Review

Tutorial on Deep Generative Model

Loc Nguyen [*]

Posted Date: 21 May 2024

doi: 10.20944/preprints202405.1348.v1

Keywords: generative artificial intelligence; deep neural network; deep generative model; data generation

Review

Tutorial on Deep Generative Model

Loc Nguyen

Loc Nguyen's Academic Network, Vietnam; Email: ng_phloc@yahoo.com; Homepage: www.locnguyen.net

Abstract: Artificial intelligence (AI) is a current trend in computer science, which extends itself its amazing capacities to other technologies such as mechatronics and robotics. Going beyond technological applications, the philosophy behind AI is that there is a vague and potential convergence of artificial manufacture and natural world although the limiting approach may be still very far away, but why? The implicit problem is that Darwin theory of evolution focuses on natural world where breeding conservation is the cornerstone of the existence of creature world but there is no similar concept of breeding conservation in artificial world whose things are created by human. However, after developing for a long time until now, AI issues an interesting concept of generation in which artifacts created by computer science can derive their new generations inheriting their aspects / characteristics. Such generated artifacts make us look back on offsprings by the process of breeding conservation in natural world. Therefore, it is possible to think that AI generation, which is a recent subject of AI, is a significant development in computer science as well as high-tech domain. AI generation does not help us to reach near biological evolution even in the case that AI can combine with biological technology but, AI generation can help us to extend our viewpoint about Darwin theory of evolution as well as there may exist some uncertain relationship between man-made world and natural world. Anyhow AI generation is a current important subject in AI and there are two main generative models in computer science: 1) generative model that applies large language model into generating natural language texts understandable by human and 2) generative model that applies deep neural network into generating digital content such as sound, image, and video. This technical report focuses on deep generative model (DGM) for digital content generation, which is a short summary of approaches to implement DGMs. Researchers can read this work as an introduction to DGM with easily understandable explanations.

Keywords: generative artificial intelligence; deep neural network; deep generative model; data generation

1. Introduction to Deep Generative Model (DGM)

By informal statement, generative artificial intelligence (GenAI) applications aim to reproduce original artifacts such as images, sounds, music, texts, and speeches into a new artifact with some changes. The problem is that reproduction or generation, which is not duplication, indeed derives a new piece of content which is large or small from whole content of the original artifacts. For example, given a smiling face of a specific person, GenAI application will generate a crying face of the same person. As a subdomain of GenAI, deep generative model (DGM) applies deep neural network (DNN) into generating artifacts but many deep generative models (DGMs) are also relevant to applied statistics. Note, DNN is an artificial neural network having many hidden layers, besides input layer and output layer. Training deep neural network or learning deep neural network is known as deep learning. Given random variable vector $\boldsymbol{x} = (x_1, x_2,\dots, x_m)^T$ presenting any digital artifact or any digital data such as image and sound, let $P(\boldsymbol{x})$ be probability density function (PDF) of $\boldsymbol{x}$ but it is difficult to estimate such probabilistic distribution $P(\boldsymbol{x})$ because data $\boldsymbol{x}$ is complicated with suppose that $\boldsymbol{x}$ belongs to the real field R^m where m is high dimension and so, $P(\boldsymbol{x})$ is called intractable PDF. Suppose there is another random variable vector $\mathbf{z} = (z_1, z_2,\dots, z_n)^T$ belonging to the real field R^n where n is low dimension ($n < m$) so that PDF of $\mathbf{z}$ denoted $P(\mathbf{z})$ is tractable and it is possible to understand $P(\mathbf{z})$. Moreover, it is most important that suppose there is a function $g(\mathbf{z} \mid \Phi) = \boldsymbol{x}$ that maps tractable data $\mathbf{z}$ to intractable data $\boldsymbol{x}$ where Φ is parameter of such mapping function. For some illustrations or examples in this report, random variable vector $\boldsymbol{x}$ is flattened from two-dimension image. As a convention, the function $g(\mathbf{z} \mid \Phi) = \boldsymbol{x}$ is called generator of $\boldsymbol{x}$ and its parameter Φ is called generator parameter (Ruthotto & Haber, 2021, p. 2).

$$g: R^n \rightarrow R^m \text{ such that } \boldsymbol{x} = g(\boldsymbol{z}|\Phi) \text{ where } \boldsymbol{z} \in Z \subseteq R^n, \boldsymbol{x} \in X \subseteq R^m$$

Where Z and X are domains of tractable data x and intractable data $\mathbf{z}$ with note that Z is called latent space and X is called sample space by convention. When Z is called latent space, tractable PDF $P(\mathbf{z})$ is called latent distribution. Because $g(\mathbf{z} \mid \Phi)$ is essentially vector-by-vector function whose input and output are vectors, it should have denoted as $\boldsymbol{g}(\mathbf{z} \mid \Phi)$. However, it is still denoted $g(\mathbf{z} \mid \Phi)$ in context DNN because there are two reasons: 1) $\boldsymbol{g}(\mathbf{z} \mid \Phi)$ is not bijection and 2) the output $\boldsymbol{x}$ of $\boldsymbol{g}(\mathbf{z} \mid \Phi)$ can be considered as scalar variable x corresponding to an output neuron of output layer in neural network. Therefore, $g(\mathbf{z} \mid \Phi)$ also implies a vector-by-scalar function whose first-order derivative can be considered as gradient vector although the first-order derivative of vector-by-vector function $\boldsymbol{g}(\mathbf{z} \mid \Phi)$ is Jacobian matrix. Note, in mathematical, the first-order derivative of scalar-by-vector function is called gradient vector and the first-order derivative of vector-by-vector function is called Jacobian matrix.

The ultimate purpose of any DGM is to determine parameter Φ because generator $g(\mathbf{z} \mid \Phi)$ is defined based on Φ. In DGM, generator $g(\mathbf{z} \mid \Phi)$ is constructed by a deep neural network (DNN) and its parameter Φ is essentially weights of such DNN. When $g(\mathbf{z} \mid \Phi)$ is constructed by DNN, $g(\mathbf{z} \mid \Phi)$ is not totally equal to $\boldsymbol{x}$ as $g(\mathbf{z} \mid \Phi) = \boldsymbol{x}$ but it is expected that $g(\mathbf{z} \mid \Phi)$ is approximated to $\boldsymbol{x}$ in practice:

$$g(\boldsymbol{z}|\Phi) = \boldsymbol{x}' \cong \boldsymbol{x}$$

Note that there are many DGMs and some of them do not require explicit definition of the PDF $P(\mathbf{z})$ of tractable data $\mathbf{z}$ but how to estimate generator parameter Φ for determining generator $g(\mathbf{z} \mid \Phi) = \boldsymbol{x}$ is always concerned. When $g(\mathbf{z} \mid \Phi)$ was determined, we can easily randomize some random tractable data $\mathbf{z}'$ according to the known tractable PDF $P(\mathbf{z})$ and then it is totally possible to generate new artifact $\boldsymbol{x}'$ by $\boldsymbol{x}' = g(\mathbf{z}' \mid \Phi)$ so that $\boldsymbol{x}'$ is generated data / derived data of original intractable data $\boldsymbol{x}$ with expectation that probability distribution of $\boldsymbol{x}'$ is approximate to the true distribution $P(\boldsymbol{x})$ of $\boldsymbol{x}$. The process to randomize $\mathbf{z}'$ is called sampling tractable data ($\mathbf{z}$). When $g(\mathbf{z} \mid \Phi)$ is modeled by a DNN, how to estimate parameter Φ is essentially to train such DNN and hence, the DNN is denoted as generator function $g(\mathbf{z} \mid \Phi)$ for a convention, which is called generator DNN $g(\mathbf{z} \mid \Phi)$. Here we identify generator function with DNN.

Intractable PDF $P(\boldsymbol{x})$ of $\boldsymbol{x}$ is specified (Ruthotto & Haber, 2021, p. 3) based on law of total probability as follows:

$$P(\boldsymbol{x}) = \int_{\boldsymbol{z}} P_g(\boldsymbol{x}|\boldsymbol{z}) P(\boldsymbol{z})\, d\boldsymbol{z}$$

Where $P_g(\boldsymbol{x} \mid \mathbf{z})$ is conditional PDF of $\boldsymbol{x}$ given $\mathbf{z}$, which implies that $P_g(\boldsymbol{x} \mid \mathbf{z})$ depends on generator $g(\mathbf{z} \mid \Phi)$ too because random variable $\boldsymbol{x}$ inside condition PDF $P_g(\boldsymbol{x} \mid \mathbf{z})$ is generated from $\mathbf{z}$. Note, notation $P(.)$ denotes probability distribution or probability density function (PDF) in this research. Therefore, it is possible to denote such conditional PDF as $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$.

$$P_g(\boldsymbol{x}|\boldsymbol{z}) = P(\boldsymbol{x}|\Phi, \boldsymbol{z})$$

Such that:

$$P(\boldsymbol{x}) = \int_{\boldsymbol{z}} P(\boldsymbol{x}|\Phi, \boldsymbol{z}) P(\boldsymbol{z})\, d\boldsymbol{z} \tag{1.1}$$

This implies intractable PDF $P(\boldsymbol{x})$ can be known via tractable PDF $P(\mathbf{z})$ and conditional PDF $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$; however it is really difficult to compute $P(\boldsymbol{x})$ due to complication of the integral but this difficulty is unimportant because the purpose of DGM is to estimate generator $g(\mathbf{z} \mid \Phi)$. As a convention, the conditional PDF $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is called likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$. Indeed, $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is likelihood function of intractable data $\boldsymbol{x}$ given tractable data **z**, which indicates how close generated data $\boldsymbol{x}' = g(\mathbf{z} \mid \Phi)$ to $\boldsymbol{x}$. In practice tractable PDF $P(\mathbf{z})$ is predefined and likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is determined based on generator DNN $g(\mathbf{z} \mid \Phi)$. For instance, $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is assumed to be normal distribution (Gaussian distribution) with mean $\boldsymbol{\mu}$ and variance σ^2 in popular as follows (Ruthotto & Haber, 2021, p. 3):

$$P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = (2\pi\sigma^2)^{-\frac{m}{2}}\exp\left(-\frac{\|g(\boldsymbol{z}|\Phi)-\boldsymbol{x}-\boldsymbol{\mu}\|^2}{2\sigma^2}\right)$$

Let $\boldsymbol{\mu} = \mathbf{0}$ and σ^2=1 for optimization, we have:

$$P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = (2\pi)^{-\frac{m}{2}}\exp\left(-\frac{\|g(\boldsymbol{z}|\Phi)-\boldsymbol{x}-\boldsymbol{\mu}\|^2}{2}\right)$$

Where notation ||.|| denotes norm of vector. For instance, Euclidean norm of intractable data $\boldsymbol{x}$ is:

$$\|\boldsymbol{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_m^2}$$

That tractable PDF $P(\mathbf{z})$ is predefined (constant with regard to Φ and $\boldsymbol{x}$) and likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is assumed to distribute normally indicates that intractable PDF $P(\boldsymbol{x})$ is implied by the simpler conditional PDF $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ with support of generator DNN $g(\mathbf{z} \mid \Phi)$; in other words, $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is probability distribution of $\boldsymbol{x}$ from viewpoint of DNN $g(\mathbf{z} \mid \Phi)$ indeed where **z** is totally determined in latent space Z and $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is really simpler with support of DNN $g(\mathbf{z} \mid \Phi)$. Because how to determine generator $g(\mathbf{z} \mid \Phi)$ is to estimate parameter Φ, it is easy to calculate Φ as maximizer of likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$, which is the optimization problem as follows:

$$\Phi^* = \underset{\Phi}{\mathrm{armax}}\, P(\boldsymbol{x}|\Phi,\boldsymbol{z}) \tag{1.2}$$

Taking natural logarithm of likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ aims to easily determine Φ by maximizing the log-likelihood function $\log(P(\boldsymbol{x} \mid \Phi, \mathbf{z}))$ as follows:

$$\Phi^* = \underset{\Phi}{\mathrm{armax}}\, P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \underset{\Phi}{\mathrm{armax}}(\log P(\boldsymbol{x}|\Phi,\boldsymbol{z})) = \underset{\Phi}{\mathrm{armax}}\left(-\frac{1}{2}\|g(\boldsymbol{z}|\Phi)-\boldsymbol{x}\|^2\right)$$

Note,

$$\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = -\frac{m}{2}\log(2\pi\sigma^2) - \frac{\|g(\boldsymbol{z}|\Phi)-\boldsymbol{x}-\boldsymbol{\mu}\|^2}{2\sigma^2}$$

Let $\boldsymbol{\mu} = \mathbf{0}$ and σ^2=1 for optimization, we have:

$$\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = -\frac{m}{2}\log(2\pi) - \frac{\|g(\boldsymbol{z}|\Phi)-\boldsymbol{x}\|^2}{2}$$

This implies the minimization problem as follows:

$$\Phi^* = \underset{\Phi}{\mathrm{armin}}\frac{1}{2}\|g(\boldsymbol{z}|\Phi)-\boldsymbol{x}\|^2$$

As a result, the estimation of generator parameter Φ based on maximum likelihood estimation (MLE) with assumption of normal distribution of generator $g(\mathbf{z} \mid \Phi)$ turns back minimization of error function $½||g(\mathbf{z} \mid \Phi) - \boldsymbol{x}||^2$ which is popular technique in learning DNN by backpropagation algorithm because $½||g(\mathbf{z} \mid \Phi) - \boldsymbol{x}||^2$ is, indeed, quadratic error function in neural network where $g(\mathbf{z} \mid \Phi)$ and $\boldsymbol{x}$ are output and real output of a neuron, respectively with note that the output $g(\mathbf{z} \mid \Phi)$ is calculated by propagation rule and the real output $\boldsymbol{x}$ is from training data. In other words, MLE is entry point to estimate generator parameter Φ which is weights of DNN $g(\mathbf{z} \mid \Phi)$ that is learned fully by backpropagation algorithm (Nguyen, 2023, pp. 8-20). Therefore, please pay attention to the association of MLE and backpropagation algorithm for determining totally generator $g(\mathbf{z} \mid \Phi)$, in which $g(\mathbf{z} \mid \Phi)$ and $\boldsymbol{x}$ are output at real output of neurons at the output layer of DNN so that backpropagation algorithm can be applied successively. Note, *error function* is also called *loss function*. Backpropagation algorithm is often associated with stochastic gradient descent (SGD) method to optimize loss function.

Let $\nabla P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ be gradient of *likelihood* $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ which is first-order derivative of $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ with regard to parameter Φ where $\boldsymbol{x}$ and $\mathbf{z}$ are samples as follows:

$$\nabla P(\boldsymbol{x}|\Phi, z) = \frac{dP(\boldsymbol{x}|\Phi, z)}{d\Phi}$$

Stochastic gradient descent (*SGD*) method (Nguyen, 2023, pp. 22-27) estimates Φ by iterative process to update successively Φ at every iteration as follows:

$$\Phi = \Phi + \gamma\nabla P(\boldsymbol{x}|\Phi, z)$$

Where $0 < \gamma \leq 1$ is learning rate. SGD, which is an iterative process, pushes candidate solution at each iteration along the direction which is opposite to gradient of target function for minimization or has the same direction to gradient of target function for maximization with note that the step length is represented by learning rate. In practice, likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is replaced by its natural logarithm as follows:

$$\Phi = \Phi + \gamma\nabla \log P(\boldsymbol{x}|\Phi, \boldsymbol{z}) \quad (1.3)$$

Where $\nabla P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is gradient of the log-likelihood function $\log P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ with regard to Φ. Note the estimation equation above mentions maximization problem according to MLE method and hence, if error function denoted $\varepsilon(\boldsymbol{x} \mid \Phi, \mathbf{z})$ which is the function related to $g(\mathbf{z} \mid \Phi)$ and $\boldsymbol{x}$ like $\varepsilon(\boldsymbol{x} \mid \Phi, \mathbf{z}) = ½||g(\mathbf{z} \mid \Phi) - \boldsymbol{x}||^2$ aforementioned, then SGD modifies a little bit the estimation equation as follows:

$$\Phi = \Phi - \gamma\nabla\varepsilon(\boldsymbol{x}|\Phi, \boldsymbol{z}) \quad (1.4)$$

Where $\nabla\varepsilon(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is gradient of error $\varepsilon(\boldsymbol{x} \mid \Phi, \mathbf{z})$.

$$\nabla\varepsilon(\boldsymbol{x}|\Phi, z) = \frac{d\varepsilon(\boldsymbol{x}|\Phi, z)}{d\Phi}$$

The main difference between maximizing likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ and minimizing error $\varepsilon(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is the changing from the sign "+" regarding maximization problem to the sign "–" regarding minimization problem, which is the essence of gradient descent method. In the example of assuming normal distribution, likelihood maximization is the same to error minimization but likelihood maximization gives broader applications to estimate generator parameter Φ within context of DNN along with *backpropagation algorithm* to train DNN. Besides, it is possible to consider error function is the minus opposite of likelihood function:

$$\varepsilon(\boldsymbol{x}|\Phi, z) = -P(\boldsymbol{x}|\Phi, \boldsymbol{z})$$

It is better that error is the minus opposite of log-likelihood function:

$$\varepsilon(\boldsymbol{x}|\Phi, z) = -\log P(\boldsymbol{x}|\Phi, \boldsymbol{z})$$

Moreover, there many ways to define likelihood and error and so, the way to define them will contribute to form a concrete DGM, besides how to specify and design generator DNN $g(\mathbf{z} \mid \Phi)$. When Φ is weight vector consisting of many weights of entire DNN, only elemental sub-weights at the output layer are estimated by SGD which maximizes likelihood or minimizes error:

$$\Phi_{\text{output}} = \Phi_{\text{output}} + \gamma\nabla P(\boldsymbol{x}|\Phi, z)$$

Or

$$\Phi_{\text{output}} = \Phi_{\text{output}} - \gamma\nabla\varepsilon(\boldsymbol{x}|\Phi, \boldsymbol{z})$$

Then backpropagation algorithm continues to update remaining sub-weights at hidden layers based on such determined sub-weights at the output layer. Therefore, for convenience, we only focus on likelihood maximization or error minimization and parameter Φ represents entire weights of DNN with assertion that backpropagation algorithm is always feasible. It means that there are two important equivalent estimation equations as follows:

$$\Phi = \Phi + \gamma\nabla P(\boldsymbol{x}|\Phi, \boldsymbol{z})$$

$$\Phi = \Phi - \gamma\nabla\varepsilon(\boldsymbol{x}|\Phi, \boldsymbol{z})$$

In similar to:

$$\Phi = \Phi + \gamma\nabla \log P(\boldsymbol{x}|\Phi, \boldsymbol{z})$$

When DGM is trained with big data, training data is fed to DGM at very time point i as a pair $\boldsymbol{d}^{(i)} = (\boldsymbol{x}^{(i)}, \mathbf{z}^{(i)})$ and therefore, a set of pairs over N time points is called epoch. As a convention, epoch of size

N is denoted as $\boldsymbol{D} = (\boldsymbol{d}^{(1)} = (\boldsymbol{x}^{(1)}, \mathbf{z}^{(1)}), \boldsymbol{d}^{(2)} = (\boldsymbol{x}^{(2)}, \mathbf{z}^{(2)}),\ldots, \boldsymbol{d}^{(N)} = (\boldsymbol{x}^{(N)}, \mathbf{z}^{(N)}))$. An interesting result from SGD is that DGM can be learned with epoch $\boldsymbol{D}$ without significant change as follows:

$$\Phi = \Phi + \gamma\nabla\frac{1}{N}\sum_{i=1}^{N} P\left(\boldsymbol{x}^{(i)}\middle|\Phi,\boldsymbol{z}^{(i)}\right) = \Phi + \gamma\frac{1}{N}\sum_{i=1}^{N} \nabla P\left(\boldsymbol{x}^{(i)}\middle|\Phi,\boldsymbol{z}^{(i)}\right)$$

Therefore, training data is counted according to every epoch $\boldsymbol{D}$ instead of every pair ($\boldsymbol{x}$, $\mathbf{z}$) so that $\boldsymbol{D}$ is fed to SGD at every time point k. Moreover, it is essential that SGD aims to update current parameter at current iteration based on previous parameter at previous iteration. Exactly, let $\Phi^{(k+1)}$ be generator parameter at the (k+1)th iteration, then $\Phi^{(k+1)}$ is calculated based on previous generator parameter $\Phi^{(k)}$ at the kth iteration as follows:

$$\Phi^{(k+1)} = \Phi^{(k)} + \gamma\nabla\frac{1}{N}\sum_{i=1}^{N} P\left(\boldsymbol{x}^{(i)}\middle|\Phi^{(k)},\boldsymbol{z}^{(i)}\right)$$

The equation above is the most precise equation for parameter estimation with SGD, which is called epoch estimation with note that SGD is an iterative process. It can also be replaced by following equations:

$$\Phi^{(k+1)} = \Phi^{(k)} + \gamma\nabla\frac{1}{N}\sum_{i=1}^{N} \log P\left(\boldsymbol{x}^{(i)}\middle|\Phi^{(k)},\boldsymbol{z}^{(i)}\right)$$

$$\Phi^{(k+1)} = \Phi^{(k)} - \gamma\nabla\frac{1}{N}\sum_{i=1}^{N} \varepsilon\left(\boldsymbol{x}^{(i)}\middle|\Phi^{(k)},\boldsymbol{z}^{(i)}\right)$$

The first equation

$$\Phi^{(k+1)} = \Phi^{(k)} + \gamma\nabla\frac{1}{N}\sum_{i=1}^{N} \log P\left(\boldsymbol{x}^{(i)}\middle|\Phi^{(k)},\boldsymbol{z}^{(i)}\right) \tag{1.5}$$

Which is most popular. Moreover, the index k indicates time point as well as iteration of SGD. If learning rate γ is varied at every iteration as $\gamma^{(k)}$, we have:

$$\Phi^{(k+1)} = \Phi^{(k)} + \gamma^{(k)}\nabla\frac{1}{N}\sum_{i=1}^{N} \log P\left(\boldsymbol{x}^{(i)}\middle|\Phi^{(k)},\boldsymbol{z}^{(i)}\right)$$

There are two problems related to construct a DGM: 1) how to define likelihood or error to train generator DNN $g(\mathbf{z} \mid \Phi)$ and 2) how to define tractable PDF $P(\mathbf{z})$ which implies the way to randomize $\mathbf{z}$. The second problem relates to assert qualification of random data $\mathbf{z}'$ and hence, the second problem is stated as qualification problem of how to qualify random data. Therefore, the two problems of constructing DGM are 1) how to train generator DNN $g(\mathbf{z} \mid \Phi)$ and 2) how to qualify such training task which often relates to another optimization task or another training task. Some basic principles related to DGM are introduced in this section but the two problems cannot be mentioned because there are many specific DGMs which have own specifications. Anyhow generator likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ based on definition of generator $g(\mathbf{z} \mid \Phi)$ is always important regardless that if it is specified explicitly and thus, suppose it was defined, then SGD is favorite method to optimize it. As an example aforementioned, suppose $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ distributes normally with mean $\boldsymbol{\mu}$ and variance σ^2 in some DGM as follows (Ruthotto & Haber, 2021, p. 3):

$$P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = (2\pi\sigma^2)^{-\frac{m}{2}}\exp\left(-\frac{\|g(\boldsymbol{z}|\Phi) - \boldsymbol{x} - \boldsymbol{\mu}\|^2}{2\sigma^2}\right)$$

Generator log-likelihood is natural logarithm of generator likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$:

$$\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = -\frac{m}{2}\log(2\pi\sigma^2) - \frac{\|g(\boldsymbol{z}|\Phi) - \boldsymbol{x} - \boldsymbol{\mu}\|^2}{2\sigma^2}$$

Gradient of this log-likelihood with regard to Φ is:

$$\nabla \log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \frac{d \log P(\boldsymbol{x}|\Phi,\boldsymbol{z})}{d\Phi} = -\frac{\left(g(\mathbf{z}|\Phi) - \boldsymbol{x} - \boldsymbol{\mu}\right)}{\sigma^2}\frac{dg(\mathbf{z}|\Phi)}{d\Phi}$$

Where $dg(\mathbf{z} \mid \Phi) / d\Phi$ is differential of $g(\mathbf{z} \mid \Phi)$ with regard to Φ. Let $\boldsymbol{\mu} = \mathbf{0}$ and $\sigma^2=1$ for optimization, we have:

$$\nabla \log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \frac{d \log P(\boldsymbol{x}|\Phi,\boldsymbol{z})}{d\Phi} = -\left(g(\mathbf{z}|\Phi) - \boldsymbol{x}\right)\frac{dg(\mathbf{z}|\Phi)}{d\Phi}$$

As usual, estimation equation resulted from SGD is:

$$\Phi = \Phi + \gamma \nabla \log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \Phi + \gamma\left(\boldsymbol{x} - g(\mathbf{z}|\Phi)\right)\frac{dg(\mathbf{z}|\Phi)}{d\Phi}$$

There is a question that how to calculate the differential $dg(\mathbf{z} \mid \Phi) / d\Phi$. Indeed, it is not difficult to calculate it in context of neural network associated with backpropagation algorithm so that the last output layer as well as last neuron o of DNN $g(\mathbf{z} \mid \Phi)$ is acted by activation function $a(.)$ as follows:

$$a(o) = a(\boldsymbol{w}^T\boldsymbol{i})$$

$$o = \boldsymbol{w}^T\boldsymbol{i}$$

Where $\boldsymbol{i}$ is input of the last layer o and weight parameter $\boldsymbol{w}$ is a part of entire parameter Φ and hence, we need to focus on calculating differential $da(o) / d\boldsymbol{w}$ which is equivalent to differential $dg(\mathbf{z} \mid \Phi) / d\Phi$ so that backpropagation algorithm will solve remaining parts of entire parameter Φ.

$$\frac{da(o)}{d\boldsymbol{w}} \cong \frac{dg(\mathbf{z}|\Phi)}{d\Phi}$$

Indeed, we have:

$$\frac{da(o)}{d\boldsymbol{w}} = \frac{da(\boldsymbol{w}^T\boldsymbol{i})}{d\boldsymbol{w}} = \frac{da(o)}{do}\boldsymbol{i}^T = a'(o)\boldsymbol{i}^T$$

Note, the subscript "T" denotes transposition operator of vector and matrix in which row vector becomes column vector and vice versa. It is easy to calculate the derivative $a'(o)$ when activation function was specified, for instance, if $a(o)$ is sigmoid function, we have:

$$a(o) = y = \frac{1}{1+e^{-o}}$$

$$a'(o) = \frac{1}{1+e^{-o}}\left(1 - \frac{1}{1+e^{-o}}\right) = a(o)\left(1 - a(o)\right) = y(1-y)$$

In practice, y is replaced by $a(y)$ in order to prevent o from being out of space:

$$a'(o) \cong a(y)\left(1 - a(y)\right) = a\left(a(o)\right)\left(1 - a\left(a(o)\right)\right)$$

As a result, we have:

$$\frac{dg(\mathbf{z}|\Phi)}{d\Phi} \cong a\left(a(o)\right)\left(1 - a\left(a(o)\right)\right)\boldsymbol{i}^T$$

For fast computation, it is possible to set the derivative $a'(o)$ to be small enough constants like 1 such that $dg(\mathbf{z} \mid \Phi) / d\Phi = \boldsymbol{i}^T$.

Suppose some other DGM assumes that $\boldsymbol{x}$ is binary ($\boldsymbol{x} = 0$ or $\boldsymbol{x} = 1$) and follows Bernoulli (Ruthotto & Haber, 2021, p. 3) and so, its generator DNN $g(\mathbf{z} \mid \Phi)$ derives values in interval [0, 1]. In other words, image of $g(\mathbf{z} \mid \Phi)$ is the real number interval [0, 1], which leads to a specification that $g(\mathbf{z} \mid \Phi)$ is probability of the event $\boldsymbol{x}$=1 with note that $\boldsymbol{x}$ is scalar variable (x) for convenience:

$$g(\mathbf{z}|\Phi) = P(\boldsymbol{x} = 1)$$

Because $g(\mathbf{z} \mid \Phi)$ becomes a (scalar) random variable whose value is probability, it is possible to identify $g(\mathbf{z} \mid \Phi)$ with its parameter Φ as a convention:

$$\Phi = g(\mathbf{z}|\Phi) = P(\boldsymbol{x} = 1)$$

Given N trials with binary values of $\boldsymbol{x}$, let $N(\boldsymbol{x})$ be the number of event $\boldsymbol{x}$=1 among N trials, then generator likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is specified according to Bernoulli distribution as follows:

$$P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \Phi^{N(\boldsymbol{x})}(1-\Phi)^{N-N(\boldsymbol{x})}$$

The generator log-likelihood is:

$$\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = N(\boldsymbol{x})\log\Phi + \big(N - N(\boldsymbol{x})\big)\log(1-\Phi)$$

Gradient of the generator log-likelihood with regard to Φ is:

$$\nabla \log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \frac{d\log P(\boldsymbol{x}|\Phi,\boldsymbol{z})}{d\Phi} = \frac{N(\boldsymbol{x})}{\Phi} - \frac{N-N(\boldsymbol{x})}{1-\Phi} = \frac{N(\boldsymbol{x})-N\Phi}{\Phi(1-\Phi)}$$

As a result, estimation equation resulted from SGD is:

$$\Phi = \Phi + \gamma\nabla\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \Phi + \gamma\frac{N(\boldsymbol{x})-N\Phi}{\Phi(1-\Phi)}$$

Although normal distribution and Bernoulli distribution are two popular distributions to specify generator likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$, there are other specifications which depend on specific DGM.

