entropy

MDPI

*Review*

# An Overview of Variational Autoencoders for Source Separation, Finance, and Bio-Signal Applications

**Aman Singh and Tokunbo Ogunfunmi ***

Department of Electrical & Computer Engineering, Santa Clara University, Santa Clara, CA 95053, USA; asingh9@scu.edu

* Correspondence: togunfunmi@scu.edu

**Abstract:** Autoencoders are a self-supervised learning system where, during training, the output is an approximation of the input. Typically, autoencoders have three parts: Encoder (which produces a compressed latent space representation of the input data), the Latent Space (which retains the knowledge in the input data with reduced dimensionality but preserves maximum information) and the Decoder (which reconstructs the input data from the compressed latent space). Autoencoders have found wide applications in dimensionality reduction, object detection, image classification, and image denoising applications. Variational Autoencoders (VAEs) can be regarded as enhanced Autoencoders where a Bayesian approach is used to learn the probability distribution of the input data. VAEs have found wide applications in generating data for speech, images, and text. In this paper, we present a general comprehensive overview of variational autoencoders. We discuss problems with the VAEs and present several variants of the VAEs that attempt to provide solutions to the problems. We present applications of variational autoencoders for finance (a new and emerging field of application), speech/audio source separation, and biosignal applications. Experimental results are presented for an example of speech source separation to illustrate the powerful application of variants of VAE: VAE, $\beta$-VAE, and ITL-AE. We conclude the paper with a summary, and we identify possible areas of research in improving performance of VAEs in particular and deep generative models in general, of which VAEs and generative adversarial networks (GANs) are examples.

**Keywords:** variational autoencoders; deep learning; volatility surfaces; speech source separation; EEG; EMG; ECG; generative models



**Citation:** Singh, A.; Ogunfunmi, T. An Overview of Variational Autoencoders for Source Separation, Finance, and Bio-Signal Applications. *Entropy* **2022**, *24*, 55. https://doi.org/10.3390/e24010055

Academic Editor: Sotiris Kotsiantis

Received: 6 November 2021
Accepted: 21 December 2021
Published: 28 December 2021



## 1. Introduction

One of the distinct traits of human intelligence is the ability to imagine and synthesize. Generative modeling in machine learning aims to train algorithms to synthesize completely new data, such as audio, text, and images; it does so by estimating the density of the data, and then sampling from that estimated density. The deep learning [1] revolution has led to breakthroughs in generative modeling with deep generative models such as variational autoencoders, generative stochastic networks, neural autoregressive models, and generative adversarial networks.

Variational autoencoders combine Bayesian variational inference with deep learning [2]; like the autoencoder, it has an encoder and decoder, but it aims to learn the probability distribution through amortized variational inference and the reparameterization trick. Information theory is a key component of variational inference because it involves minimizing the KL Divergence between the posterior distribution and variational posterior. The generative adversarial network is a framework for training two models (a discriminator and a generator) simultaneously with an adversarial process.

There has been lots of research on deep generative models for image, text, and audio generation. Generative adversarial networks (GANs) tend to outperform variational autoencoders in image fidelity, while a variational autoencoder is more stable and is better for estimating the probability distribution itself. Neural autoregressive models are powerful at

density estimation but often slower than VAEs in sampling. Often times, these different frameworks have been integrated to compliment the different strengths and ameliorate the weaknesses, such as with the adversarial autoencoder, VAE-GAN, VAE with inverse autoregressive flow, and PixelVAE.

Source separation, especially blind source separation, has long been a problem of interest in the signal processing community. The cocktail party problem is a common example of the blind source separation problem. It is when a listener is at a cocktail party where there are many people speaking concurrently, and the listener must try to follow one of the conversations. It seems like an easy task for humans, but, for computers, it is more difficult. Source separation has applications in music/speech/audio data, EEG signals, ECG signals, and image processing. In the past, methods like Independent Component Analysis (ICA) and Non Negative Matrix Factorization (NNMF) have been the state-of-the-art methods for this problem. More recently, deep learning methods have improved our solution, including variational autoencoders, generative adversarial networks, and recurrent neural networks.

For applications in finance (which is relatively new), VAEs are used to generate synthetic volatility surfaces for options trading.

Bio-signal applications of VAE include detection of serious diseases using electrocardiogram (ECG) signals, data augmentation of bio-signals and improving electroencephalography (EEG)-based speech recognition systems, etc.

The key contributions of our paper include:

(1) A general overview of autoencoders and VAEs (2) A comprehensive survey of applications of the variational autoencoders for speech source separation, data augmentation and dimensionality reduction in finance, and biosignal analysis. (3) A comprehensive survey of variational autoencoder variants. (4) Experiments and analysis of results in speech source separation using various VAE models. (5) While multiple survey papers have covered the VAE [3,4], this paper has a special focus on time series/signal processing applications and information-theoretic interpretations.

The paper is organized as follows: Section 2 provides a general background information and Section 3 discusses Variational Inference. Section 4 presents the Variational Autoencoder while Section 5 discusses Problems with the VAE. Several variants of the VAEs are presented in Section 6. Three interesting applications of VAEs are discussed in Section 7. Experimental results on speech source separation are discussed in Sections 8 and 9 concludes the paper.

*Notations*

The following notation will be used throughout this paper:

- Lower case $p$, $q$, $f$, $\gamma$, $\psi$, or $p(.)$, $q(.)$, $f(.)$, $\gamma(.)$, $\psi(.)$, to denote probability density functions (PDFs) or probability mass functions (PMFs).
- Random variables are written in lower case italicized letters, for example, $x$, with the exception of $\epsilon$, which will represent both a random variable and its instance.
- Deterministic scalar terms, including realizations of random variables, are written in lower case letters, for example, x. Greek alphabets $\alpha$, $\beta$, $\theta$, and $\phi$ will also denote deterministic scalar terms. Deterministic scalar terms that are not realizations of random variables can also be written in upper case letters, such as N.
- Random vectors are written in lower case italicized bold letters, for example, $\boldsymbol{x}$.
- if we have a random vector $\boldsymbol{x}$, then its $j$th component will be noted $x_j$.
- Ordinary vectors are written in lowercase bold letters or bold Greek alphabets $\boldsymbol{\theta}$, $\boldsymbol{\mu}$, and $\boldsymbol{\phi}$. A realization of a random vector $\boldsymbol{x}$ will be written as $\mathbf{x}$.
- Matrices are written in uppercase bold italics, such as $\boldsymbol{W}$. $\boldsymbol{I}$ denotes the identity matrix.
- If we have two random variables $x$ and $y$ with probability functions $p(x)$ and $p(y)$, we can write their joint probability function as $p(x, y)$. Their conditional probability distribution function is written as $p(x|y)$.
- If we have a PMF/PDF $p(x = \mathrm{x})$, $p(\mathrm{x})$ is the shorthand notation. For $p(x|y = \mathrm{y})$, $p(x|\mathrm{y})$ is the shorthand. For $p(x = \mathrm{x}|y = \mathrm{y})$, $p(\mathrm{x}|\mathrm{y})$ is the shorthand.

- If we have $p(.;\boldsymbol{\theta})$ or $p_{\boldsymbol{\theta}}(\mathrm{x})$, this denotes that the PDF/PMF $p$ is parameterized by $\boldsymbol{\theta}$—similarly with $q_{\boldsymbol{\phi}}(\mathrm{z})$. However, there are exceptions in some contexts. $p_x(\mathrm{x})$ will be shorthand for a distribution $p(\mathrm{x})$ associated with random variable $x$; $p_z(\mathrm{z})$ will be shorthand for a distribution $p$ associated with random variable $z$; and $p_{x|z}(\mathrm{x} \mid \mathrm{z}_i)$ denotes a conditional distribution of random variable $x$ given $z$=$\mathrm{z}_i$. The term $p_{(x,y)}(\mathrm{x},\mathrm{y})$ is a joint PDF/PMF between random variables $x$ and $y$. The term $p_{(x,z)}(\mathrm{x},\mathrm{z})$ is a joint PDF/PMF between random variables $x$ and $z$. In Section 2.8, $p_g(\mathrm{z})$ would represent the distribution that is the input to the generator in the GAN, while $p_d(\mathrm{x})$ represents the data distribution.
- range($x$) denotes the range of random variable $x$ and dom($x$) denotes the domain of random variable $x$.
- If we have a dataset $\mathcal{X} = \left\{\mathrm{x}^{(i)}\right\}_{i=1}^{N}$, where there are N independent and identically distributed (i.i.d.) realizations/observations of some continuous or discrete random variable $x$, the $i$th observation is denoted as $\mathrm{x}^{(i)}$.
- If we have a dataset $\mathcal{X} = \left\{\mathbf{x}^{(i)}\right\}_{i=1}^{N}$, where there are N i.i.d. realizations/observations of some continuous or discrete random vector $\mathbf{x}$, the $i$th observation is denoted as $\mathbf{x}^{(\mathbf{i})}$. If this our dataset, then $x_j^{(i)}$ represents the $j$th component of the $i$th observation.
- diag ($\mathbf{x}$) is a diagonal matrix, with the diagonal values being the values of vector $\mathbf{x}$ and det ($\boldsymbol{W}$) is the determinant of matrix $\boldsymbol{W}$.

## 2. Background

In this section, we summarize the prerequisite information for understanding variational autoencoders and its many extensions.

### *2.1. Distances and Information-Theoretic Measures*

Information theory is key to machine learning, from the usage of information-theoretic measures for the loss functions [5–7] to use for analysis through the information bottleneck framework [8–10]. It is also key to the development of the VAE [2]. Therefore, it is recommended to know key measures in information theory.

#### 2.1.1. Shannon's Entropy

If you have a random variable $x$, Shannon's Entropy is a measure of uncertainty of $x$, with a PDF of $p(\mathrm{x})$ [11]. It can also be thought of as a way to measure the uncertainty in a random vector or random process $\boldsymbol{x}$, which has a joint PDF $p(\mathbf{x})$. It is also a generalization of the variance of a process.

