale, scale, n)[::-1] #reverse the order

    for i, yi in enumerate(grid_y):
        for j, xi in enumerate(grid_x):
            z_sample = torch.tensor([[xi, yi]], dtype=torch.float)
            x_decoded = model.decode(z_sample)
            digit = x_decoded[0].detach().cpu().reshape(digit_size, digit_size)
            figure[i * digit_size : (i + 1) * digit_size, j * digit_size : (j + 1) * d
igit_size,] = 1-digit

    plt.figure(figsize=(figsize, figsize))
    plt.title('Generated samples')
    start_range = digit_size // 2
    end_range = n * digit_size + start_range
    pixel_range = np.arange(start_range, end_range, digit_size)
    sample_range_x = np.round(grid_x, 1)
    sample_range_y = np.round(grid_y, 1)
    plt.xticks(pixel_range, sample_range_x)
    plt.yticks(pixel_range, sample_range_y)
    plt.xlabel("z [0]")
    plt.ylabel("z [1]")
    plt.imshow(figure, cmap="Greys_r")
    plt.show()

plot_reconstructed(vae, scale=4.)
```



(a) z: (-4,-4) to (4,4)

(b) z: (-2,-2) to (2,2) (zoom-in)

Fig.11.11 Generated samples in Z-space

Plot the datapoints from the test dataset in the latent z-space, as shown in Fig.11.12. We can see that the datapoints for the same digit (indicated by the color) distribute roughly in a cluster.

```
def plot_latent(vae, data, num_batches=100):
    for i, (x, y) in enumerate(data):
        mu, log_var = vae.encode(x.view(-1, 784))
        z = vae.reparameterization(mu, log_var)
        z = z.detach().numpy()
        plt.scatter(z[:, 0], z[:, 1], c=y, cmap='tab10')
        if i > num_batches:
            break
    plt.colorbar()
    plt.xlabel("z [0]")
    plt.ylabel("z [1]")
    plt.show()
plot_latent(vae, test_loader)
```

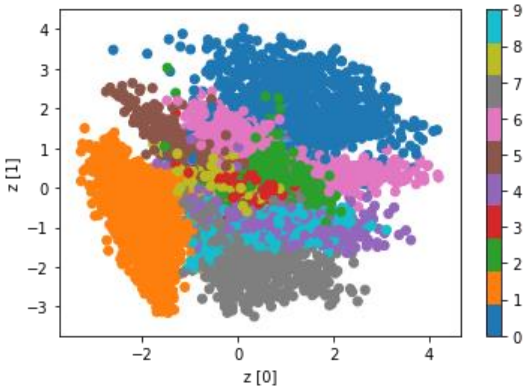


Fig.11.12 Test dataset mapped into z-space

11.4.3 Conditional VAE

To generate a sample for a particular class, we can extend the VAE to a conditional VAE. In the conditional VAE setting, we are given a complete dataset $\{(\mathbf{x}^{(n)}, \mathbf{c}^{(n)}), n = 1,2, \dots, N\}$, where $\mathbf{c}^{(n)}$ is the label. By adding the condition $\mathbf{c}$ to (11.82), the resulting ELBO can be represented by

$$ELBO(\theta, \phi; \mathbf{x}, \mathbf{c}) = \ln p_\theta(\mathbf{x}|\mathbf{c}) - KL\left(q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{c})||p_\theta(\mathbf{z}|\mathbf{x}, \mathbf{c})\right) \quad (11.84a)$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{c})}[\ln p_\theta(\mathbf{x}|\mathbf{z}, \mathbf{c})\ ] - KL\left(q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{c})||p_\theta(\mathbf{z}|\mathbf{c})\right) \quad (11.84b)$$

where $p_\theta(\mathbf{z}|\mathbf{c})$ is assumed to be a standard Gaussian distribution. Thus, we add the label as a part of both the inputs of the encoder and the decode. The architecture of conditional VAE is illustrated by Fig. 11.12.

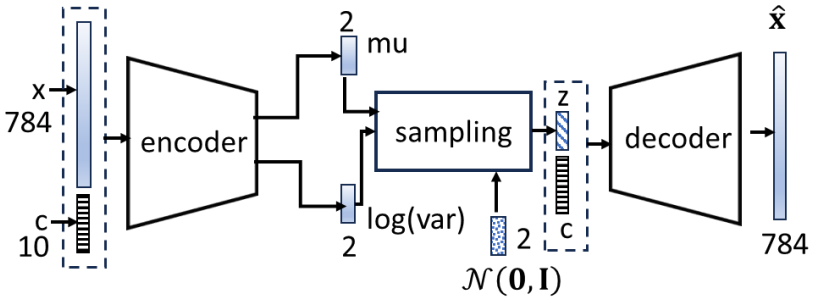


Fig.11.13 Conditional VAE

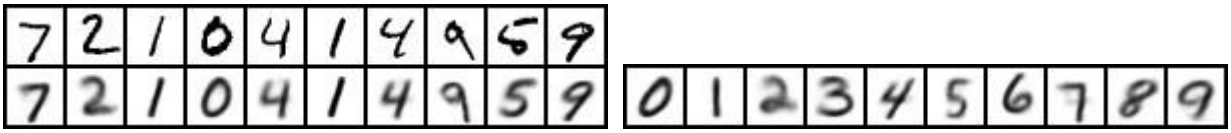
In our implementation of the conditional VAE on MNIST dataset, we use a 10-element one-hot code for each label. The label is concatenated with $\mathbf{x}$ (or $\mathbf{z}$) to generate the input for the encoder (or the decoder).