Given epoch $\boldsymbol{D} = (\boldsymbol{d}^{(1)} = (\boldsymbol{x}^{(1)}, \mathbf{z}^{(1)}), \boldsymbol{d}^{(2)} = (\boldsymbol{x}^{(2)}, \mathbf{z}^{(2)}),\ldots, \boldsymbol{d}^{(N)} = (\boldsymbol{x}^{(N)}, \mathbf{z}^{(N)}))$ implies that the epoch is created or sent by equilateral distribution $1/N$ but in general case, $\boldsymbol{D}$ can follow an arbitrary distribution denoted by PDF $P(\boldsymbol{d})$, which makes the optimization problem and the SGD estimation changed a little bit by theoretical expectation given distribution $P(\boldsymbol{d})$.

$$\Phi^* = \underset{\Phi}{\mathrm{armax}}\, E\big(P(\boldsymbol{x}|\Phi,\boldsymbol{z})\big|P(\boldsymbol{d})\big)$$

$$\Phi = \Phi + \gamma\nabla E\big(P(\boldsymbol{x}|\Phi,\boldsymbol{z})\big|P(\boldsymbol{d})\big)$$

Where,

$$E\big(P(\boldsymbol{x}|\Phi,\boldsymbol{z})\big|P(\boldsymbol{d})\big) = \int_{\boldsymbol{d}} P(\boldsymbol{x}|\Phi,\boldsymbol{z})P(\boldsymbol{d})d\boldsymbol{d}$$

However, there is no significant change in aforementioned practical technique to estimate parameters.

Turning back the assumption that generator likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ distributes normally with mean $\boldsymbol{\mu}$ and variance σ^2 in some DGM as follows (Ruthotto & Haber, 2021, p. 3):

$$P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = (2\pi\sigma^2)^{-\frac{m}{2}}\exp\left(-\frac{\|g(\boldsymbol{z}|\Phi)-\boldsymbol{x}-\boldsymbol{\mu}\|^2}{2\sigma^2}\right)$$

This assumption is not totally exact because the distribution above mentions the error $g(\mathbf{z} \mid \Phi) - \boldsymbol{x}$ between generated data $g(\mathbf{z} \mid \Phi)$ and real data $\boldsymbol{x}$. Exactly, generator likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is defined as distribution of the error $\boldsymbol{\varepsilon} = g(\mathbf{z} \mid \Phi) - \boldsymbol{x}$ and such error distribution is assumed to follow normal distribution with mean $\boldsymbol{\mu}$ and variance σ^2.

$$P(\boldsymbol{\varepsilon}|\boldsymbol{\mu},\sigma^2) = P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = (2\pi\sigma^2)^{-\frac{m}{2}}\exp\left(-\frac{\|g(\boldsymbol{z}|\Phi)-\boldsymbol{x}-\boldsymbol{\mu}\|^2}{2\sigma^2}\right)$$

Therefore, setting error mean and error variance to be zero and one as $\boldsymbol{\mu} = \mathbf{0}$, $\sigma^2 = 1$ is for best optimization because of smallest error mean $\mathbf{0}$ but the setting is not totally diverse in data generation.

$$P(\boldsymbol{\varepsilon}|\boldsymbol{\mu},\sigma^2) = P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = (2\pi)^{-\frac{m}{2}}\exp\left(-\frac{\|g(\boldsymbol{z}|\Phi)-\boldsymbol{x}\|^2}{2}\right)$$

When learning generator DNN by backpropagation algorithm associated with SGD, it is possible to estimate dynamically $\boldsymbol{\mu}$ and σ^2 by maximum likelihood estimation (MLE) method. Given epoch $\boldsymbol{D} = (\boldsymbol{d}^{(1)} = (\boldsymbol{x}^{(1)}, \mathbf{z}^{(1)}), \boldsymbol{d}^{(2)} = (\boldsymbol{x}^{(2)}, \mathbf{z}^{(2)}),\ldots, \boldsymbol{d}^{(N)} = (\boldsymbol{x}^{(N)}, \mathbf{z}^{(N)}))$, error mean and error variance are estimated as follows:

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N}\left(g\left(\mathbf{z}^{(i)}\middle|\Phi\right) - \boldsymbol{x}^{(i)}\right)$$
$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}\left\|g\left(\mathbf{z}^{(i)}\middle|\Phi\right) - \boldsymbol{x}^{(i)} - \hat{\boldsymbol{\mu}}\right\|^2 \quad (1.6)$$

When error mean and error variance are dynamically estimated instead of fixing them by zero and unit, generator DNN $g(\mathbf{z} \mid \Phi)$ may produce new data in high diversity, which is similar to add noises to generated data. In other words, estimation of error mean and error variance based on epoch makes the data generation more diverse because **z** may be randomized in interval [**0**, **1**] although DGMs try to diversify **z** or $\boldsymbol{x}$ like Variational Autoencoders (VAE) and Generative Adversarial Network (GAN). Note, if **z** is randomized only in interval [**0**, **1**], generated data $\boldsymbol{x}' = g(\mathbf{z} \mid \Phi)$ may not be different much from sample $\boldsymbol{x}$ in epoch in case that error mean $\boldsymbol{\mu}$ and error variance σ^2 are fixed by **0** and 1. However, quality of data generation is the best with zero error mean **0**.

Recall that the two problems of constructing DGM are 1) how to train generator DNN $g(\mathbf{z} \mid \Phi)$ and 2) how to qualify such training task which often relates to another optimization task or another training task. The first problem relates to how to establish generator likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ which is the probability density function (PDF) of intractable $\boldsymbol{x}$ given tractable data **z** and this establishment is based on generator DNN $g(\mathbf{z} \mid \Phi)$. However, there are some DGMs do not specify explicitly the density function $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$, which is cause of the fact that there are two DGM approaches: 1) DGM specifies explicitly generator PDF $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ and 2) vice versa. In group of explicit PDF approach, there are two built-in approaches: 1) tractable density DGM specifies explicitly well-known distributions for generator likelihood and 2) approximate density DGM tries to estimate approximately generator PDF $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ or derive other PDF that is similar to $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$. In general, there are three main approaches for constructing DGM such as tractable density DGM, approximate density DGM, and implicit density DGM which are mentioned in next sections. Following figure depicts taxonomy of DGM (Oussidi & Elhassouny, 2018, p. 7) by Goodfellow.

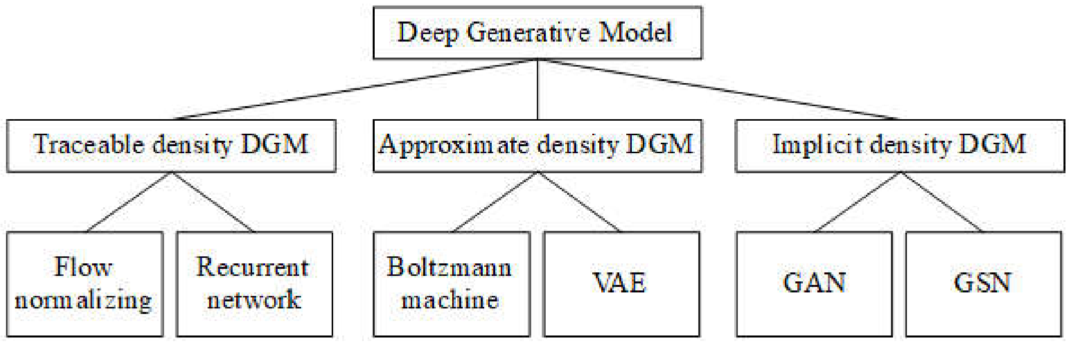

Figure 1. 1. Taxonomy of DGM.

Especially, if data is image, there is another categorization way that there are two main approaches: 1) pixel density approach tries to model pixel distribution and 2) block density approach tries to model entire image distribution as any data distribution. In other words, likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is defined based on probabilistic distribution of pixels where $\boldsymbol{x}$ is considered as set of pixels according to pixel density approach. On the other hand, block density approach considers likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is PDF of a block or entire image (unified big block) where $\boldsymbol{x}$ is considered as any arbitrary data. A usual, pixel density approach belongs to tractable density approach of the first categorization.

2. Tractable Density DGM

According to tractable density approach, DGMs specify explicitly generator PDF $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ with note that PDF is abbreviation of probability density function. Recall that the two problems of constructing DGM are 1) how to train generator DNN $g(\mathbf{z} \mid \Phi)$ and 2) how to qualify such training task which often relates to another optimization task or another training task. However, the two problems are merged into the first problem which is to train $g(\mathbf{z} \mid \Phi)$ according to normalizing flow technique in which $g(\mathbf{z} \mid \Phi)$ is invertible given tractable data $\mathbf{z}$ and intractable data $\boldsymbol{x}$ have the same dimension n. Therefore, latent space Z and sample space X are the same with dimension n. As a convention, $g^{-1}(\boldsymbol{x} \mid \Phi)$ is called inversed generator.

$$\begin{aligned}\boldsymbol{x} &= g(\boldsymbol{z}|\Phi)\\ \boldsymbol{z} &= g^{-1}(\boldsymbol{x}|\Phi)\end{aligned} \quad (2.1)$$

Where,

$$\begin{aligned}&\boldsymbol{z} = (z_1, z_2, \dots, z_n)^T, \boldsymbol{x} = (x_1, x_2, \dots, x_n)^T\\ &\boldsymbol{x} \in X \subseteq R^n\\ &\boldsymbol{z} \in Z \subseteq R^n\\ &X = Z\end{aligned}$$

Note, the subscript "T" denotes transposition operator of vector and matrix in which row vector becomes column vector and vice versa. Because $g(\mathbf{z} \mid \Phi)$ is essentially vector-by-vector function whose input and output are vectors, it should have denoted as $\boldsymbol{g}(\mathbf{z} \mid \Phi)$, especially, when $\boldsymbol{g}(\mathbf{z} \mid \Phi)$ here is bijection. However, it is still denoted $g(\mathbf{z} \mid \Phi)$ for convenience. Therefore, the first-order derivative of vector-by-vector function $g(\mathbf{z} \mid \Phi)$ here is Jacobian matrix but is stilled called gradient. Note, in mathematical, the first-order derivative of scalar-by-vector function is called gradient vector and the first-order derivative of vector-by-vector function is called Jacobian matrix.

As a result, **normalizing flow** (**NL**) technique focuses on maximizing intractable PDF $P(\boldsymbol{x})$ now called sample PDF or sample likelihood rather than maximizing generator likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ because $P(\boldsymbol{x})$ is now proportional to tractable PDF $P(\mathbf{z})$ and $P(\boldsymbol{x})$ is stronger than $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$. When $P(\boldsymbol{x})$ has generator parameter Φ, it is denoted as $P(\boldsymbol{x} \mid \Phi)$. According to applied statistics literature, sample likelihood $P(\boldsymbol{x} \mid \Phi)$ is determined based on tractable PDF $P(\mathbf{z})$ and generator $g(\mathbf{z} \mid \Phi)$ as follows (Ruthotto & Haber, 2021, p. 7):

$$P(\boldsymbol{x}|\Phi) = P(\boldsymbol{z})|\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x}|\Phi)| = P\big(g^{-1}(\boldsymbol{x}|\Phi)\big)|\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x}|\Phi)| \quad (2.2)$$

Where |.| or det(.) denotes determinant of square matrix with note that the gradient $\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x} \mid \Phi)$ of the inverse $g^{-1}(\boldsymbol{x} \mid \Phi)$ is Jacobian matrix which is the first-order derivative of $g^{-1}(\boldsymbol{x} \mid \Phi)$ with regard to $\boldsymbol{x}$. As a convention, $\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x} \mid \Phi)$ is called inversed gradient because $\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x} \mid \Phi)$ is the first-order derivative of inversed generator $g^{-1}(\boldsymbol{x} \mid \Phi)$ with regard to $\boldsymbol{x}$.

$$\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x}|\Phi) = \frac{dg^{-1}(\boldsymbol{x}|\Phi)}{d\boldsymbol{x}}$$

The equation of sample likelihood $P(\boldsymbol{x} \mid \Phi)$ is much more definite than the integral formulation of $P(\boldsymbol{x})$ as aforementioned

$$P(\boldsymbol{x}) = \int_{\boldsymbol{z}} P(\boldsymbol{x}|\Phi, \boldsymbol{z}) P(\boldsymbol{z})\, d\boldsymbol{z}$$

It is explained from the equation of sample likelihood $P(\boldsymbol{x} \mid \Phi)$ that given source and target with a function from source to target, target distribution is calculated by multiplying source distribution with determinant of gradient of inversed function.

$$P(\boldsymbol{x}|\Phi) = P\big(g^{-1}(\boldsymbol{x}|\Phi)\big)|\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x}|\Phi)|$$

For optimization, $P(\mathbf{z})$ is assumed to follow standard normal distribution with mean 0 and variance 1:

$$P(\boldsymbol{z}) = (2\pi)^{-\frac{n}{2}} \exp\left(-\frac{1}{2}\|\boldsymbol{z}\|^2\right)$$

Such that:

$$P(\boldsymbol{x}|\Phi) = (2\pi)^{-\frac{n}{2}}\exp\left(-\frac{1}{2}\|g^{-1}(\boldsymbol{x}|\Phi)\|^2\right)|\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)|$$

Where notation ||.|| denotes norm of vector. Exactly, $P(\mathbf{z})$ follows standard normal distribution with mean vector **0** and identity covariance matrix I. Sample log-likelihood is derived by taking natural logarithm of sample likelihood:

$$\log P(\boldsymbol{x}|\Phi) = -\frac{n}{2}\log 2\pi - \frac{1}{2}\|g^{-1}(\boldsymbol{x}|\Phi)\|^2 + \log|\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)|$$

NL aims to maximize sample log-likelihood so as to estimate generator parameter Φ:

$$\Phi^* = \underset{\Phi}{\text{armax}}\log P(\boldsymbol{x}|\Phi)$$

Stochastic gradient descent (SGD) method is used to estimate Φ by iterative process to update successively Φ at every iteration as follows:

$$\Phi = \Phi + \gamma\nabla\log P(\boldsymbol{x}|\Phi)$$

Where $\nabla\log P(\boldsymbol{x} \mid \Phi)$ which is called sample log-likelihood gradient is gradient of sample log-likelihood $\log P(\boldsymbol{x} \mid \Phi)$ and γ ($0 < \gamma \leq 1$) is learning rate. Note that SGD, which is an iterative process, pushes candidate solution at each iteration along the direction which is opposite to gradient of target function for minimization or has the same direction to gradient of target function for maximization with note that the step length is represented by learning rate. Given epoch of size N is denoted as $\boldsymbol{D} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)},\ldots, \boldsymbol{x}^{(N)})$, the estimation equation of Φ is extended exactly as epoch estimation at every iteration of SGD:

$$\Phi^{(k+1)} = \Phi^{(k)} + \gamma\nabla\frac{1}{N}\sum_{i=1}^{N}\log P\left(\boldsymbol{x}^{(i)}\middle|\Phi^{(k)}\right)$$

It is necessary to determine sample log-likelihood gradient $\nabla\log P(\boldsymbol{x} \mid \Phi)$ with regard to parameter Φ. Due to (Nguyen, Matrix Analysis and Calculus, 2015, pp. 45-46):

$$\frac{d\log(|\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)|)}{d\Phi} = \frac{d\log(|\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)|)}{d\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)}\frac{d\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)}{d\Phi}$$

$$= \frac{d|\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)|}{d\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)}\frac{1}{|\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)|}\frac{d\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)}{d\Phi}$$

$$= \frac{|\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)|\left(\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)\right)^{-1}}{|\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)|}\frac{d\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)}{d\Phi}$$

$$= \left(\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)\right)^{-1}\frac{d\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)}{d\Phi}$$

And

$$\frac{d\frac{1}{2}\|g^{-1}(\boldsymbol{x}|\Phi)\|^2}{d\Phi} = g^{-1}(\boldsymbol{x}|\Phi)\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)$$

We have following equation to calculate log-likelihood gradient $\nabla\log P(\boldsymbol{x} \mid \Phi)$:

$$\nabla\log P(\boldsymbol{x}|\Phi) = \frac{d\log P(\boldsymbol{x}|\Phi)}{d\Phi} = \left(\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)\right)^{-1}\frac{d\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)}{d\Phi} - g^{-1}(\boldsymbol{x}|\Phi)\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x}|\Phi)$$

The notation $(\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x} \mid \Phi))^{-1}$ denotes inverse of matrix $\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x} \mid \Phi)$. Because $\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x} \mid \Phi)$ called Jacobian matrix is a square matrix, the derivative $d\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x} \mid \Phi) / d\Phi$ is determined by taking first-order derivative for every element of $\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x} \mid \Phi)$ with regard to Φ, which produces a tensor. Therefore, $d\nabla_{\boldsymbol{x}}g^{-1}(\boldsymbol{x} \mid \Phi) / d\Phi$ is the second-order derivative of inversed generator $g^{-1}(\boldsymbol{x} \mid \Phi)$ with regard to $\boldsymbol{x}$ and Φ. Let

$$\nabla_\Phi \nabla_x g^{-1}(\boldsymbol{x}|\Phi) = \frac{d\nabla_x g^{-1}(\boldsymbol{x}|\Phi)}{d\Phi}$$

It is possible to calculate this second-order derivative if inversed gradient $\nabla_x g^{-1}(\boldsymbol{x} \mid \Phi)$ is determined. Log-likelihood gradient $\nabla \log P(\boldsymbol{x} \mid \Phi)$ is rewritten:

$$\nabla \log P(\boldsymbol{x}|\Phi) = \left(\nabla_x g^{-1}(\boldsymbol{x}|\Phi)\right)^{-1} \nabla_\Phi \nabla_x g^{-1}(\boldsymbol{x}|\Phi) - g^{-1}(\boldsymbol{x}|\Phi) \nabla_x g^{-1}(\boldsymbol{x}|\Phi) \quad (2.3)$$

Where:

$$\nabla_x g^{-1}(\boldsymbol{x}|\Phi) = \frac{dg^{-1}(\boldsymbol{x}|\Phi)}{d\boldsymbol{x}}$$

$$\nabla_\Phi \nabla_x g^{-1}(\boldsymbol{x}|\Phi) = \frac{d\nabla_x g^{-1}(\boldsymbol{x}|\Phi)}{d\Phi}$$

According to traditional neural network, let φ_i be the ith row vector of matrix Φ, then generator $g(\boldsymbol{x} \mid \varphi)$ is linear composition as follows:

$$g(\boldsymbol{z}|\varphi) = a(\varphi_i^T \boldsymbol{z} + \delta_i) = x_i$$

Where δ_i is the ith bias parameter associated with each x_i. Note, x_i is the ith elemental variable in $\boldsymbol{x}$ whereas activation $a(.)$ is invertible, whose inverse is $a^{-1}(.)$. In traditional neural network, x_i represents a neuron or unit. Due to:

$$a^{-1}(x_i) = \varphi_i^T \boldsymbol{z} + \delta_i = \sum_{t=1}^{n} \varphi_{it} z_t + \delta_i = \varphi_{ij} z_j + \sum_{t=1, t\neq j}^{n} \varphi_{it} z_t + \delta_i$$

When $\varphi_i = (\varphi_{i1}, \varphi_{i2},\ldots, \varphi_{in})^T$ and $\mathbf{z} = (z_1, z_2,\ldots, z_n)^T$, without loss of generality, given φ_{ij} and z_j are the jth elements of φ_i and $\mathbf{z}$, respectively we have fine-tuned inversed generator $g^{-1}(x_i \mid \varphi_{ij})$:

$$g^{-1}(x_i|\varphi_{ij}) = z_j = \frac{a^{-1}(x_i) - b_{ij} - \delta_i}{\varphi_{ij}}$$

Where,

$$b_{ij} = \sum_{t=1, t\neq j}^{n} \varphi_{it} z_t$$

It is easy to calculate the inversed gradient:

$$\nabla_{x_i} g^{-1}(x_i|\varphi_{ij}) = \frac{d\left(a^{-1}(x_i) - b_{ij} - \delta_i\right)/dx_i}{\varphi_{ij}} = \frac{\left(a^{-1}(x_i)\right)'}{\varphi_{ij}}$$

Where $a^{-1}(x_i)$ is the first-order derivative of $a^{-1}(.)$ at x_i. The second-order derivative $\nabla_{\varphi_{ij}} \nabla_{x_i} g^{-1}(x_i|\varphi_{ij})$ is determined as follows:

$$\nabla_{\varphi_{ij}} \nabla_{x_i} g^{-1}(x_i|\varphi_{ij}) = \frac{d\nabla_{x_i} g^{-1}(x_i|\varphi_{ij})}{d\varphi_{ij}} = -\frac{\left(a^{-1}(x_i)\right)'}{\varphi_{ij}^2}$$

Log-likelihood gradient with regard to φ_{ij} is fine-tuned as $\nabla \log P(x_i \mid \varphi_{ij})$ is expended again:

$$\nabla \log P(x_i|\varphi_{ij}) = -\frac{\varphi_{ij}}{\left(a^{-1}(x_i)\right)'} \frac{\left(a^{-1}(x_i)\right)'}{\varphi_{ij}^2} - \frac{a^{-1}(x_i)}{\varphi_{ij}} \frac{\left(a^{-1}(x_i)\right)'}{\varphi_{ij}}$$

$$= -\frac{1}{\varphi_{ij}} - \frac{a^{-1}(x_i)\left(a^{-1}(x_i)\right)'}{\varphi_{ij}^2} = -\frac{\varphi_{ij} + a^{-1}(x_i)\left(a^{-1}(x_i)\right)'}{\varphi_{ij}^2}$$

Because δ_i is the ith bias parameter, the second-order derivative $\nabla_{\delta_i}\nabla_{x_i}g^{-1}\left(x_i \middle| \varphi_{ij}\right)$ is determined as follows:

$$\nabla_{\delta_i}\nabla_{x_i}g^{-1}(x_i|\delta_i) = \frac{d\nabla_{x_i}g^{-1}(x_i|\delta_i)}{d\delta_i} = 0$$

Where,

$$\nabla_{x_i}g^{-1}(x_i|\delta_i) = \frac{d\left(a^{-1}(x_i) - b_{ij} - \delta_i\right)/dx_i}{\varphi_{ij}} = \frac{\left(a^{-1}(x_i)\right)'}{\varphi_{ij}}$$

$$g^{-1}(x_i|\delta_i) = z_j = \frac{a^{-1}(x_i) - b_{ij} - \delta_i}{\varphi_{ij}}$$

Log-likelihood gradient with regard to δ_i is fine-tuned as $\nabla\log P(x_i \mid \delta_i)$ is expended again:

$$\nabla\log P(x_i|\delta_i) = -\frac{\varphi_{ij}}{\left(a^{-1}(x_i)\right)'}0 - \frac{a^{-1}(x_i)}{\varphi_{ij}}\frac{\left(a^{-1}(x_i)\right)'}{\varphi_{ij}} = -\frac{a^{-1}(x_i)\left(a^{-1}(x_i)\right)'}{\varphi_{ij}^2}$$

In general, log-likelihood gradient $\nabla\log P(x_i \mid \varphi_{ij}, \delta_i)$ is specified as follows:

$$\begin{gathered}\nabla\log P\left(x_i\middle|\varphi_{ij}\right) = -\frac{1}{\varphi_{ij}}\left(1 + \frac{a^{-1}(x_i)\left(a^{-1}(x_i)\right)'}{\varphi_{ij}}\right), \forall i,j \\ \nabla\log P(x_i|\delta_i) = -\frac{a^{-1}(x_i)\left(a^{-1}(x_i)\right)'}{\varphi_{ij}^2}, \forall i\end{gathered} \tag{2.4}$$

Where $a(.)$ and $a^{-1}(.)$ are invertible activation function and its inverse and,

$$\begin{gathered}x_i \in \boldsymbol{x} = (x_1, x_2, \dots, x_n)^T \\ \varphi_{ij} \in \varphi_i = (\varphi_{i1}, \varphi_{i2}, \dots, \varphi_{in})^T \\ \varphi_i \in \Phi \\ x_i = a(\varphi_i^T\boldsymbol{z} + \delta_i)\end{gathered}$$

SGD estimation is fine-tuned as follows:

$$\varphi_{ij} = \varphi_{ij} + \gamma\nabla\log P\left(x_i\middle|\varphi_{ij}\right) = \varphi_{ij} - \gamma\frac{1}{\varphi_{ij}}\left(1 + \frac{a^{-1}(x_i)\left(a^{-1}(x_i)\right)'}{\varphi_{ij}}\right), \forall i,j$$

$$\delta_i = \delta_i + \gamma\nabla\log P(x_i|\delta_i) = \delta_i - \gamma\frac{a^{-1}(x_i)\left(a^{-1}(x_i)\right)'}{\varphi_{ij}^2}, \forall i$$

Given epoch of size N is denoted as $\boldsymbol{D} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)},\dots, \boldsymbol{x}^{(N)})$, the estimation equation of φ_{ij} and δ_i is extended exactly as epoch estimation at every uth iteration of SGD with regard to log-likelihood gradient $\nabla\log P(x_i \mid \varphi_{ij}, \delta_i)$.

$$\begin{aligned}\varphi_{ij}^{(u+1)} &= \varphi_{ij}^{(u)} + \gamma\frac{1}{N}\nabla\sum_{v=1}^{N}\log P\left(x_i^{(v)}\middle|\varphi_{ij}^{(u)}\right) \\ &= \varphi_{ij}^{(u)} - \gamma\frac{1}{\varphi_{ij}^{(u)}}\left(1 + \frac{1}{N\varphi_{ij}^{(u)}}\sum_{v=1}^{N}a^{-1}\left(x_i^{(v)}\right)\left(a^{-1}\left(x_i^{(v)}\right)\right)'\right), \forall i,j\end{aligned}$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 May 2024 doi:10.20944/preprints202405.1348.v1

$$\delta_i^{(u+1)} = \delta_i^{(u)} + \gamma \frac{1}{N} \nabla \sum_{v=1}^{N} \log P\left(x_i^{(v)} \middle| \delta_i^{(u)}\right)$$

$$= \delta_i^{(u)} - \gamma \frac{1}{N\left(\varphi_{ij}^{(u)}\right)^2} \sum_{v=1}^{N} a^{-1}\left(x_i^{(v)}\right)\left(a^{-1}\left(x_i^{(v)}\right)\right)', \forall i$$

Where $x_i^{(v)}$ is the ith element of $\boldsymbol{x}^{(v)}$ in epoch. As a result, NL trained with SGD is specified as follows:

Initialize all φ_{ij}, δ_i and set $u = 0$.

Repeat

Sampling epoch $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)},\ldots, \boldsymbol{x}^{(N)})$ or receiving epoch $\boldsymbol{X}$ from big data / data stream.

$$\varphi_{ij}^{(u+1)} = \varphi_{ij}^{(u)} - \gamma \frac{1}{\varphi_{ij}^{(u)}}\left(1 + \frac{1}{N\varphi_{ij}^{(u)}} \sum_{v=1}^{N} a^{-1}\left(x_i^{(v)}\right)\left(a^{-1}\left(x_i^{(v)}\right)\right)'\right), \forall i, j$$

$$\delta_i^{(u+1)} = \delta_i^{(u)} - \gamma \frac{1}{N\left(\varphi_{ij}^{(u)}\right)^2} \sum_{v=1}^{N} a^{-1}\left(x_i^{(v)}\right)\left(a^{-1}\left(x_i^{(v)}\right)\right)', \forall i$$

Increase $u = u + 1$.

Until some terminating conditions are met.

Note, a terminating condition is customized, for example, parameters φ_{ij} and δ_i are not changed significantly or there is no more coming epoch $\boldsymbol{X}$. Moreover, the index u indicates time point as well as iteration of SGD. After finite NL is trained, it can generate new data $\boldsymbol{x}'$ by generator $g(\mathbf{z} \mid \Phi) = \boldsymbol{x}'$ with any $\mathbf{z}$ randomized from standard normal distribution with mean $\mathbf{0}$ and variance 1.