Discrete Shannon's Entropy, for discrete random variable $x$, with a PMF $p(\mathrm{x})$, is defined as:

$$\mathbb{H}(x) = -\sum_{\mathrm{x} \in range(x)} p(\mathrm{x}) \log(p(\mathrm{x})) \tag{1}$$

Continuous (Differential) Shannon's Entropy, with the PDF $p(\mathrm{x})$, for random variable $x$, is given by

$$\mathbb{H}(x) = -\int_{range(x)} p(\mathrm{x}) \log(p(\mathrm{x})) d\mathrm{x} \tag{2}$$

We can also rewrite both discrete and differential Entropy as:

$$\mathbb{H}(x) = -\mathbb{E}_{p(x)}[\log(p(x))] \tag{3}$$

2.1.2. Shannon's Joint Entropy

The Joint Entropy $\mathbb{H}(x,y)$ of a pair of discrete random variables $(x,y)$ with a joint distribution $p_{(x,y)}(\mathrm{x},\mathrm{y})$ is defined as

$$\mathbb{H}(x,y) = -\sum_{x\in range(x)}\sum_{y\in range(y)} p_{(x,y)}(\mathrm{x},\mathrm{y})\log(p_{(x,y)}(\mathrm{x},\mathrm{y})) \tag{4}$$

We can also write the Joint Entropy as

$$\mathbb{H}(x,y) = -\mathbb{E}_{p(x,y)}[\log(p(x,y))] \tag{5}$$

and

$$\mathbb{H}(x,y) = \mathbb{H}(y) + \mathbb{H}(x|y) \tag{6}$$

If we have $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \sim p(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_n) = p(\mathbf{x})$, where $\boldsymbol{x}$ is a discrete random vector, then

$$\mathbb{H}(x_1, x_2, \ldots, x_n) = -\sum_{\mathrm{x}_1,\mathrm{x}_2,\ldots,\mathrm{x}_n} p(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_n)\log(p(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_n)) \tag{7}$$

$$\mathbb{H}(\boldsymbol{x}) = -\sum_{\mathbf{x}\in range(\boldsymbol{x})} p(\mathbf{x})\log(p(\mathbf{x})) \tag{8}$$

If we have $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \sim p(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_n) = p(\mathbf{x})$, where $\boldsymbol{x}$ is a continuous random vector, then

$$\mathbb{H}(x_1, \ldots, x_n) = -\int p(\mathrm{x}_1, \ldots, \mathrm{x}_n)\log(p(\mathrm{x}_1, \ldots, \mathrm{x}_n))d\mathrm{x}_1 \ldots d\mathrm{x}_n \tag{9}$$

$$\mathbb{H}(\boldsymbol{x}) = -\int p(\mathbf{x})\log(p(\mathbf{x}))d\mathbf{x} \tag{10}$$

2.1.3. Shannon's Conditional Entropy

If we have $(x,y) \sim p_{(x,y)}(\mathrm{x},\mathrm{y})$, where $x$ and $y$ are discrete random variables, the conditional Entropy $\mathbb{H}(y|x)$ is:

$$\begin{aligned}
\mathbb{H}(y|x) &= \sum_{\mathrm{x}\in range(x)} p(\mathrm{x})\mathbb{H}(y|x=\mathrm{x}) \\
&= -\sum_{\mathrm{x}\in range(x)} p(\mathrm{x}) \sum_{\mathrm{y}\in range(y)} p_{y|x}(\mathrm{y}|\mathrm{x})\log(p_{y|x}(\mathrm{y}|\mathrm{x})) \\
&= -\sum_{\mathrm{x}\in range(x)}\sum_{\mathrm{y}\in range(y)} p_{(x,y)}(\mathrm{x},\mathrm{y})\log(p_{y|x}(\mathrm{y}|\mathrm{x})) \\
&= -\mathbb{E}_{p(x,y)}[\log(p(y|x))]
\end{aligned}$$

The conditional Entropy $\mathbb{H}(x|y)$ is:

$$\begin{aligned}
\mathbb{H}(x|y) &= \sum_{\mathrm{y}\in range(y)} p(\mathrm{y})\mathbb{H}(x|y=\mathrm{y}) \\
&= -\sum_{\mathrm{y}\in range(y)} p(\mathrm{y}) \sum_{\mathrm{x}\in range(x)} p_{x|y}(\mathrm{x}|\mathrm{y})\log(p_{x|y}(\mathrm{x}|\mathrm{y})) \\
&= -\sum_{\mathrm{y}\in range(y)}\sum_{\mathrm{x}\in range(x)} p_{(x,y)}(\mathrm{x},\mathrm{y})\log(p_{x|y}(\mathrm{x}|\mathrm{y})) \\
&= -\mathbb{E}_{p(x,y)}[\log(p(x|y))]
\end{aligned}$$

Conditional Entropy can be thought of as the "*expected value of the entropies of the conditional distributions, averaged over the conditioning random variable*" [11].

If $(x, y)$ are continuous random variables with the joint PDF $p(x, y)$, the conditional differential Entropy $\mathbb{H}(x|y)$ is

$$\mathbb{H}(x|y) = -\int p(\mathrm{x}, \mathrm{y}) \log(p(\mathrm{x}|\mathrm{y}))d\mathrm{x}d\mathrm{y}.$$

Since the joint PDF has the property $p(x|y) = p(x, y)/p(y)$, $\mathbb{H}(x|y) = \mathbb{H}(x, y) - \mathbb{H}(y)$.

2.1.4. Kullback–Leiber (KL) Divergence

If you have two probability distributions, $p$ and $q$, the KL Divergence measures the similarity between the two distributions; however, it is asymmetric [11]. It is also non-negative.

For discrete random variables with PMFs $p$ and $q$, the discrete KL Divergence is given by

$$\mathbb{D}_{KL}(p\|q) = \sum_{\mathrm{x} \in range(x)} p(\mathrm{x}) \log\left(\frac{p(\mathrm{x})}{q(\mathrm{x})}\right) \tag{11}$$

If $p$ and $q$ are distributions of a continuous random variable $x$, the continuous KL Divergence is given by

$$\mathbb{D}_{KL}(p\|q) = \int_{-\infty}^{\infty} p(\mathrm{x}) \log\left(\frac{p(\mathrm{x})}{q(\mathrm{x})}\right) d\mathrm{x} \tag{12}$$

We can also write the KL Divergence as

$$\mathbb{D}_{KL}(p\|q) = \mathbb{E}_{p(x)}\left[\log\left(\frac{p(x)}{q(x)}\right)\right] \tag{13}$$

$$\mathbb{D}_{KL}(q\|p) = \mathbb{E}_{q(x)}\left[\log\left(\frac{q(x)}{p(x)}\right)\right] \tag{14}$$

KL Divergence can be used when we are approximating a probability distribution $p$ with another probability distribution $q$. We can use $\mathbb{D}_{KL}(p\|q)$, called the forward $KL$, or $\mathbb{D}_{KL}(q\|p)$, which is called the reverse KL. Minimizing the forward KL with respect to the approximate distribution $q$ is called moment projection [12]. In the case where $p$ is positive but $q$ is $0, \log\left(\frac{p}{q}\right)$ becomes $\infty$, so then the support of $p$ is overestimated in the approximation $q$. Minimizing the reverse KL with respect to the approximate distribution $q$ is called information projection. In the case where $q$ is positive but $p$ is 0, $\log\left(\frac{q}{p}\right)$ becomes $\infty$, so the approximation $q$ does not include any input where $p$ is 0.

2.1.5. Mutual Information

Mutual Information of random variables $x$ and $y$, denoted $\mathbb{I}(x; y)$, measures the information that $x$ and $y$ share and the dependence between them [11]. Intuitively, it is how much knowing one random variable decreases uncertainty in the other one; this can be seen by the following formula:

$$\mathbb{I}(x; y) = \mathbb{H}(y) - \mathbb{H}(y|x) = \mathbb{H}(x) - \mathbb{H}(x|y) \tag{15}$$

Mutual Information can also be written as

$$\mathbb{I}(x; y) = \mathbb{H}(x) + \mathbb{H}(y) - \mathbb{H}(x, y) \tag{16}$$

If $x$ and $y$ are discrete random variables, their discrete Mutual Information is given by

$$\mathbb{I}(x; y) = \sum_{\mathrm{y} \in range(y)} \sum_{\mathrm{x} \in range(x)} p_{(x,y)}(\mathrm{x}, \mathrm{y}) \log\left(\frac{p_{(x,y)}(\mathrm{x}, \mathrm{y})}{p_x(\mathrm{x}) p_y(\mathrm{y})}\right) \tag{17}$$

$$\mathbb{I}(x;y) = \sum_{\mathrm{y}\in range(y)} \sum_{\mathrm{x}\in range(x)} p_{(x,y)}(x,\mathrm{y}) \log\left(\frac{p_{(x,y)}(\mathrm{x},\mathrm{y})}{p_x(\mathrm{x})p_y(\mathrm{y})}\right) \tag{18}$$

where $p_{(x,y)}(\mathrm{x},\mathrm{y})$ is the joint PMF between $x$ and $y$. If $x$ and $y$ are continuous random variables, their continuous Mutual Information is

$$\mathbb{I}(\mathrm{x};\mathrm{y}) = \iint_{range(x),range(y)} p_{(x,y)}(\mathrm{x},\mathrm{y}) \log\left(\frac{p_{(x,y)}(\mathrm{x},\mathrm{y})}{p_x(\mathrm{x})p_y(\mathrm{y})}\right) d\mathrm{x}d\mathrm{y} \tag{19}$$

$\mathbb{I}(\mathrm{x};\mathrm{y}) = 0$ if and only if $x$ and $y$ are independent. Mutual Information is non-negative and symmetric.

2.1.6. Cross-Entropy

For two PMFs p and q, the Cross-Entropy is defined as:

$$\mathbb{H}(p,q) = -\sum_{\mathrm{x}\in range(x)} p(\mathrm{x}) \log(q(\mathrm{x})) \tag{20}$$

2.1.7. Jensen–Shannon (JS) Divergence

Given PDFs $p$ and $q$, we have $\psi = \frac{1}{2}(p+q)$. The JS Divergence is

$$\mathrm{JSD}(p\|q) = \frac{1}{2}\mathbb{D}_{KL}(p\|\psi) + \frac{1}{2}\mathbb{D}_{KL}(q\|\psi) \tag{21}$$

This is also a symmetric measure.

2.1.8. Renyi's Entropy

Renyi's Entropy is a generalization of Shannon's Entropy [6]. Discrete Renyi's Entropy for PMF $p(\mathrm{x})$ is given by

$$h_\alpha(X) = \frac{1}{\alpha-1}\log\left(\sum_{k=1}^{\mathrm{N}}(p^\alpha(\mathrm{x}))\right), \alpha > 0 \tag{22}$$

Continuous Renyi's Entropy for PDF $p(\mathrm{x})$ is given by

$$h_\alpha(x) = \frac{1}{\alpha-1}\log\left(\int(p^\alpha(\mathrm{x})d\mathrm{x})\right), \alpha > 0 \tag{23}$$

When $\alpha \to 1$, Renyi's Entropy converges to Shannon's Entropy. When $\alpha = 2$, it becomes quadratic entropy. The term $\mathbb{V}_\alpha(x)$ is called information potential:

$$\mathbb{V}_\alpha(x) = \sum_{k=1}^{N} p^\alpha(\mathrm{x}) = \sum_{i=1}^{N} p^{\alpha-1}(\mathrm{x})p(\mathrm{x}) = \mathbb{E}\left[p^{\alpha-1}(x)\right] \tag{24}$$

If $\alpha$ = 2, it becomes the quadratic information potential (QIP):

$$\mathbb{V}_\alpha(x) = \sum_{k=1}^{N} p^\alpha(\mathrm{x}) = \sum_{i=1}^{N} p^{\alpha-1}(\mathrm{x})p(\mathrm{x}) = \mathbb{E}\left[p^{\alpha-1}(x)\right] \tag{25}$$

We first look at the continuous Quadratic Entropy, which is a case of $\alpha$-Renyi Entropy where $\alpha$= 2:

$$h_2(x) = -\log\left(\int p^2(\mathrm{x})d\mathrm{x}\right) = -\log(\mathbb{E}[p(x)]) \tag{26}$$

$\mathbb{V}_2(x) = \mathbb{E}_{p(x)}[p(x)]$ is the QIP; it is the expected value of the PDF if $x$ is continuous, or PMF if $x$ is discrete. From this point on, all information potentials will be QIPs. In

addition, the subscripts will denote the PDF/PMF associated with the QIP; $\mathbb{V}_p$ will be the QIP associated with $p$.