```
def one_hot(labels, class_size):
    targets = torch.zeros(labels.size(0), class_size)
    for i, label in enumerate(labels):
        targets[i, label] = 1
    return targets
```

```
class CVAE(nn.Module):
    def __init__(self, x_dim, h_dim1, h_dim2, z_dim, class_size):
        super(CVAE, self).__init__()

        # encoder
        self.encoder = nn.Sequential(
            nn.Linear(x_dim+class_size, h_dim1),
            nn.LeakyReLU(0.2),
            nn.Linear(h_dim1, h_dim2),
            nn.LeakyReLU(0.2)
            )

        # latent mean and variance
        self.mean_layer = nn.Linear(h_dim2, z_dim)
        self.logvar_layer = nn.Linear(h_dim2, z_dim)

        # decoder
        self.decoder = nn.Sequential(
            nn.Linear(z_dim+class_size, h_dim2),
            nn.LeakyReLU(0.2),
            nn.Linear(h_dim2, h_dim1),
            nn.LeakyReLU(0.2),
```

```
            nn.Linear(h_dim1, x_dim),
            nn.Sigmoid()
            )

    def encode(self, x, c):    #q(z|x,c)
        x = torch.cat([x, c], 1) # (bs, feature_size+class_size)
        x = self.encoder(x)
        mu, log_var = self.mean_layer(x), self.logvar_layer(x)
        return mu, log_var

    # sample z from q(z/x) by reparameterization trick
    def reparameterization(self, mu, log_var):
        std = torch.exp(0.5*log_var)
        eps = torch.randn_like(std)
        z = eps.mul(std).add_(mu)
        return z

    def decode(self, z, c):    # p(x|z, c)
        x = torch.cat([z, c], 1) # (bs, latent_size+class_size)
        return self.decoder(x)