It is interesting that log-likelihood gradient $\nabla \log P(x_i \mid \varphi_{ij})$ is determined based on inversed gradient $\nabla_{x_i} g^{-1}(x_i|\varphi_{ij})$. Therefore, how to estimate generator parameter Φ by SGD estimation focuses on calculating inversed gradient $\nabla_{x_i} g^{-1}(x_i|\varphi_{ij})$ which is central point of normalizing flow (NL) technique. Moreover, how to calculate $\nabla_{x_i} g^{-1}(x_i|\varphi_{ij})$ is based on how to determine inversed generator $g^{-1}(\boldsymbol{x} \mid \Phi)$. In other words, the main problem of NL is how to determine inversed generator $g^{-1}(\boldsymbol{x} \mid \Phi)$ because it is easy to calculate gradient of function $f(\boldsymbol{x}) = g^{-1}(\boldsymbol{x} \mid \Phi)$ with regard to $\boldsymbol{x}$. Especially, when generator $g(\boldsymbol{x} \mid \Phi)$ is implemented by DNN, NL will have some special techniques so that determining its inverse $g^{-1}(\boldsymbol{x} \mid \Phi)$ is easier. One of these technique is *finite normalizing flow* (finite NL) in which generator $g(\boldsymbol{x} \mid \Phi)$ is implemented by a DNN having K layers from layer 1 to layer K where layer 0 is input layer with note that each layer is represented by partial generator function f_k (Ruthotto & Haber, 2021, p. 8):

$$g(\mathbf{z}|\Phi) = f_K \circ f_{K-1} \circ \ldots \circ f_2 \circ f_1(\mathbf{z}) \tag{2.5}$$

Note, all layers f_k have the same number of neurons which is the dimension n. Because f_k is essentially vector-by-vector function whose input and output are vectors, it should have denoted as $\boldsymbol{f}_k$, especially, when f_k here is bijection. However, it is still denoted f_k for convenience. Let $\mathbf{z}^{(k+1)}$ be output of partial generator f_k, we have:

$$\mathbf{z}^{(k+1)} = f_k\left(\mathbf{z}^{(k)}\right)$$

$$\mathbf{z} = \mathbf{z}^{(1)} = \mathbf{z}^{(0)} = f_0\left(\mathbf{z}^{(0)}\right) \equiv f_0$$

$$\boldsymbol{x} = \mathbf{z}^{(K+1)} = f_K\left(\mathbf{z}^{(K)}\right)$$

Inversed generator $g^{-1}(\boldsymbol{x} \mid \Phi)$ representing inversed DNN is determined:

$$g^{-1}(\boldsymbol{x}|\Phi) = f_1^{-1} \circ f_2^{-1} \circ \ldots \circ f_{K-1}^{-1} \circ f_K^{-1}(\boldsymbol{x}) \tag{2.6}$$

Each f_k^{-1} is called inversed partial generator function which is the inverse of partial generator function f_k. Let $\boldsymbol{x}^{(k-1)}$ be output of partial generator f_k^{-1}, we have:

$$\boldsymbol{x}^{(k-1)} = f_k^{-1}\left(\boldsymbol{x}^{(k)}\right)$$

$$\boldsymbol{x} = \boldsymbol{x}^{(K+1)} = \boldsymbol{x}^{(K)} = f_{K+1}^{-1}\left(\boldsymbol{x}^{(K+1)}\right) \equiv f_{K+1}^{-1}$$

$$\boldsymbol{z} = \boldsymbol{x}^{(0)} = f_2^{-1}\left(\boldsymbol{x}^{(1)}\right)$$

"Input layer" of "inversed DNN" is f_{K+1}^{-1}. The inversed generator DNN may be pseudo in case that only one generator DNN is designed so that inversed generator function f_k^{-1} is existent. An interesting result of the design of finite NL is that inversed gradient $\nabla g^{-1}(\boldsymbol{x} \mid \Phi)$ is product of gradients of inversed partial generator f_k^{-1}.

$$\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x}|\Phi) = \prod_{k=1}^{K} \nabla_{\boldsymbol{x}^{(k)}} f_k^{-1}\left(\boldsymbol{x}^{(k)}\middle|\Phi^{(k)}\right)$$

$$\log\left|\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x}|\Phi)\right| = \sum_{k=1}^{K} \left|\nabla_{\boldsymbol{x}^{(k)}} f_k^{-1}\left(\boldsymbol{x}^{(k)}\middle|\Phi^{(k)}\right)\right|$$

Where $\Phi^{(k)}$ is parameter of f_k. It is now necessary to determine fine-tuned partial inversed gradient $\nabla_{x_i^{(k)}} f_k^{-1}\left(x_i^{(k)}\middle|\varphi_{ij}^{(k)}\right)$ in order to determine fine-tuned partial log-likelihood gradient $\nabla\log P(x_i^{(k)} \mid \varphi_{ij}^{(k)})$ where $x_i^{(k)}$ is an elemental variable in $\boldsymbol{x}^{(k)} = (x_1^{(k)}, x_2^{(k)},\ldots, x_n^{(k)})^T$ and $\varphi_{ij}^{(k)}$ is the jth element in $\varphi_i^{(k)} = (\varphi_{i1}^{(k)}, \varphi_{i2}^{(k)},\ldots, \varphi_{in}^{(k)})^T$ with note that $\varphi_i^{(k)}$ is the ith row vector of matrix $\Phi^{(k)}$. Moreover, let $\delta_i^{(k)}$ be the ith bias parameter associated with each $x_i^{(k)}$. Without loss of generality, given $\varphi_j^{(k)}$, $\delta_i^{(k)}$, and $z_j^{(k)}$ along with invertible activation $a(.)$, we have fine-tuned inversed generator $g^{-1}(x_i^{(k)} \mid \varphi_{ij}^{(k)}, \delta_i^{(k)})$.

$$f_k^{-1}\left(x_i^{(k)}\middle|\varphi_{ij}^{(k)}\right) = z_j^{(k)} = \frac{a^{-1}\left(x_i^{(k)}\right) - b_{ij}^{(k)} - \delta_i^{(k)}}{\varphi_{ij}^{(k)}}$$

Where,

$$b_{ij}^{(k)} = \sum_{t=1,t\neq j}^{n} \varphi_{it}^{(k)} z_t^{(k)}$$

Where $z_i^{(k)}$ is the ith elemental variable in $\mathbf{z}^{(k)} = (z_1^{(k)}, z_2^{(k)},\ldots, z_n^{(k)})^T$. By similar way aforementioned, log-likelihood gradient $\nabla\log P(x_i^{(k)} \mid \varphi_{ij}^{(k)}, \delta_i^{(k)})$ is specified as follows:

$$\nabla \log P\left(x_i^{(k)}\middle|\varphi_{ij}^{(k)}\right) = -\frac{1}{\varphi_{ij}^{(k)}}\left(1 + \frac{a^{-1}\left(x_i^{(k)}\right)\left(a^{-1}\left(x_i^{(k)}\right)\right)'}{\varphi_{ij}^{(k)}}\right), \forall i,j$$

$$\nabla \log P\left(x_i^{(k)}\middle|\delta_i^{(k)}\right) = -\frac{a^{-1}\left(x_i^{(k)}\right)\left(a^{-1}\left(x_i^{(k)}\right)\right)'}{\left(\varphi_{ij}^{(k)}\right)^2}, \forall i \tag{2.7}$$

Where $a(.)$ and $a^{-1}(.)$ are invertible activation function and its inverse and,

$$x_i^{(k)} \in \boldsymbol{x}^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right)^T$$

$$\varphi_{ij}^{(k)} \in \varphi_i^{(k)} = \left(\varphi_{i1}^{(k)}, \varphi_{i2}^{(k)}, \ldots, \varphi_{in}^{(k)}\right)^T$$

$$\varphi_i^{(k)} \in \Phi^{(k)}$$

$$x_i^{(k)} = a\left(\left(\varphi_i^{(k)}\right)^T \boldsymbol{z} + \delta_i^{(k)}\right)$$

SGD estimation is fine-tuned as follows:

$$\varphi_{ij}^{(k)} = \varphi_{ij}^{(k)} + \gamma\nabla \log P\left(x_i^{(k)}\middle|\varphi_{ij}^{(k)}\right) = \varphi_{ij}^{(k)} - \gamma\frac{1}{\varphi_{ij}^{(k)}}\left(1 + \frac{a^{-1}\left(x_i^{(k)}\right)\left(a^{-1}\left(x_i^{(k)}\right)\right)'}{\varphi_{ij}^{(k)}}\right), \forall i,j$$

$$\delta_i^{(k)} = \delta_i^{(k)} + \gamma\nabla \log P\left(x_i^{(k)}\middle|\delta_i^{(k)}\right) = \delta_i^{(k)} - \gamma\frac{a^{-1}\left(x_i^{(k)}\right)\left(a^{-1}\left(x_i^{(k)}\right)\right)'}{\left(\varphi_{ij}^{(k)}\right)^2}, \forall i$$

Given epoch of size N is denoted as $\boldsymbol{D} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)},\ldots, \boldsymbol{x}^{(N)})$, the estimation equation of $\varphi_{ij}^{(k)}$ and $\delta_i^{(k)}$ is extended exactly as epoch estimation at every uth iteration of SGD with regard to log-likelihood gradient $\nabla\log P(x_i^{(k)} \mid \varphi_{ij}^{(k)}, \delta_i^{(k)})$:

$$\left(\varphi_{ij}^{(k)}\right)^{(u+1)} = \left(\varphi_{ij}^{(k)}\right)^{(u)} + \gamma\frac{1}{N}\nabla\sum_{v=1}^{N}\log P\left(\left(x_i^{(k)}\right)^{(v)}\middle|\left(\varphi_{ij}^{(k)}\right)^{(u)}\right)$$

$$= \left(\varphi_{ij}^{(k)}\right)^{(u)}$$

$$-\gamma\frac{1}{\left(\varphi_{ij}^{(k)}\right)^{(u)}}\left(1\right.$$

$$\left.+\frac{1}{N\left(\varphi_{ij}^{(k)}\right)^{(u)}}\sum_{v=1}^{N}a^{-1}\left(\left(x_i^{(k)}\right)^{(v)}\right)\left(a^{-1}\left(\left(x_i^{(k)}\right)^{(v)}\right)\right)'\right), \forall i,j$$

$$\left(\delta_i^{(k)}\right)^{(u+1)} = \left(\delta_i^{(k)}\right)^{(u)} + \gamma\frac{1}{N}\nabla\sum_{v=1}^{N}\log P\left(\left(x_i^{(k)}\right)^{(v)}\middle|\left(\delta_i^{(k)}\right)^{(u)}\right)$$

$$= \left(\delta_i^{(k)}\right)^{(u)} - \gamma\frac{1}{N\left(\left(\varphi_{ij}^{(k)}\right)^{(u)}\right)^2}\sum_{v=1}^{N}a^{-1}\left(\left(x_i^{(k)}\right)^{(v)}\right)\left(a^{-1}\left(\left(x_i^{(k)}\right)^{(v)}\right)\right)', \forall i$$

Where $(x_i^{(k)})^{(v)}$ is the ith element of $\boldsymbol{x}^{(v)}$ in epoch with regard to inversed generator f_k^{-1}. As a result, finite NL trained with SGD is specified as follows:

Initialize all $\varphi_{ij}^{(k)}$, $\delta_i^{(k)}$ and set $u = 0$.

Repeat

Sampling epoch $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)},\ldots, \boldsymbol{x}^{(N)})$ or receiving epoch $\boldsymbol{X}$ from big data / data stream.

$$\left(\varphi_{ij}^{(k)}\right)^{(u+1)} = \left(\varphi_{ij}^{(k)}\right)^{(u)} - \gamma \frac{1}{\left(\varphi_{ij}^{(k)}\right)^{(u)}}\left(1 + \frac{1}{N\left(\varphi_{ij}^{(k)}\right)^{(u)}}\sum_{v=1}^{N} a^{-1}\left(\left(x_i^{(k)}\right)^{(v)}\right)\left(a^{-1}\left(\left(x_i^{(k)}\right)^{(v)}\right)\right)'\right), \forall i,j$$

$$\left(\delta_i^{(k)}\right)^{(u+1)} = \left(\delta_i^{(k)}\right)^{(u)} - \gamma \frac{1}{N\left(\left(\varphi_{ij}^{(k)}\right)^{(u)}\right)^2}\sum_{v=1}^{N} a^{-1}\left(\left(x_i^{(k)}\right)^{(v)}\right)\left(a^{-1}\left(\left(x_i^{(k)}\right)^{(v)}\right)\right)', \forall i$$

Increase $u = u + 1$.

Until some terminating conditions are met.

Note, a terminating condition is customized, for example, parameters $\varphi_{ij}^{(k)}$ and $\delta_i^{(k)}$ are not changed significantly or there is no more coming epoch $\boldsymbol{X}$. Moreover, the index u indicates time point as well as iteration of SGD. After finite NL is trained, it can generate new data $\boldsymbol{x}'$ by generator $g(\mathbf{z} \mid \Phi) = \boldsymbol{x}'$ with any $\mathbf{z}$ randomized from standard normal distribution with mean $\mathbf{0}$ and variance 1.

Because it is not easy to calculate inversed gradient $\nabla_x g^{-1}(\boldsymbol{x} \mid \Phi)$ as well as its determinant $|\nabla_x g^{-1}(\boldsymbol{x} \mid \Phi)|$ according to finite NL except the decomposition technique above of entire matrix parameter Φ into partial vector parameters $\varphi_i^{(k)}$, there is technique called real RVP (Ruthotto & Haber, 2021, p. 9) which defines each layer or partial generator $f_k(\mathbf{z}^{(k)})$ by special way where $\mathbf{z}^{(k)}$ is split into two parts such as $\mathbf{z}_1^{(k)}$ and $\mathbf{z}_2^{(k)}$ so that:

$$f_k\left(\boldsymbol{z}^{(k)}\right) = f_k\begin{pmatrix}\boldsymbol{z}_1^{(k)} \\ \boldsymbol{z}_2^{(k)}\end{pmatrix} = \begin{pmatrix}\boldsymbol{z}_1^{(k)} \\ \boldsymbol{z}_2^{(k)} \otimes \exp\left(\boldsymbol{s}_k\left(\boldsymbol{z}_1^{(k)}\right)\right) + \boldsymbol{t}_k\left(\boldsymbol{z}_1^{(k)}\right)\end{pmatrix} \tag{2.8}$$

Of course, we have:

$$f_k\left(\boldsymbol{z}^{(k)}\right) = f_k\begin{pmatrix}\boldsymbol{z}_1^{(k)} \\ \boldsymbol{z}_2^{(k)}\end{pmatrix} = \begin{pmatrix}\boldsymbol{z}_1^{(k)} \\ \boldsymbol{z}_2^{(k)} \otimes \exp\left(\boldsymbol{s}_k\left(\boldsymbol{z}_1^{(k)}\right)\right) + \boldsymbol{t}_k\left(\boldsymbol{z}_1^{(k)}\right)\end{pmatrix} = \begin{pmatrix}\boldsymbol{z}_1^{(k+1)} \\ \boldsymbol{z}_2^{(k+1)}\end{pmatrix} = \boldsymbol{z}^{(k+1)}$$

Where $\boldsymbol{s}_k$ and $\boldsymbol{t}_k$ are two neural networks for scaling and translation, whose inputs and outputs have the same dimension. The operator $\otimes$ denotes component-wise multiplication of two vectors where every pair of two corresponding elements of the two vectors are multiplied together, for instance, given two arbitrary vectors $\boldsymbol{u} = (u_1, u_2,\ldots, u_n)^T$ and $\boldsymbol{v} = (v_1, v_2,\ldots, v_n)^T$, we have $\boldsymbol{u}\otimes\boldsymbol{v} = (u_1v_1, u_2v_2,\ldots, u_nv_n)^T$. Moreover, the exponential function exp(.) above whose input is vector produces a vector by taking exponential function over every element of input vector. Inversed generator f_k^{-1} is specified from generator f_k.

$$f_k^{-1}\left(\boldsymbol{x}^{(k)}\right) = f_k^{-1}\begin{pmatrix}\boldsymbol{x}_1^{(k)} \\ \boldsymbol{x}_2^{(k)}\end{pmatrix} = \begin{pmatrix}\boldsymbol{x}_1^{(k)} \\ \left(\boldsymbol{x}_2^{(k)} - \boldsymbol{t}_k\left(\boldsymbol{x}_1^{(k)}\right)\right) \otimes \exp\left(-\boldsymbol{s}_k\left(\boldsymbol{x}_1^{(k)}\right)\right)\end{pmatrix} \tag{2.9}$$

Of course, we have:

$$f_k^{-1}\left(\boldsymbol{x}^{(k)}\right) = f_k^{-1}\begin{pmatrix}\boldsymbol{x}_1^{(k)} \\ \boldsymbol{x}_2^{(k)}\end{pmatrix} = \begin{pmatrix}\boldsymbol{x}_1^{(k)} \\ \left(\boldsymbol{x}_2^{(k)} - \boldsymbol{t}_k\left(\boldsymbol{x}_1^{(k)}\right)\right) \otimes \exp\left(-\boldsymbol{s}_k\left(\boldsymbol{x}_1^{(k)}\right)\right)\end{pmatrix} = \begin{pmatrix}\boldsymbol{x}_1^{(k-1)} \\ \boldsymbol{x}_2^{(k-1)}\end{pmatrix} = \boldsymbol{x}^{(k-1)}$$

Inversed gradient $\nabla_{\boldsymbol{x}^{(k)}} f_k^{-1}(\boldsymbol{x}^{(k)})$ is the 2x2 Jacobian matrix determined as follows:

$$\nabla_{\boldsymbol{x}^{(k)}} f_k^{-1}(\boldsymbol{x}^{(k)}) = \begin{pmatrix} 1 & 0 \\ \dfrac{d\left(\boldsymbol{x}_2^{(k)} - \boldsymbol{t}_k\left(\boldsymbol{x}_1^{(k)}\right)\right) \otimes \exp\left(-\boldsymbol{s}_k\left(\boldsymbol{x}_1^{(k)}\right)\right)}{d\boldsymbol{x}_1^{(k)}} & \exp\left(-\boldsymbol{s}_k\left(\boldsymbol{x}_1^{(k)}\right)\right) \end{pmatrix}$$

It is interesting that taking determinant of inversed gradient $\nabla_x f_k^{-1}(\boldsymbol{x}^{(k)})$ becomes simple:

$$\left|\nabla_{\boldsymbol{x}^{(k)}} f_k^{-1}(\boldsymbol{x}^{(k)})\right| = \exp\left(-\boldsymbol{s}_k\left(\boldsymbol{x}_1^{(k)}\right)\right) \tag{2.10}$$

When this determinant is determined, it is possible to maximize log-likelihood log$P(\boldsymbol{x} \mid \Phi)$ to estimate Φ where Φ here are weights of scaling neural network $\boldsymbol{s}_k$ and translation neural network $\boldsymbol{t}_k$. Log-likelihood log$P(\boldsymbol{x} \mid \Phi)$ is written:

$$\begin{aligned}\log P(\boldsymbol{x}|\Phi) &= -\frac{n}{2}\log 2\pi - \frac{1}{2}\|g^{-1}(\boldsymbol{x}|\Phi)\|^2 + \log|\nabla_{\boldsymbol{x}} g^{-1}(\boldsymbol{x}|\Phi)| \\ &= \frac{n}{2}\log 2\pi - \frac{1}{2}\|g^{-1}(\boldsymbol{x}|\Phi)\|^2 + \sum_{k=1}^{K}\log\left|\nabla_{\boldsymbol{x}^{(k)}} f_k^{-1}(\boldsymbol{x}^{(k)})\right| \\ &= \frac{n}{2}\log 2\pi - \frac{1}{2}\|g^{-1}(\boldsymbol{x}|\Phi)\|^2 - \sum_{k=1}^{K}\boldsymbol{s}_k\left(\boldsymbol{x}_1^{(k)}\right)\end{aligned}$$

Because parameter Φ is now only weights of scaling neural network $\boldsymbol{s}_k$ and translation neural network $\boldsymbol{t}_k$, maximizing log-likelihood log$P(\boldsymbol{x} \mid \Phi)$ is now to optimize (train) $\boldsymbol{s}_k$ and $\boldsymbol{t}_k$ by some algorithms like backpropagation algorithm.

$$\left(\Phi_{\boldsymbol{s}}^{(k)}\right)^* = \underset{\Phi_s^{(k)}}{\text{optimize}}\, \boldsymbol{s}_k\left(\boldsymbol{x}_1^{(k)} \middle| \Phi_{\boldsymbol{s}}^{(k)}\right)$$

$$\left(\Phi_{\boldsymbol{t}}^{(k)}\right)^* = \underset{\Phi_t^{(k)}}{\text{optimize}}\, \boldsymbol{t}_k\left(\boldsymbol{x}_1^{(k)} \middle| \Phi_{\boldsymbol{t}}^{(k)}\right)$$

Beside finite NL there is another NL technique called continuous NL but it is not mentioned here because continuous NL is relevant to hazard problem of differential equation which is not main subject of DNN.

Recall that there are three main approaches for constructing DGM such as tractable density DGM, approximate density DGM, and implicit density DGM. However, if data is image, there is another categorization way that there are two main approaches: 1) pixel density approach tries to model pixel distribution and 2) block density approach tries to model entire image distribution as any data distribution. In other words, likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is defined based on probabilistic distribution of pixels where $\boldsymbol{x}$ is considered as set of pixels according to pixel density approach. On the other hand, block density approach considers likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is PDF of a block or entire image (unified big block) where $\boldsymbol{x}$ is considered as any arbitrary data. For instance, NL belongs to both tractable density DGM and block density approach. It is interesting that pixel density approach also belongs to tractable density approach because its PDF is defined obviously. Moreover, pixel density approach merges the two problems of training generator $g(\mathbf{z} \mid \Phi)$ and qualifying such training task into the first problem which is to train $g(\mathbf{z} \mid \Phi)$ by learning sample PDF $P(\boldsymbol{x})$ because $P(\boldsymbol{x})$ or $P(\boldsymbol{x} \mid \Phi)$ now replaces $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$.

Shortly, **pixel density** (**PD**) approach defines $P(\boldsymbol{x})$ as product of all pixel distribution. Concretely, let $\boldsymbol{x} = (x_1, x_2,\ldots, x_{n^2})^T$ denote an image whose every ith pixel is represented by elemental variable x_i and $P(\boldsymbol{x})$ called image PDF is defined according to joint probability rule as follows:

$$P(\boldsymbol{x}) = \prod_{i=1}^{n^2} P(x_i|x_{i-1}, x_{i-2}, \ldots, x_1) \tag{2.11}$$

Where n implies image width with suppose that image width and image height are equal for convenience, and

$$P(x_1|x_0) = P(x_1)$$

In other words, image PDF $P(\boldsymbol{x})$ is product of all conditional PDFs $P(x_i \mid x_{i-1}, x_{i-2},\ldots, x_1)$ where every $P(x_i \mid x_{i-1}, x_{i-2},\ldots, x_1)$ is called conditional pixel PDF or pixel PDF in short. There is neither tractable data **z** nor explicit generator $g(\mathbf{z} \mid \Phi)$ for generating new data in PD because generation task is based on the entire PDF $P(\boldsymbol{x})$. For instance, without loss of generality, if we randomize k first pixels $x_1, x_2,\ldots, x_k$, we can generate $n^2–k$ remaining pixels by the recurrent process: determining $P(x_{k+1} \mid x_k, x_{k-1},\ldots, x_1)$ based on x_1 to x_k, generating x_{k+1} according to $P(x_{k+1} \mid x_k, x_{k-1},\ldots, x_1)$ and determining $P(x_{k+2} \mid x_{k+1}, x_k,\ldots, x_1)$ based on x_1 to x_{k+1}, generating x_{k+2} according to $P(x_{k+2} \mid x_{k+1}, x_k,\ldots, x_1)$ and determining $P(x_{k+3} \mid x_{k+2}, x_{k+1},\ldots, x_1)$ based on x_1 to $x_{k+2},\ldots$, generating x_{n^2-1} according to $P(x_{n^2-1} \mid x_{n^2-2}, x_{n^2-3},\ldots, x_1)$ and determining $P(x_{n^2} \mid x_{n^2-1}, x_{n^2-2},\ldots, x_1)$ based on x_1 to x_{n^2-1}, generating the last x_{n^2} according to $P(x_{n^2} \mid x_{n^2-1}, x_{n^2-2},\ldots, x_1)$. By another viewpoint, the joint probability of $n^2–k$ remaining pixels denoted $P(x_k, x_{k+1},\ldots, x_{n^2})$ is determined and then, $n^2–k$ remaining pixels are generated according to this joint probability. Indeed, the joint probability $P(x_k, x_{k+1},\ldots, x_{n^2})$ of $n^2–k$ remaining pixels is totally determined when $P(\boldsymbol{x})$ and k probabilities $P(x_i \mid x_{i-1}, x_{i-2},\ldots, x_1)$ are determined where i is from 1 to k.

$$P(x_{k+1}, x_{k+2}, \ldots, x_{n^2}) = \prod_{i=k+1}^{n^2} P(x_i|x_{i-1}, x_{i-2}, \ldots, x_1) = \frac{P(\boldsymbol{x})}{\prod_{i=1}^{k} P(x_i|x_{i-1}, x_{i-2}, \ldots, x_1)}$$

Because there are a large number of pixels in a large image which produces a large number of pixel PDFs as well as every pixel PDF $P(x_i \mid x_{i-1}, x_{i-2},\ldots, x_1)$ of a given pixel x_i is itself also complicated with a lot of its previous pixels $x_{i-1}, x_{i-2},\ldots, x_1$, there are many techniques proposed to PD in order to decrease complexity and increase computation effectiveness. Anyhow, the equation of image PDF $P(\boldsymbol{x})$ above is important one in theory. One of PD techniques is to apply long short-term memory (LSTM) (Theis & Bethge, 2015) into modeling and learning sample PDF $P(\boldsymbol{x})$.

The default artificial neural network is feedforward neural network where data is fed to input layer which, in turn, is evaluated and passed across hidden layers to output layer in one-way direction, finally. However, there is an extension of neural network, which is called recurrent neural work (RNN), where an output can be turned back in order to feed on network as input. In other words, RNN has circle, which allow that output can become input. For convenience and easy explanation, given T time points $t = 1, 2,\ldots, T$, current state of a RNN at time point t is represented by three layers such as input layer $\boldsymbol{x}_t$, hidden layer $\boldsymbol{h}_t$, and output layer $\boldsymbol{o}_t$ without loss of generality with note that $\boldsymbol{h}_t$ can represent many hidden layers when RNN is a DNN too. Obviously, RNN is an extension of neural network because every triple $(\boldsymbol{x}_t, \boldsymbol{h}_t, \boldsymbol{o}_t)$ is, essentially, a feedforward neural network, even a DNN. Hidden layer $\boldsymbol{h}_t$ as well as output layer $\boldsymbol{o}_t$ at current state t is calculated based on both current input layer $\boldsymbol{x}_t$ and previous hidden layer $\boldsymbol{h}_{t-1}$ of previous state at time point $t–1$. Without loss of generality, input layer, hidden layer, and output layer are considered as input neuron, hidden neuron, and output neuron for convenience (Wikipedia, Recurrent neural network, 2005).

$$\begin{aligned} \boldsymbol{h}_t &= \sigma_{\boldsymbol{h}}(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{h}}) \\ \boldsymbol{o}_t &= \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{b}_{\boldsymbol{o}}) \end{aligned} \quad (2.12)$$

Where $\boldsymbol{W}_h$ is weight matrix of current hidden neuron $\boldsymbol{h}_t$ regarding current input neuron $\boldsymbol{x}_t$, $\boldsymbol{U}_h$ is weight matrix of current hidden neuron $\boldsymbol{h}_t$ regarding previous hidden neuron $\boldsymbol{h}_{t-1}$, and $\boldsymbol{b}_h$ is bias vector of current hidden neuron $\boldsymbol{h}_t$ whereas $\boldsymbol{W}_o$ is weight matrix of current output neuron $\boldsymbol{o}_t$ regarding current hidden neuron $\boldsymbol{h}_t$ and $\boldsymbol{b}_o$ is bias vector of current output neuron $\boldsymbol{o}_t$. Moreover, $\sigma_h(.)$ and $\sigma_o(.)$ are activation functions of $\boldsymbol{h}_t$ and $\boldsymbol{o}_t$, respectively with note that $\sigma_h(.)$ and $\sigma_o(.)$ are vector-by-vector functions. Backpropagation algorithm can be applied into learning RNN as usual. It is interesting that structure of RNN defined by the triple $(\boldsymbol{x}_t, \boldsymbol{h}_t, \boldsymbol{o}_t)$ is not changed but its parameters $\boldsymbol{W}_h$, $\boldsymbol{U}_h$, $\boldsymbol{b}_h$, $\boldsymbol{W}_o$, and $\boldsymbol{b}_o$ are changed by backpropagation algorithm when RNN is learned. Of course, values of the triple $(\boldsymbol{x}_t, \boldsymbol{h}_t, \boldsymbol{o}_t)$ are changed over time points. Note, $\boldsymbol{W}_h$, $\boldsymbol{U}_h$, and $\boldsymbol{W}_o$ are matrix parameters and $\boldsymbol{b}_h$ and $\boldsymbol{b}_o$ are vector parameters whereas $\boldsymbol{x}_t$, $\boldsymbol{h}_t$, and $\boldsymbol{o}_t$ are vector variables.

Long short-term memory (LSTM) is an extension of RNN, which implies that RNN is used to implement short-term memory so that the short-term memory can last for a longer time through T time points $t = 1, 2,\ldots, T$ built in RNN. Consequently, the short-term memory is represented by a so-called cell associated with three gates such as input gate, forget gate, and output gate. Cell represents information piece stored in memory at current time (Wikipedia, 2007). Input gate controls which new information to be put to cell, forget gate decides which information to be discarded, and output gate controls which information to be sent to next state (Wikipedia, 2007). As a convention, the cell at current state t is represented by the pair $(\boldsymbol{c}_t, \boldsymbol{h}_t)$ whereas input gate, forget gate, and output gate are represented by vector variables $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, and $\boldsymbol{o}_t$, respectively. Note, let $\boldsymbol{g}_t$ and $\boldsymbol{c}_t$ be cell input activation variable and cell state variable where cell input activation variable $\boldsymbol{g}_t$ represents the activated input part of a cell, which is the important input part being different from the forgotten part, whereas cell state variable $\boldsymbol{c}_t$ represents real information stored in cell which is, exactly, the short memory at current state. In literature, $\boldsymbol{g}_t$ is also called cell gate. Some LSTM variants merge $\boldsymbol{g}_t$ and $\boldsymbol{c}_t$ into the same cell state variable. Although output gate $\boldsymbol{o}_t$ represents which information to be sent to next state, it is consolidated with current cell memory $\boldsymbol{c}_t$ in order to produce the real output information $\boldsymbol{h}_t$ which represents bright and clear-cut memory. In other words, given cell $(\boldsymbol{c}_t, \boldsymbol{h}_t)$, then $\boldsymbol{c}_t$ represents the real information stored in memory and $\boldsymbol{h}_t$ represents the clear-cut memory which displays brightly at the outside for next state. It is possible to consider that $\boldsymbol{c}_t$ is evaluated value of cell t and $\boldsymbol{h}_t$ is predictive value of cell t. Following equations specify LSTM based on specification of RNN (Wikipedia, 2007), which indicates how to calculate cell and gates.

$$\begin{aligned}
\boldsymbol{i}_t &= \sigma_{\boldsymbol{i}}(\boldsymbol{W}_{\boldsymbol{i}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{i}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{i}}) \\
\boldsymbol{f}_t &= \sigma_{\boldsymbol{f}}\left(\boldsymbol{W}_{\boldsymbol{f}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{f}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{f}}\right) \\
\boldsymbol{o}_t &= \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{o}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{o}}) \\
\boldsymbol{g}_t &= \sigma_{\boldsymbol{g}}\left(\boldsymbol{W}_{\boldsymbol{g}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{g}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{g}}\right) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \otimes \boldsymbol{g}_t \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \otimes \sigma_{\boldsymbol{h}}(\boldsymbol{c}_t)
\end{aligned} \tag{2.13}$$

Note, weight matrix $\boldsymbol{W}_i$, weight matrix $\boldsymbol{U}_i$, and bias vector $\boldsymbol{b}_i$ are parameters of input gate $\boldsymbol{i}_t$. Weight matrix $\boldsymbol{W}_f$, weight matrix $\boldsymbol{U}_f$, and bias vector $\boldsymbol{b}_f$ are parameters of forget gate $\boldsymbol{f}_t$. Weight matrix $\boldsymbol{W}_o$, weight matrix $\boldsymbol{U}_o$, and bias vector $\boldsymbol{b}_o$ are parameters of output gate $\boldsymbol{o}_t$. Weight matrix $\boldsymbol{W}_g$, weight matrix $\boldsymbol{U}_g$, and bias vector $\boldsymbol{b}_g$ are parameters of cell gate $\boldsymbol{g}_t$. Vector variables $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, and $\boldsymbol{o}_t$ are often in range $[\boldsymbol{0}, \boldsymbol{1}]$ whereas vector variables $\boldsymbol{c}_t$ and $\boldsymbol{h}_t$ are often in range $[\boldsymbol{-1}, \boldsymbol{1}]$. Activation functions $\sigma_i(.)$, $\sigma_f(.)$, and $\sigma_o(.)$ are often sigmoid (logistic) functions whereas activation functions $\sigma_g(.)$ and $\sigma_h(.)$ are hyperbolic tangent functions. The operator $\otimes$ denotes component-wise multiplication of two vectors where every pair of two corresponding elements of the two vectors are multiplied together, for instance, given two arbitrary vectors $\boldsymbol{u} = (u_1, u_2,\ldots, u_n)^T$ and $\boldsymbol{v} = (v_1, v_2,\ldots, v_n)^T$, we have $\boldsymbol{u}\otimes\boldsymbol{v} = (u_1v_1, u_2v_2,\ldots, u_nv_n)^T$. Note, backpropagation algorithm can be applied into learning LSTM as usual.