2.1.9. Renyi's Cross-Entropy

Renyi's Cross-Entropy for two PDFs is given by:

$$h_2(p,q) = -\log\left(\int p(\mathrm{x})q(\mathrm{x})d\mathrm{x}\right) \tag{27}$$

The cross information potential is given by:

$$\mathbb{V}_2(p,q) = \left(\int p(\mathrm{x})q(\mathrm{x})d\mathrm{x}\right) \tag{28}$$

From this point, all information potentials in this paper will be quadratic. In addition, the subscripts will denote the PDF/PMF associated with the QIP; $\mathbb{V}_p$ will be the QIP associated with $p$; $\mathbb{V}_q$ will be the QIP associated with $q$; $\mathbb{V}_c$ will be the cross information potential associated with $p$ and $q$.

2.1.10. Renyi's $\alpha$-Divergence

For two PMFs, $p(\mathrm{x})$ and $q(\mathrm{x})$, the formula is:

$$\begin{aligned}\mathbb{D}_\alpha(p(\mathrm{x});q(\mathrm{x})) &= \frac{1}{\alpha-1}\log\left(\sum_{\mathrm{x}\in range(x)}\sum_{\mathrm{y}\in range(y)} p(\mathrm{x})\left(\frac{p(\mathrm{x})}{q(\mathrm{x})}\right)^{\alpha-1}\right) \\ &= \frac{1}{\alpha-1}\log\left(\sum_{\mathrm{x}\in range(x)}\sum_{\mathrm{y}\in range(y)} \frac{p^\alpha(\mathrm{x})}{q^{\alpha-1}(\mathrm{x})}\right)\end{aligned} \tag{29}$$

When $\alpha \to 1$, it converges to KL Divergence.

For two PDFs $p(\mathrm{x})$ and $q(\mathrm{x})$, the formula is:

$$\begin{aligned}\mathbb{D}_\alpha(p(\mathrm{x});q(\mathrm{x})) &= \frac{1}{\alpha-1}\log\left(\int p(\mathrm{x})\left(\frac{p(\mathrm{x})}{q(\mathrm{x})}\right)^{\alpha-1}d\mathrm{x}\right) \\ &= \frac{1}{\alpha-1}\log\left(\int \frac{(p(\mathrm{x}))^\alpha}{(q(\mathrm{x}))^{\alpha-1}}d\mathrm{x}\right)\end{aligned} \tag{30}$$

2.1.11. Euclidean Divergence

The Euclidean Divergence between PDFs $p(\mathrm{x})$ and $q(\mathrm{x})$ is given by the following formula:

$$\mathbb{D}_{ED}(p(\mathrm{x});q(\mathrm{x})) = \int (p(\mathrm{x})-q(\mathrm{x}))^2 d\mathrm{x} \tag{31}$$

$$= \int (p(\mathrm{x}))^2 d\mathrm{x} + \int (q(\mathrm{x}))^2 d\mathrm{x} - 2\int (p(\mathrm{x})q(\mathrm{x}))d\mathrm{x}$$

If we want to express the Euclidean Divergence between $p$ and $q$ in terms of QIP, it is given by:

$$\mathbb{D}_{ED}(p(\mathrm{x})\|q(\mathrm{x})) = \mathbb{V}_p + \mathbb{V}_q - 2\mathbb{V}_c \tag{32}$$

The Euclidean Divergence is a symmetric measure.

2.1.12. Cauchy–Schwarz Divergence

The Cauchy–Schwarz (CS) Divergence, for probability density functions $p(\mathrm{x})$ and $q(\mathrm{x})$, is given by the following formula [6]:

$$\mathbb{D}_{CS}(p(\mathrm{x}); q(\mathrm{x})) = \frac{-1}{2} \log\left( \frac{\left(\int p(\mathrm{x}) q(\mathrm{x}) d\mathrm{x}\right)^2}{\int p^2(\mathrm{x}) d\mathrm{x} \int q^2(\mathrm{x}) d\mathrm{x}} \right) \tag{33}$$

If we have PDFs $p(\mathrm{x})$ and $q(\mathrm{x})$, and want to express the CS Divergence between them in terms of QIP, it is given by:

$$\mathbb{D}_{CS}(p(\mathrm{x}) \| q(\mathrm{x})) = \log\left( \frac{\mathbb{V}_p \mathbb{V}_q}{\mathbb{V}_c^2} \right) \tag{34}$$

Unlike KL Divergence and Renyi's $\alpha$-Divergence, CS Divergence is a symmetric measure.

*2.2. Monte Carlo*

Monte Carlo methods [13,14] are methods using random simulations; they are often used to estimate integrals. This is useful in statistics for estimating expected values. In machine learning, it is especially useful for the case of gradient estimation.

Given the fact that we have an integrable function $\mathrm{g} : [0,1]^{\mathrm{d}} \mapsto \mathbb{R}$, we can look at the following integral [14]:

$$\mathrm{A} = \int_{[0,1]^{\mathrm{d}}} \mathrm{g}(\mathrm{x}) \mathrm{dx}$$

If the domain of the integral is in $\mathbb{R}^{\mathrm{d}}$, the change of variables can change the domain to $[0,1]^{\mathrm{d}}$.

We can generate an i.i.d sequence $\{u_1, \ldots, u_N\}$ from a standard uniform distribution over $[0,1]^{\mathrm{d}}$. Then, the Monte Carlo estimator is given by

$$\mathrm{A_N} = \frac{1}{\mathrm{N}} \sum_{n=1}^{\mathrm{N}} \mathrm{g}(u_n) \tag{35}$$

We can also use a more general Monte Carlo estimator for integration [15]. We can find a PDF $f$ of random variable $z \in [0,1]^{\mathrm{d}}$ such that $f > 0$ on $[0,1]^{\mathrm{d}}$ and $\int_{[0,1]^{\mathrm{d}}} f(\mathrm{x}) d\mathrm{x} = 1$.

Given that $\mathrm{h}(\mathrm{x}) = \mathrm{g}(\mathrm{x})/f(\mathrm{x})$, our integral becomes

$$\mathrm{A} = \int_{[0,1]^{\mathrm{d}}} \mathrm{h}(\mathrm{x}) f(\mathrm{x}) d\mathrm{x} = \mathbb{E}[\mathrm{h}(z)] = \mathbb{E}_f[\mathrm{h}(z)]$$

The Monte Carlo estimator is then given by

$$\mathrm{A_N} = \frac{1}{\mathrm{N}} \sum_{k=1}^{\mathrm{N}} h(x_k) \tag{36}$$

Thus, the steps are

(1) sample i.i.d sequence $\{x_1, x_2 \ldots x_{\mathrm{N}}\} \sim f$
(2) Plug i.i.d. sequence into the estimator given by Equation (36).

If we set $f(\mathrm{x})$ =1 over the region $[0,1]^{\mathrm{d}}$, we arrive at Equation (35). From the Law of Large Numbers, $A_{\mathrm{N}}$ for both Monte Carlo estimators converge to A when $\mathrm{n} \to \infty$, and the convergence rate does not depend on dimension d; this provides an advantage over traditional integration.

*2.3. Autoencoders*

The Autoencoder is a self supervised learning algorithm that is used for lossy data compression [16]. The compression is specific to the data that is used to train the model.

There is an encoder that creates a compressed representation; this representation goes into the decoder and outputs a reconstructed input. This algorithm test label is the data input itself. Autoencoders can be considered a nonlinear Principal Component Analysis (PCA). Often times, the compressed representation has a smaller dimension than the input and output. Figure 1 shows the architecture of an Autoencoder with the MNIST data set as the data.

The encoder and decoder are typically multilayered perceptrons (MLPs), but they can be replaced with convolutional neural networks, which becomes a Convolutional Autoencoder. The convolutional autoencoder is better with reconstructing image data. The use of convolution in deep learning actually refers to what is known as cross-correlation in signal processing terminology [1]. The LSTM-Autoencoder uses LSTMs instead of MLPs for the encoder and decoder.

One important variation of the Autoencoder is the Denoising Autoencoder (DAE) [17]; the DAE is used to clean data that is corrupted by noise. Random noise is added to the input, but the reconstruction loss is between the clean input and the output. Some noise that can be added includes salt & pepper noise, additive white Gaussian noise (AWGN), and masking noise.

Discrete-time white noise is a zero mean discrete-time random process with finite variance whose samples are serially uncorrelated random variables. AWGN is discrete-time white noise that is Gaussian and additive. Additive implies it is added to the original signal. We add AWGN to the original signal/image $\mathbf{x}$. If our signal is a 1D discrete time series, the AWGN vector added to the signal can be represented as $\boldsymbol{w} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$. To introduce masking noise into $\mathbf{x}$, a certain fraction of the elements of $\mathbf{x}$ are randomly chosen and set to 0. Salt & pepper noise is when a certain fraction of the elements of $\mathbf{x}$ are randomly chosen and set to the min or max possible value. This is chosen by a fair coin flip. We can also convolve the input $\mathbf{x}$ with a Gaussian filter, blurring the input [18]. In the context of MNIST data set, we can corrupt the data by adding a block of white pixels to the center of the digits [18]. Salt & pepper noise and masking noise both corrupt a fraction of the elements in a signal significantly, while not affecting the others. By denoising, we are attempting to recover the original values of the elements that were corrupted. The only scenario where this is possible is if, in high dimensional probability distributions, there is a dependency between dimensions. What we expect when training the DAE is that it learns these dependencies. Thus, for low dimensional distribution, it does not make sense to use the DAE approach as much.

Convolutional Denoising Autoencoders (CDAEs) are DAEs with convolutional layers. Stacked denoising autoencoders (SAE) are when we are stacking layers of DAEs.

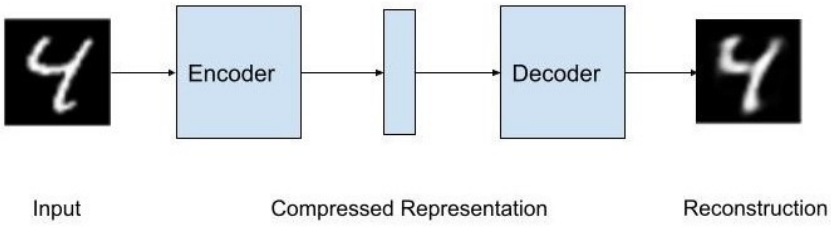



**Figure 1.** Autoencoder with MNIST input. The input is fed into the encoder. The encoder outputs a compressed representation, which is then inputted into the decoder. The decoder outputs a reconstruction of the input.

### *2.4. Bayesian Networks*

For any joint probability distribution, their independence/dependence relationships can be depicted using Probabilistic Graphical Models. When the relationships are represented via directed acyclical graphs (DAGs), the graphical models are known as Bayesian Networks. Other names for Bayesian Networks include Directed Graphical Models and

Belief Networks. To illustrate the use of Bayesian Networks, we will use an example from [19].

A woman named Tracey notices that her lawn is wet in the morning. She wonders whether it is from the rain or her accidentally leaving the sprinklers on the previous night. She then sees that her neighbor Jack also has a wet lawn. Her conclusion was that it rained last night.

Our variables are:

$r \in \{0, 1\}$, where $r = 1$ denotes it was raining last night, 0 denotes it was not raining last night.

$s \in \{0, 1\}$, where $s = 1$ denotes Tracey left the sprinklers on the previous night, and 0 otherwise.

$j \in \{0, 1\}$, where $j = 1$ indicates that Jack's lawn is wet, and 0 otherwise.

$t \in \{0, 1\}$, where $t = 1$ Denotes that Tracey's grass is wet, and 0 otherwise.