    def forward(self, x, c):
        mu, log_var = self.encode(x.view(-1, 784), c)
        z = self.reparameterization(mu, log_var)
        y = self.decode(z,c)
        return y, mu, log_var
```

The conditional VAE can be trained in the similar way in Section 11.4.2. (see Exercise) After 5 epochs, the results are shown in Fig.11.14.



(a) original (1st row) and generated samples (2nd row) (b) generated samples for given labels

Fig.11.14 The samples generated by the conditional VAE

Summary and Further Reading

This chapter serves as an introduction to generative models. The major theme of this chapter is the expectation-maximization (EM) algorithm for Gaussian mixture models and the development of variational auto-encoders.

First, we briefly introduced the basics of generative models, including graphical models, latent variables, Bayes' inference, evidence lower bound, etc., which form a foundation for advanced generative models such as variational auto-encoders (VAEs) and generative adversarial networks (GANs). The Gaussian mixture model is a good introductory example of generative models for us to present these basics. Given a Gaussian mixture model, the EM algorithm was derived to learn the probability distribution of a dataset, i.e., to find the optimal values for the parameters of the Gaussian mixture model, by an iterative scheme.

Then, the general EM algorithm was presented. The goal of the general EM algorithm is to maximize the evidence lower bound (ELBO) iteratively by selecting a currently optimal (or a better) inference distribution $q(\mathbf{z})$ and searching currently optimal (or better) parameter values θ. To deal with the intractability, we use neural networks to perform approximate inference. This leads to the framework of variational auto-encoder.

Finally, we implement a variational auto-encoder (VAE) on MNIST dataset in PyTorch. To generate a sample for a given class, we extended the basic VAE to a conditional VAE.

A comprehensive treatment of Gaussian mixture models and EM algorithm are given by Chapter 9 in the book (Bishop 2009). The relation between K-means and EM is discussed. The VAE was originally proposed by Kingma and Welling (2014), and extensively treated later by Kingma and Welling (2019).

Files:

C:\Users\weido\ch11_generative\

Ch11_cvae_weidong.ipynb, ch11_vae_weidong.ipynb, gmm.ipynb,

C:\Users\weido\ch11_generative\samples\

Some results

References

Bishop, C. M. (2009), Pattern recognition and machine learning. Springer, 2009.

Kingma, D. P. and Welling, M. (2014). "Auto-Encoding Variational Bayes". International Conference on Learning Representations. arXiv:1312.6114 **[stat.ML]**

Kingma, D. P. and Welling M. (2019), "An Introduction to Variational Autoencoders", Foundations and Trends in Machine Learning: Vol. 12 (2019): No. 4, pp 307-392. arXiv:1906.02691 **[cs.LG]**

Exercises

11.1 Discuss why the gradient descent (or ascent) method cannot be directly applied to maximize the log-likelihood of a Gaussian mixture model.

$$\ln p(\mathbf{X};\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) = \ln \prod_{n=1}^{N} p\left(\mathbf{x}^{(n)};\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}\right) = \sum_{n=1}^{N} \ln \left[\sum_{k=1}^{K} \pi_k \mathbb{N}\left(\mathbf{x}^{(n)};\boldsymbol{\mu_k},\boldsymbol{\Sigma_k}\right)\right]$$

11.2 Assume that we have a Gaussian mixture model in (11.7), re-written here as

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \, \mathbb{N}(\mathbf{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) \tag{11.7}$$

Given a complete data set $S = \{(\mathbf{x}_n, \mathbf{z}_n), n = 1,2,\dots,N\}$. Find the solution (i.e., π_k $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$) for maximizing the log likelihood function in (11.9)

$$\ln p(\mathbf{X};\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) = \ln \prod_{n=1}^{N} p(\mathbf{x}_n;\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) \tag{11.9}$$

(ref: Bishop pp.442 (9.37), and pp.93 (2.121), pp.94 (2.122))

11.3 Prove Jensen's inequality. The Jensen's inequality is stated as: if function f is a *convex* function, and **x** is a random variable, then

$$\mathbb{E}[f(\mathbf{x})] \geq f(\mathbb{E}[\mathbf{x}])$$

11.4 The KL divergence is defined as

$$KL(p||q) = \sum_{z} p(\mathbf{z}) \ln \frac{p(\mathbf{z})}{q(\mathbf{z})}$$