By applying LSTM into pixel density (PD) approach for modeling DGM, each pixel x_i is represented by cell $\boldsymbol{c}_i$ when pixel index i is considered as time point t. Because each cell $\boldsymbol{c}_i$ is dependent on its one right previous cell $\boldsymbol{c}_{i-1}$ whereas conditional pixel PDF $P(\boldsymbol{x}_i \mid \boldsymbol{x}_{i-1}, \boldsymbol{x}_{i-2},\ldots, \boldsymbol{x}_1)$ of pixel $\boldsymbol{x}_i$ is dependent on $i-1$ previous pixels $\boldsymbol{x}_{i-1}, \boldsymbol{x}_{i-2},\ldots, \boldsymbol{x}_1$, Markov property is applied so that conditional pixel PDF of pixel $\boldsymbol{x}_i$ depends on only one previous pixel $\boldsymbol{x}_{i-1}$.

$$P(\boldsymbol{x}_i|\boldsymbol{x}_{i-1}, \boldsymbol{x}_{i-2}, \ldots, \boldsymbol{x}_1) = P(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$$

It is now possible to apply LSTM to model PD by matching each pixel $\boldsymbol{x}_i$ with each cell $\boldsymbol{c}_i$ so that cell $\boldsymbol{c}_i$ is considered as evaluated value of pixel $\boldsymbol{x}_i$ as well as each $\boldsymbol{h}_i$ is predictive value of pixel $\boldsymbol{x}_i$. Because image is two-dimension array, each pixel $\boldsymbol{x}_{ij}$ or each cell $\boldsymbol{c}_{ij}$ is indexed by two indices i and j following image height and image width. The event that cell $\boldsymbol{c}_{ij}$ or $\boldsymbol{c}_{i,j}$ indexed by two indices i and j makes LSTM extended into two-dimension LSTM as follows:

$$\begin{aligned}
\boldsymbol{i}_{i,j} &= \sigma_{\boldsymbol{i}}\left(\boldsymbol{W}_{\boldsymbol{i}}\boldsymbol{x}_{i,j} + \boldsymbol{U}_{\boldsymbol{i}}\boldsymbol{h}_{i,j-1} + \boldsymbol{V}_{\boldsymbol{i}}\boldsymbol{h}_{i-1,j} + \boldsymbol{b}_{\boldsymbol{i}}\right)\\
\boldsymbol{f}_{i,j} &= \sigma_{\boldsymbol{f}}\left(\boldsymbol{W}_{\boldsymbol{f}}\boldsymbol{x}_{i,j} + \boldsymbol{U}_{\boldsymbol{f}}\boldsymbol{h}_{i,j-1} + \boldsymbol{V}_{\boldsymbol{f}}\boldsymbol{h}_{i-1,j} + \boldsymbol{b}_{\boldsymbol{f}}\right)\\
\boldsymbol{o}_{i,j} &= \sigma_{\boldsymbol{o}}\left(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{x}_{i,j} + \boldsymbol{U}_{\boldsymbol{o}}\boldsymbol{h}_{i,j-1} + \boldsymbol{V}_{\boldsymbol{o}}\boldsymbol{h}_{i-1,j} + \boldsymbol{b}_{\boldsymbol{o}}\right)\\
\boldsymbol{g}_{i,j} &= \sigma_{\boldsymbol{g}}\left(\boldsymbol{W}_{\boldsymbol{g}}\boldsymbol{x}_{i,j} + \boldsymbol{U}_{\boldsymbol{g}}\boldsymbol{h}_{i,j-1} + \boldsymbol{V}_{\boldsymbol{g}}\boldsymbol{h}_{i-1,j} + \boldsymbol{b}_{\boldsymbol{g}}\right)\\
\boldsymbol{c}_{i,j} &= \boldsymbol{f}_{i,j}\otimes\left(\boldsymbol{c}_{i,j-1} + \boldsymbol{c}_{i-1,j}\right) + \boldsymbol{i}_{i,j}\otimes\boldsymbol{g}_{i,j}\\
\boldsymbol{h}_{i,j} &= \boldsymbol{o}_{i,j}\otimes\sigma_{\boldsymbol{h}}\left(\boldsymbol{c}_{i,j}\right)
\end{aligned} \tag{2.14}$$

The equations above specify core ideology of PD associated with two-dimension LSTM where the contextual meaning of weight and bias parameters $\boldsymbol{W}_{(.)}$, $\boldsymbol{U}_{(.)}$, $\boldsymbol{V}_{(.)}$, and $\boldsymbol{b}_{(.)}$ is not changed with note that $\boldsymbol{W}_{(.)}$, $\boldsymbol{U}_{(.)}$, and $\boldsymbol{V}_{(.)}$ are weight matrices regarding current pixel, previous pixel (i, j–1), and previous pixel (i–1, j), respectively. In literature, such PD is called *PixelRNN* associated with diagonal two-dimension LSTM (Oord, Kalchbrenner, & Kavukcuoglu, 2016, pp. 3-4). According to diagonal two-dimension LSTM each pixel (i, j) at ith row and jth column has two previous neighbors such as previous left pixel (i, j–1) and previous upper pixel (i–1, j). For extension, each pixel (i, j) can have up four previous neighbors such as pixel (i, j–1), pixel (i–1, j–1), pixel (i–1, j), and pixel (i–1, j+1). Following figure depicts PixelRNN with diagonal two-dimension LSTM (Oord, Kalchbrenner, & Kavukcuoglu, 2016, p. 4).

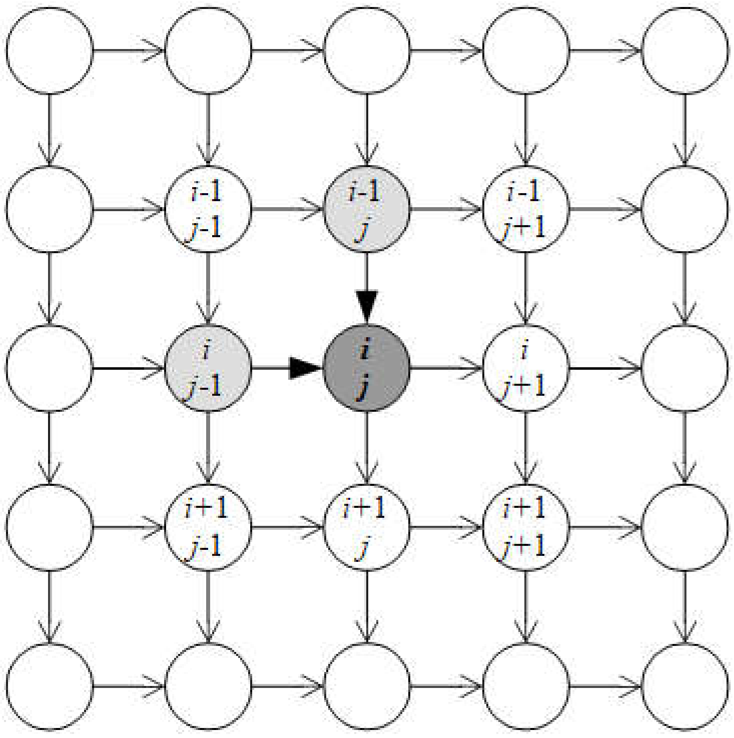

Figure 2. 1. PixelRNN with diagonal two-dimension LSTM.

It is easy to add more weight parameters to these extensive cases. For example, cell gates and cell state with regard to the four previous neighbors are specified as follows:

$$\boldsymbol{i}_{i,j} = \sigma_{\boldsymbol{i}}\left(\boldsymbol{W}_{\boldsymbol{i}}\boldsymbol{x}_{i,j} + \boldsymbol{U}_{\boldsymbol{i}}\boldsymbol{h}_{i,j-1} + \boldsymbol{V}_{\boldsymbol{i}}\boldsymbol{h}_{i-1,j} + \boldsymbol{R}_{\boldsymbol{i}}\boldsymbol{h}_{i-1,j-1} + \boldsymbol{S}_{\boldsymbol{i}}\boldsymbol{h}_{i-1,j+1} + \boldsymbol{b}_{\boldsymbol{i}}\right)$$

$$\boldsymbol{f}_{i,j} = \sigma_{\boldsymbol{f}}\left(\boldsymbol{W}_{\boldsymbol{f}}\boldsymbol{x}_{i,j} + \boldsymbol{U}_{\boldsymbol{f}}\boldsymbol{h}_{i,j-1} + \boldsymbol{V}_{\boldsymbol{f}}\boldsymbol{h}_{i-1,j} + \boldsymbol{R}_{\boldsymbol{f}}\boldsymbol{h}_{i-1,j-1} + \boldsymbol{S}_{\boldsymbol{f}}\boldsymbol{h}_{i-1,j+1} + \boldsymbol{b}_{\boldsymbol{f}}\right)$$

$$\boldsymbol{o}_{i,j} = \sigma_{\boldsymbol{o}}\left(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{x}_{i,j} + \boldsymbol{U}_{\boldsymbol{o}}\boldsymbol{h}_{i,j-1} + \boldsymbol{V}_{\boldsymbol{o}}\boldsymbol{h}_{i-1,j} + \boldsymbol{R}_{\boldsymbol{o}}\boldsymbol{h}_{i-1,j-1} + \boldsymbol{S}_{\boldsymbol{o}}\boldsymbol{h}_{i-1,j+1} + \boldsymbol{b}_{\boldsymbol{i}}\right)$$

$$\boldsymbol{g}_{i,j} = \sigma_{\boldsymbol{g}}\left(\boldsymbol{W}_{\boldsymbol{g}}\boldsymbol{x}_{i,j} + \boldsymbol{U}_{\boldsymbol{g}}\boldsymbol{h}_{i,j-1} + \boldsymbol{V}_{\boldsymbol{g}}\boldsymbol{h}_{i-1,j} + \boldsymbol{R}_{\boldsymbol{g}}\boldsymbol{h}_{i-1,j-1} + \boldsymbol{S}_{\boldsymbol{g}}\boldsymbol{h}_{i-1,j+1} + \boldsymbol{b}_{\boldsymbol{g}}\right)$$

$$\boldsymbol{c}_{i,j} = \boldsymbol{f}_{i,j} \otimes \left(\boldsymbol{c}_{i,j-1} + \boldsymbol{c}_{i-1,j} + \boldsymbol{c}_{i-1,j-1} + \boldsymbol{c}_{i-1,j+1}\right) + \boldsymbol{i}_{i,j} \otimes \boldsymbol{g}_{i,j}$$

$$\boldsymbol{h}_{i,j} = \boldsymbol{o}_{i,j} \otimes \sigma_{\boldsymbol{h}}\left(\boldsymbol{c}_{i,j}\right)$$

Where matrices $\boldsymbol{R}_{(.)}$ and $\boldsymbol{S}_{(.)}$ are additional weight parameters regarding two new neighbor pixels such as pixel (i–1, j–1) and pixel (i–1, j+1).

Recall that $\boldsymbol{c}_{i,j}$ is considered as evaluated value of pixel $\boldsymbol{x}_{i,j}$ and $\boldsymbol{h}_{i,j}$ is predictive value of pixel $\boldsymbol{x}_{i,j}$. It is interesting that $\boldsymbol{h}_{i,j}$ is generated pixel within the aforementioned generation process by PD. Turning back the generation process, without loss of generality, given k randomized pixels $\boldsymbol{x}_{i-1,1}$, $\boldsymbol{x}_{i-1,2}$,.., $\boldsymbol{x}_{i-1,j+1}$,…, $\boldsymbol{x}_{i,1}$, $\boldsymbol{x}_{i,2}$,…, $\boldsymbol{x}_{i,j}$, we will generate the next pixel $\boldsymbol{x}_{i,j+1}$. Firstly, PD model must be trained by some dataset as a set of images. Secondly, k randomized pixels $\boldsymbol{x}_{i-1,1}$, $\boldsymbol{x}_{i-1,2}$,.., $\boldsymbol{x}_{i-1,j+1}$,…, $\boldsymbol{x}_{i,1}$, $\boldsymbol{x}_{i,2}$,…, $\boldsymbol{x}_{i,j}$ are fed to PD again so as to update k sets of parameters $\boldsymbol{W}_{(.)}$, $\boldsymbol{U}_{(.)}$, and $\boldsymbol{b}_{(.)}$ as well as compute k predictive values $\boldsymbol{h}_{i-1,1}$, $\boldsymbol{h}_{i-1,2}$,.., $\boldsymbol{h}_{i-1,j+1}$,…, $\boldsymbol{h}_{i,1}$, $\boldsymbol{h}_{i,2}$,…, $\boldsymbol{h}_{i,j}$. Finally, it is possible to determine the predictive value $\boldsymbol{h}_{i,j+1}$ of the next pixel (i, j+1) given $\boldsymbol{x}_{i,j+1}$, $\boldsymbol{h}_{i,j}$, and $\boldsymbol{h}_{i-1,j+1}$ along with learned parameters of two-dimension LSTM PD. It is important to note that $\boldsymbol{x}_{i,j+1}$ is randomized arbitrarily whereas $\boldsymbol{h}_{i,j}$ and $\boldsymbol{h}_{i-1,j+1}$ were computed previously. Obviously, it is easy to generate next predictive values $\boldsymbol{h}_{i,j+2}$, $\boldsymbol{h}_{i,j+2}$,…, $\boldsymbol{h}_{i+1,j}$, $\boldsymbol{h}_{i+1,j+1}$, etc. by the similar process. Note, backpropagation algorithm can be applied into learning two-dimension LSTM as usual. Note, backpropagation algorithm is often associated with stochastic gradient descent (SGD) method and so, please pay attention to SGD.

3. Approximate Density DGM

According to approximate density approach, DGMs try to estimate approximately generator PDF $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ or derive other PDF that is similar to $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ with note that PDF is abbreviation of probability density function.

Recall that there are two problems related to construct a DGM: 1) how to define likelihood or error to train generator DNN $g(\mathbf{z} \mid \Phi)$ and 2) how to define tractable PDF $P(\mathbf{z})$ which implies the way to randomize $\mathbf{z}$. The second problem relates to assert qualification of random data $\mathbf{z}'$ and hence, the second problem is stated as qualification problem of how to qualify random data. According to implicit density approach, a discrimination DNN is used to qualify randomized data $\mathbf{z}$ instead of defining tractable PDF $P(\mathbf{z})$ by Generative Adversarial Network (GAN) which is a typical method belonging to implicit density approach. In different way belonging to this approximate density approach, **Variational Autoencoders** (**VAE**) method developed by Kingma and Welling (Kingma & Welling, 2022) proposed another DNN called encoder $f(\boldsymbol{x} \mid \Theta)$ to expectedly convert intractable data $\boldsymbol{x}$ into tractable data $\mathbf{z}$. In other words, encoder $f(\boldsymbol{x} \mid \Theta)$ approximates tractable data $\mathbf{z}$ by encoded data $\mathbf{z}'$.

$$f(\boldsymbol{x}|\Theta) = \mathbf{z}' \cong \mathbf{z} \quad (3.1)$$

It is easy to recognize that encoder $f(\boldsymbol{x} \mid \Theta)$ is an approximation of the inverse of generator $g(\mathbf{z} \mid \Phi)$ when $g(\mathbf{z} \mid \Phi)$ is invertible where $\boldsymbol{x}$-dimension m is larger than $\mathbf{z}$-dimension n ($m > n$), which is the reason that generator $g(\mathbf{z} \mid \Phi)$ is called decoder $g(\mathbf{z} \mid \Phi)$ in VAE. Like decoder $g(\mathbf{z} \mid \Phi)$, encoder $f(\boldsymbol{x} \mid \Theta)$ is modeled by a so-called encoder DNN whose weights are parameter Θ called encoder parameter and so parameter Φ is called decoder parameter in VAE. By following the fact that encoder $f(\boldsymbol{x} \mid \Theta)$ approximates tractable data $\mathbf{z}$ by encoded data $\mathbf{z}'$, tractable PDF $P(\mathbf{z})$ is approximated by a so-called encoder PDF $P_f(\mathbf{z}')$.

$$P_f(\mathbf{z}') \cong P(\mathbf{z})$$

Because encoder $f(\boldsymbol{x} \mid \Theta)$ depends on its parameter Θ, we can denote:

$$P_f(\mathbf{z}') = P(\mathbf{z}'|\Theta, \boldsymbol{x})$$

Essential, encoder PDF $P(\mathbf{z}' \mid \Theta, \boldsymbol{x})$ is likelihood function of $\mathbf{z}'$ given $\boldsymbol{x}$ which is conditional PDF of $\mathbf{z}'$ given $\boldsymbol{x}$ and hence, $P(\mathbf{z}' \mid \Theta, \boldsymbol{x})$ is called encoder likelihood which depends on encoder $f(\boldsymbol{x} \mid \Theta)$, of course. On the other hand, $P(\mathbf{z}' \mid \Theta, \boldsymbol{x})$ is posterior PDF of tractable data given tractable data $\boldsymbol{x}$ where $P(\mathbf{z})$ is prior PDF of tractable data. In practice, $\mathbf{z}'$ is assumed to conform multivariate normal distribution and therefore, let $\mu(x)$ and $\Sigma(x)$ be mean vector and covariance matrix of $\mathbf{z}'$. Encoder likelihood $P(\mathbf{z}' \mid \Theta, x)$ becomes $P(\mathbf{z}' \mid \Theta, \mu(x \mid \Theta), \Sigma(x \mid \Theta))$ so that output of encoder DNN $f(x \mid \Theta)$ is mean $\mu(x \mid \Theta)$ and covariance matrix $\Sigma(x \mid \Theta)$ while its input is x and its weights are Θ, of course.

$$f(\boldsymbol{x}|\Theta) = \begin{pmatrix} \mu(\boldsymbol{x}|\Theta) \\ \mu(\boldsymbol{x}|\Theta) \end{pmatrix}$$
$$P(\mathbf{z}'|\Theta, \boldsymbol{x}) = P\big(\mathbf{z}'\big|\Theta, \mu(\boldsymbol{x}), \Sigma(\boldsymbol{x})\big) = \mathcal{N}\big(\mathbf{z}\big|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\big) \tag{3.2}$$

Note, $\mathcal{N}(.)$ denotes normal distribution and thus, $\mathcal{N}(\mathbf{z} \mid \mu(x \mid \Theta), \Sigma(x \mid \Theta))$ represents encoder likelihood. That $\mathcal{N}(\mathbf{z} \mid \mu(x \mid \Theta), \Sigma(x \mid \Theta))$ is encoder likelihood is an important improvement in developing VAE because encoder DNN $f(x \mid \Theta)$ is learned by minimizing a so-called encoder error which is represented by the difference between encoder likelihood and predefined tractable PDF $P(\mathbf{z})$. Let $\text{KL}(\mathcal{N}(\text{z} \mid \mu(x \mid \Theta), \Sigma(x \mid \Theta)) \mid P(\mathbf{z}))$ be Kullback-Leibler divergence of encoder likelihood $\mathcal{N}(\text{z} \mid \mu(x \mid \Theta), \Sigma(x \mid \Theta))$ and predefined tractable PDF $P(\mathbf{z})$. As a result, $\text{KL}(\mathcal{N}(\mathbf{z} \mid \mu(x \mid \Theta), \Sigma(x \mid \Theta)) \mid P(\mathbf{z}))$ becomes an ideal encoder error, which is called encoder KL divergence. The smaller the encoder KL divergence is, the closer the encoder likelihood $\mathcal{N}(\text{z} \mid \mu(x \mid \Theta), \Sigma(x \mid \Theta))$ is to tractable PDF $P(\mathbf{z})$, the better the encoder DNN $f(x \mid \Theta)$ is.

$$\varepsilon(\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)|\Theta) = \text{KL}\left(\mathcal{N}\big(\mathbf{z}\big|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\big)\Big|P(\boldsymbol{z})\right) \tag{3.3}$$

Therefore, encoder KL divergence $\text{KL}(\mathcal{N}(\mathbf{z} \mid \mu(x \mid \Theta), \Sigma(x \mid \Theta)) \mid P(\mathbf{z}))$ is minimized by stochastic gradient descent (SGD) method in order to estimate decoder parameter Θ for training encoder DNN $f(x \mid \Theta)$ as follows:

$$\Theta^* = \underset{\Theta}{\text{armin}}\, \text{KL}\left(\mathcal{N}\big(\mathbf{z}\big|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\big)\Big|P(\boldsymbol{z})\right)$$

Which results estimation equation according to SGD:

$$\Theta = \Theta - \gamma\nabla\text{KL}\left(\mathcal{N}\big(\mathbf{z}\big|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\big)\Big|P(\boldsymbol{z})\right)$$

Where $\nabla\text{KL}(\mathcal{N}(\text{z} \mid \mu(x \mid \Theta), \Sigma(x \mid \Theta)) \mid P(\mathbf{z}))$ is gradient of encoder KL divergence $\text{KL}(\mathcal{N}(\mathbf{z} \mid \mu(x \mid \Theta), \Sigma(x \mid \Theta)) \mid P(\mathbf{z}))$ with regard to $\mu(x \mid \Theta)$ and $\Sigma(x \mid \Theta)$ while γ is learning rate. Recall that SGD, which is an iterative process, pushes candidate solution at each iteration along the direction which is opposite to gradient of target function for minimization or has the same direction to gradient of target function for maximization with note that the step length is represented by learning rate. We have:

$$\nabla\text{KL}\left(\mathcal{N}\big(\mathbf{z}\big|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\big)\Big|P(\boldsymbol{z})\right) = \frac{d\text{KL}\left(\mathcal{N}\big(\mathbf{z}\big|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\big)\Big|P(\boldsymbol{z})\right)}{d\Theta}$$

There can be no change in estimating decoder parameter Φ within VAE so that decoder error $\varepsilon(x \mid \Phi, \mathbf{z}) = ½||g(\mathbf{z} \mid \Phi) - x||^2$ is minimized to produce optimal Φ.

$$\Phi^* = \underset{\Phi}{\text{armin}} \frac{1}{2}\|g(\mathbf{z}|\Phi) - \boldsymbol{x}\|^2$$

Which results estimation equation according to SGD:

$$\Phi = \Phi - \gamma\nabla\frac{1}{2}\|g(\mathbf{z}|\Phi) - \boldsymbol{x}\|^2$$

Recall that generator $g(\mathbf{z} \mid \Phi)$ is called decoder $g(\mathbf{z} \mid \Phi)$ in VAE. As a result, encoder parameter Θ and decoder parameter Φ are estimated as follows:

$$\Theta = \Theta - \gamma\nabla\text{KL}\left(\mathcal{N}\big(\mathbf{z}\big|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\big)\Big|P(\boldsymbol{z})\right)$$
$$\Phi = \Phi - \gamma\nabla\frac{1}{2}\|g(\mathbf{z}|\Phi) - \boldsymbol{x}\|^2 = \Phi - \gamma\big(g(\mathbf{z}|\Phi) - \boldsymbol{x}\big)\frac{dg(\mathbf{z}|\Phi)}{d\Phi}$$

Where $dg(\mathbf{z} \mid \Phi) / d\Phi$ is differential of $g(\mathbf{z} \mid \Phi)$ with regard to Φ while $0 < \gamma \leq 1$ is learning rate and tractable PDF $P(\mathbf{z})$ is predefined with note that VAE replaces tractable PDF $P(\mathbf{z})$ by likelihood $P(\mathbf{z}' \mid \Theta, \mu(x \mid \Theta), \Sigma(x \mid \Theta))$ with fixed $P(\mathbf{z})$. As usual, $P(\mathbf{z})$ is assumed to conform standard normal distribution with mean $\mathbf{0}$ and covariance matrix I.

$$P(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, I)$$

This implies:

$$\Theta = \Theta - \gamma \nabla \mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, I)\right)$$

Where I is identity matrix:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

It is easier to determine gradient of encoder KL divergence $\nabla\mathrm{KL}(N(\mu(x \mid \Theta), \Sigma(x \mid \Theta)) \mid \mathcal{N}(\mathbf{z} \mid \mathbf{0}, I))$ with regard to Θ between the multivariate normal distribution $\mathcal{N}(\mu(x), \Sigma(x) \mid \Theta)$ and the standard multivariate normal distribution $\mathcal{N}(\mathbf{z} \mid \mathbf{0}, I)$. We have following equation to calculate such gradient (Kingma & Welling, 2022, p. 5), (Doersch, 2016, p. 9), (Nguyen, 2015, p. 43):

$$\nabla \mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, I)\right) = \frac{d\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, I)\right)}{d\Theta}$$

$$= \begin{pmatrix} \mu(\boldsymbol{x}|\Theta)\dfrac{d\mu(\boldsymbol{x}|\Theta)}{d\Theta} \\ \dfrac{1}{2}\left(I - \left(\Sigma(\boldsymbol{x}|\Theta)\right)^{-1}\right)\dfrac{d\Sigma(\boldsymbol{x}|\Theta)}{d\Theta} \end{pmatrix}^T$$

$$= \left(\mu(\boldsymbol{x}|\Theta)\frac{d\mu(\boldsymbol{x}|\Theta)}{d\Theta}, \frac{1}{2}\left(I - \left(\Sigma(\boldsymbol{x}|\Theta)\right)^{-1}\right)\frac{d\Sigma(\boldsymbol{x}|\Theta)}{d\Theta}\right)$$

Where $(\Sigma(x \mid \Theta))^{-1}$ is inverse of covariance matrix $\Sigma(x \mid \Theta)$ and the subscript "T" denotes transposition operator of matrix and vector whereas $d\mu(x \mid \Theta) / d\Theta$ and $d\Sigma(x \mid \Theta) / d\Theta$ are differentials of $\mu(x \mid \Theta)$ and $\Sigma(x \mid \Theta)$ with regard to Θ, respectively. As a result, encoder parameter Θ and decoder parameter Φ are totally estimated according to SGD as follows:

$$\Theta = \Theta - \gamma \begin{pmatrix} \mu(\boldsymbol{x}|\Theta)\dfrac{d\mu(\boldsymbol{x}|\Theta)}{d\Theta} \\ \dfrac{1}{2}\left(I - \left(\Sigma(\boldsymbol{x}|\Theta)\right)^{-1}\right)\dfrac{d\Sigma(\boldsymbol{x}|\Theta)}{d\Theta} \end{pmatrix}$$

$$\Phi = \Phi - \gamma\left(g(\boldsymbol{z}|\Phi) - \boldsymbol{x}\right)\frac{dg(\boldsymbol{z}|\Phi)}{d\Phi}$$

The estimation equations above are simple explanation of VAE but its formal construction is more complicated. We begin the aforementioned intractable PDF $P(x)$ specified by law of total probability:

$$P(\boldsymbol{x}) = \int_{\boldsymbol{z}} P(\boldsymbol{x}|\Phi, \boldsymbol{z})P(\boldsymbol{z})\,d\boldsymbol{z}$$

However, $P(x)$ is interpreted by another way which is based on Bayes' rule within VAE:

$$P(\boldsymbol{x}) = \frac{P(\boldsymbol{x}, \boldsymbol{z})}{P(\boldsymbol{z}|\boldsymbol{x})}$$

Because the conditional probability $P(\mathbf{z} \mid x)$ is arbitrary without formal specification, it should be approximated by another PDF denoted $Q(\mathbf{z} \mid x)$ with assumption that the PDF $Q(\mathbf{z} \mid x)$ has formal specification like normal distribution.

$$Q(\boldsymbol{z}|\boldsymbol{x}) \cong P(\boldsymbol{z}|\boldsymbol{x})$$

Logarithm of intractable PDF $P(x)$ is specified as follows (Ruthotto & Haber, 2021, p. 13):

$$\log P(\boldsymbol{x}) = \log \frac{P(\boldsymbol{x},\boldsymbol{z})}{P(\boldsymbol{z}|\boldsymbol{x})} = \log\left(\frac{P(\boldsymbol{x},\boldsymbol{z})}{Q(\boldsymbol{z}|\boldsymbol{x})}\frac{Q(\boldsymbol{z}|\boldsymbol{x})}{P(\boldsymbol{z}|\boldsymbol{x})}\right)$$

This implies:

$$\log P(\boldsymbol{x}) = \log \frac{P(\boldsymbol{x},\boldsymbol{z})}{Q(\boldsymbol{z}|\boldsymbol{x})} + \log \frac{Q(\boldsymbol{z}|\boldsymbol{x})}{P(\boldsymbol{z}|\boldsymbol{x})} \tag{3.4}$$