We can represent this with a joint probability function $p(\mathrm{t}, \mathrm{j}, \mathrm{r}, \mathrm{s})$. Using the chain rule of probability, we can decompose this into:

$$\begin{aligned} p(\mathrm{t}, \mathrm{j}, \mathrm{r}, \mathrm{s}) &= p(\mathrm{t}|\mathrm{j}, \mathrm{r}, \mathrm{s}) p(\mathrm{j}, \mathrm{r}, \mathrm{s}) \\ &= p(\mathrm{t}|\mathrm{j}, \mathrm{r}, s) p(\mathrm{j}|\mathrm{r}, \mathrm{s}) p(\mathrm{r}, \mathrm{s}) \\ &= p(\mathrm{t}|\mathrm{j}, \mathrm{r}, \mathrm{s}) p(\mathrm{j}|\mathrm{r}, \mathrm{s}) p(\mathrm{r}|\mathrm{s}) p(\mathrm{s}) \end{aligned}$$

However, we can simplify this further by looking at the constraints. We know that the status of Tracey's lawn does not depend on Jack's; it depends on whether it rained and whether she left the sprinkler on. Thus, then:

$$p(\mathrm{t}|\mathrm{j}, \mathrm{r}, \mathrm{s}) = p(\mathrm{t}|\mathrm{r}, \mathrm{s})$$

We can also assume that the only variable affecting the status of Jack's lawn is whether it was raining the night before. Thus, then:

$$p(\mathrm{j}|\mathrm{r}, \mathrm{s}) = p(\mathrm{j}|\mathrm{r})$$

We assume that the rain is affected by the sprinkler.

$$p(\mathrm{r}|\mathrm{s}) = p(\mathrm{r})$$

Thus, our simplified model is:

$$p(\mathrm{t}, \mathrm{j}, \mathrm{r}, \mathrm{s}) = p(\mathrm{t}|\mathrm{j}, \mathrm{r}, \mathrm{s}) p(\mathrm{j}|\mathrm{r}, \mathrm{s}) p(\mathrm{r}|\mathrm{s}) p(\mathrm{s}) = p(\mathrm{t}|\mathrm{r}, \mathrm{s}) p(\mathrm{j}|\mathrm{r}) p(\mathrm{r}) p(\mathrm{s})$$

We can represent this with a Bayesian Network, as shown in Figure 2. Each node in this graph represents a variable from the joint distribution. Notice that there is a directed edge from $r$ to $j$. This means that the $r$ is the parent node of $j$, while $j$ is the child node of the $r$. If any variable is a parent of another variable, it means it is on the right side of the conditional bar; like for $p(\mathrm{j}|\mathrm{r})$, $r$ is on the right side of the conditional bar.

If we have a set of random variables $\{x_1, \ldots, x_M\}$ with certain conditional independence assumptions, we can represent their joint distribution as

$$p_{\boldsymbol{\theta}}(\mathrm{x}_1, \ldots, \mathrm{x}_M) = \prod_{j=1}^{M} p_{\boldsymbol{\theta}}\left(\mathrm{x}_j | Parents\left(x_j\right)\right) \tag{37}$$

Similarly, for a set of random vectors, $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$ we can represent their joint distributions as:

$$p_{\boldsymbol{\theta}}(\mathbf{x}_1, \ldots, \mathbf{x}_M) = \prod_{j=1}^{M} p_{\boldsymbol{\theta}}\left(\mathbf{x}_j | Parents\left(x_j\right)\right) \tag{38}$$

For root nodes in the Bayesian Network, the set of parents is the empty set. Thus, they are marginal distributions. $Parents(x_j)$ denotes the set of parent variables for node $x_j$ in the Bayesian Network.

Initially, the parameterization of each conditional probability distribution was done with a lookup table or a linear model. In deep learning, we can use neural networks to parameterize conditional distributions; this is more flexible. The meaning of a neural network parameterizing a PDF is that it is part of the function that computes the PDF [20].

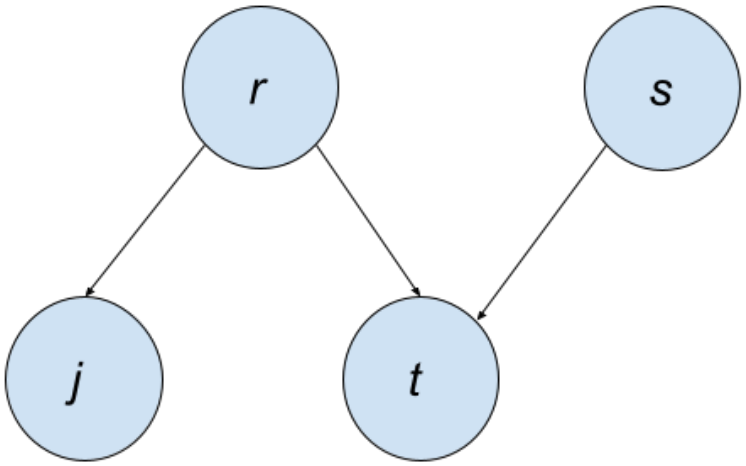



**Figure 2.** Bayesian Network of $p(\mathrm{t}|\mathrm{j},\mathrm{r},\mathrm{s})$. Since $r$ affects $j$ and $t$, it is a parent node of those two. $s$ is also a parent node of $t$.

### 2.5. Generative Models vs. Discriminative Models

Given the PDF $p(\mathbf{x},\mathrm{y})$, we generate a dataset $\mathcal{D} = \left\{\mathbf{x}^{(i)},\mathrm{y}^{(i)}\right\}_{i=1}^{N}$. We have the realizations of an i.i.d sequence $\mathcal{X} = \left\{\mathbf{x}^{(i)}\right\}_{i=1}^{N}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and each $\mathbf{x}^{(i)}$ has a label $\mathrm{y}^{(i)}$ associated with it [21]. A generative model would attempt to learn $p(\mathbf{x},\mathrm{y})$. It would then generate new examples $\mathbf{x}$ from estimated distribution. The term $p(\mathrm{y}|\mathbf{x})$ would be a discriminative model; it attempts to estimate the label generating probability distribution function. A discriminative model can predict $y$ given examples $\boldsymbol{x}$, but it cannot generate a sample of $\boldsymbol{x}$.

There are three types of generative models typically used in deep learning [22]: latent variable models, neural autoregressive models, and implicit models.

### 2.6. Latent Variable Models

Latent variables are underlying variables; often times, they are not directly observable. An example given by Dr. Nuno Vasconcelos [23] is the bridge example. There is a bridge, and there are weight sensors measuring the weight of each car. However, we do not have a camera, so we do not know what type of car it is; it could be a compact, sedan, station wagon, pick up, or van. Thus, the hidden/latent variable, represented by random variable $z$, is the type of the car, and the observed variable is the weight measured, represented by random variable $x$. Figure 3 shows the process of data generation. Our example can be represented by the Bayesian Network in Figure 4a. The latent variables and observed variables can also be random vectors, denoted as $\boldsymbol{z}$ and $\boldsymbol{x}$.

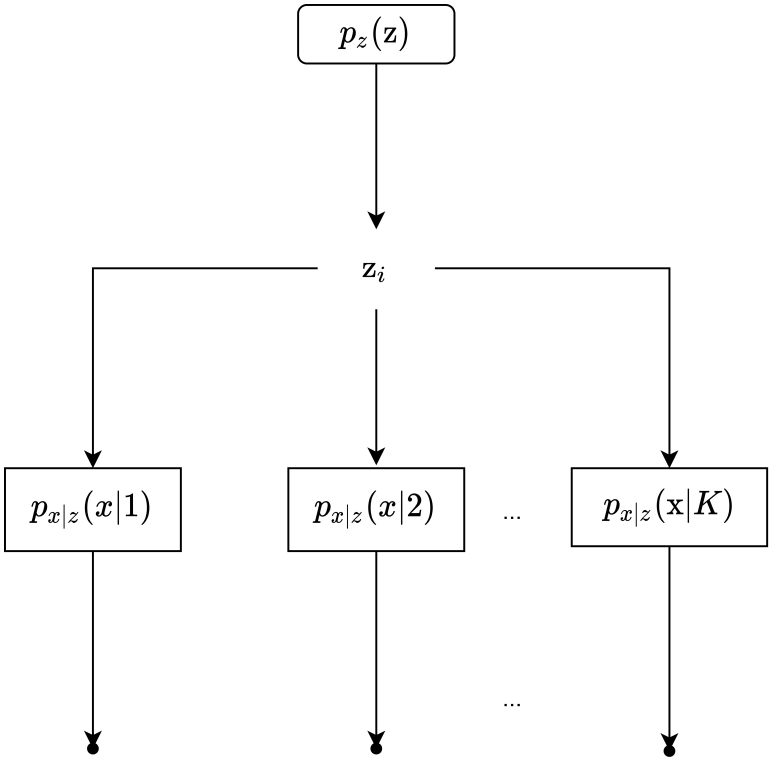



**Figure 3.** This shows the data generating process for a scenario with a latent variable. First, $z_i$ is sampled from $p(z)$; we assume $z_i$ can take K values from 1 to K. Then, given $z_i$, we can generate $x$.

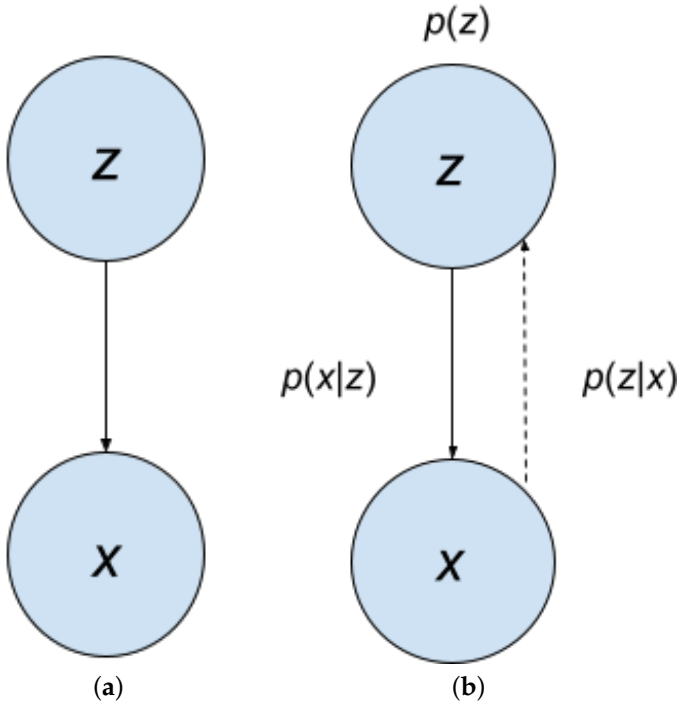



**Figure 4.** (**a**) A Bayesian Network representation of latent variable $z$ and observation variable $x$. They can also be random vectors $\boldsymbol{x}$ and $\boldsymbol{z}$; (**b**) another Bayesian Network for $z$ and $x$, this time showing the dotted arrow for inference.