Prove that 1) $KL(p||q) \geq 0, \forall\, p, q$; and 2) $KL(p||q) = 0$, if and only if $p(\mathbf{z}) = q(\mathbf{z}), \forall \mathbf{z}$.

11.5 Generate a dataset by sampling a Gaussian mixture model. Suppose we know all the parameters for the Gaussian mixture model, assuming K=3, $x \in \mathbb{R}$

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \, \mathbb{N}(\mathbf{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$$

1) Specify any reasonable values for $\pi_k \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$, k=1,2,3.
2) Sample 10 datapoints from each Gaussian component, and obtain a complete dataset consisting of 30 datapoints by combining all these samples and an incomplete dataset for the same 30 datapoints.
3) Plot the probability density function $p(\mathbf{x})$ and the datapoints of the dataset generated in 2), similar to Fig.11.4.

11.6 Given the complete dataset generated in Exercise 11.5 and the Gaussian mixture model (K=3) (but all parameters are unknown), estimate the parameters $\pi_k \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$, k=1,2,3. (hint: use maximal likelihood estimate).

11.7 Given the incomplete dataset generated in Exercise 11.5 and the Gaussian mixture model (K=3) (all parameters are unknown).

1) write a Python program to implement the EM algorithm to estimate the parameters $\pi_k \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$, k=1,2,3.

2) In the same coordinate system, plot the true log likelihood $\ln\big(p(\mathbf{X}|\theta)\big)$ using the parameters you used in Exercises 11.5 and the ELBO curve $\mathcal{L}_{q^{old}}(\theta)$ (see equation (11.41)) for a particular set of values of θ^{old} (you arbitrarily select a set of appropriate numerical values for θ^{old}). (refer to Fig.11.5)

11.8 If u$\sim U(0,1)$, what is the pdf of x $= -\frac{1}{\lambda}\ln(1-\text{u})$?

11.9 The reparameterization trick in our text is applied to the factorized Gaussian posterior $q_\phi(\mathbf{z}|\mathbf{x})$, i.e., we assume that the covariance matrix is diagonal, $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\mathbf{z}, diag(\boldsymbol{\sigma}_\mathbf{z}{}^2)\right)$

$$\boldsymbol{z} = \boldsymbol{\mu}_\mathbf{z} + \boldsymbol{\sigma}_\mathbf{z} \odot \boldsymbol{\epsilon} \qquad \textbf{where} \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

In this exercise, we extend the reparameterization trick for a full covariance Gaussian posterior. Specifically, we assume the Gaussian posterior $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$. A reparameterization trick of this distribution is given by

$$\boldsymbol{z} = \boldsymbol{\mu} + \mathbf{L} \odot \boldsymbol{\epsilon} \qquad \textbf{where} \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

where $\mathbf{L}$ is a lower (or upper) triangular matrix, with non-zero entries on the diagonal. The off-diagonal elements define the correlations (covariances) of the elements in $\mathbf{z}$.

Find the matrix $\mathbf{L}$ in terms of $\boldsymbol{\Sigma}$. (refer, Kingma, D. P. and Welling M. (2019))

11.10 Prove (11.76) and (11.77).

$$\int q_\phi(\mathbf{z}|\mathbf{x}) \ln p_\theta(\mathbf{z})\, dz = \int \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\mathbf{z}, diag(\boldsymbol{\sigma}_\mathbf{z}{}^2)\right) \ln \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})\, d\boldsymbol{z}$$

$$= -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\sum_{k=1}^{K}\left(\sigma_{zk}^2 + \mu_{zk}^2\right) \tag{11.76}$$

$$\int q_\phi(\mathbf{z}|\mathbf{x}) \ln q_\phi(\mathbf{z}|\mathbf{x})\, dz = \int \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\mathbf{z}, diag(\boldsymbol{\sigma}_\mathbf{z}{}^2)\right) \ln \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_\mathbf{z}, diag(\boldsymbol{\sigma}_\mathbf{z}{}^2)\right) d\boldsymbol{z}$$

$$= -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\sum_{k=1}^{K}\left(1 + \ln \sigma_{zk}^2\right) \tag{11.77}$$

11.11 Compute the KL divergence between two Gaussian distributions $KL\left(\mathcal{N}(\mathbf{x}; \boldsymbol{\mu_0}, \boldsymbol{\Sigma}_0)||\mathcal{N}(\mathbf{x}; \boldsymbol{\mu_1}, \boldsymbol{\Sigma}_1)\right)$ in a closed form.

(ans: Tutorial on Variational Autoencoders, CARL DOERSCH https://arxiv.org/pdf/1606.05908.pdf)

11.12 Train and test the conditional VAE given in Section 11.4.3.