The second term log($Q(\mathbf{z} \mid x) / P(\mathbf{z} \mid x)$) is not variant because $Q(\mathbf{z} \mid x)$ is approximated to $P(\mathbf{z} \mid x)$. Therefore, the first term log($P(x, \mathbf{z}) / Q(\mathbf{z} \mid x)$ is called variation lower bound or *evidence lower bound* because it is variant. Let $l(x, \mathbf{z})$ be loss function or error function on VAE which is defined as the minus opposite of expectation of the evidence lower bound log($P(x, \mathbf{z}) / Q(\mathbf{z} \mid x)$ given PDF $Q(\mathbf{z} \mid x)$ with note that $Q(\mathbf{z} \mid x)$ has formal probabilistic distribution.

$$l(\boldsymbol{x},\boldsymbol{z}) = -E_{Q(\boldsymbol{z}|\boldsymbol{x})}\left(\log\frac{P(\boldsymbol{x},\boldsymbol{z})}{Q(\boldsymbol{z}|\boldsymbol{x})}\right) = -\int_{\boldsymbol{z}} Q(\boldsymbol{z}|\boldsymbol{x}) \log \frac{P(\boldsymbol{x}|\boldsymbol{z})P(\boldsymbol{z})}{Q(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z}$$

Loss function $l(x, \mathbf{z})$ is expended as follows:

$$l(\boldsymbol{x},\boldsymbol{z}) = -\log P(\boldsymbol{x}|\boldsymbol{z})\int_{\boldsymbol{z}} Q(\boldsymbol{z}|\boldsymbol{x})d\boldsymbol{z} + \int_{\boldsymbol{z}} Q(\boldsymbol{z}|\boldsymbol{x})\left(\log\frac{Q(\boldsymbol{z}|\boldsymbol{x})}{P(\boldsymbol{z})}\right)d\boldsymbol{z}$$

$$= -\log P(\boldsymbol{x}|\boldsymbol{z}) + \int_{\boldsymbol{z}} Q(\boldsymbol{z}|\boldsymbol{x})\left(\log\frac{Q(\boldsymbol{z}|\boldsymbol{x})}{P(\boldsymbol{z})}\right)d\boldsymbol{z}$$

Because $Q(\mathbf{z} \mid x)$ and $P(x \mid \mathbf{z})$ depend on encoder $f(x \mid \Theta)$ and decoder $g(\mathbf{z} \mid \Phi)$, respectively, their parameters are Θ and Φ, respectively.

$$Q(\boldsymbol{z}|\boldsymbol{x}) = Q(\boldsymbol{z}|\Theta,\boldsymbol{x})$$
$$P(\boldsymbol{x}|\boldsymbol{z}) = P(\boldsymbol{x}|\Phi,\boldsymbol{z})$$

Exactly, $Q(\mathbf{z} \mid \Theta, x)$ is encoder likelihood which is the same to the aforementioned $P(\mathbf{z}' \mid \Theta, x)$ except that it is focused that $Q(\mathbf{z} \mid \Theta, x)$ has formal probabilistic specification like normal distribution. Loss function $l(\Theta, \Phi \mid x, \mathbf{z})$, which is now function of encoder parameter Θ and decoder parameter Φ, is written as follows (Ruthotto & Haber, 2021, p. 14):

$$l(\Theta,\Phi|\boldsymbol{x},\boldsymbol{z}) = -\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) + \int_{\boldsymbol{z}} Q(\boldsymbol{z}|\Theta,\boldsymbol{x})\left(\log\frac{Q(\boldsymbol{z}|\Theta,\boldsymbol{x})}{P(\boldsymbol{z})}\right)d\boldsymbol{z}$$

Firstly, please pay attention to the first term loss function $l(\Theta, \Phi \mid x, \mathbf{z})$ where $P(x \mid \Phi, \mathbf{z})$ depends only on Φ although it can be considered as a conditional PDF of x given $\mathbf{z}$ because $P(x \mid \Phi, \mathbf{z})$ is defined for output layer containing only x of decoder DNN $g(x \mid \Phi)$ whose input is x. Therefore, we had the following assertion:

$$\int_{\boldsymbol{z}} Q(\boldsymbol{z}|\boldsymbol{x}) \log P(\boldsymbol{x}|\Phi,\boldsymbol{z})\, d\boldsymbol{z} = \log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) \int_{\boldsymbol{z}} Q(\boldsymbol{z}|\boldsymbol{x})d\boldsymbol{z} = \log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \log P(\boldsymbol{x}|\Phi)$$

Secondly, the second term in loss function $l(\Theta, \Phi \mid x, \mathbf{z})$ is, actually, Kullback-Leibler divergence of encoder likelihood $Q(\mathbf{z} \mid \Theta, x)$ and predefined tractable PDF $P(\mathbf{z})$, which measure the difference between $Q(\mathbf{z} \mid \Theta, x)$ and $P(\mathbf{z})$. As a convention, this Kullback-Leibler divergence is called encoder KL divergence which is an ideal encoder error.

$$\mathrm{KL}\big(Q(\boldsymbol{z}|\Theta,\boldsymbol{x})\big|P(\boldsymbol{z})\big) = E_{Q(\boldsymbol{z}|\boldsymbol{x})}\left(\log\frac{Q(\boldsymbol{z}|\Theta,\boldsymbol{x})}{P(\boldsymbol{z})}\right) = \int_{\boldsymbol{z}} Q(\boldsymbol{z}|\Theta,\boldsymbol{x})\left(\log\frac{Q(\boldsymbol{z}|\Theta,\boldsymbol{x})}{P(\boldsymbol{z})}\right)d\boldsymbol{z}$$

The smaller the encoder KL divergence is, the closer the encoder likelihood $Q(\mathbf{z} \mid \Theta, x)$ is to tractable PDF $P(\mathbf{z})$, the better the encoder DNN $f(x \mid \Theta)$ is. Loss function is rewritten again:

$$l(\Theta, \Phi|\boldsymbol{x}, \boldsymbol{z}) = -\log P(\boldsymbol{x}|\Phi, \boldsymbol{z}) + \text{KL}\big(Q(\boldsymbol{z}|\Theta, \boldsymbol{x})\big|P(\boldsymbol{z})\big)$$

Or,

$$l(\Theta, \Phi|\boldsymbol{x}, \boldsymbol{z}) = -\log P(\boldsymbol{x}|\Phi) + \text{KL}\big(Q(\boldsymbol{z}|\Theta)\big|P(\boldsymbol{z})\big) \quad (3.5)$$

According to the two problem of construct a DGM, the first term –log(P($\boldsymbol{x}$ | Φ, **z**)) in loss function indicates the first problem of how to train decoder DNN g(**z** | Φ) which is called reconstruction error in literature and the second term KL(Q(**z** | Θ, $\boldsymbol{x}$) | P(**z**)) in loss function indicates the second problem of how to qualify training task for training encoder DNN f($\boldsymbol{x}$ | Θ) which is called regularity in literature. Loss function l(Θ, Φ | $\boldsymbol{x}$, **z**) is minimized to estimate Θ and Φ as follows:

$$\begin{aligned}\Theta^* &= \underset{\Theta}{\text{armin}}\, l(\Theta, \Phi|\boldsymbol{x}, \boldsymbol{z})\\ \Phi^* &= \underset{\Phi}{\text{armin}}\, l(\Theta, \Phi|\boldsymbol{x}, \boldsymbol{z})\end{aligned} \quad (3.6)$$

Because P($\boldsymbol{x}$ | Θ, **z**) depends only on Θ and encoder KL divergence KL(Q(**z** | Θ, $\boldsymbol{x}$) | P(**z**)) depends only on Φ, the optimization problem is specified as follows:

$$\Theta^* = \underset{\Theta}{\text{armin}}\, \text{KL}\big(Q(\boldsymbol{z}|\Theta, \boldsymbol{x})\big|P(\boldsymbol{z})\big)$$

$$\Phi^* = \underset{\Phi}{\text{armin}}(-\log P(\boldsymbol{x}|\Phi, \boldsymbol{z})) = \underset{\Phi}{\text{armax}}(\log P(\boldsymbol{x}|\Phi, \boldsymbol{z}))$$

Which results estimation equations according to SGD:

$$\begin{aligned}\Theta &= \Theta - \gamma\nabla\text{KL}\big(Q(\boldsymbol{z}|\Theta, \boldsymbol{x})\big|P(\boldsymbol{z})\big)\\ \Phi &= \Phi + \gamma\nabla\log P(\boldsymbol{x}|\Phi, \boldsymbol{z})\end{aligned} \quad (3.7)$$

Where ∇KL(Q(**z** | Θ, $\boldsymbol{x}$) | P(**z**)) is gradient of encoder KL divergence KL(Q(**z** | Θ, $\boldsymbol{x}$) | P(**z**)) with regard to encoder parameter Θ. Note that tractable PDF P(**z**) is predefined (fixed). While Q(**z** | Θ, $\boldsymbol{x}$) is called encoder likelihood, P($\boldsymbol{x}$ | Φ, **z**) is called decoder likelihood. On the other hand, while P(**z**) is prior PDF of intractable data **z**, then Q(**z** | Θ, $\boldsymbol{x}$) is approximated posterior PDF of **z** given $\boldsymbol{x}$ where both P(**z**) and Q(**z** | Θ, $\boldsymbol{x}$) have formal probabilistic specifications and moreover, P(**z**) is fixed (predefined).

$$Q(\boldsymbol{z}|\Theta, \boldsymbol{x}) = P(\boldsymbol{z}'|\Theta, \boldsymbol{x}) \cong P(\boldsymbol{z}|\Theta, \boldsymbol{x})$$

Both P(**z** | Θ, $\boldsymbol{x}$) and Q(**z** | Θ, $\boldsymbol{x}$) are encoder likelihood as well as posterior PDF of tractable data **z** but Q(**z** | Θ, $\boldsymbol{x}$) is approximated one whose probabilistic distribution is specified formally. Therefore (Ruthotto & Haber, 2021, p. 16), randomized data **z**′ in latent space Z is sampled from approximated distribution Q(**z** | Θ, $\boldsymbol{x}$) instead of sampling from true distribution P(**z** | Θ, $\boldsymbol{x}$).

Given epoch of size N is denoted as $\boldsymbol{D} = (\boldsymbol{d}^{(1)} = (\boldsymbol{x}^{(1)}, \mathbf{z}^{(1)}), \boldsymbol{d}^{(2)} = (\boldsymbol{x}^{(2)}, \mathbf{z}^{(2)}),\ldots, \boldsymbol{d}^{(N)} = (\boldsymbol{x}^{(N)}, \mathbf{z}^{(N)}))$, the estimation equations of Θ and Φ are extended exactly as epoch estimation at every iteration of SGD:

$$\Theta^{(k+1)} = \Theta^{(k)} - \gamma\nabla\frac{1}{N}\sum_{i=1}^{N}\text{KL}\left(Q\big(\boldsymbol{z}^{(i)}\big|\Theta^{(k)}, \boldsymbol{x}^{(i)}\big)\middle|P(\boldsymbol{z})\right)$$

$$\Phi^{(k+1)} = \Phi^{(k)} + \gamma\nabla\frac{1}{N}\sum_{i=1}^{N}\log P\big(\boldsymbol{x}^{(i)}\big|\Phi, \boldsymbol{z}^{(i)}\big)$$

Please distinguish that the tractable data $\mathbf{z}^{(i)}$ in the first equation above follows distribution P(**z**) but the tractable data $\mathbf{z}^{(i)}$ in the second equation above follows distribution Q(**z** | Θ, $\boldsymbol{x}$). As a result, VAE trained with SGD is specified as follows:

Initialize Θ and Φ and set k = 0.

Repeat

Sampling epoch $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)},\ldots, \boldsymbol{x}^{(N)})$ or receiving epoch $\boldsymbol{X}$ from big data / data stream.

Randomize random epoch $\boldsymbol{Z} = (\mathbf{z}^{(1)}, \mathbf{z}^{(2)},\ldots, \mathbf{z}^{(N)})$ in which each $\mathbf{z}^{(i)}$ is randomized from distribution $Q(\mathbf{z} \mid \Theta^{(k)}, \boldsymbol{x}^{(i)})$.

$$\Theta^{(k+1)} = \Theta^{(k)} - \gamma\nabla\frac{1}{N}\sum_{i=1}^{N}\mathrm{KL}\left(Q\left(\boldsymbol{z}^{(i)}\middle|\Theta^{(k)},\boldsymbol{x}^{(i)}\right)\middle|P(\boldsymbol{z})\right)$$

$$\Phi^{(k+1)} = \Phi^{(k)} + \gamma\nabla\frac{1}{N}\sum_{i=1}^{N}\log P\left(\boldsymbol{x}^{(i)}\middle|\Phi,\boldsymbol{z}^{(i)}\right)$$

Increase $k = k + 1$.

Until some terminating conditions are met.

Note, a terminating condition is customized, for example, parameters Θ and Φ are not changed significantly or there is no more coming epoch $\boldsymbol{X}$. Moreover, the index k indicates time point as well as iteration of SGD. Because PDF $P(\mathbf{z})$ is predefined, it is easy to calculate encoder KL divergence $\mathrm{KL}(Q(\mathbf{z}^{(i)} \mid \Theta^{(k)}, \boldsymbol{x}^{(i)}) \mid P(\mathbf{z}))$ but it is necessary to define $P(\boldsymbol{x})$ by well-known distribution. However, randomizing random epoch $\mathbf{Z} = (\mathbf{z}^{(1)}, \mathbf{z}^{(2)},\ldots, \mathbf{z}^{(N)})$ from distribution $Q(\mathbf{z} \mid \Theta^{(k)}, \boldsymbol{x}^{(i)})$ is not easy and so, VAE trained with SGD will be fine-tuned. It is interesting that when $Q(\mathbf{z} \mid \Theta^{(k)}, \boldsymbol{x}^{(i)})$ is posterior PDF of $\mathbf{z}$ and $P(\mathbf{z})$ is prior PDF of $\mathbf{z}$, the event that $\mathbf{z}$ is randomized from the posterior PDF $Q(\mathbf{z} \mid \Theta^{(k)}, \boldsymbol{x}^{(i)})$ and $Q(\mathbf{z} \mid \Theta^{(k)}, \boldsymbol{x}^{(i)})$ itself is updated continuously based on its previous evidence $\boldsymbol{x}^{(i)}$ over SGD iterations implies that VAE conforms Bayesian statistics in estimation. Moreover, $P(\mathbf{z})$ is an alignment that $Q(\mathbf{z} \mid \Theta^{(k)}, \boldsymbol{x}^{(i)})$ adjusts itself with support of encoder KL divergence $\mathrm{KL}(Q(\mathbf{z}^{(i)} \mid \Theta^{(k)}, \boldsymbol{x}^{(i)}) \mid P(\mathbf{z}))$.

Because encoder likelihood $Q(\mathbf{z} \mid \Theta, \boldsymbol{x})$ must always have formal probabilistic distribution, it is assumed to follow multivariate normal distribution in practice. Therefore, let $\mu(\boldsymbol{x} \mid \Theta)$ and $\Sigma(\boldsymbol{x} \mid \Theta)$ be mean vector and covariance matrix of $\mathbf{z}$, then encoder likelihood $Q(\mathbf{z} \mid \Theta, \boldsymbol{x})$ becomes $Q(\mathbf{z} \mid \mu(\boldsymbol{x} \mid \Theta), \Sigma(\boldsymbol{x} \mid \Theta))$ so that output of encoder DNN $f(\boldsymbol{x} \mid \Theta)$ is mean $\mu(\boldsymbol{x} \mid \Theta)$ and covariance matrix $\Sigma(\boldsymbol{x} \mid \Theta)$ while its input is $\boldsymbol{x}$ and its weights are Θ, of course. Please pay attention to the fact that output of encoder DNN $f(\boldsymbol{x} \mid \Theta)$ is now $\mu(\boldsymbol{x} \mid \Theta)$ and $\Sigma(\boldsymbol{x} \mid \Theta)$ which are corresponding to $\mathbf{z}$. Moreover, $\mu(\boldsymbol{x} \mid \Theta)$ and $\Sigma(\boldsymbol{x} \mid \Theta)$ are functions of $\boldsymbol{x}$, whose parameter is Θ.

$$\begin{gathered} f(\boldsymbol{x}|\Theta) = \begin{pmatrix}\mu(\boldsymbol{x}|\Theta)\\ \Sigma(\boldsymbol{x}|\Theta)\end{pmatrix} \sim \boldsymbol{z} \\ Q\big(\boldsymbol{z}\big|\mu(\boldsymbol{x}|\Theta),\Sigma(\boldsymbol{x}|\Theta)\big) = \mathcal{N}\big(\boldsymbol{z}\big|\mu(\boldsymbol{x}|\Theta),\Sigma(\boldsymbol{x}|\Theta)\big) \end{gathered} \quad (3.8)$$

Note, $\mathcal{N}(\mathbf{z} \mid \mu(\boldsymbol{x} \mid \Theta), \Sigma(\boldsymbol{x} \mid \Theta))$ denotes multivariate normal distribution with mean $\mu(\boldsymbol{x} \mid \Theta)$ and covariance matrix $\Sigma(\boldsymbol{x} \mid \Theta)$.

$$\mathcal{N}\big(\boldsymbol{z}\big|\mu(\boldsymbol{x}|\Theta),\Sigma(\boldsymbol{x}|\Theta)\big) \\ = (2\pi)^{-\frac{n}{2}}|\Sigma(\boldsymbol{x}|\Theta)|^{-\frac{1}{2}}\exp\left(-\frac{1}{2}\big(\boldsymbol{z}-\mu(\boldsymbol{x}|\Theta)\big)^{T}\big(\Sigma(\boldsymbol{x}|\Theta)\big)^{-1}\big(\boldsymbol{z}-\mu(\boldsymbol{x}|\Theta)\big)\right)$$

Note, dimension of tractable data $\mathbf{z}$ is n. Moreover, notation |.| or notation det(.) denotes determinant of matrix whereas $(\Sigma(\boldsymbol{x} \mid \Theta))^{-1}$ is inverse of covariance matrix $\Sigma(\boldsymbol{x} \mid \Theta)$ and the subscript "T" denotes transposition operator of matrix and vector. It is easy to recognize that $\mathbf{z}'$ is approximation of $\mathbf{z}$. When tractable PDF $P(\mathbf{z})$ is fixed, it is often assumed to follow multivariate normal distribution with predefined mean μ_0 and predefined covariance matrix Σ_0 as follows:

$$P(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\mu_0,\Sigma_0) = \mathcal{N}(\mu_0,\Sigma_0) = (2\pi)^{-\frac{n}{2}}|\Sigma_0|^{-\frac{1}{2}}\exp\left(-\frac{1}{2}(\boldsymbol{z}-\mu_0)^T\Sigma_0^{-1}(\boldsymbol{z}-\mu_0)\right)$$

Encoder KL divergence $\mathrm{KL}(Q(\mathbf{z} \mid \Theta, \boldsymbol{x}) \mid P(\mathbf{z}))$ between $Q(\mathbf{z} \mid \Theta, \boldsymbol{x})$ and $P(\mathbf{z})$ becomes encoder KL divergence $\mathrm{KL}(Q(\mathbf{z} \mid \mu(\boldsymbol{x} \mid \Theta), \Sigma(\boldsymbol{x} \mid \Theta)) \mid P(\mathbf{z}))$ between $Q(\mathbf{z} \mid \mu(\boldsymbol{x} \mid \Theta), \Sigma(\boldsymbol{x} \mid \Theta))$ and $P(\mathbf{z})$ as follows:

$$\mathrm{KL}\left(Q\big(\boldsymbol{z}\big|\mu(\boldsymbol{x}|\Theta),\Sigma(\boldsymbol{x}|\Theta)\big)\middle|P(\boldsymbol{z})\right) = \mathrm{KL}\left(\mathcal{N}\big(\boldsymbol{z}\big|\mu(\boldsymbol{x}|\Theta),\Sigma(\boldsymbol{x}|\Theta)\big)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0,\Sigma_0)\right)$$

Which is, essentially, encoder KL divergence between two normal distributions, $\mathrm{KL}(\mathcal{N}(\mathbf{z} \mid \mu(\boldsymbol{x} \mid \Theta), \Sigma(\boldsymbol{x} \mid \Theta)) \mid \mathcal{N}(\mu_0, \Sigma_0))$. As a convention, this divergence is called encoder KL divergence which is determined in literature as follows (Doersch, 2016, p. 9):

$$\mathrm{KL}\left(\mathcal{N}\big(\boldsymbol{z}\big|\mu(\boldsymbol{x}|\Theta),\Sigma(\boldsymbol{x}|\Theta)\big)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0,\Sigma_0)\right)$$

$$= \frac{1}{2}\left(\mathrm{tr}\left(\Sigma_0^{-1}\Sigma(\boldsymbol{x}|\Theta)\right) + (\mu(\boldsymbol{x}|\Theta) - \mu_0)^T\Sigma_0^{-1}(\mu(\boldsymbol{x}|\Theta) - \mu_0) + \log\left(\frac{|\Sigma_0|}{|\Sigma(\boldsymbol{x}|\Theta)|}\right) - n\right)$$

Where tr(.) denotes trace operator of square matrix which is sum of elements on main diagonal, for instance, given $n \times n$ matrix A, then tr(A) = $a_{11} + a_{22} + \ldots + a_{nn}$ with note that a_{ij} is the element at row i and column j. Moreover, notation |.| or notation det(.) denotes determinant of matrix. Gradient of encoder KL divergence consists of two elemental gradients with regard to mean $\mu(x \mid \Theta)$ and covariance matrix $\Sigma(x \mid \Theta)$.

$$\nabla\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right)$$

$$= \begin{pmatrix} \nabla_\mu\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right) \\ \nabla_\Sigma\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right) \end{pmatrix}^T$$

Where,

$$\nabla_\mu\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right)$$

$$= \frac{d\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right)}{d\mu(\boldsymbol{x}|\Theta)}\frac{d\mu(\boldsymbol{x}|\Theta)}{d\Theta}$$

$$\nabla_\Sigma\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right)$$

$$= \frac{d\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right)}{d\Sigma(\boldsymbol{x}|\Theta)}\frac{d\Sigma(\boldsymbol{x}|\Theta)}{d\Theta}$$

Where $d\mu(x \mid \Theta) / d\Theta$ and $d\Sigma(x \mid \Theta) / d\Theta$ are differentials of $\mu(x \mid \Theta)$ and $\Sigma(x \mid \Theta)$ with regard to Θ, respectively. It is not difficult to calculate KL gradient ∇_μ:

$$\nabla_\mu\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right) = (\mu(\boldsymbol{x}|\Theta) - \mu_0)^T\Sigma_0^{-1}\frac{d\mu(\boldsymbol{x}|\Theta)}{d\Theta} \quad (3.9)$$

Due to (Nguyen, Matrix Analysis and Calculus, 2015, p. 35):

$$\frac{d(\mu(\boldsymbol{x}|\Theta) - \mu_0)^T\Sigma_0^{-1}(\mu(\boldsymbol{x}|\Theta) - \mu_0)}{d\mu(\boldsymbol{x}|\Theta)} = 2(\mu(\boldsymbol{x}|\Theta) - \mu_0)^T\Sigma_0^{-1}$$

It is not difficult to calculate KL gradient ∇_Σ too:

$$\nabla_\Sigma\mathrm{KL}\left(\mathcal{N}\left(\boldsymbol{z}\middle|\mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\right)\middle|\mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right) = \frac{1}{2}\left(\Sigma_0^{-1} - \left(\Sigma(\boldsymbol{x}|\Theta)\right)^{-1}\right)\frac{d\Sigma(\boldsymbol{x}|\Theta)}{d\Theta} \quad (3.10)$$

Due to (Nguyen, Matrix Analysis and Calculus, 2015, pp. 45-46):

$$\frac{d\mathrm{tr}\left(\Sigma_0^{-1}\Sigma(\boldsymbol{x}|\Theta)\right)}{d\Sigma(\boldsymbol{x}|\Theta)} = \Sigma_0^{-1}$$

$$\frac{d\log(|\Sigma(\boldsymbol{x}|\Theta)|)}{d\Sigma(\boldsymbol{x}|\Theta)} = \frac{d|\Sigma(\boldsymbol{x}|\Theta)|}{d\Sigma(\boldsymbol{x}|\Theta)}\frac{1}{|\Sigma(\boldsymbol{x}|\Theta)|} = \frac{|\Sigma(\boldsymbol{x}|\Theta)|\left(\Sigma(\boldsymbol{x}|\Theta)\right)^{-1}}{|\Sigma(\boldsymbol{x}|\Theta)|} = \left(\Sigma(\boldsymbol{x}|\Theta)\right)^{-1}$$

As a result, encoder parameter Θ consists of two elemental parameters according to with regard to mean $\mu(x \mid \Theta)$ and covariance matrix $\Sigma(x \mid \Theta)$ as follows:

$$\Theta = \begin{pmatrix} \Theta_\mu \\ \Theta_\Sigma \end{pmatrix}$$

Where,

$$\Theta_\mu = (\mu_1, \mu_2, \ldots, \mu_n)^T$$

$$\Theta_\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_n^2 \end{pmatrix}$$

Note, given random vector **z** = (z_1, z_2,…, z_n)^T whose elements z_i are random variables too, σ_{ij} where $i \neq j$ is covariance between two random variables z_i and z_j and σ_i^2 is variance of random variable z_i. It is easy to calculate encoder parameters Θ_μ and Θ_Σ by SGD estimation:

$$\Theta_\mu = \Theta_\mu - \gamma \nabla_\mu \mathrm{KL}\left(\mathcal{N}\big(\boldsymbol{z} \big| \mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\big) \middle| \mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right)$$

$$= \Theta_\mu - \gamma \Sigma_0^{-1}\big(\mu(\boldsymbol{x}|\Theta_\mu) - \mu_0\big)\frac{d\mu(\boldsymbol{x}|\Theta_\mu)}{d\Theta_\mu}$$

$$\Theta_\Sigma = \Theta_\Sigma - \gamma \nabla_\Sigma \mathrm{KL}\left(\mathcal{N}\big(\boldsymbol{z} \big| \mu(\boldsymbol{x}|\Theta), \Sigma(\boldsymbol{x}|\Theta)\big) \middle| \mathcal{N}(\boldsymbol{z}|\mu_0, \Sigma_0)\right)$$

$$= \Theta_\Sigma - \gamma \frac{1}{2}\left(\Sigma_0^{-1} - \big(\Sigma(\boldsymbol{x}|\Theta_\Sigma)\big)^{-1}\right)\frac{d\Sigma(\boldsymbol{x}|\Theta_\Sigma)}{d\Theta_\Sigma}$$

Where $d\mu(\boldsymbol{x} \mid \Theta_\mu) / d\Theta_\mu$ and $d\Sigma(\boldsymbol{x} \mid \Theta_\Sigma) / d\Theta_\Sigma$ are differentials of $\mu(\boldsymbol{x} \mid \Theta_\mu)$ and $\Sigma(\boldsymbol{x} \mid \Theta_\Sigma)$ with regard to Θ_μ and Θ_Σ, respectively. In practice, P(**z**) is assumed to conform standard normal distribution with zero mean μ_0 = **0** and identity covariance matrix $\Sigma_0 = I$ where I is identity matrix so that encoder parameters Θ_μ and Θ_Σ are computed effectively.

$$\begin{aligned} \Theta_\mu &= \Theta_\mu - \gamma\mu(\boldsymbol{x}|\Theta_\mu)\frac{d\mu(\boldsymbol{x}|\Theta_\mu)}{d\Theta_\mu} \\ \Theta_\Sigma &= \Theta_\Sigma - \gamma\frac{1}{2}\left(I - \big(\Sigma(\boldsymbol{x}|\Theta_\Sigma)\big)^{-1}\right)\frac{d\Sigma(\boldsymbol{x}|\Theta_\Sigma)}{d\Theta_\Sigma} \end{aligned} \quad (3.11)$$

In order to improve more computational effectiveness, it is possible to suppose that elemental variables z_i in **z** = (z_1, z_2,…, z_n)^T within context P(**z**) are mutually independent so that covariance σ_{ij} between two variables z_i and z_j where $i \neq j$ is 0, which results that there only exist variances σ_i^2 of z_i. Covariance matrix $\Sigma(\boldsymbol{x} \mid \Theta)$ becomes diagonal matrix:

$$\Sigma(\boldsymbol{x}|\Theta_\Sigma) = \big(\boldsymbol{\sigma}^2(\boldsymbol{x}|\Theta_\Sigma)\big)_{nxn} = \begin{pmatrix} \sigma_1^2(\boldsymbol{x}|\Theta_\Sigma) & 0 & \cdots & 0 \\ 0 & \sigma_2^2(\boldsymbol{x}|\Theta_\Sigma) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n^2(\boldsymbol{x}|\Theta_\Sigma) \end{pmatrix}$$

Note,

$$\boldsymbol{\sigma}^2(\boldsymbol{x}|\Theta_\Sigma) = \begin{pmatrix} \sigma_1^2(\boldsymbol{x}|\Theta_\Sigma) \\ \sigma_2^2(\boldsymbol{x}|\Theta_\Sigma) \\ \vdots \\ \sigma_n^2(\boldsymbol{x}|\Theta_\Sigma) \end{pmatrix}$$

Where $\sigma_i^2(\boldsymbol{x} \mid \Theta)$ is variance of elemental variable x_i in **z** = (z_1, z_2,…, z_n)^T given $\boldsymbol{x}$ according to encoder $f(\boldsymbol{x} \mid \Theta)$. As a result, encoder parameter Θ_Σ, which is now diagonal matrix represented by its diagonal vector $\Theta_{\boldsymbol{\sigma}^2}$, is computed easier.

$$\Theta_{\boldsymbol{\sigma}^2} = \Theta_{\boldsymbol{\sigma}^2} - \gamma\frac{1}{2}\left(\mathbf{1} - \big(\boldsymbol{\sigma}^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2})\big)^{-1}\right)\frac{d\boldsymbol{\sigma}^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2})}{d\Theta_{\boldsymbol{\sigma}^2}}$$

Where,

$$\Theta_\Sigma = (\Theta_{\boldsymbol{\sigma}^2})_{nxn} = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n^2 \end{pmatrix}$$

$$\Theta_{\boldsymbol{\sigma}^2} = (\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2)^T$$

$$\boldsymbol{\sigma}^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2}) = \begin{pmatrix} \sigma_1^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2}) \\ \sigma_2^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2}) \\ \vdots \\ \sigma_n^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2}) \end{pmatrix}$$

$$\left(\boldsymbol{\sigma}^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2})\right)^{-1} = \begin{pmatrix} 1/\sigma_1^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2}) \\ 1/\sigma_2^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2}) \\ \vdots \\ 1/\sigma_n^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2}) \end{pmatrix}$$

$$\mathbf{1} = (1,1,\dots,1)^T$$

In general, estimation equations for encoder parameter $\Theta = (\Theta_\mu, \Theta_{\boldsymbol{\sigma}^2})^T$ are specified as follows:

$$\Theta_\mu = \Theta_\mu - \gamma\mu(\boldsymbol{x}|\Theta_\mu)\frac{d\mu(\boldsymbol{x}|\Theta_\mu)}{d\Theta_\mu}$$
$$\Theta_{\boldsymbol{\sigma}^2} = \Theta_{\boldsymbol{\sigma}^2} - \gamma\frac{1}{2}\left(\mathbf{1} - \left(\boldsymbol{\sigma}^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2})\right)^{-1}\right)\frac{d\boldsymbol{\sigma}^2(\boldsymbol{x}|\Theta_{\boldsymbol{\sigma}^2})}{d\Theta_{\boldsymbol{\sigma}^2}} \quad (3.12)$$

Where $d\boldsymbol{\sigma}^2(\boldsymbol{x} \mid \Theta_{\boldsymbol{\sigma}^2}) / d\Theta_{\boldsymbol{\sigma}^2}$ is differential of $\boldsymbol{\sigma}^2(\boldsymbol{x} \mid \Theta_{\boldsymbol{\sigma}^2})$ with regard to $\Theta_{\boldsymbol{\sigma}^2}$.