Sampling our observation takes two steps. First, a sample $\boldsymbol{z}$ comes from the probability distribution $p_{\boldsymbol{z}}(\mathbf{z})$. Then, a sample $\mathbf{x}$ comes from $p_{\boldsymbol{x}|\boldsymbol{z}}(\mathbf{x}|\mathbf{z})$. This is also called generation, represented by $z \rightarrow x$ in Figure 4a,b; it is represented by the solid arrow. With $p_{\boldsymbol{z}}(\mathbf{z})$ with

$p_{x|z}(\mathbf{x}|\mathbf{z})$, we can obtain the joint density $p_{(x,z)}(\mathbf{x},\mathbf{z})$. Then, by marginalizing the joint density, we get $p_{\boldsymbol{x}}(\mathbf{x})$. Then, from there, we can get $p(\mathbf{z}|\mathbf{x})$. Obtaining $p(\mathbf{z}|\mathbf{x})$ is represented by $x \rightarrow z$. It is the inverse of generation, and is called inference. In Figure 4b, inference is represented by the dotted arrow. Inference can be obtained using Bayes Rule:

$$p_{z|x}(\mathbf{z}|\mathbf{x}) = \frac{p_{x|z}(\mathbf{x}|\mathbf{z})p_z(\mathbf{z})}{p_x(\mathbf{x})} \tag{39}$$

Often times, calculating $p_{\boldsymbol{x}}(\mathbf{x})$ is intractable, making inference intractable through this method. This leads to approximation methods. Markov Chain Monte Carlo (MCMC) methods [13] are a common collection of methods [24] used for this. However, variational inference is a quicker family of methods. They do not guarantee that we will create exact samples from the target density function asymptotically like MCMC methods [25].

If we parameterize our model with $\boldsymbol{\theta}$, we use $p_{\boldsymbol{\theta}}(\mathrm{x})$; this means that the PDF $p(\mathrm{x})$ is associated with random variable $x$ and has parameters represented by $\boldsymbol{\theta}$. We would attempt to learn $\boldsymbol{\theta}$ using maximum likelihood estimation.

When we have a latent variable model $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$, where a deep neural network parameterizes $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$, it is called a deep latent variable model (DLVM) [20].

An example of a DLVM is given by the following [26]:

$$\boldsymbol{z} = (z_1, z_2, \cdots, z_K) \sim p(\mathbf{z};\beta) = \prod_{k=1}^{K} \beta_k^{z_k}(1-\beta_k)^{1-z_k}$$
$$\boldsymbol{x} = (x_1, x_2, \cdots, x_L) \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \Leftrightarrow \mathrm{Bernoulli}(\mathrm{x}_i; \mathrm{DNN}(\mathbf{z}))$$

We have a random vector $\boldsymbol{z}$, of length K, sampled from a multivariate Bernoulli distribution. This is then fed into a neural network, denoted by DNN, which outputs random vector $\mathbf{x}$. The neural network can have L output units with sigmoid activations.

Another example of DLVM is the following [27]: If the observation data $\mathbf{x}$ of size L is binary data, the latent space is Gaussian latent space, and the observation model is a factorized multivariate Bernoulli distribution, we have the following formulas:

$$\begin{aligned} p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; 0, \boldsymbol{I}) \\ \mathbf{a} &= \mathrm{DNN}_{\boldsymbol{\theta}}(\mathbf{z}) \\ \log(p(\mathbf{x}|\mathbf{z})) &= \sum_{j=1}^{\mathrm{L}} \log(p(\mathrm{x}_j|\mathbf{z})) = \sum_{j=1}^{\mathrm{L}} \log(\mathrm{Bernoulli}(\mathrm{x}_j; \mathrm{a}_j)) \\ &= \sum_{j=1}^{\mathrm{L}} \mathrm{x}_j \log(\mathrm{a}_j) + (1-\mathrm{x}_j)\log(1-\mathrm{a}_j) \end{aligned}$$

where $\mathrm{a}_j$ is a value between 0 and 1 and $\mathbf{a}$ is a vector with $\mathrm{a}_j$'s; it can be implemented by having the output layer of the neural network have sigmoid activation functions.

A third example of a DLVM is where $\boldsymbol{z}$ is a Gaussian distribution, and $p(\mathbf{x}|\mathbf{z})$ can be a neural network with a softmax activation function for its output layer [26]. Our generative model in the case of latent variable models learns the joint PDF $p_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{x})$.

Overall, latent variable training can be summarized by the following four steps [26]:

(1) Sampling
$\boldsymbol{z} \sim p_z(\mathbf{z})$
$\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
(2) Evaluate likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_z p_z(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
(3) Train $\arg\max_{\theta} \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) = \sum_i \log \sum_z p_z(\mathbf{z})p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(\mathbf{i})}|\mathbf{z}\right)$
(4) Representation $\boldsymbol{x} \rightarrow \boldsymbol{z}$

Common latent variable models in deep learning include energy-based models, variational autoencoders, and flow models [28,29]. The VAE explicitly models the density of the distribution, so it has a prescribed Bayesian Network.

### *2.7. Neural Autoregressive Models*

In time series analysis, an autoregressive model of order p is denoted as AR(p) [30]. If we have a time series $\{y[\mathrm{n}], y[\mathrm{n}-1], \ldots, y[\mathrm{n}-\mathrm{p}]\}$, it is an AR(p) process if it satisfies the following equation:

$$y[\mathrm{n}] = \sum_{j=1}^{\mathrm{p}} \mathrm{a_j} y[\mathrm{n}-\mathrm{j}] + w[\mathrm{n}] + \mu \tag{40}$$

$y[n]$ denotes the value of $y$ (a scalar value) at time n. $y[n]$ is a linear combination of the p past values of $y$, weighted by scalar coefficients $\mathrm{a}_j$ plus some white noise $w[n]$ and the mean $\mu$ of the process ($\mathbb{E}[y[\mathrm{n}]] = \mu$).

In neural networks, there is a subtype of the AR model, called the neural autoregressive model or autoregressive neural density estimators; in deep learning literature, this subtype is often just called an autoregressive model. The neural autoregressive model involves using a Bayesian Network structure where the conditional probabilities are set to neural networks.

If we are given a Bayesian Network representation for a model, we can get a tractable gradient for our log likelihood by setting the conditional probability distributions to neural networks [31]:

$$\log(p_{\boldsymbol{\theta}}(\mathbf{x})) = \sum_{i=1}^{d} \log(p_{\boldsymbol{\theta}}(\mathbf{x}_i | Parents(\boldsymbol{x}_i))) \tag{41}$$

Parent denotes the parent nodes of $x_i$. If we assume that our Bayesian Network is fully expressive, any joint probability distribution can be decomposed to a product of conditionals, using the probability chain rule and conditional independence assumptions:

$$\log(p(\mathbf{x})) = \sum_{i=1}^{d} \log(p(\mathbf{x}_i | \mathbf{x}_{1:i-1})) \tag{42}$$

This is called a neural autoregressive model. Common neural autoregressive models include Neural Autoregressive Distribution Estimation (NADE) [32,33], Masked Autoencoder for Distribution Estimation (MADE) [34], Deep AutoRegressive Networks (DARN) [35], PixelRNN [36], PixelCNN [36,37], and WaveNet [38].

Neural AR models are slower because they sequentially generate from one dimension at a time. They also tend to model local structure better than global structure.

### *2.8. Generative Adversarial Networks (GANs)*

Two major implicit models are Generative Stochastic Networks (GSNs) and GANs [39,40].

A GAN trains a generative model *Gen* and a discriminative model *Dis* simultaneously. *Gen* attempts to estimate the distribution of the data, while *Dis* tries to estimate the probability that the data came from the training set rather than *Gen*. *Gen* tries to maximize the probability that the discriminator makes a mistake. Typically for a GAN, *Dis* is only used during training, but, afterwards, it is discarded.

The Bayesian Network in Figure 4a from Section 2.6 can also represent the generator from a GAN. This is because a random noise term $z$, often sampled from a uniform distribution, is the input to *Gen*, which outputs synthetic data $Gen(\mathbf{z})$; we can also write $Gen(\mathbf{z})$ as $\hat{\mathbf{x}}$. The GAN implicitly models the distribution and so it has an implicit Bayesian network [41]. Thus, the GAN is both an implicit model and has a latent variable model.

The algorithm is outlined in Algorithm 1. The noise prior $p_g(\mathbf{z})$ is the input to the generator; it is often typically a uniform distribution. The data generating distribution is denoted $p_{\mathrm{d}}(\mathbf{x})$; it is the data behind our real distribution. The hyperparameter k denotes how many steps the discriminator is applied. $\boldsymbol{\theta}_G$ represents the weights and biases of *Gen*,

while $\boldsymbol{\theta}_D$ represents the weights and biases of *Dis*; the subscripts denote their association with the discriminator and generator. Figure 5 shows the architecture.

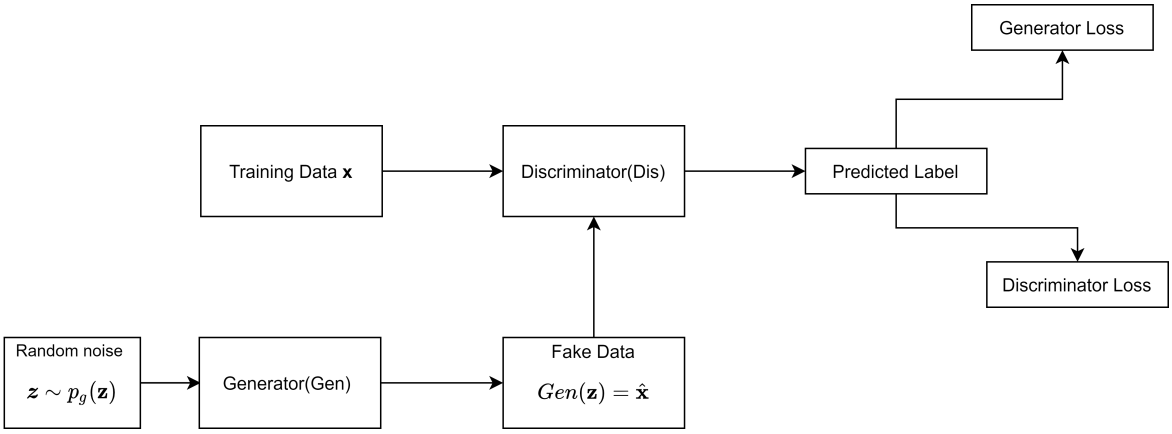



**Figure 5.** GAN architecture. A random noise vector drives our generator to create fake data. The fake data and training data are sent to the discriminator, which attempts to classify which data are fake or real.

**Algorithm 1** Original GAN algorithm

**for do** # of training iterations:
  **for** k **do** steps
    Sample minibatch $\left\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\right\} \sim p_g(\mathbf{z})$
    Sample minibatch $\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right\} \sim p_{\mathrm{d}}(\mathbf{x})$
    Update the weights of the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} \left[\log\left(Dis\left(\boldsymbol{x}^{(i)}\right)\right) + \log\left(1 - Dis\left(Gen\left(\mathbf{z}^{(i)}\right)\right)\right)\right]$$

  **end for**
  Sample minibatch $\left\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\right\} \sim p_g(\mathbf{z})$
  Update the weights of the generator by descending its stochastic gradient:

$$\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - Dis\left(Gen\left(\mathbf{z}^{(i)}\right)\right)\right).$$

**end for**

Gradient updates can use any rule. The loss function is similar to JS Divergence. When *Dis* is optimal, the weights of *Gen* are updated in a way that it minimizes the JS Divergence.

The original GAN had issues such as mode collapse, convergence, and vanishing gradients. The Wasserstein GAN [42] is a class of GAN meant to improve on these flaws; it uses Wasserstein distance instead. Conditional GANs (CGANs) [43] are another type of GAN that attempts to alleviate some of the flaws of the GAN.

The Deep Convolutional GAN (DCGAN) [44] is what many GANs are based on; it uses ADAM to optimize, uses an all convolutional network [45], and has batch normalization [46] in most layers for *Dis* and *Gen*. *Gen*'s last and *Dis*'s first layer are not batch normalized. Other GAN methods include Periodic Spatial GAN (PSGAN) [47], INFOGAN [48], CycleGAN [49], StyleGAN [50], and Self-Attention GAN (SAGAN) [51].

### *2.9. Gradient Estimation*

#### 2.9.1. REINFORCE Estimator/Score Function

Two important gradient estimators are the score function and the pathwise gradient estimator [22,26,52,53]. The score function (also known as the REINFORCE estimator) can handle non-differentiable functions; the downside is that it has a high variance.

If you have a function $p_{\boldsymbol{\theta}}(\mathbf{x})$, which is a PDF of random variable $\boldsymbol{x}$ parameterized by $\boldsymbol{\theta}$, then the score function is $\nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\mathbf{x}))$. This is the derivative of the log of our PDF w.r.t to $\boldsymbol{\theta}$. This score function can be written as:

$$\nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\mathbf{x})) = \frac{\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \tag{43}$$

The score function's expectation is zero:

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\boldsymbol{x}))] = \int p_{\boldsymbol{\theta}}(\mathbf{x}) \frac{\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})} d\mathbf{x} = \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\mathbf{x}) d\mathbf{x} = \nabla_{\boldsymbol{\theta}} 1 = \mathbf{0}$$

The score function's variance is the Fisher information. The estimator for the score function can be derived as follows:

$$\begin{aligned} &\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\mathrm{f}(\boldsymbol{x})] = \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\mathbf{x}) \mathrm{f}(\mathbf{x}) d\mathbf{x} = \int \mathrm{f}(\mathbf{x}) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{x}) d\mathbf{x} \\ &= \int p_{\boldsymbol{\theta}}(\mathbf{x}) \mathrm{f}(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\mathbf{x})) d\mathbf{x} \\ &= \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\mathrm{f}(\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\boldsymbol{x}))] \\ &\approx \frac{1}{\mathrm{L}} \sum_{\mathrm{l}=1}^{\mathrm{L}} \mathrm{f}\left(\boldsymbol{x}^{(l)}\right) \nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(l)}\right)); \quad \boldsymbol{x}^{(l)} \sim p_{\boldsymbol{\theta}}(\mathbf{x}) \end{aligned} \tag{44}$$

#### 2.9.2. Pathwise Gradient Estimator

The pathwise gradient estimator is also known as the reparameterization trick or the pathwise derivative estimator. However, it has low variance, so it is a common choice. More details about this estimator will be shown in the next section. For a continuous distribution for $\boldsymbol{x}$, direct sampling has an equivalent indirect process:

$$\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}) \quad \equiv \quad \mathbf{x} = \mathrm{g}(\boldsymbol{\epsilon}, \boldsymbol{\theta}), \quad \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$$

This statement means an indirect way to create samples $\mathbf{x}$ from $p_{\boldsymbol{\theta}}(\mathbf{x})$ is to sample from $p(\boldsymbol{\epsilon})$ first; this distribution is independent of $\boldsymbol{\theta}$. The next step is to apply a transformation with $\mathrm{g}(\boldsymbol{\epsilon}, \boldsymbol{\theta})$ which is deterministic. This can be called a sampling path or a sampling process.

For indirect sampling from a Gaussian distribution denoted by $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{C})$, we can reparameterize it by making $\mathrm{g}(\boldsymbol{\epsilon}, \boldsymbol{\theta})$ a location scale transformation, given by $\mathrm{g}(\boldsymbol{\epsilon}, \boldsymbol{\theta}) = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{\epsilon}$, where $\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{C}$. $\boldsymbol{L}$ is a lower triangular matrix with nonzero diagonal values and $\epsilon$ is sampled from a standard isotropic multivariate Gaussian $p(\boldsymbol{\epsilon}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

We can derive the gradient estimator by the following:

$$\begin{aligned} &\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\mathrm{f}(\boldsymbol{x})] = \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\mathbf{x}) \mathrm{f}(\mathbf{x}) d\mathbf{x} \\ &= \nabla_{\boldsymbol{\theta}} \int p(\boldsymbol{\epsilon}) \mathrm{f}(\mathrm{g}(\boldsymbol{\epsilon}, \boldsymbol{\theta})) d\boldsymbol{\epsilon} \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\theta}} \mathrm{f}(\mathrm{g}(\boldsymbol{\epsilon}, \boldsymbol{\theta}))] \\ &\approx \frac{1}{\mathrm{L}} \sum_{l=1}^{\mathrm{L}} \nabla_{\boldsymbol{\theta}} \mathrm{f}\left(\mathrm{g}\left(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{\theta}\right)\right); \quad \boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon}) \end{aligned}$$

Thus, our pathwise gradient estimator where our $\boldsymbol{x}$ is distributed according to $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{C})$ is given by:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\mathrm{f}(\boldsymbol{x})] \approx \frac{1}{\mathrm{L}} \sum_{l=1}^{\mathrm{L}} \nabla_{\boldsymbol{\theta}} \mathrm{f}\Big(\mathrm{g}\Big(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{\theta}\Big)\Big); \quad \boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon}) \tag{45}$$

where $\mathrm{g}(\boldsymbol{\epsilon}, \boldsymbol{\theta}) = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{\epsilon}, \boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{C}, p(\boldsymbol{\epsilon}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

**3. Variational Inference**

In Bayesian statistics, parameters that we estimate are random variables instead of deterministic variables. For latent random vector $\boldsymbol{z}$ and observation variables/vector $\boldsymbol{x}$,$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is known as the posterior distribution; $p(\mathbf{z})$ is the prior distribution, $p_{\boldsymbol{\theta}}(\mathbf{x})$ is the model evidence or marginal likelihood, and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is the likelihood. We perform updates on prior $p_{\boldsymbol{z}}(\mathbf{z})$ using Bayesian rule.

Variational inference is a particular method for approximating the posterior distribution. We approximate the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ with the approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{z})$; it is also known as the variational posterior, where $\boldsymbol{\phi}$ represents the variational parameters. We will optimize over $\boldsymbol{\phi}$ so we can fit the variational posterior to the real posterior. Any valid distribution for $q_{\boldsymbol{\phi}}(\mathbf{z})$ can be used as long as we can sample data from it and we can compute $\log(q_{\boldsymbol{\phi}}(\mathbf{z}))$ and $\nabla_{\boldsymbol{\phi}} \log(q_{\boldsymbol{\phi}}(\mathbf{z}))$. Thus, we want to solve the following for all $\mathbf{x}^{(i)}$

$$\min_{q_{\boldsymbol{\phi}}(z)} \mathbb{D}_{KL}\Big(q_{\boldsymbol{\phi}}(\mathbf{z}) \| p_{\boldsymbol{\theta}}\Big(\mathbf{z}|\mathbf{x}^{(\mathbf{i})}\Big)\Big) \tag{46}$$

We are taking the reverse-KL Divergence between $p$ and $q$. There are different posteriors for each datapoint $x^{(i)}$, so we learn a different $\boldsymbol{\phi}$ for each datapoint. To make this calculation more quick, we can use an amortized formulation for variational inference:

$$\min_{\phi} \sum_{i} \mathbb{D}_{KL}\Big(q_{\boldsymbol{\phi}}\Big(\mathbf{z}|\mathbf{x}^{(\mathbf{i})}\Big) \| p_{\boldsymbol{\theta}}\Big(\mathbf{z}|\mathbf{x}^{(\mathbf{i})}\Big)\Big) \tag{47}$$

In this formulation, we predict $\boldsymbol{\phi}$ with a neural network, called an inference network; variational parameters $\boldsymbol{\phi}$ refer to the parameters of this inference network. The downside of this formulation is less precision. With this, we can derive the Evidence Lower Bound (ELBO):

$$\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) = \int_{-\infty}^{\infty} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log\left(\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right) d\mathbf{z} = \quad \mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}\left[\log\left(\frac{q_{\boldsymbol{\phi}}(z|\mathbf{x})}{p_{\boldsymbol{\theta}}(z|\mathbf{x})}\right)\right]$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}[\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})) - \log(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\mathbf{x}))] = \mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}\left[\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})) - \log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}|z)p(z)}{p_{\boldsymbol{\theta}}(\mathbf{x})}\right)\right]$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}[\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})) - \log(p_{\boldsymbol{\theta}}(\mathbf{x}|z)) - \log(p(z)) + \log(p_{\boldsymbol{\theta}}(\mathbf{x}))]$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}[\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})) - \log(p_{\boldsymbol{\theta}}(\mathbf{x}|z)) - \log(p(z))] + \log(p_{\boldsymbol{\theta}}(\mathbf{x}))$$

$$= -\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) + \log(p_{\boldsymbol{\theta}}(\mathbf{x}))$$

where $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}[-\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})) + \log(p_{\boldsymbol{\theta}}(\mathbf{x}|z)) + \log(p(z))]$.
Since $\mathbb{D}_{KL}((q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \geq 0$, we have $-\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) + \log(p_{\boldsymbol{\theta}}(\mathbf{x})) \geq 0$.

Thus, we arrive at:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) \leq \log(p_{\boldsymbol{\theta}}(\mathbf{x})) \tag{48}$$

or

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}[-\log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})) + \log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})) + \log(p(\mathbf{z}))] \leq \log(p_{\boldsymbol{\theta}}(\mathbf{x})) \tag{49}$$

This is called the Evidence Lower Bound (ELBO), or variational lower bound.

We can derive a second formulation as follows:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\left[-\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})) + \log(p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z})) + \log(p(\boldsymbol{z}))\right] =$$
$$\mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})}\right)\right] \leq \log(p_{\boldsymbol{\theta}}(\mathbf{x}))$$

Thus, we arrive at:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right)\right] \leq \log(p_{\boldsymbol{\theta}}(\mathbf{x})) \tag{50}$$

We also derive a third formulation as follows:

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(z|\mathbf{x})}\left[-\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})) + \log(p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z})) + \log(p(\boldsymbol{z}))\right]$$
$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}[\log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}))] + \mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\left[\log\left(\tfrac{p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})}\right)\right]$$
$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}[\log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}))] - \mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|\mathrm{p}(\boldsymbol{z}))$$

Thus, we arrive at:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}[\log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}))] - \mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|\mathrm{p}(\mathbf{z})) \leq \log(p_{\boldsymbol{\theta}}(\mathbf{x})) \tag{51}$$

We would train our model by maximizing the ELBO $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})$ with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, which are our model parameters and variational parameters.

Taking the gradient of ELBO w.r.t. to $\boldsymbol{\theta}$ is easily calculated.

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \nabla_{\boldsymbol{\theta}}\mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}\left[\log\left(\tfrac{p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})}\right)\right]$$
$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(z|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}}\log(p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{z}))]$$
$$\approx \tfrac{1}{\mathrm{L}}\textstyle\sum_{l=1}^{\mathrm{L}}\nabla_{\boldsymbol{\theta}}\log(p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{z}^{(l)})) \text{ with } \boldsymbol{z}^{(l)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$$

Thus, we estimate the gradient w.r.t to $\boldsymbol{\theta}$ using the formula

$$\frac{1}{\mathrm{L}}\sum_{l=1}^{\mathrm{L}}\nabla_{\theta}\log(p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{z}^{(l)})) \text{ with } \boldsymbol{z}^{(l)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \tag{52}$$

We can generate samples from $q$ and use it to calculate each $\nabla_{\theta}\log(p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}^{(l)}))$, and average these individual gradients to estimate the gradient of the ELBO w.r.t $\boldsymbol{\theta}$.