There can be no change in estimating decoder parameter Φ within VAE so that decoder log-likelihood $\log(P(\boldsymbol{x} \mid \Phi, \mathbf{z}))$ is maximized.

$$\Phi^* = \underset{\Phi}{\mathrm{armin}}(-\log P(\boldsymbol{x}|\Phi,\boldsymbol{z})) = \underset{\Phi}{\mathrm{armax}}(\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}))$$

As usual, decoder likelihood $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ is assumed to distribute normally with mean $\boldsymbol{\delta}$ and variance σ^2.

$$P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = (2\pi\sigma^2)^{-\frac{m}{2}}\exp\left(-\frac{\|g(\boldsymbol{z}|\Phi) - \boldsymbol{x} - \boldsymbol{\delta}\|^2}{2\sigma^2}\right)$$

Which implies decoder log-likelihood $\log(P(\boldsymbol{x} \mid \Phi, \mathbf{z}))$ as follows:

$$\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = -\frac{m}{2}\log(2\pi\sigma^2) - \frac{\|g(\boldsymbol{z}|\Phi) - \boldsymbol{x} - \boldsymbol{\delta}\|^2}{2\sigma^2}$$

Where $||.||$ denotes Euclidean norm of vector. Gradient of decoder log-likelihood is:

$$\nabla\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \frac{d\log P(\boldsymbol{x}|\Phi,\boldsymbol{z})}{d\Phi} = -\frac{(g(\boldsymbol{z}|\Phi) - \boldsymbol{x} - \boldsymbol{\delta})}{\sigma^2}\frac{dg(\boldsymbol{z}|\Phi)}{d\Phi}$$

Where $dg(\mathbf{z} \mid \Phi) / d\Phi$ is differential of $g(\mathbf{z} \mid \Phi)$ with regard to Φ. Let $\boldsymbol{\delta} = \mathbf{0}$ and σ^2=1 optimization, we have:

$$\nabla\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = -(g(\boldsymbol{z}|\Phi) - \boldsymbol{x})\frac{dg(\boldsymbol{z}|\Phi)}{d\Phi}$$

Which implies estimation equation for decoder parameter Φ by SGD as follows:

$$\Phi = \Phi + \gamma\nabla\log P(\boldsymbol{x}|\Phi,\boldsymbol{z}) = \Phi - \gamma(g(\boldsymbol{z}|\Phi) - \boldsymbol{x})\frac{dg(\boldsymbol{z}|\Phi)}{d\Phi}$$

Because data $\mathbf{z}$ in the decoder estimation equation above follows encoder likelihood $Q(\mathbf{z} \mid \Theta, \mu(\boldsymbol{x} \mid \Theta_\mu), \Sigma(\boldsymbol{x} \mid \Theta_\Sigma)) = \mathcal{N}(\mathbf{z} \mid \mu(\boldsymbol{x} \mid \Theta_\mu), \Sigma(\boldsymbol{x} \mid \Theta_\Sigma))$ rather than tractable PDF $P(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mu_0, \Sigma_0)$, it is denoted as $\mathbf{z}'$ such that:

$$\Phi = \Phi + \gamma \nabla \log P(\boldsymbol{x}|\Phi, \mathbf{z}') = \Phi - \gamma\big(g(\mathbf{z}'|\Phi) - \boldsymbol{x}\big)\frac{dg(\mathbf{z}'|\Phi)}{d\Phi}$$

Given epoch of size N is denoted as $\boldsymbol{D} = (\boldsymbol{d}^{(1)} = (\boldsymbol{x}^{(1)}, \mathbf{z}'^{(1)}), \boldsymbol{d}^{(2)} = (\boldsymbol{x}^{(2)}, \mathbf{z}'^{(2)}),\ldots, \boldsymbol{d}^{(N)} = (\boldsymbol{x}^{(N)}, \mathbf{z}'^{(N)}))$, the estimation equations of Θ and Φ are extended exactly as epoch estimation at every iteration of SGD:

$$\Theta_\mu^{(k+1)} = \Theta_\mu^{(k)} - \gamma\frac{1}{N}\sum_{i=1}^{N}\mu\left(\boldsymbol{x}^{(i)}\middle|\Theta_\mu^{(k)}\right)\frac{d\mu\left(\boldsymbol{x}^{(i)}\middle|\Theta_\mu^{(k)}\right)}{d\Theta_\mu}$$

$$\Theta_{\boldsymbol{\sigma}^2}^{(k+1)} = \Theta_{\boldsymbol{\sigma}^2}^{(k)} - \gamma\frac{1}{2N}\sum_{i=1}^{N}\left(\mathbf{1} - \left(\boldsymbol{\sigma}^2\left(\boldsymbol{x}^{(i)}\middle|\Theta_{\boldsymbol{\sigma}^2}^{(k)}\right)\right)^{-1}\right)\frac{d\boldsymbol{\sigma}^2\left(\boldsymbol{x}^{(i)}\middle|\Theta_{\boldsymbol{\sigma}^2}^{(k)}\right)}{d\Theta_{\boldsymbol{\sigma}^2}} \quad (3.13)$$

$$\Phi^{(k+1)} = \Phi^{(k)} - \gamma\frac{1}{N}\sum_{i=1}^{N}\left(g\left(\mathbf{z}'^{(i)}\middle|\Phi^{(k)}\right) - \boldsymbol{x}^{(i)}\right)\frac{dg\left(\mathbf{z}'^{(i)}\middle|\Phi^{(k)}\right)}{d\Phi}$$

As a result, VAE trained with SGD is specified as follows:

Initialize $\Theta = (\Theta_\mu, \Theta_{\boldsymbol{\sigma}^2})^T$ and Φ and set $k = 0$.

Repeat

Sampling epoch $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)},\ldots, \boldsymbol{x}^{(N)})$ or receiving epoch $\boldsymbol{X}$ from big data / data stream.

$$\Theta_\mu^{(k+1)} = \Theta_\mu^{(k)} - \gamma\frac{1}{N}\sum_{i=1}^{N}\mu\left(\boldsymbol{x}^{(i)}\middle|\Theta_\mu^{(k)}\right)\frac{d\mu\left(\boldsymbol{x}^{(i)}\middle|\Theta_\mu^{(k)}\right)}{d\Theta_\mu}$$

$$\Theta_{\boldsymbol{\sigma}^2}^{(k+1)} = \Theta_{\boldsymbol{\sigma}^2}^{(k)} - \gamma\frac{1}{2N}\sum_{i=1}^{N}\left(\mathbf{1} - \left(\boldsymbol{\sigma}^2\left(\boldsymbol{x}^{(i)}\middle|\Theta_{\boldsymbol{\sigma}^2}^{(k)}\right)\right)^{-1}\right)\frac{d\boldsymbol{\sigma}^2\left(\boldsymbol{x}^{(i)}\middle|\Theta_{\boldsymbol{\sigma}^2}^{(k)}\right)}{d\Theta_{\boldsymbol{\sigma}^2}}$$

Randomize random epoch $\mathbf{Z} = (\mathbf{z}^{(1)}, \mathbf{z}^{(2)},\ldots, \mathbf{z}^{(N)})$ from standard normal distribution $P(\mathbf{z}) = \mathcal{N}(\mathbf{0}, I)$ with mean $\mathbf{0}$ and identity covariance matrix I. For each randomized data $\mathbf{z}^{(i)}$, let $\mathbf{z}'^{(i)}$ be calculated based on $\mathbf{z}^{(i)}$ so that $\mathbf{z}'^{(i)}$ follows multivariate normal distribution $Q(\mathbf{z}' \mid \mu(\boldsymbol{x} \mid \Theta_\mu), \Sigma(\boldsymbol{x} \mid \Theta_\Sigma)) = \mathcal{N}(\mathbf{z}' \mid \mu(\boldsymbol{x} \mid \Theta_\mu), \Sigma(\boldsymbol{x} \mid \Theta_\Sigma))$ with mean $\mu(\boldsymbol{x} \mid \Theta_\mu)$ and covariance matrix $\Sigma(\boldsymbol{x} \mid \Theta_\Sigma)$ with note that $\Theta_\Sigma = (\Theta_{\boldsymbol{\sigma}^2})_{n\mathrm{x}n}$ is diagonal matrix.

$$\mathbf{z}'^{(i)} = \mu\left(\boldsymbol{x}^{(i)}\middle|\Theta_\mu^{(k)}\right) + \left(\Sigma\left(\boldsymbol{x}^{(i)}\middle|\Theta_\Sigma^{(k)}\right)\right)^{\frac{1}{2}}\mathbf{z}^{(i)}$$

$$\Phi^{(k+1)} = \Phi^{(k)} - \gamma\frac{1}{N}\sum_{i=1}^{N}\left(g\left(\mathbf{z}'^{(i)}\middle|\Phi^{(k)}\right) - \boldsymbol{x}^{(i)}\right)\frac{dg\left(\mathbf{z}'^{(i)}\middle|\Phi^{(k)}\right)}{d\Phi}$$

Increase $k = k + 1$.

Until some terminating conditions are met.

Note, a terminating condition is customized, for example, parameters Θ and Φ are not changed significantly or there is no more coming epoch $\boldsymbol{X}$. Moreover, the index k indicates time point as well as iteration of SGD. Because it is not easy to randomize $\mathbf{z}$ according to normal distribution $Q(\mathbf{z} \mid \mu(\boldsymbol{x} \mid \Theta_\mu), \Sigma(\boldsymbol{x} \mid \Theta_\Sigma)) = \mathcal{N}(\mathbf{z} \mid \mu(\boldsymbol{x} \mid \Theta_\mu), \Sigma(\boldsymbol{x} \mid \Theta_\Sigma))$ with mean $\mu(\boldsymbol{x} \mid \Theta_\mu)$ and covariance matrix $\Sigma(\boldsymbol{x} \mid \Theta_\Sigma)$, there is a trick that simple data $\mathbf{z}$ is randomized firstly by simple normal distribution $P(\mathbf{z}) = \mathcal{N}(\mathbf{0}, I)$ with mean $\mathbf{0}$ and identity covariance matrix I and, then random data $\mathbf{z}'$ is calculated based on $\mathbf{z}$ and $\mu(\boldsymbol{x} \mid \Theta_\mu)$, $\Sigma(\boldsymbol{x} \mid \Theta_\Sigma)$ as follows:

$$\mathbf{z}' = \mu\big(\boldsymbol{x}\big|\Theta_\mu\big) + \big(\Sigma(\boldsymbol{x}|\Theta_\Sigma)\big)^{\frac{1}{2}}\mathbf{z} \quad (3.14)$$

Such that $\mathbf{z}'$ follows normal distribution $\mathcal{N}(\mathbf{z}' \mid \mu(\boldsymbol{x} \mid \Theta_\mu), \Sigma(\boldsymbol{x} \mid \Theta_\Sigma))$ with mean $\mu(\boldsymbol{x} \mid \Theta_\mu)$ and covariance matrix $\Sigma(\boldsymbol{x} \mid \Theta_\Sigma)$ according to some rule of normal distribution in applied statistics (Hardle

& Simar, 2013, p. 157). The notation $A = \Sigma(\boldsymbol{x} \mid \Theta_\Sigma)^{1/2}$ implies $AA = \Sigma(\boldsymbol{x} \mid \Theta_\Sigma)$ and so, we can consider it as square root of $\Sigma(\boldsymbol{x} \mid \Theta_\Sigma)$. Calculating this square root is not so easy because of complexity of singular decomposition for calculating it. Fortunately, it is easier to calculate the square root when Θ_Σ was simplified by diagonal elements $(\sigma^2(\boldsymbol{x} \mid \Theta_\Sigma))_{nxn}$. Indeed, we have:

$$\left(\Sigma(\boldsymbol{x}|\Theta_\Sigma)\right)^{\frac{1}{2}} = \left(\left(\boldsymbol{\sigma}^2(\boldsymbol{x}|\Theta_\Sigma)\right)_{nxn}\right)^{\frac{1}{2}} = \begin{pmatrix} \sqrt{\sigma_1^2(\boldsymbol{x}|\Theta_\Sigma)} & 0 & \cdots & 0 \\ 0 & \sqrt{\sigma_2^2(\boldsymbol{x}|\Theta_\Sigma)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{\sigma_n^2(\boldsymbol{x}|\Theta_\Sigma)} \end{pmatrix}$$

Where,

$$\Theta_\Sigma = (\Theta_{\boldsymbol{\sigma}^2})_{nxn} = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n^2 \end{pmatrix}$$

Following figure depicts VAE.

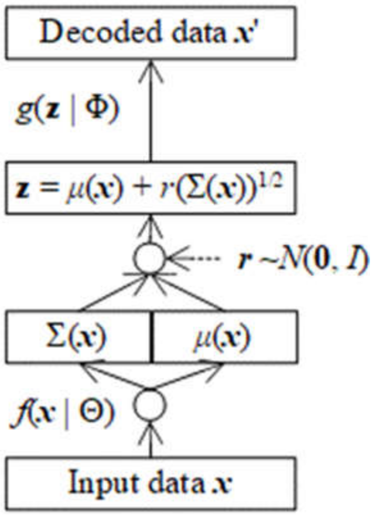

Figure 3. 1. Variational Autoencoders (VAE).

There is a question that how to calculate the differentials $d\mu(\boldsymbol{x} \mid \Theta_\mu) / d\Theta_\mu$, $d\sigma^2(\boldsymbol{x} \mid \Theta_{\boldsymbol{\sigma}^2}) / d\Theta_{\boldsymbol{\sigma}^2}$, and $dg(\mathbf{z}' \mid \Phi) / d\Phi$. Indeed, it is not difficult to calculate them in context of neural network associated with backpropagation algorithm so that the last output layer as well as last neuron o of any DNN $f(\boldsymbol{x} \mid \Theta)$ or $g(\mathbf{z} \mid \Phi)$ is acted by activation function $a(.)$ as follows:

$$a(o) = a(\boldsymbol{w}^T\boldsymbol{i})$$
$$o = \boldsymbol{w}^T\boldsymbol{i}$$

Where $\boldsymbol{i}$ is input of the last layer o and weight parameter $\boldsymbol{w}$ is a part of entire parameter Θ or Φ and hence, we need to focus on calculating differential $da(o) / d\boldsymbol{w}$ which is equivalent to any differential $d\mu(\boldsymbol{x} \mid \Theta_\mu) / d\Theta_\mu$, $d\sigma^2(\boldsymbol{x} \mid \Theta_{\boldsymbol{\sigma}^2}) / d\Theta_{\boldsymbol{\sigma}^2}$, or $dg(\mathbf{z}' \mid \Phi) / d\Phi$ so that backpropagation algorithm will solve remaining parts of entire parameter Θ or Φ.

$$\frac{da(o)}{d\boldsymbol{w}} \cong \text{any differential}$$

Indeed, we have:

$$\frac{da(o)}{d\boldsymbol{w}} = \frac{da(\boldsymbol{w}^T\boldsymbol{i})}{d\boldsymbol{w}} = \frac{da(o)}{do}\boldsymbol{i}^T = a'(o)\boldsymbol{i}^T$$

Note, the subscript "*T*" denotes transposition operator of vector and matrix in which row vector becomes column vector and vice versa. It is easy to calculate the derivative $a'(o)$ when activation function was specified, for instance, if $a(o)$ is sigmoid function, we have:

$$a(o) = y = \frac{1}{1+e^{-o}}$$

$$a'(o) = \frac{1}{1+e^{-o}}\left(1 - \frac{1}{1+e^{-o}}\right) = a(o)\big(1-a(o)\big) = y(1-y)$$

In practice, y is replaced by $a(y)$ in order to prevent o from being out of space:

$$a'(o) \cong a(y)\big(1-a(y)\big) = a\big(a(o)\big)\Big(1-a\big(a(o)\big)\Big)$$

As a result, we have:

$$\text{any differential} \cong a\big(a(o)\big)\Big(1-a\big(a(o)\big)\Big)\boldsymbol{i}^T$$

For fast computation, it is possible to set the derivative $a'(o)$ to be small enough constants like 1 such that any differential is $\boldsymbol{i}^T$.

Given epoch $\boldsymbol{D} = (\boldsymbol{d}^{(1)} = (\boldsymbol{x}^{(1)}, \mathbf{z}^{(1)}), \boldsymbol{d}^{(2)} = (\boldsymbol{x}^{(2)}, \mathbf{z}^{(2)}),\ldots, \boldsymbol{d}^{(N)} = (\boldsymbol{x}^{(N)}, \mathbf{z}^{(N)}))$ implies that the epoch is created or sent by equilateral distribution $1/N$ but in general case, $\boldsymbol{D}$ can follow an arbitrary distribution denoted by PDF $P(\boldsymbol{d})$, which makes the optimization problem and the SGD estimation changed a little bit by theoretical expectation given distribution $P(\boldsymbol{d})$.

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}}\, E\left(\mathrm{KL}\big(Q(\boldsymbol{z}|\Theta,\boldsymbol{x})\big|P(\boldsymbol{z})\big)\Big|P(\boldsymbol{d})\right)$$

$$\Phi^* = \underset{\Phi}{\operatorname{argmax}}\, E\big(\log P(\boldsymbol{x}|\Phi,\boldsymbol{z})\,\big|P(\boldsymbol{d})\big)$$

Where,

$$E\left(\mathrm{KL}\big(Q(\boldsymbol{z}|\Theta,\boldsymbol{x})\big|P(\boldsymbol{d})\big)\Big|P(\boldsymbol{d})\right) = \int_{\boldsymbol{d}} \mathrm{KL}\big(Q(\boldsymbol{z}|\Theta,\boldsymbol{x})\big|P(\boldsymbol{d})\big)P(\boldsymbol{d})d\boldsymbol{d}$$

$$E\big(\log P(\boldsymbol{x}|\Phi,\boldsymbol{z})\,\big|P(\boldsymbol{d})\big) = \int_{\boldsymbol{d}} \log P(\boldsymbol{x}|\Phi,\boldsymbol{z})\,P(\boldsymbol{d})d\boldsymbol{d}$$

However, there is no significant change in aforementioned practical technique to estimate parameters.

Recall that the default artificial neural network is feedforward neural network where data is fed to input layer which, in turn, is evaluated and passed across hidden layers to output layer in one-way direction, finally. However, there is an extension of neural network, which is called recurrent neural work (RNN), where an output can be turned back in order to feed on network as input. In other words, RNN has circle, which allow that output can become input. There are many kinds of RNN, for instance, long short-term memory is a case of RNN aforementioned. Boltzmann machine (Wikipedia, Boltzmann machine, 2004) is another variant of RNN, in which there is no separation of inputs from outputs. Like Hopfield network (Wikipedia, Hopfield network, 2004), every neuron (unit) in Boltzmann machine connects to all remaining neurons. In other words, Boltzmann machine applies an interesting aspect that all input neurons are output neurons too.

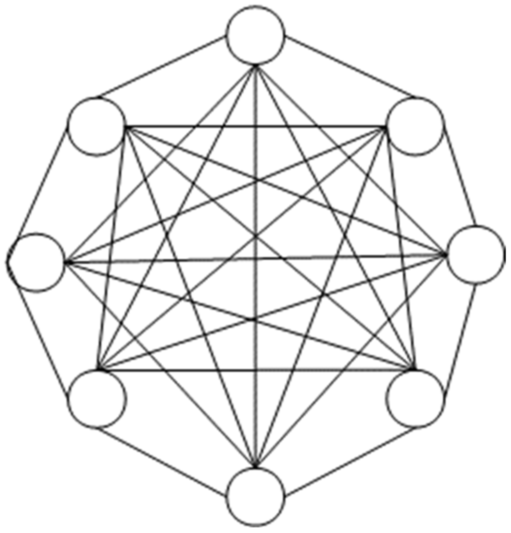

Figure 3. 2. Topology of Hopfield network and Boltzmann machine.

Boltzmann machine named by the name of Austrian physicist Ludwig Eduard Boltzmann, also called Sherrington-Kirkpatrick model with external field or stochastic Ising-Lenz-Little model, is a stochastic spin-glass model with an external field and classified as a Markov random filed too. For easy explanation, Boltzmann machine simulates spinning glass process or annealing metal process, in which melt glass or melt metal will be frozen or get stable at some energy and some temperature where such energy and temperature are called stable energy and stable temperature at stable state of glass. The annealing process aims to reach the stable state of metal (glass) at which time the metal is frozen. Given concrete temperature, the smaller the energy is, the more stable the metal state is. Similarly, given concrete energy, the smaller the temperature is, the more stable the metal state is. Therefore, annealing process is cooling process where probability of metal state, which is proportional to energy and temperature, follows the so-called Boltzmann distribution specified as follows:

$$P(s)=\frac{\exp\left(-\frac{E(s)}{\kappa T}\right)}{\sum_{i=1}^{M}\exp\left(-\frac{E(s)}{\kappa T}\right)} \quad (3.15)$$

Where $P(s)$ is probability of current state s and $E(s)$ is energy applied to metal at state s given temperature T while κ is Boltzmann constant and M is the number of states. Note, T can be considered as a parameter. If the denominator is constant, Boltzmann probability is approximated as follows:

$$P(s)\cong\exp\left(-\frac{E(s)}{\kappa T}\right)$$

In annealing process, if next energy is concerned by observing current energy because of successive annealing process, energy deviation or energy difference $\Delta E(s, s_{new})$ between current energy $E(s)$ and next energy $E(s_{new})$ is concerned so that Boltzmann probability derives a so-called acceptance probability $P(s, s_{new}, T)$ as follows:

$$P(s,s_{new},T)=\begin{cases}1 \text{ if } \Delta E(s,s_{new})<0\\ \exp\left(-\frac{\Delta E(s,s_{new})}{\kappa T}\right) \text{ otherwise}\end{cases}$$

Where,

$$\Delta E(s,s_{new})=E(s_{new})-E(s)$$

Given a certain temperature T, the larger the acceptance probability is, the higher likely the annealing process stops, the higher the likelihood of stability is. In other words, acceptance probability $P(s, s_{new}, T)$ decides whether or not the new state s_{new} is moved next in annealing process. When applied into solving optimization problem as well as learning problem, simulated annealing (SA) algorithm codes candidate solution as states. Indeed, SA is iterative process including many enough iterations where SA decreases temperature T at each iteration and then, randomize a new state s_{new} and calculates energy $E(s_{new})$ of the new state. Whether or not the new state (new candidate solution) s_{new} is based on the acceptance probability $P(s, s_{new}, T)$ based on current state s, new state s_{new}, current temperature T. If the new candidate solution s_{new} is selected as current solution, SA will decrease temperature in the next iteration. Following is pseudo code of SA:

```
Initialize current temperature $T$ by highest temperature $T_0$ as $T = T_0$.
Repeat
Decrease current temperature, for example, $T$ = decrease($T$).
Select a random neighbor of current state as $s_{new}$ = neighbor($s$).
If $P(s, s_{new}, T)$ is larger than a predefined threshold then
    $s = s_{new}$
End if
Until terminating conditions are met.
```

The terminating conditions can be that best state (solution) is reached, the current state s is good enough, the current temperature T is low enough, or the number of iterations is large enough. A usual, given a maximum iteration number K and the current iteration number k, the temperature decreasing function can be defined as follows:

$$\text{decrease}(T) = T - \frac{k}{K}T$$

It is easy to infer that it is possible to set the initial temperature to be the maximum number of iterations as $T_0 = K$ in practice. There is no significant change when applying SA into training Boltzmann machine where the most important problem is how to specify energy of Boltzmann machine. Fortunately, global energy of Boltzmann machine inherits from global energy of Hopfield network because Boltzmann machine is a type of Hopfield network which in turn is a variant of RNN. Suppose an entire Boltzmann machine is represented by a vector $\boldsymbol{x} = (x_1, x_2,\ldots, x_n)$ in which each x_i is a neuron or unit. It is exact that a certain state of Boltzmann machine is represented by $\boldsymbol{x}$ which is evaluated at certain time point. It is possible to denote current state of Boltzmann machine as $\boldsymbol{x}$ instead. For convenience, the next state of Boltzmann machine is denoted $\boldsymbol{x}'$. Energy $E(\boldsymbol{x})$ of Boltzmann machine at state $\boldsymbol{x}$ is defined based on global energy of Hopfield network as follows (Hinton, 2007, p. 2):

$$E(\boldsymbol{x}) = -\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} w_{ij}x_ix_j - \sum_{i=1}^{n} b_ix_i$$

Note, w_{ij} is weight between neuron x_i and neuron x_j whereas b_i is bias of neuron x_i. As usual, biases b_i are considered as parameters like weights w_{ij}. Because there are $n(n–1)/2$ connections as well as $n(n–1)/2$ weights, the equation of energy is rewritten for convenience as follows (Wikipedia, Boltzmann machine, 2004):

$$E(\boldsymbol{x}) = -\sum_{i<j} w_{ij}x_ix_j - \sum_{i} b_ix_i \tag{3.16}$$

All weights w_{ij} compose weight matrix $\boldsymbol{W} = (w_{ij})_{n\text{x}n}$ whose elements on diagonal are zero. Note, $\boldsymbol{W}$ is $n\text{x}n$ symmetric matrix.

$$w_{ij} = w_{ji}, \forall i, j$$
$$w_{ii} = 0, \forall i \text{ by convention}$$

Every neuron x_i is evaluated by propagation rule:

$$x_i = \sum_{j=1,j\neq i}^{n} w_{ij}x_j$$

Neurons in traditional Boltzmann machine are binary variables such that xi belongs to {0, 1} but it is extended to allow neurons *x_i* to belong to arbitrary real interval and so, suppose every x_i ranges in interval [0, 1] without loss of generality. Rectified Linear Unit (ReLU) function is used to ramp x_i in interval [0, 1] so as to modify the propagation rule a little bit but learning algorithm mentioned later is not changed because the first-order derivative of ReLU function within valid domain [0, 1] is 1.

$$x_i = \text{ReLU}\left(\sum_{j=1,j\neq i}^{n} w_{ij}x_j\right)$$

Where

$$\text{ReLU}(x) = \begin{cases} x \text{ if } 0 \leq x \leq 1 \\ 0 \text{ if } x < 0 \\ 1 \text{ if } x > 1 \end{cases}$$

It implies:

$$x_i = \text{ReLU}(x_i)$$

So that the propagation rule is not changed in theory:

$$x_i = \sum_{j=1,j\neq i}^{n} w_{ij}x_j$$

Based on definition of global energy, Boltzmann probability density function (PDF) of Boltzmann machine is determined as follows:

$$P(\boldsymbol{x}|\boldsymbol{W},\boldsymbol{b}) = \frac{\exp\left(-\frac{E(\boldsymbol{x}|\boldsymbol{W},\boldsymbol{b})}{\kappa T}\right)}{\int_{\boldsymbol{x}} \exp\left(-\frac{E(\boldsymbol{x}|\boldsymbol{W})}{\kappa T}\right) d\boldsymbol{x}}$$

Recall that:

$$E(\boldsymbol{x}|\boldsymbol{W},\boldsymbol{b}) = -\sum_{i<j} w_{ij}x_ix_j - \sum_i b_ix_i$$

$$\boldsymbol{b} = (b_1, b_2, \dots, b_n)^T$$

Within context of DGM, such PDF is generator likelihood whose parameter is Φ = (*W*, ***b***).

$$P(\boldsymbol{x}|\Phi) = P(\boldsymbol{x}|\boldsymbol{W},\boldsymbol{b}) = \frac{\exp\left(-\frac{E(\boldsymbol{x}|\boldsymbol{W},\boldsymbol{b})}{\kappa T}\right)}{\int_{\boldsymbol{x}} \exp\left(-\frac{E(\boldsymbol{x}|\boldsymbol{W},\boldsymbol{b})}{\kappa T}\right) d\boldsymbol{x}}$$

Because the denominator is constant with regard to *W* and ***b***, Boltzmann PDF is approximated as follows:

$$P(\boldsymbol{x}|\boldsymbol{W},\boldsymbol{b}) \cong \exp\left(-\frac{E(\boldsymbol{x}|\boldsymbol{W},\boldsymbol{b})}{\kappa T}\right) \quad (3.17)$$

For learning Boltzmann, maximum likelihood estimation (MLE) method (Goodfellow, Bengio, & Courville, Deep Learning, 2016, p. 655) is applied into estimating weight parameter ***W*** and bias parameter ***b*** by maximizing Boltzmann PDF with regard to w_{ij} and b_i.

$$w_{ij}^* = \max_{w_{ij}} P(\boldsymbol{x}|\boldsymbol{W},\boldsymbol{b})$$

$$b_i^* = \max_{b_i} P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b})$$

By taking natural logarithm of Boltzmann PDF, the optimization becomes easier to be solved.

$$w_{ij}^* = \max_{w_{ij}} \log P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b})$$

$$b_i^* = \max_{b_i} \log P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b})$$

Where log$P(\boldsymbol{x} \mid \boldsymbol{W}, \boldsymbol{b})$ is called Boltzmann log-likelihood or Boltzmann log-PDF.

$$\log P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b}) \cong -\frac{E(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b})}{\kappa T}$$

The first-order partial derivatives of Boltzmann log-likelihood are:

$$\nabla_{w_{ij}} \log P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b}) \cong \frac{\partial \log P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b})}{\partial w_{ij}} = \frac{\partial \log E(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b})}{\partial w_{ij}} = \frac{x_i x_j}{\kappa T}$$

$$\nabla_{b_i} \log P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b}) \cong \frac{\partial \log P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b})}{\partial b_i} = \frac{\partial \log E(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b})}{\partial b_i} = \frac{x_i}{\kappa T}$$

As a convention, these first-order partial derivatives are called (partial) gradients. By applying stochastic gradient descent (SGD) method into estimating w_{ij} and b_i given Boltzmann log-likelihood, we have:

$$\begin{gathered} w_{ij} = w_{ij} + \gamma \nabla_{w_{ij}} \log P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b}) = w_{ij} + \gamma \frac{x_i x_j}{\kappa T} \\ b_i = b_i + \gamma \nabla_{b_i} \log P(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b}) = b_i + \gamma \frac{x_i}{\kappa T} \end{gathered} \quad (3.18)$$

Where $0 < \gamma \leq 1$ is learning rate. It is easy to recognize that the estimation equations above confirm Hebbian learning rule in which the strength of connection represented by weight is consolidated by agreement of two nodes to which the connection is attached. As a result, Boltzmann machine trained with SGD is specified as follows:

Initialize $\boldsymbol{W}$ and set $k = 0$.