Estimating $\nabla_{\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})$ is more difficult. This is because we cannot bring the gradient inside the expectation because the expectation is a function of $\boldsymbol{\phi}$.

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

The score function and pathwise gradient estimator are both possible methods to estimate the gradient. The score function can apply to latent variables that are continuous or discrete. The pathwise gradient estimator applied to continuous latent variables and requires that the function being estimated is differentiable. For the pathwise gradient estimator, we reparameterize a sample from $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ by expressing it as a function of a sample $\boldsymbol{\epsilon}$ from some fixed distribution $p(\boldsymbol{\epsilon})$:

$$\mathbf{z} = \mathrm{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \tag{53}$$

$p(\boldsymbol{\epsilon})$ is independent of $\mathbf{x}$ or $\boldsymbol{\phi}$. We can bring $\nabla_{\boldsymbol{\phi}}$ inside the expectation because $p(\boldsymbol{\epsilon})$ does not depend on $\phi$. We assume $g(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})$ is differentiable with respect to $\boldsymbol{\phi}$. Thus, our

pathwise gradient estimator for $\nabla_{\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x})$, where $\boldsymbol{z}$ is supposed to be sampled from $\mathcal{N}(\mathbf{z};\boldsymbol{\mu},\boldsymbol{C})$, can be derived using Equation (45):

$$\nabla_{\boldsymbol{\phi}}\mathbb{E}_{q_{\boldsymbol{\phi}}(z)}[\mathrm{f}(\boldsymbol{z})] \approx \frac{1}{\mathrm{L}}\sum_{l=1}^{\mathrm{L}}\nabla_{\boldsymbol{\phi}}\mathrm{f}\Big(\mathrm{g}\Big(\boldsymbol{\epsilon}^{(l)};\boldsymbol{\phi}\Big)\Big); \quad \boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon}) \tag{54}$$

where $\boldsymbol{z} = \mathrm{g}(\boldsymbol{\epsilon},\boldsymbol{\phi},\boldsymbol{x}) = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{\epsilon}, \boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{C}, p(\boldsymbol{\epsilon}) = \mathcal{N}(\mathbf{0},\mathbf{I})$

We can choose our $q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}\big(\mathbf{z};\boldsymbol{\mu},\mathrm{diag}\big(\sigma^2\big)\big)$, which is a multivariate Gaussian distribution with a diagonal matrix as its covariance matrix. $\mu$ is the mean vector and $\sigma^2$ is a vector that creates covariance matrix $\mathrm{diag}\big(\sigma^2\big)$, which we can sample using

$$\boldsymbol{z} = \mathrm{g}(\boldsymbol{\epsilon},\boldsymbol{\phi},\boldsymbol{x}) = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon} = \boldsymbol{\mu} + \boldsymbol{\sigma}\odot\boldsymbol{\epsilon} \tag{55}$$

where $\mathrm{diag}(\boldsymbol{\sigma}^2) = \boldsymbol{L}\boldsymbol{L}^T$, and $\odot$ is an elementwise multiplication operator. If we have $z \sim \mathcal{N}\big(\mu,\sigma^2\big)$, we can reparameterize it by

$$z = \mu + \sigma\epsilon, \epsilon \sim p(\boldsymbol{\epsilon}) = \mathcal{N}(0,1)$$

If we are not using amortized variational inference, the formula for ELBO is different.

## 4. The Variational Autoencoder

The Variational Autoencoder uses an inference network as its encoder. The VAE has a MLP encoder and an MLP decoder [2]. The encoder is the variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and is an inference/recognition model. The decoder is a generative model, and it represents the likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

A joint inference distribution can be defined as

$$q_{\phi}(\mathbf{x},\mathbf{z}) \equiv p_{\boldsymbol{\theta}}(\mathrm{x})q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$$

$q_{\boldsymbol{\phi}}(\mathbf{z})$ is called the aggregated posterior, given by the following:

$$\int_{range(\boldsymbol{x})} p_{\boldsymbol{\theta}}(\mathbf{x})q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})d\mathbf{x} \equiv \int_{range(\boldsymbol{x})} q_{\phi}(\mathbf{x},\mathbf{z})d\mathbf{x} \equiv q_{\boldsymbol{\phi}}(\mathbf{z}) \tag{56}$$

The prior is a standard multivariate isotropic Gaussian distribution, $\mathrm{p}_z(\mathrm{z}) = \mathcal{N}(\mathbf{z};\mathbf{0},\boldsymbol{I})$, while the likelihood function $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is a Gaussian distribution or a Bernoulli distribution. The Gaussian likelihood is $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}\big(\mathbf{x};\boldsymbol{\mu}_{decoder},diag(\sigma^2_{decoder})\big)$, which is a multivariate Gaussian with a diagonal covariance matrix. The posterior distribution can be any PDF, but is assumed to be approximately a multivariate Gaussian with a diagonal covariance matrix. The variational posterior is also taken to be a multivariate Gaussian with a diagonal covariance matrix, given by $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}\big(\mathbf{z};\boldsymbol{\mu},\mathrm{diag}\big(\sigma^2\big)\big)$; $\boldsymbol{\mu}$ and $\sigma^2$ from the variational posterior are outputs of the encoder.

The weights and biases for the encoder are the variational parameters $\boldsymbol{\phi}$, while the weights and biases for the decoder are the model parameters $\boldsymbol{\theta}$.

We sample $\mathbf{z}$ from the encoder using the reparameterization trick; so the encoder outputs $\boldsymbol{\mu}$ and $\sigma^2$, and generates $\boldsymbol{\epsilon}$ from $\mathcal{N}(0,\boldsymbol{I})$. The variable $\boldsymbol{z}$ is the input to the decoder, which then generates a new example of $\boldsymbol{x}$. Equation (55) is used.

Note that, in practice, the encoder outputs $\log(\sigma^2)$ instead of $\sigma^2$ to ensure positive values for $\sigma^2$. We can then retrieve $\sigma^2$ by taking the exponential of $\log(\sigma^2)$. We can also retrieve $\sigma$ by taking the exponential of $(0.5)\log(\sigma^2)$. In the literature, they often refer to the output as $\sigma$ instead of $\sigma^2$.

If we assume that our encoder has one hidden layer, and the distribution is multivariate Gaussian with a diagonal covariance matrix, we can write it and the sampling process as:

$$\begin{aligned}\mathbf{h} &= \tanh(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)\\ \boldsymbol{\mu} &= \mathbf{W}_2\mathbf{h} + \mathbf{b}_2\\ \log(\boldsymbol{\sigma}^2) &= \mathbf{W}_3\mathbf{h} + \mathbf{b}_3\\ \mathbf{z} &= \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}\end{aligned}$$

$\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$ are the weights and biases of the encoder MLP, so they are variational parameters $\boldsymbol{\phi}$.

An encoder with any number of hidden layers can be summarized with the following equations:

$$\begin{aligned}\boldsymbol{\epsilon} &\sim \mathcal{N}(0, \mathbf{I})\\ (\boldsymbol{\mu}, \log(\boldsymbol{\sigma}^2)) &= \text{ EncoderNeuralNet }_{\phi}(\mathbf{x})\\ \mathbf{z} &= \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}\end{aligned}$$

For the decoder, we have two choices.

(1) Multivariate Bernoulli MLP for decoder:

The likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is a multivariate Bernoulli. With decoder input z, the probabilities of the decoder are calculated with the MLP. $\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2\}$ are the weights and biases of the decoder MLP. The hidden layer has a tanh activation function, while the output layer has a sigmoid activation function $sig(.)$. The output is plugged into the log likelihood, getting a Cross-Entropy (CE) function.

$$\begin{aligned}\mathbf{h} &= \tanh(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1)\\ \mathbf{y} &= sig(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)\\ \log(p(\mathbf{x}|\mathbf{z})) &= \sum_{i=1}^{D} \mathrm{x}_i \log(\mathrm{y}_i) + (1 - \mathrm{x}_i) \cdot \log(1 - \mathrm{y}_i)\end{aligned}$$

The equations for more hidden layers can be written as

$$\begin{aligned}\mathbf{y} &= \text{ DecoderNeuralNet }_{\boldsymbol{\theta}}(\mathbf{z})\\ \log(p(\mathbf{x}|\mathbf{z})) &= \sum_{j=1}^{\mathrm{d}} \log(p(\mathrm{x}_j|\mathbf{z})) = \sum_{j=1}^{\mathrm{d}} \log(\text{Bernoulli}(\mathrm{x}_j; \mathrm{y}_j))\\ &= \sum_{j=1}^{\mathrm{d}} (\mathrm{x}_j \log(\mathrm{y}_j) + (1 - \mathrm{x}_j) \log(1 - \mathrm{y}_j))\end{aligned}$$

The variable d is the dimensionality of $\mathbf{x}$ and Bernoulli $(.; \mathrm{y})$ is the Bernoulli PMF. $\forall \mathrm{y}_j \in \mathbf{y}$ : $0 \leq y_j \leq 1$. This $\mathbf{y}$ can be implemented by making the last layer of the decoder a sigmoid function. This is similar to the second example of a DLVM in Section 2.6.

(2) Gaussian MLP as decoder

This is the case where the decoder distribution is a multivariate Gaussian with a diagonal covariance structure:

$$\begin{aligned}
\mathbf{h} &= \tanh(\mathbf{W}_3\mathbf{z} + \mathbf{b}_3) \\
\boldsymbol{\mu}_{decoder} &= \mathbf{W}_4\mathbf{h} + \mathbf{b}_4 \\
\log(\sigma^2_{decoder}) &= \mathbf{W}_5\mathbf{h} + \mathbf{b}_5 \\
\log(p(\mathbf{x}|\mathbf{z})) &= \log(\mathcal{N}\Big(\mathbf{x}; \boldsymbol{\mu}, diag(\boldsymbol{\sigma}^2_{decoder})\Big)) \\
&= \frac{-L}{2} log(2\pi) + \frac{-1}{2}\sum \sigma^2_{decoder,i} + \sum \frac{(x_i - \mu_{decoder,i})^2}{-2\sigma^2_{decoder,i}}
\end{aligned}$$

where $\{\mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_5, \mathbf{b}_3, \mathbf{b}_4, \mathbf{b}_5\}$ are the weights and biases of the decoder MLP, so they are the model parameters $\boldsymbol{\theta}$.

Those are the derivations for the forward propagation in the VAE. Figure 6 shows the architecture of the VAE for a forward pass, excluding the ELBO calculation.

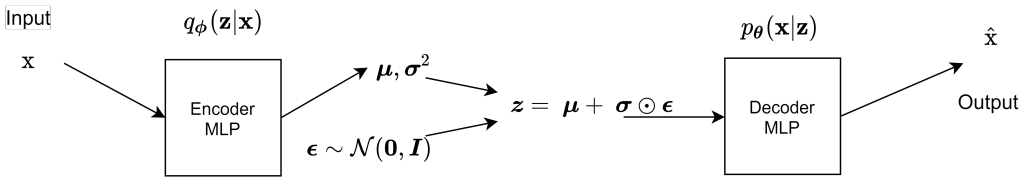



**Figure 6.** Diagram of the variational autoencoder. The input $\mathbf{x}$ goes into the encoder, which then creates a sample of $z$ using the reparameterization trick. $\mathbf{z}$ is the input to the decoder, which outputs a new example of $\mathbf{x}$, denoted as $\tilde{\mathbf{x}}$.