Repeat

Data (state) $\boldsymbol{x}$ is received from some real sample, or it can be kept intact.

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} + \gamma \frac{x_i x_j}{\kappa T}$$

$$b_i^{(k+1)} = b_i^{(k)} + \gamma \frac{x_i}{\kappa T}$$

Increase $k = k + 1$.

Until some terminating conditions are met.

Note, a terminating condition is customized, for example, parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ are not changed significantly, the maximum number of iterations is reached, or Boltzmann machine gets stable. The terminating condition that Boltzmann machine gets stable receives more concerns because stability is important property of spinning glass process or annealing process that Boltzmann machine. However, checking the stability in which global energy $E(\boldsymbol{x})$ is not changed may consume a lot of iterations. Fortunately, SA can be incorporated into SGD so as to derive a more effective estimation. Boltzmann machine trained with SGD and SA is specified as follows:

Initialize current temperature T by highest temperature T_0 as $T = T_0$.

Repeat

Data (state) $\boldsymbol{x}$ is received from some real sample, or it can be kept intact.

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} + \gamma \frac{x_i x_j}{\kappa T}$$
$$b_i^{(k+1)} = b_i^{(k)} + \gamma \frac{x_i}{\kappa T}$$

Evaluate Boltzmann machine given current parameter $\boldsymbol{W}^{(k+1)}$ and $\boldsymbol{b}^{(k+1)}$ so as to produce a new state $\boldsymbol{x}'$:

$$x_i' = \text{ReLU}\left(\sum_{j=1, j\neq i}^{n} w_{ij}^{(k+1)} x_j\right), \text{for all } x_i' \text{ not from real sample}$$

If $P(\boldsymbol{x}, \boldsymbol{x}', T \mid \boldsymbol{W}^{(k+1)}, \boldsymbol{b}^{(k+1)})$ is larger than a predefined threshold then

$\boldsymbol{x} = \boldsymbol{x}'$

Decrease current temperature, for example, T = decrease(T).

End if

Increase $k = k + 1$.

Until terminating conditions are met.

The terminating conditions can be that best state ($\boldsymbol{x}'$) is reached, the current state $\boldsymbol{x}$ is good enough, or the current temperature T is low enough. These terminating conditions reflect the stable state of Boltzmann machine. A usual, given a maximum iteration number K and the current iteration number k, the temperature decreasing function can be defined as follows:

$$\text{decrease}(T) = T - \frac{k}{K} T$$

Of course, the acceptance probability is:

$$P(\boldsymbol{x}, \boldsymbol{x}', T | \boldsymbol{W}, \boldsymbol{b}) = \begin{cases} 1 \text{ if } \Delta E(\boldsymbol{x}, \boldsymbol{x}' | \boldsymbol{W}, \boldsymbol{b}) < 0 \\ \exp\left(-\dfrac{\Delta E(\boldsymbol{x}, \boldsymbol{x}' | \boldsymbol{W}, \boldsymbol{b})}{\kappa T}\right) \text{ otherwise} \end{cases} \quad (3.19)$$

Where,

$$\Delta E(\boldsymbol{x}, \boldsymbol{x}' | \boldsymbol{W}, \boldsymbol{b}) = E(\boldsymbol{x}' | \boldsymbol{W}, \boldsymbol{b}) - E(\boldsymbol{x} | \boldsymbol{W}, \boldsymbol{b})$$

There is a so-called restricted Boltzmann machine (RBM) in which neurons are separated into two groups such as input group denoted $\boldsymbol{x}_z$ and hidden group denoted $\boldsymbol{x}_x$.

$$\boldsymbol{x} = (\boldsymbol{x}_z, \boldsymbol{x}_x)^T$$

The training algorithm by incorporation of SA and SGD is not changed except that there is neither connection between input neurons and input neurons nor connection between hidden neurons and hidden neurons. In other words, all connections are made between input group and hidden group, for instance, suppose cardinality of input group is k then, the number of connections is $k(n - k)$. Therefore, the two groups are considered layers such as input layer and hidden layer. Of course, both layers are output layers because connections in Boltzmann machine are two-direction connections whereas feed-forward neural network accepts only one-direction connections. RBM is trained faster than traditional Boltzmann machine because its number of connections is smaller. Moreover, it is clear to apply RBM into DGM because generator function in DGM $\boldsymbol{x} = g(\mathbf{z} \mid \Phi)$ is modeled by RBM whose input is input group $\boldsymbol{x}_z$ and whose output is output group $\boldsymbol{x}_x$ such as $\boldsymbol{x}_x = g(\boldsymbol{x}_z \mid \boldsymbol{W}, \boldsymbol{b})$ where $\boldsymbol{x}_x$ is calculated by evaluating RBM given input $\boldsymbol{x}_z$.

$$\boldsymbol{x}_x = g(\boldsymbol{x}_z | \boldsymbol{W}, \boldsymbol{b}) \stackrel{\text{def}}{=} \text{evaluate RBM at } \boldsymbol{x}_z \text{ to produce } \boldsymbol{x}_x$$

The reason that the RBM approach for DGM is classified into approximate density DGM is that generator likelihood $P(\boldsymbol{x} \mid \boldsymbol{W}, \boldsymbol{b})$ is defined indirectly based on the energy $E(\boldsymbol{x} \mid \boldsymbol{W}, \boldsymbol{b})$. Of course, $\boldsymbol{x}_z$ is randomized such that $\boldsymbol{x}_x$ is generated data.

4. Implicit Density DGM

According to implicit density approach, DGMs do not specify explicitly generator PDF $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$, which does not means that such PDF is not existent but it is simple that such PDF is not concerned. Note, PDF is abbreviation of probability density function.

Recall that there are two problems related to construct a DGM: 1) how to define likelihood or error to train generator DNN $g(\mathbf{z} \mid \Phi)$ and 2) how to define tractable PDF $P(\mathbf{z})$ which implies the way to randomize **z**. The second problem relates to assert qualification of random data **z**′ and hence, the second problem is stated as qualification problem of how to qualify random data. However, it is essential that the qualification problem aims to improve generator DNN $g(\mathbf{z} \mid \Phi)$ because $g(\mathbf{z} \mid \Phi)$ translate intractable **z** into tractable $\boldsymbol{x}$. **Generative Adversarial Network (GAN)** developed by Goodfellow et al. (Goodfellow, et al., 2014) aims to reinforcing quality of generator DNN $g(\mathbf{z} \mid \Phi) = \boldsymbol{x}' \approx \boldsymbol{x}$ by adding a so-called discriminator which is a discrimination function $d(\boldsymbol{x} \mid \Psi): \boldsymbol{x} \rightarrow [0, 1]$ from concerned data $\boldsymbol{x}$ or $\boldsymbol{x}'$ to range [0, 1] in which $d(\boldsymbol{x} \mid \Psi)$ can distinguish fake data from real data. In other words, the larger result the discriminator $d(\boldsymbol{x}' \mid \Psi)$ derives, the more realistic the generated data $\boldsymbol{x}'$ is. Obviously, discriminator $d(\boldsymbol{x} \mid \Psi)$ is implemented by a DNN whose weights are Ψ called discriminator parameter with note that this discriminator DNN has only one output neuron denoted d_0.

$$d: R^m \rightarrow [0,1] \text{ such that } d_0 = d(\boldsymbol{x}|\Psi) \text{ where } d_0 \in [0,1], \boldsymbol{x} \in X \subseteq R^m \quad (4.1)$$

Actually, the task of discriminator $d(\boldsymbol{x} \mid \Psi)$ is classification task with regard to class d_0 belonging to interval [0, 1]. GAN does not establish explicitly PDFs of generator $g(\mathbf{z} \mid \Phi)$ and discriminator $d(\boldsymbol{x} \mid \Psi)$ such as $P(\boldsymbol{x} \mid \Phi, \mathbf{z})$ and $P(d_0 \mid \Psi, \boldsymbol{x})$ and hence, GAN does not define explicitly and separately likelihoods / errors of $g(\mathbf{z} \mid \Phi)$ and $d(\boldsymbol{x} \mid \Psi)$ too. Indeed, GAN instead unifies optimization constraints of $g(\mathbf{z} \mid \Phi)$ and $d(\boldsymbol{x} \mid \Psi)$ into a target function $l(\Phi, \Psi \mid \boldsymbol{x}, \mathbf{z})$.

$$l(\Phi, \Psi|\boldsymbol{x}, \boldsymbol{z}) = \log\big(d(\boldsymbol{x}|\Psi)\big) + \log\big(1 - d(g(\boldsymbol{z}|\Phi)|\Psi)\big) \quad (4.2)$$

Indeed, target function $l(\Phi, \Psi \mid \boldsymbol{x}, \mathbf{z})$ is error function and so it is called loss function in literature. As a result, GAN tries to optimize dually generator parameter Φ and discriminator parameter Ψ so that optimal estimate Φ^* and optimal estimate Ψ^* are minimizer and maximizer of loss function $l(\Phi, \Psi \mid \boldsymbol{x}, \mathbf{z})$ with expectation that Nash equilibrium will be achieved at the saddle point (Φ^*, Ψ^*) with note that loss function $l(\Phi, \Psi \mid \boldsymbol{x}, \mathbf{z})$ is function of Φ and Ψ given data $\boldsymbol{x}$ and **z**.

$$\begin{aligned}
\Phi^* &= \underset{\Phi}{\operatorname{argmin}}\, l(\Phi, \Psi^*|\boldsymbol{x}, \boldsymbol{z}) = \underset{\Phi}{\operatorname{argmin}} \log\big(1 - d(g(\boldsymbol{z}|\Phi)|\Psi^*)\big) \\
\Psi^* &= \underset{\Psi}{\operatorname{argmax}}\, l(\Phi^*, \Psi|\boldsymbol{x}, \boldsymbol{z}) = \underset{\Psi}{\operatorname{argmax}}\Big(\log\big(d(\boldsymbol{x}|\Psi)\big) + \log\big(1 - d(g(\boldsymbol{z}|\Phi^*)|\Psi)\big)\Big)
\end{aligned} \quad (4.3)$$

This is min-max problem in game theory (Goodfellow, et al., 2014):

$$\min_{\Phi} \max_{\Psi} l(\Phi, \Psi|\boldsymbol{x}, \boldsymbol{z})$$

Which results estimation equation according to stochastic gradient descent (SGD) method:

$$\Phi = \Phi - \gamma \nabla_{\Phi} l(\Phi, \Psi^*|\boldsymbol{x}, \boldsymbol{z})$$
$$\Psi = \Psi + \gamma \nabla_{\Psi} l(\Phi^*, \Psi|\boldsymbol{x}, \boldsymbol{z})$$

Where γ is learning rate. Recall that SGD, which is an iterative process, pushes candidate solution at each iteration along the direction which is opposite to gradient of target function for minimization or has the same direction to gradient of target function for maximization with note that the step length is represented by learning rate. Note, $\nabla_\Phi l(\Phi, \Psi^* \mid \boldsymbol{x}, \mathbf{z})$ is gradient of loss function $l(\Phi, \Psi^* \mid \boldsymbol{x}, \mathbf{z})$ fixed Ψ^* with regard to generator parameter Φ and $\nabla_\Psi l(\Phi^*, \Psi \mid \boldsymbol{x}, \mathbf{z})$ is gradient of loss function $l(\Phi^*, \Psi \mid \boldsymbol{x}, \mathbf{z})$ fixed Φ^* with regard to discriminator parameter Ψ as follows:

$$\nabla_{\Phi} l(\Phi, \Psi^*|\boldsymbol{x}, \boldsymbol{z}) = \frac{dl(\Phi, \Psi^*|\boldsymbol{x}, \boldsymbol{z})}{d\Phi} = \frac{\partial \log\big(1 - d(g(\boldsymbol{z}|\Phi)|\Psi^*)\big)}{\partial \Phi}$$
$$\nabla_{\Psi} l(\Phi^*, \Psi|\boldsymbol{x}, \boldsymbol{z}) = \frac{dl(\Phi^*, \Psi|\boldsymbol{x}, \boldsymbol{z})}{d\Psi} = \frac{\partial\Big(\log\big(d(\boldsymbol{x}|\Psi)\big) + \log\big(1 - d(g(\boldsymbol{z}|\Phi^*)|\Psi)\big)\Big)}{\partial \Psi}$$

Therefore, the estimation equation is rewritten as follows:

$$\begin{aligned}\Phi &= \Phi - \gamma\nabla_\Phi \log\big(1 - d(g(\boldsymbol{z}|\Phi)|\Psi^*)\big)\\ \Psi &= \Psi + \gamma\nabla_\Psi\big(\log\big(d(\boldsymbol{x}|\Psi)\big) + \log\big(1 - d(g(\boldsymbol{z}|\Phi^*)|\Psi)\big)\big)\end{aligned} \tag{4.4}$$

According to equations above, real data x aims to maximize discriminator $d(x \mid \Psi)$ and in opposite, generated data $x' = g(\mathbf{z} \mid \Phi)$ aims to minimize discriminator $d(x' \mid \Psi)$. Although both GAN and VAE use two DNNs for data generation but the underlying theory of GAN is slightly more succinct than VAE because there is no requirement of specifying probabilistic distribution $P(\mathbf{z})$ of tractable $\mathbf{z}$. As a convention, the gradient $\nabla_\Phi(\log(1 - d(g(\mathbf{z} \mid \Phi) \mid \Psi^*)))$ related to generator parameter Θ is called generator gradient and the gradient $\nabla_\Psi(\log(d(x \mid \Psi)) + \log(1 - d(g(\mathbf{z} \mid \Phi^*) \mid \Psi)))$ related to discriminator parameter Θ is called discriminator gradient.

Given epoch of size N is denoted as $\boldsymbol{D} = ((x^{(1)}, \mathbf{z}^{(1)}), (x^{(2)}, \mathbf{z}^{(2)}),\ldots, (x^{(N)}, \mathbf{z}^{(N)}))$, the estimation equations of Φ and Ψ are extended exactly as epoch estimation at every iteration of SGD:

$$\Phi^{(k+1)} = \Phi^{(k)} - \gamma\nabla_\Phi \frac{1}{N}\sum_{i=1}^{N} \log\left(1 - d\big(g\big(\boldsymbol{z}^{(i)}\big|\Phi^{(k)}\big)\big|\Psi^{(k)}\big)\right)$$

$$\Psi^{(k+1)} = \Psi^{(k)} + \gamma\nabla_\Psi \frac{1}{N}\sum_{i=1}^{N} \left(\log\left(d\big(\boldsymbol{x}^{(i)}\big|\Psi^{(k)}\big)\right) + \log\left(1 - d\big(g\big(\boldsymbol{z}^{(i)}\big|\Phi^{(k)}\big)\big|\Psi^{(k)}\big)\right)\right)$$

As a result, GAN trained with SGD is specified as follows:

Initialize Φ and Ψ and set $k = 0$.

Repeat

Sampling epoch $X = (x^{(1)}, x^{(2)},\ldots, x^{(N)})$ or receiving epoch X from big data / data stream.

Randomize random epoch $Z = (\mathbf{z}^{(1)}, \mathbf{z}^{(2)},\ldots, \mathbf{z}^{(N)})$ from standard normal distribution $P(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, I)$ with mean $\mathbf{0}$ and identity covariance matrix I.

$$\Phi^{(k+1)} = \Phi^{(k)} - \gamma\nabla_\Phi \frac{1}{N}\sum_{i=1}^{N} \log\left(1 - d\big(g\big(\boldsymbol{z}^{(i)}\big|\Phi^{(k)}\big)\big|\Psi^{(k)}\big)\right)$$

$$\Psi^{(k+1)} = \Psi^{(k)} + \gamma\nabla_\Psi \frac{1}{N}\sum_{i=1}^{N} \left(\log\left(d\big(\boldsymbol{x}^{(i)}\big|\Psi^{(k)}\big)\right) + \log\left(1 - d\big(g\big(\boldsymbol{z}^{(i)}\big|\Phi^{(k)}\big)\big|\Psi^{(k)}\big)\right)\right)$$

Increase $k = k + 1$.

Until some terminating conditions are met.

Note, a terminating condition is customized, for example, parameters Φ and Ψ are not changed significantly or there is no more coming epoch X. Moreover, the index k indicates time point as well as iteration of SGD.

Recall that the estimation equations of generator parameter Φ and discriminator parameter Ψ are:

$$\begin{aligned}\Phi &= \Phi - \gamma\nabla_\Phi \log\big(1 - d(g(\boldsymbol{z}|\Phi)|\Psi^*)\big)\\ \Psi &= \Psi + \gamma\nabla_\Psi\big(\log\big(d(\boldsymbol{x}|\Psi)\big) + \log\big(1 - d(g(\boldsymbol{z}|\Phi^*)|\Psi)\big)\big)\end{aligned}$$

It is necessary to calculate generator gradient and discriminator gradient. Indeed, we have:

$$\nabla_\Phi \log\big(1 - d(g(\boldsymbol{z}|\Phi)|\Psi^*)\big) = \frac{\partial \log\big(1 - d(g(\boldsymbol{z}|\Phi)|\Psi^*)\big)}{\partial\Phi} = -\frac{\partial d(g(\boldsymbol{z}|\Phi)|\Psi^*)/\partial\Phi}{1 - d(g(\boldsymbol{z}|\Phi)|\Psi^*)}$$

$$\begin{aligned}&\nabla_\Psi\big(\log\big(d(\boldsymbol{x}|\Psi)\big) + \log\big(1 - d(g(\boldsymbol{z}|\Phi^*)|\Psi)\big)\big)\\ &\quad= \frac{\partial\big(\log\big(d(\boldsymbol{x}|\Psi)\big) + \log\big(1 - d(g(\boldsymbol{z}|\Phi^*)|\Psi)\big)\big)}{\partial\Psi}\\ &\quad= \frac{\partial d(\boldsymbol{x}|\Psi)/\partial\Psi}{d(\boldsymbol{x}|\Psi)} - \frac{\partial d(g(\boldsymbol{z}|\Phi^*)|\Psi)/\partial\Psi}{1 - d(g(\boldsymbol{z}|\Phi^*)|\Psi)}\end{aligned}$$

Where ∂*d*(.) / ∂Φ and ∂*d*(.) / ∂Ψ denotes differentials of discriminator function with regard to Φ and Ψ, respectively.

Given epoch of size *N* is denoted as $\boldsymbol{D} = ((\boldsymbol{x}^{(1)}, \mathbf{z}^{(1)}), (\boldsymbol{x}^{(2)}, \mathbf{z}^{(2)}),\ldots, (\boldsymbol{x}^{(N)}, \mathbf{z}^{(N)}))$, the estimation equations of Φ and Ψ are extended exactly as epoch estimation at every iteration of SGD:

$$\begin{gathered}\Phi^{(k+1)} = \Phi^{(k)} + \gamma\frac{1}{N}\sum_{i=1}^{N}\frac{\partial d\left(g\left(\boldsymbol{z}^{(i)}\middle|\Phi^{(k)}\right)\middle|\Psi^{(k)}\right)/\partial\Phi}{1-d\left(g\left(\boldsymbol{z}^{(i)}\middle|\Phi^{(k)}\right)\middle|\Psi^{(k)}\right)}\\ \Psi^{(k+1)} = \Psi^{(k)} + \gamma\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial d\left(\boldsymbol{x}^{(i)}\middle|\Psi^{(k)}\right)/\partial\Psi}{d\left(\boldsymbol{x}^{(i)}\middle|\Psi^{(k)}\right)}-\frac{\partial d\left(g\left(\boldsymbol{z}^{(i)}\middle|\Phi^{(k)}\right)\middle|\Psi^{(k)}\right)/\partial\Psi}{1-d\left(g\left(\boldsymbol{z}^{(i)}\middle|\Phi^{(k)}\right)\middle|\Psi^{(k)}\right)}\right)\end{gathered} \tag{4.5}$$

As a result, GAN trained with SGD is specified as follows:

Initialize Φ and Ψ and set *k* = 0.

Repeat

Sampling epoch $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)},\ldots, \boldsymbol{x}^{(N)})$ or receiving epoch ***X*** from big data / data stream.

Randomize random epoch $\boldsymbol{Z} = (\mathbf{z}^{(1)}, \mathbf{z}^{(2)},\ldots, \mathbf{z}^{(N)})$ from standard normal distribution $P(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, I)$ with mean **0** and identity covariance matrix *I*.

$$\Phi^{(k+1)} = \Phi^{(k)} + \gamma\frac{1}{N}\sum_{i=1}^{N}\frac{\partial d\left(g\left(\boldsymbol{z}^{(i)}\middle|\Phi^{(k)}\right)\middle|\Psi^{(k)}\right)/\partial\Phi}{1-d\left(g\left(\boldsymbol{z}^{(i)}\middle|\Phi^{(k)}\right)\middle|\Psi^{(k)}\right)}$$

$$\Psi^{(k+1)} = \Psi^{(k)} + \gamma\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial d\left(\boldsymbol{x}^{(i)}\middle|\Psi^{(k)}\right)/\partial\Psi}{d\left(\boldsymbol{x}^{(i)}\middle|\Psi^{(k)}\right)}-\frac{\partial d\left(g\left(\boldsymbol{z}^{(i)}\middle|\Phi^{(k)}\right)\middle|\Psi^{(k)}\right)/\partial\Psi}{1-d\left(g\left(\boldsymbol{z}^{(i)}\middle|\Phi^{(k)}\right)\middle|\Psi^{(k)}\right)}\right)$$

Increase *k* = *k* + 1.

Until some terminating conditions are met.

Note, a terminating condition is customized, for example, parameters Φ and Ψ are not changed significantly or there is no more coming epoch ***X***. Moreover, the index *k* indicates time point as well as iteration of SGD.

Following figure depicts GAN.

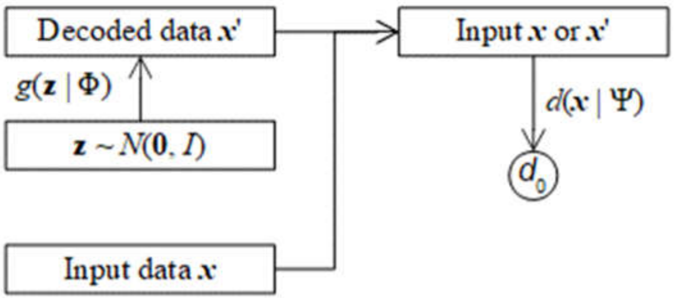

Figure 4. 1. Generative Adversarial Network (GAN).

There is a question that how to calculate the differentials ∂*d*(.) / ∂Φ and ∂*d*(.) / ∂Ψ. Indeed, it is not difficult to calculate them in context of neural network associated with backpropagation algorithm so that the last output layer as well as last neuron *o* of any DNN $f(\boldsymbol{x} \mid \Theta)$ or $g(\mathbf{z} \mid \Phi)$ is acted by activation function *a*(.) as follows:

$$a(o) = a\left(\boldsymbol{w}^T\boldsymbol{i}\right)$$
$$o = \boldsymbol{w}^T\boldsymbol{i}$$

Where ***i*** is input of the last layer *o* and weight parameter ***w*** is a part of entire parameter Φ or Ψ and hence, we need to focus on calculating differential *da*(*o*) / *d**w*** which is equivalent to any differential ∂*d*(.) / ∂Φ or ∂*d*(.) / ∂Ψ so that backpropagation algorithm will solve remaining parts of entire parameter Φ or Ψ.

$$\frac{da(o)}{d\boldsymbol{w}} \cong \text{any differential}$$

Indeed, we have:

$$\frac{da(o)}{d\boldsymbol{w}} = \frac{da(\boldsymbol{w}^T\boldsymbol{i})}{d\boldsymbol{w}} = \frac{da(o)}{do}\boldsymbol{i}^T = a'(o)\boldsymbol{i}^T$$

Note, the subscript "T" denotes transposition operator of vector and matrix in which row vector becomes column vector and vice versa. It is easy to calculate the derivative $a'(o)$ when activation function was specified, for instance, if $a(o)$ is sigmoid function, we have:

$$a(o) = y = \frac{1}{1+e^{-o}}$$

$$a'(o) = \frac{1}{1+e^{-o}}\left(1 - \frac{1}{1+e^{-o}}\right) = a(o)\big(1-a(o)\big) = y(1-y)$$

In practice, y is replaced by $a(y)$ in order to prevent o from being out of space:

$$a'(o) \cong a(y)\big(1-a(y)\big) = a\big(a(o)\big)\Big(1 - a\big(a(o)\big)\Big)$$

As a result, we have:

$$\text{any differential} \cong a\big(a(o)\big)\Big(1 - a\big(a(o)\big)\Big)\boldsymbol{i}^T$$

For fast computation, it is possible to set the derivative $a'(o)$ to be small enough constants like 1 such that any differential is $\boldsymbol{i}^T$.

Given epoch $\boldsymbol{D} = (\boldsymbol{d}^{(1)} = (\boldsymbol{x}^{(1)}, \mathbf{z}^{(1)}), \boldsymbol{d}^{(2)} = (\boldsymbol{x}^{(2)}, \mathbf{z}^{(2)}),\ldots, \boldsymbol{d}^{(N)} = (\boldsymbol{x}^{(N)}, \mathbf{z}^{(N)}))$ implies that the epoch is created or sent by equilateral distribution $1/N$ but in general case, $\boldsymbol{D}$ can follow an arbitrary distribution denoted by PDF $P(\boldsymbol{d})$, which makes loss function $l(\Phi, \Psi)$ changed a little bit by theoretical expectation given distribution $P(\boldsymbol{d})$.

$$l(\Phi,\Psi) = \int_{\boldsymbol{d}} \Big(\log\big(d(\boldsymbol{x}|\Psi)\big) + \log\big(1 - d(g(\mathbf{z}|\Phi)|\Psi)\big)\Big) d\boldsymbol{d}$$

Suppose $\boldsymbol{x}$ and $\mathbf{z}$ distribute separately as $P(\boldsymbol{x})$ and $Q(\mathbf{z})$ such that $P(\boldsymbol{x})$ is called original data PDF and $Q(\mathbf{z})$ is called generated data PDF, we have:

$$l(\Phi,\Psi) = \int_{\boldsymbol{x}} P(\boldsymbol{x})\log\big(d(\boldsymbol{x}|\Psi)\big)\,d\boldsymbol{x} + \int_{\mathbf{z}} Q(\mathbf{z})\log\big(1 - d(g(\mathbf{z}|\Phi)|\Psi)\big)\,d\mathbf{z} \quad (4.6)$$

Although there is no significant change in aforementioned practical technique to estimate parameters, it is necessary to research original data PDF $P(\boldsymbol{x})$ and generated data PDF $Q(\mathbf{z})$ as well as expectation form of loss function $l(\Phi, \Psi)$ so as to prove convergence of GAN. Recall that the min-max problem is:

$$\min_{\Phi}\max_{\Psi} l(\Phi,\Psi)$$

That is:

$$\Phi^* = \underset{\Phi}{\operatorname{argmin}}\, l(\Phi,\Psi^*)$$

$$\Psi^* = \underset{\Psi}{\operatorname{argmax}}\, l(\Phi^*,\Psi)$$

The convergence of GAN is equivalent to the convergence of this min-max problem. In other words, Goodfellow et al. (Goodfellow, et al., 2014) proved the existence of global optimal value l^* such that min-max problem approach l^* as follows:

$$l^* = l(\Phi^*, \Psi^*) = \min_{\Phi} l(\Phi, \Psi^*) = \min_{\Phi} \max_{\Psi} l(\Phi, \Psi) \quad (4.7)$$

Because **z** is generated by distribution $Q(\mathbf{z})$ and $g(\mathbf{z} \mid \Phi)$ is valuated as $\boldsymbol{x}$ as $g(\mathbf{z} \mid \Phi) = \boldsymbol{x}$, loss function $l(\Phi, \Psi)$ is rewritten by changing variable (Goodfellow, et al., 2014, p. 5).

$$l(\Phi, \Psi) = \int_{\boldsymbol{x}} P(\boldsymbol{x}) \log\big(d(\boldsymbol{x}|\Psi)\big)\, d\boldsymbol{x} + \int_{\boldsymbol{x}} Q(\boldsymbol{x}) \log\big(1 - d(\boldsymbol{x}|\Psi)\big)\, d\boldsymbol{x}$$

$$= \int_{\boldsymbol{x}} \Big(P(\boldsymbol{x}) \log\big(d(\boldsymbol{x}|\Psi)\big) + Q(\boldsymbol{x}) \log\big(1 - d(\boldsymbol{x}|\Psi)\big)\Big) d\boldsymbol{x}$$

In mathematical literature, function $a\log(y) + b\log(1-y)$ gets maximal at maximizer $y = a/(a+b)$ such that:

$$d(\boldsymbol{x}|\Psi^*) = \frac{P(\boldsymbol{x})}{P(\boldsymbol{x}) + Q(\boldsymbol{x})}$$

Therefore, we have (Goodfellow, et al., 2014, p. 5):

$$\max_{\Psi} l(\Phi, \Psi) = l(\Phi, \Psi^*) = \int_{\boldsymbol{x}} \left(P(\boldsymbol{x}) \log\left(\frac{P(\boldsymbol{x})}{P(\boldsymbol{x}) + Q(\boldsymbol{x})}\right) + Q(\boldsymbol{x}) \log\left(\frac{Q(\boldsymbol{x})}{P(\boldsymbol{x}) + Q(\boldsymbol{x})}\right)\right) d\boldsymbol{x}$$

$$=$$

$$= \int_{\boldsymbol{x}} P(\boldsymbol{x}) \log\left(\frac{2P(\boldsymbol{x})}{P(\boldsymbol{x}) + Q(\boldsymbol{x})}\right) d\boldsymbol{x} + \int_{\boldsymbol{x}} Q(\boldsymbol{x}) \log\left(\frac{2Q(\boldsymbol{x})}{P(\boldsymbol{x}) + Q(\boldsymbol{x})}\right) d\boldsymbol{x} - \log 4$$

$$= \mathrm{KL}\left(P(\boldsymbol{x}) \middle| \frac{P(\boldsymbol{x}) + Q(\boldsymbol{x})}{2}\right) + \mathrm{KL}\left(Q(\boldsymbol{x}) \middle| \frac{P(\boldsymbol{x}) + Q(\boldsymbol{x})}{2}\right) - \log 4$$

Where KL(.) denotes Kullback-Leibler divergence of two distributions. The sum of two KL divergences above is a so-called Jensen-Shannon divergence of original data distribution $P(\boldsymbol{x})$ and generated data distribution $Q(\mathbf{z})$, denoted $\mathrm{JS}(P(\boldsymbol{x}) \mid Q(\boldsymbol{x}))$. Therefore, we have (Goodfellow, et al., 2014, p. 5):

$$\max_{\Psi} l(\Phi, \Psi) = l(\Phi, \Psi^*) = \mathrm{JS}\big(P(\boldsymbol{x}) \big| Q(\boldsymbol{x})\big) - \log 4$$

Because Jensen-Shannon divergence is always nonnegative, we have (Goodfellow, et al., 2014, p. 5):

$$\max_{\Psi} l(\Phi, \Psi) = l(\Phi, \Psi^*) \geq -\log 4$$

The sign "=" occurs because Jensen-Shannon divergence is zero if the two distributions are equal, for instance, $P(\boldsymbol{x}) = Q(\boldsymbol{x})$. Therefore, $l(\Phi, \Psi^*)$ has maximal value –log4. In other words, we have:

$$l^* = l(\Phi^*, \Psi^*) = \min_{\Phi} \max_{\Psi} l(\Phi, \Psi) = -\log 4 \quad (4.8)$$

Due to the existence of global optimal value $l^* = -\log 4$, the convergence of GAN is asserted.

5. Conclusions

Recall that there are three main approaches for constructing deep generative model (DGM) such as tractable density DGM, approximate density DGM, and implicit density DGM. When skimming these approaches, it is easy to recognize that applied statistical problems such as probability distribution and parameter estimation are often mentioned but the effectiveness of a deep generative model is also dependent on how to structure the deep neural network (DNN) and how to train such network. Anyhow data generation function called generator is always defined by DNN in DGM. Backpropagation (BP) algorithm associated with stochastic gradient descent (SGD) method is focused as typical example in this research but there are some more effective training algorithms. Essential, training DNN generator is unsupervised learning task because there is no data class in DGM although generating data distribution (distribution of tractable data **z**) is often assumed to follow normal distribution whereas BP belongs to supervised learning algorithm. This is the reason that the two problems of constructing DGM are 1) how to train generator DNN $g(\mathbf{z} \mid \Phi)$ and 2) how to qualify such training task which often relates to another optimization task or another training task so that the qualification task tries to attach supervised learning BP to unsupervised learning mechanism. For instance, PixelRNN allows output data becomes input data by recurrent neural network, VAE applies Kullback-Leibler divergence into forming data distribution as implicit data class, and GAN issues target function with expectation of Nash equilibrium. Essentially, these mechanisms make the exchange or transformation between supervised learning and unsupervised learning, which plays the role of a hinge for creating the DNN generator.

One of problems issued by BP is the zero derivative problem when SGD cannot improve parameters after some large enough iteration because the gradient (derivative) approaches zero at that time. In other words, SGD may not converge even though there is a large enough number of iterations. Moreover, basic DGM approaches mentioned here require a continuous data provision for training DNN, which consumes more resources than reinforcement learning.

Appendices

A1. Backpropagation Algorithm

Because backpropagation (*BP*) algorithm is often associated with stochastic gradient descent (SGD) method to optimize loss function, it is necessary to describe a little bit BP and SGD, especially, in case of DGM where artificial neural network is deep neural network (DNN) with many hidden layers so that learning DNN (training DNN) is essential to deep learning. A neural network often has one input layer, one output layer, and hidden layers. The simplest neural network has one input layer and one output layer without a hidden layer. A DNN is a neural network often having many enough hidden layers. Each layer has a list of units called neurons and there are full connections of neurons between two successive layers. Feed-forward network which is the neural network whose connections are one-way from input layer to hidden layers to output layer is focused here. BP is a reverse process that begins from output layer back to input layer. Without loss of generality, input neurons, hidden neurons, and output neurons are concerned rather than input layer, hidden layers, and output layers, respectively because BP processes layers by processing neurons of layers. As a convention, let i, h, and o denote indices of input neurons, hidden neurons, and output neurons, respectively. Let x and y denote input variable and output variable of neurons. For instance, x_i and y_i are input and output of an input neuron and x_h and y_h are input and output of a hidden neuron whereas x_o and y_o are input and output of an output neuron, respectively. Because BP is a reverse process that begins from output layer back to input layer, output neuron is concerned firstly by starting its propagation rule as follows:

$$x_o = \sum_h w_{ho} y_h + \theta_o$$

$$y_o = f(x_o)$$

Where w_{ho} and θ_o are weights of connections between previous hidden neurons h and current output neurons o while θ_o is bias of current output neurons o. Moreover, f denotes activation function that squashes input into valid range, which is often sigmoid (logistic) function or hyperbolic tangent function. Some literature documents use letter b to denote such bias. BP aims to learn the parameters such as connection weights $w_{(.)(.)}$ and biases $\theta_{(.)}$ from sample data. Note, propagation rule is cornerstone of evaluating neural network. Let v_o is real value of output neuron o, error function $\varepsilon(y_o)$ of output neuron is half the square deviation between y_o and v_o.

$$\varepsilon(y_o) = \frac{1}{2}(v_o - y_o)^2 = \frac{1}{2}(y_o - v_o)^2$$

Note, y_o is variable and v_o represents sample data. Weight parameter and bias parameter are estimated by minimizing output error function $\varepsilon(y_o)$ according to BP.

$$w_{ho}^* = \underset{w_{ho}}{\operatorname{argmin}} \frac{1}{2}(y_o - v_o)^2$$

$$\theta_o^* = \underset{\theta_o}{\operatorname{argmin}} \frac{1}{2}(y_o - v_o)^2$$

Minimizing output error function $\varepsilon(y_o)$ is equivalent to maximizing likelihood (PDF) of random variable y_o. Indeed, output likelihood $P(y_o)$ is specified as follows:

$$P(y_o) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y_o - v_o)^2\right)$$

Where,

$$y_o = f(x_o) = f\left(\sum_h w_{ho} y_h + \theta_o\right)$$

Note, mean and variance of output likelihood $P(y_o)$ are 0 and 1, respectively. Exactly, $P(y_o)$ is probability density function (PDF) of the error y_o–v_o. By maximum likelihood estimation (MLE) method, maximizing $P(y_o)$ is to maximize its natural logarithm log$P(y_o)$:

$$\log P(y_o) = -\frac{1}{2}\pi - \frac{1}{2}(y_o - v_o)^2$$

Because π is constant, it is obtained:

$$w_{ho}^* = \underset{w_{ho}}{\operatorname{argmax}} \log P(y_o) = \underset{w_{ho}}{\operatorname{argmax}} \left(-\frac{1}{2}(y_o - v_o)^2\right)$$

$$\theta_o^* = \underset{\theta_o}{\operatorname{argmax}} \log P(y_o) = \underset{\theta_o}{\operatorname{argmax}} \left(-\frac{1}{2}(y_o - v_o)^2\right)$$

This implies:

$$w_{ho}^* = \underset{w_{ho}}{\operatorname{argmin}} \frac{1}{2}(y_o - v_o)^2$$

$$\theta_o^* = \underset{\theta_o}{\operatorname{argmin}} \frac{1}{2}(y_o - v_o)^2$$

Which confirm the equivalence of error minimization and likelihood maximization. Therefore, BP becomes more potential because MLE can be extended with more techniques so that error function is defined indirectly and more flexibly via likelihood function. It is interesting that the minimization problem can be exchanged with the maximization problem. Shortly, the optimization is ultimate purpose.

The problem now is how to minimize error function $\varepsilon(y_o)$, which is the optimization problem. Fortunately, *stochastic gradient descent* (*SGD*) method is applied into solving this optimization when square error function is Lipschitz continuous and bounded. Given target function has both variable and parameter where parameter is the subject of optimization, SGD pushes parameter candidate-point along with the same direction (for maximization) or the opposite direction (for minimization) of gradient of target function. There are two important aspects of SGD: 1) the gradient is the first-order derivative of target function with regard to parameter, and 2) the variable is considered as input data which is fed by stochastic process or random way. Moreover, candidate point considered as candidate solution or candidate minimizer / maximizer of optimal parameter is pushed with step length which is coded as learning rate γ. It is proved that SGD will be converged to the optimal solution (good enough minimizer / maximizer) after many enough iterations and many enough data. Shortly, SGD updates weight parameter w_{ho} and bias parameter θ_o of output error function $\varepsilon(y_o)$ at every iteration within context of BP as follows:

$$w_{ho} = w_{ho} - \gamma \nabla_{w_{ho}} \varepsilon(y_o)$$
$$\theta_o = \theta_o - \gamma \nabla_{\theta_o} \varepsilon(y_o)$$

Where $\nabla_{w_{ho}} \varepsilon(y_o)$ and $\nabla_{\theta_o} \varepsilon(y_o)$ are gradients of output error function $\varepsilon(y_o)$ with regard to weight parameter w_{ho} and bias parameter θ_o, respectively:

$$\nabla_{w_{ho}} \varepsilon(y_o) = \frac{d\varepsilon(y_o)}{dw_{ho}}$$
$$\nabla_{\theta_o} \varepsilon(y_o) = \frac{d\varepsilon(y_o)}{d\theta_o}$$

It is not difficult to calculate these gradients. Due to chain rule of derivative and propagation rule

$$y_o = f\left(x_o = \sum_h w_{ho} y_h + \theta_o\right)$$

We have:

$$\nabla_{w_{ho}} \varepsilon(y_o) = \frac{d\varepsilon(y_o)}{dw_{ho}} = \frac{d\varepsilon(y_o)}{dy_o} \frac{dy_o}{dx_o} \frac{dx_o}{dw_{ho}} = (y_o - v_o) f'(x_o) y_h$$
$$\nabla_{\theta_o} \varepsilon(y_o) = \frac{d\varepsilon(y_o)}{d\theta_o} = \frac{d\varepsilon(y_o)}{dy_o} \frac{dy_o}{dx_o} \frac{dx_o}{d\theta_o} = (y_o - v_o) f'(x_o)$$

Note,

$$\frac{d\varepsilon(y_o)}{dy_o} = \frac{d\frac{1}{2}(y_o - v_o)^2}{dy_o} = (y_o - v_o)$$
$$\frac{dy_o}{dx_o} = \frac{df(x_o)}{dx_o} = f'(x_o)$$
$$\frac{dx_o}{dw_{ho}} = \frac{d\left(\sum_h w_{ho} y_h + \theta_o\right)}{dw_{ho}} = y_h$$
$$\frac{dx_o}{d\theta_o} = \frac{d\left(\sum_h w_{ho} y_h + \theta_o\right)}{d\theta_o} = 1$$

And due to:

$$w_{ho} = w_{ho} - \gamma \nabla_{w_{ho}} \varepsilon(y_o)$$
$$\theta_o = \theta_o - \gamma \nabla_{\theta_o} \varepsilon(y_o)$$

As a result, parameter w_{ho} and bias parameter θ_o at output neuron is updated at every iteration of SGD within context of BP as follows:

$$\begin{aligned} w_{ho} &= w_{ho} + \gamma y_h (v_o - y_o) f'(x_o) \\ \theta_o &= \theta_o + \gamma (v_o - y_o) f'(x_o) \end{aligned} \quad (A1.1)$$

Note, $f'(x_o)$ is the first derivative of activation function at x_o and γ is learning rate ($0 < \gamma \leq 1$). It is expected that weight parameter w_{ho} and bias parameter θ_o will converge the aforementioned optimizers w^*_{ho} and θ^*_o. Let δ_o denote output error, we have:

$$w_{ho} = w_{ho} + \gamma y_h \delta_o$$
$$\theta_o = \theta_o + \gamma \delta_o$$

Where,

$$\delta_o = (v_o - y_o) f'(x_o)$$

BP continues to estimate weight and bias parameters of previous neurons which are hidden neurons. Without loss of generality, given hidden neuron h whose error $\varepsilon(y_h)$ is sum of errors of all output neuron o to which such hidden neuron connects.

$$\varepsilon(y_h)=\sum_o \varepsilon(y_o)$$

Note, y_h is calculated by propagation rule as usual:

$$y_h=f\left(x_h=\sum_j w_{jh}y_j+\theta_h\right)$$

Note, y_j is output value of previous hidden neuron j which connects to current hidden neuron h. Because SGD is continuously applied into estimating weight parameters w_{jh} and bias parameter θ_h of hidden neuron h, gradients of hidden error function $\varepsilon(y_h)$ with regard to w_{jh} and θ_h need to be determined, respectively. According to chain rule of derivative, we have:

$$\nabla_{w_{jh}}\varepsilon(y_h)=\frac{d\varepsilon(y_h)}{dw_{jh}}=\frac{d\varepsilon(y_h)}{dy_h}\frac{dy_h}{dx_h}\frac{dx_h}{dw_{jh}}$$

$$\nabla_{\theta_h}\varepsilon(y_h)=\frac{d\varepsilon(y_h)}{d\theta_h}=\frac{d\varepsilon(y_h)}{dy_h}\frac{dy_h}{dx_h}\frac{dx_h}{d\theta_h}$$

Due to:

$$\frac{dy_h}{dx_h}=\frac{df(x_h)}{dx_h}=f'(x_h)$$

$$\frac{dx_h}{dw_{jh}}=\frac{d\left(\sum_j w_{jh}y_j+\theta_h\right)}{dw_{jh}}=y_j$$

$$\frac{dx_h}{d\theta_h}=\frac{d\left(\sum_j w_{jh}y_j+\theta_h\right)}{d\theta_h}=1$$

Equations of gradients $\nabla_{w_{jh}}\varepsilon(y_h)$ and $\nabla_{\theta_h}\varepsilon(y_h)$ are written:

$$\nabla_{w_{jh}}\varepsilon(y_h)=\frac{d\varepsilon(y_h)}{dy_h}y_jf'(x_h)$$

$$\nabla_{\theta_h}\varepsilon(y_h)=\frac{d\varepsilon(y_h)}{d\theta_h}=\frac{d\varepsilon(y_h)}{dy_h}f'(x_h)$$

It is now necessary to calculate the derivative $d\varepsilon(y_h)/dy_h$ of hidden error function $\varepsilon(y_h)$ with regard to y_h. Indeed, we have:

$$\frac{d\varepsilon(y_h)}{dy_h}=\frac{d\sum_o\varepsilon(y_o)}{dy_h}=\sum_o\frac{d\varepsilon(y_o)}{dy_h}=\sum_o\frac{d\varepsilon(y_o)}{dy_o}\frac{dy_o}{dx_o}\frac{dx_o}{dy_h}$$

Due to:

$$\frac{d\varepsilon(y_o)}{dy_o}\frac{dy_o}{dx_o}=\frac{d\frac{1}{2}(y_o-v_o)^2}{dy_o}\frac{df(x_o)}{dx_o}=(y_o-v_o)f'(x_o)=-\delta_o$$

This produces:

$$\frac{d\varepsilon(y_h)}{dy_h}=-\sum_o\frac{dx_o}{dy_h}\delta_o=-\sum_o\frac{d\left(\sum_h w_{ho}y_h+\theta_o\right)}{dy_h}\delta_o=-\sum_o w_{ho}\delta_o$$

Gradients $\nabla_{w_{jh}}\varepsilon(y_h)$ and $\nabla_{\theta_h}\varepsilon(y_h)$ are totally determined:

$$\nabla_{w_{jh}}\varepsilon(y_h) = -y_j\left(\sum_o w_{ho}\delta_o\right)f'(x_h)$$

$$\nabla_{\theta_h}\varepsilon(y_h) = -\left(\sum_o w_{ho}\delta_o\right)f'(x_h)$$

Due to SGD estimation:

$$w_{jh} = w_{jh} - \gamma\nabla_{w_{jh}}\varepsilon(y_h)$$

$$\theta_h = \theta_h - \gamma\nabla_{\theta_h}\varepsilon(y_h)$$

Parameter w_{jh} and bias parameter θ_h at output neuron is updated at every iteration of SGD within context of BP as follows:

$$\begin{aligned} w_{jh} &= w_{jh} + \gamma y_j\left(\sum_o w_{ho}\delta_o\right)f'(x_h) \\ \theta_o &= \theta_o + \gamma\left(\sum_o w_{ho}\delta_o\right)f'(x_h) \end{aligned} \tag{A1.2}$$

Let δ_h denote hidden error:

$$\delta_h = \left(\sum_o w_{ho}\delta_o\right)f'(x_h)$$

SGD estimation equations of hidden neurons become more succinct:

$$w_{jh} = w_{jh} + \gamma y_j\delta_h$$
$$\theta_h = \theta_h + \gamma\delta_h$$

The reverse process of BP recurrently continues to estimate other parameters of previous hidden neurons, which is an interesting and effective aspect of BP. Finally, following system of estimation equations is the summary of association of BP and SGD, in which weight parameters and bias parameters of feed-forward neural network are updated at every iteration whenever data sample v is received.

$$\begin{aligned} w_{ho} &= w_{ho} + \gamma y_h\delta_o \\ \theta_o &= \theta_o + \gamma\delta_o \\ w_{jh} &= w_{jh} + \gamma y_j\delta_h \\ \theta_h &= \theta_h + \gamma\delta_h \end{aligned} \tag{A1.3}$$

Where output error δ_o and hidden error δ_h are calculated as follows:

$$\begin{aligned} \delta_o &= (v_o - y_o)f'(x_o) \\ \delta_h &= \left(\sum_o w_{ho}\delta_o\right)f'(x_h) \end{aligned} \tag{A1.4}$$

For easy explanation, according to the BP recurrent process, weight parameters w_{kj} and bias parameter θ_j of previous hidden neuron j of hidden neuron h are calculated as follows:

$$w_{kj} = w_{kj} + \gamma y_k\delta_j$$
$$\theta_j = \theta_j + \gamma\delta_j$$

Where error δ_j of hidden neuron j is:

$$\delta_j = \left(\sum_h w_{jh}\delta_h\right)f'(x_j)$$

The back recurrent process continues until reaching input layer (exclusive). It is easy to recognize that the entry point of BP is the output error δ_o which relates to derivative of error function of output neuron at sample point v. Recall that such error function can be replaced by likelihood function instead. Therefore, δ_o is the opposite of gradient of output error function if the error function is applied to estimation within context of minimization. Otherwise, δ_o is gradient of output likelihood function if the likelihood function is applied to estimation within context of maximization. The interesting result allows us to extend BP applications by defining error function or likelihood function at output layer without changing BP recurrent process.

It is necessary to consider activation function $f(x)$ and its derivative $f'(x)$ which are evaluated at output neuron and hidden neuron as $f(x_o)$, $f(x_h)$, $f'(x_o)$, and $f'(x_h)$. For instance, if $f(x)$ is sigmoid function (logistic function), we have:

$$f(x) = y = \frac{1}{1+e^{-x}}$$

$$f'(x) = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = f(x)\big(1-f(x)\big) = y(1-y)$$

In practice, y is replaced by $f(y)$ in order to prevent o from being out of space:

$$f'(x) \cong f(y)\big(1-f(y)\big) = f\big(f(x)\big)\Big(1-f\big(f(x)\big)\Big)$$

It is possible to fix the derivative by 1 as $f'(x) = 1$ for all x for fast computation but this approximation is not optimal.

A2. Kullback-Leibler Divergence

In information theory, entropy is the quantity that indicates uncertainty of a random variable. Exactly, given random variable x and its probability density function (PDF) $P(x)$, entropy of x denoted $H(x)$ is the metric which is minus expected value of natural logarithm of x given distribution $P(x)$.

$$H(x) = -\int_x P(x)\ln P(x)\,dx$$

If random variable x is discrete, its entropy becomes simpler:

$$H(x) = -\sum_x P(x)\ln P(x)$$

It is easy to recognize that entropy $H(x)$ measures the level of uncertainty or the level of surprise for random variable x and such level is average level. As the opposite of probability, such uncertainty which measures the variation of random variable is also called information content, self-information, surprise, Shannon information, or information, in short:

$$I(x) = I\big(x\big|P(x)\big) = -\ln P(x)$$

It is interesting that information of x is defined based on probability of x although it is opposite to the probability. The larger the information of x is, the more the uncertainty of x is, the less the probability of x is. Entropy $H(x)$ is the expectation of information $I(x)$ given distribution $P(x)$.

$$H(x) = \int_x P(x)I(x)dx$$

In thermodynamics, entropy represents the level of chaos in movement of particles. *Kullback-Leibler* (*KL*) *divergence* which is defined based on concept of entropy measures the difference of two distributions. For instance, given distributions $P(x)$ and $Q(x)$, Kullback-Leibler divergence of $P(x)$ given $Q(x)$ denoted $D_{KL}(P(x) \mid Q(x))$ measures how much the distribution $P(x)$ is different from the distribution $Q(x)$ (Wikipedia, Kullback-Leibler divergence, 2004).

$$\mathrm{D}_{\mathrm{KL}}(P(x)\|Q(x)) = \int_x P(x)\ln\frac{P(x)}{Q(x)}dx$$

For convenience, we denote:

$$\mathrm{KL}\big(P(x)\big|Q(x)\big) = \int_x P(x)\ln\frac{P(x)}{Q(x)}dx \tag{A2.1}$$

The larger the KL divergence KL($P(x)$ | $Q(x)$) is, the more different focused distribution $P(x)$ is from distribution $Q(x)$. However, such difference does not represent distance metric between $P(x)$ and $Q(x)$ because KL divergence is not symmetric:

$$\mathrm{KL}\big(P(x)\big|Q(x)\big) \neq \mathrm{KL}\big(Q(x)\big|P(x)\big)$$

KL divergence does not satisfy triangle inequality too (Wikipedia, Kullback-Leibler divergence, 2004). KL divergence is expended:

$$\mathrm{KL}\big(P(x)\big|Q(x)\big) = \int_x P(x)\ln\frac{P(x)}{Q(x)}dx = \int_x P(x)\ln P(x)\,dx - \int_x P(x)\ln Q(x)\,dx$$

$$= -\int_x P(x)\ln Q(x)\,dx - H(x)$$

Let $H(x \mid Q(x))$ denote entropy of x such that x is quantified by distribution $Q(x)$.

$$H\big(x\big|Q(x)\big) = -\int_x P(x)\ln Q(x)\,dx = \int_x P(x)I\big(x\big|Q(x)\big)dx$$

Where,

$$I\big(x\big|Q(x)\big) = -\ln Q(x)$$

We obtain:

$$\mathrm{KL}\big(P(x)\big|Q(x)\big) = H\big(x\big|Q(x)\big) - H(x) = \int_x P(x)\Big(I\big(x\big|Q(x)\big) - I\big(x\big|P(x)\big)\Big)dx$$

This implies KL divergence measures the expected value of uncertainty when focused distribution $P(x)$ is replaced by distribution $Q(x)$ for quantifying random variable x. This is the reason that KL divergence is also called relative entropy. KL divergence is always nonnegative due to:

$$\int_x P(x)\ln P(x)\,dx \geq \int_x P(x)\ln Q(x)\,dx$$

Two distributions $P(x)$ and $Q(x)$ are identical if KL divergence KL($P(x)$ | $Q(x)$) is zero. Moreover, if $P(x)$ is 0 for all x then $Q(x)$ must be 0 for all x. As usual, $P(x)$ represents data distribution and $Q(x)$ represents theoretical distribution so that it is possible to compare or fit observational model with hypothesis model (Wikipedia, Kullback-Leibler divergence, 2004).

While Kullback-Leibler divergence is not a metric because it satisfies neither symmetry nor triangle inequality, *Jensen-Shannon* (*JS*) divergence is a real metric that measures the distance between two distributions although JS divergence is defined based on KL divergence. Given the two distributions $P(x)$ and $Q(x)$, its JS divergence is the following average KL divergence (Wikipedia, Jensen-Shannon divergence, 2006):

$$\mathrm{D}_{\mathrm{JS}}(P(x)\|Q(x)) = \frac{1}{2}\mathrm{D}_{\mathrm{KL}}(P(x)\|M(x)) + \frac{1}{2}\mathrm{D}_{\mathrm{KL}}(Q(x)\|M(x))$$

Where $M(x)$ is mixture distribution or mean distribution of $P(x)$ and $Q(x)$:

$$M(x) = \frac{P(x) + Q(x)}{2}$$

Note, JS divergence satisfies both symmetry and triangle inequality.

$$\mathrm{D}_{\mathrm{JS}}(P(x)\|Q(x)) = \mathrm{D}_{\mathrm{JS}}(Q(x)\|P(x))$$
$$\mathrm{D}_{\mathrm{JS}}(P(x)\|R(x)) < \mathrm{D}_{\mathrm{JS}}(P(x)\|Q(x)) + \mathrm{D}_{\mathrm{JS}}(Q(x)\|R(x))$$

For convenience, we denote:

$$\mathrm{JS}\big(P(x)\big|Q(x)\big) = \frac{1}{2}\mathrm{KL}\big(P(x)\big|M(x)\big) + \frac{1}{2}\mathrm{KL}\big(Q(x)\big|M(x)\big) \tag{A2.2}$$

JS divergence is bounded in interval [0, 1] such that $0 \leq \mathrm{JS}(P(x) \mid Q(x)) \leq 1$. Square root of JS divergence is called JS distance of two distributions.

References

1. Doersch, C. (2016, January 3). Tutorial on Variational Autoencoders. *arXiv preprint*. Retrieved from https://arxiv.org/abs/1606.05908
2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Cambridge, Massachusetts, US: The MIT Press. Retrieved from https://www.deeplearningbook.org/
3. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Weinberger (Ed.), *Advances in Neural Information Processing Systems 27 (NIPS 2014). 27*. Montreal: NeurIPS. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf
4. Hardle, W., & Simar, L. (2013). *Applied Multivariate Statistical Analysis*. Berlin, Germany: Research Data Center, School of Business and Economics, Humboldt University.
5. Hinton, G. E. (2007). Boltzmann machine. *Brain Corporation, 2*(5), 1668. doi:10.4249/scholarpedia.1668
6. Kingma, D. P., & Welling, M. (2022, December 10). Auto-Encoding Variational Bayes. *arXiv Preprint*, 1-14. doi:10.48550/arXiv.1312.6114
7. Nguyen, L. (2015). *Matrix Analysis and Calculus* (1st ed.). (C. Evans, Ed.) Hanoi, Vietnam: Lambert Academic Publishing. Retrieved March 3, 2014, from https://www.shuyuan.sg/store/gb/book/matrix-analysis-and-calculus/isbn/978-3-659-69400-4
8. Nguyen, L. (2023). *Tutorial on artificial neural network*. Loc Nguyen's Academic Network. Open Science Framework (OSF). doi:https://osf.io/k8syc
9. Oord, A. v., Kalchbrenner, N., & Kavukcuoglu, K. (2016, August 19). Pixel Recurrent Neural Networks. *arXiv preprint*, 1-11. doi:10.48550/arXiv.1601.06759
10. Oussidi, A., & Elhassouny, A. (2018). Deep generative models: Survey. *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*. Fez, Morocco: IEEE. doi:10.1109/ISACV.2018.8354080
11. Ruthotto, L., & Haber, E. (2021, April 12). An Introduction to Deep Generative Modeling. *arXiv preprint*. doi:10.48550/arXiv.2103.05180
12. Theis, L., & Bethge, M. (2015, September 18). Generative Image Modeling Using Spatial LSTMs. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Ed.), *Advances in Neural Information Processing Systems 28 (NIPS 2015). 28*, pp. 1-9. Montreal: NeurIPS. doi:10.5555/2969442.2969455
13. Wikipedia. (2004, November 13). *Boltzmann machine*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Boltzmann_machine
14. Wikipedia. (2004, November 15). *Hopfield network*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Hopfield_network
15. Wikipedia. (2004, February 13). *Kullback-Leibler divergence*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence
16. Wikipedia. (2005, April 7). *Recurrent neural network*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Recurrent_neural_network
17. Wikipedia. (2006, February 9). *Jensen-Shannon divergence*. (Wikimedia Foundation) Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence
18. Wikipedia. (2007, April 16). *Long short-term memory*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Long_short-term_memory