We will get ELBO estimates by looking at this equation:

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{z}))] - \mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_z(\mathbf{z})) \leq \log(p_{\boldsymbol{\theta}}(\mathbf{x}))$$

$\mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}[\log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}))]$ is a reconstruction loss, while $\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$ is a regularizing term.

We derive the expression to get $-\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_z(\mathbf{z}))$ [2].

$$\begin{aligned}
\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_z(\mathbf{z})) &= \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\big(\log(p_z(\mathbf{z})) - \log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\big)\mathrm{d}\mathbf{z} \\
&= \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})(\log(p_z(\mathbf{z}))\mathrm{d}\mathbf{z} - \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\mathrm{d}\mathbf{z}
\end{aligned}$$

Split into two parts:

$$\begin{aligned}
\int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})(\log(p_z(\mathbf{z}))\mathrm{d}\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \mu, \mathrm{C_v})\log(\mathcal{N}(\mathbf{z}; 0, \boldsymbol{I}))\mathrm{d}\mathbf{z} \\
&= \frac{-j}{2}\log(2\pi) + \frac{1}{2}\sum_{j=1}^{J}\Big(\mu_j^2 + \sigma_j^2\Big)
\end{aligned}$$

$$\begin{aligned}
\int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\mathrm{d}\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \mu, C_v)\log(\mathcal{N}(\mathbf{z}; \mu, C_v))\mathrm{d}\mathbf{z} \\
&= \frac{-j}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{J}\Big(1 + \log\Big(\sigma_j^2\Big)\Big)
\end{aligned}$$

We can then add the two terms:

$$
\begin{aligned}
&-\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_z(\mathbf{z})) = \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\mathrm{d}\mathbf{z} - \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})(\log(p_z(\mathbf{z}))\mathrm{d}\mathbf{z}.\\
&= \frac{-j}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{J}(1+\log(\sigma_j^2)) - \left(\frac{-j}{2}\log(2\pi) + \frac{1}{2}\sum_{j=1}^{J}(\mu_j^2+\sigma_j^2)\right)\\
&= \frac{1}{2}\sum_{j=1}^{J}(1+\log(\sigma_j^2) - \sigma_j^2 - \mu_j^2)
\end{aligned}
$$

$\sigma_j^2$ and $\mu_j$ represent the $j$th component of the $\boldsymbol{\sigma}^2$ and $\boldsymbol{\mu}$ vectors for a given datapoint inputted into the encoder.

Our total training set is $\mathcal{X} = \left\{\mathbf{x}^{(i)}\right\}_{i=1}^{\mathrm{N}}$. Our Stochastic Gradient Variational Bayes Estimator ( SGVB Estimator) estimates the lower bound for one datapoint by:

$$
\begin{aligned}
&\widetilde{\mathcal{L}}\left(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x}^{(i)}\right) = -\mathbb{D}_{KL}\left(q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(i)}\right)\|p_{\boldsymbol{\theta}}(\mathbf{z})\right) + \frac{1}{L}\sum_{l=1}^{\mathrm{L}}\left(\log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}\right)\right)\\
&\text{where } \mathbf{z}^{(i,l)} = g\left(\boldsymbol{\phi},\boldsymbol{\epsilon}^{(i,l)},\mathbf{x}^{(i)}\right) \text{ and } \boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})
\end{aligned}
$$

We can then have our SGVB estimator using minibatches from $\mathcal{X}$, where $\mathcal{X}^{(M)}$ is the M-th minibatch.

We can have $\mathcal{X}^{(M)} = \left\{\mathbf{x}^{(i)}\right\}_{i=1}^{\mathrm{M}}$, where $\mathcal{X}^{(M)}$ is a minibatch from $\mathcal{X}$ with M datapoints. Then, we can estimate the ELBO over the full dataset $\mathcal{X}$, denoted by $\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\mathcal{X})$:

$$
\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\mathcal{X}) \simeq \widetilde{\mathcal{L}}^{(M)}\left(\boldsymbol{\theta},\boldsymbol{\phi};\mathcal{X}^{(M)}\right) = \frac{\mathrm{N}}{\mathrm{M}}\sum_{i=1}^{\mathrm{M}}\widetilde{\mathcal{L}}\left(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x}^{(i)}\right) \tag{57}
$$

where $\widetilde{\mathcal{L}}^{(M)}\left(\boldsymbol{\theta},\boldsymbol{\phi};\mathcal{X}^{(M)}\right)$ is a minibatch estimator. Empirically, it has been shown that L could be set to 1 if the size of the minibatch is large.

The method used to train the VAE to find the parameters is called the Auto-Encoding Variational Bayes (AEVB) algorithm. Algorithm 2 shows the steps of the AEVB algorithm.

**Algorithm 2** AEVB algorithm using minibatches

$\boldsymbol{\theta},\boldsymbol{\phi} \leftarrow$ Initialize parameters
**for do** # of training iterations:
  $\mathcal{X}^{(M)} \leftarrow$ Random minibatch of M datapoints from full dataset $\mathcal{X}$
  Sample $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$
  Calculate minibatch estimate $\widetilde{\mathcal{L}}^{(M)}\left(\boldsymbol{\theta},\phi;\mathcal{X}^{(M)},\boldsymbol{\epsilon}\right)$
  Calculate gradients of minibatch estimator $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\widetilde{\mathcal{L}}^{(M)}\left(\boldsymbol{\theta},\phi;\mathcal{X}^{(M)},\boldsymbol{\epsilon}\right)$
  $\boldsymbol{\theta},\boldsymbol{\phi} \leftarrow$ Update parameters using gradients with methods like SGD or Adagrad
**end for**

We have presented and derived the original variational autoencoder model; however, the variational autoencoder often refers more of a general framework, where we can choose different prior, posterior and likelihood distributions, along with many other variations. Thus, the VAE framework can refer to a continuous latent variable deep learning model that uses the reparameterization trick and amortized variational inference [22].

## 5. Problems/Tradeoffs with the VAE

While being a powerful model, the VAE has multiple problems and trade-offs. We will cover variance loss, image blurriness, posterior collapse, disentanglement, the balancing

issue, the origin gravity effect, and the curse of dimensionality. We then compare the VAE with the GAN.

### *5.1. Variance Loss and Image Blurriness*

When comparing input data to generated data for the generic Autoencoders and the VAE, there is a variance loss [54]. This was empirically measured in [54]. This phenomena is possibly due to averaging.

When being used to generate new images, VAEs tend to be more blurry compared to other generative models. Variance loss is a main cause of this [54]. In [55], they find that the maximum likelihood approach is not always the cause of blurriness, it is choice of the inference distribution. They use a sequential VAE model. Choosing flexible inference models or flexible generative models in the architecture also helps to reduce this problem [27].

The VAE-GAN reduces image blurriness by replacing the reconstruction loss term with a discriminator [56]. The multi stage VAE [57], deep residual VAE, and Hierarchical VAEs such as VAE's with inverse autoregressive flows (IAF-VAE) [58] and Noveau VAE (NVAE) [59] also improve image generation quality. PixelVAE [60], 2-Stage VAE [61], and VQ-VAE are also very effective in generating good quality images.

### *5.2. Disentanglement*

How successful machine learning methods are depends on data representation. In [62], they hypothesize that the reason behind this dependence on data representation is that multiple explanatory factors of variations of the data are entangled and hidden by the representation. Representation learning can be defined as learning data representations that makes extracting useful information easier for input into predictors [62]. Three important goals of a good representation include being distributed, invariant, and having disentangled the factors of variation. Disentanglement and disentangled representations do not have agreed upon formal definitions. An intuitive definition that is commonly used is "*a disentangled representation should separate the distinct, informative factors of variations in the data*" [63].

The vanilla VAE fails to learn disentangled representations. INFOVAE [64], $\beta$-VAE [65], $\beta$-TCVAE [66], AnnealedVAE [67], DIP-VAE I/II [68], and FactorVAE [68] are VAE variants that attempt to obtain a disentangled representation, and many of them are the state of the art for disentanglement. However, according to a large-scale empirical study by Google AI, these state-of-the-art VAE models do not really learn disentangled representations in an unsupervised manner [63]; the choice of the model is not as important as the random seeds and hyperparameters; these hyperparameters do not transfer across data sets.

### *5.3. The Balancing Issue*

In the VAE loss function for context of images, the KL Divergence regularizes the latent space, while the reconstruction loss affects the quality of the image [69]. There is a tension between these two effects. If we emphasize the reconstruction loss, the reconstruction is more powerful, but the latent space shape is affected, so the capabilities of the VAE to generate new examples are negatively affected. If we emphasize the regularizing term, the disentangling becomes better and the latent space is smoother and normalized. However, it also results in the images being more blurry.

The 2-Stage VAE uses a balancing factor that it learns during training to balance these effects. In [69], they use a deterministic variable for the decoder variance to balance these factors.

### *5.4. Variational Pruning and Posterior Collapse*

Generally, in variational inference, there is a problem called variational pruning. If we rewrite ELBO as the following:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \log(p_{\boldsymbol{\theta}}(\mathbf{x})) - \mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \tag{58}$$

The $\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$ term is known as the variational gap. When we maximize ELBO, we either decrease the variational gap or increase the log evidence. Maximum likelihood training increases the log evidence. To decrease the variational gap, there are two options. The first is to update $\boldsymbol{\phi}$ to make the variational posterior closer to the real posterior. The second way is to update $\boldsymbol{\theta}$ so that the real posterior is closer to the variational posterior; this can lead to a decrease in how well the model can fit the data. This effect can be mitigated by using a more expressive posterior.

One possible consequence of this is called variational pruning; this is when latent variables are not used for the model, and the posterior becomes the same as the prior. In variational autoencoders, this is called posterior collapse. Some researchers speculate that the KL Divergence term $\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$ in the ELBO is a cause of this phenomena. Thus, this has led to a focus on reducing the effect of this KL term. The decoder becoming powerful is another cause. Lucas [70] investigated posterior collapse; initially, they investigated it for a linear VAE, and then extended their results for nonlinear VAEs. They find that, in cases where the decoder is not powerful, posterior collapse can still happen. They also formally define posterior collapse and how to measure it.

The $\delta$-VAE [71], Variational Mixture of Posteriors prior VAE (VampPrior VAE) [72], 2-Stage VAE [61], epitomic VAE (eVAE) [73], and VQ-VAE [74] are some models that attempt to deal with preventing posterior collapse.

### *5.5. Origin Gravity Effect*

The origin gravity effect is an effect in low dimensions. Since the prior is a multivariate standard normal distribution, the probabilities are centered around the origin. This pushes the points in the latent space towards the origin. Thus, even when the data are spread around multiple clusters, the Gaussian prior tends to push the clusters centers of the latent space toward the origin. Ideally, the latent space should have separate clusters and the prior should not push the mean toward the origin. We can exploit this clustering structure by using GMM based models, such as VADE and GMM-VAE [75,76].

### *5.6. Hidden Score Function*

The pathwise gradient has a hidden score function that can lead to high variance; this is discussed more in depth in Section 6.1.

### *5.7. Curse of Dimensionality*

Since the Gaussian prior has a $L_2$ norm, it suffers from the curse of dimensionality [77]. The mass of the Gaussian distribution is no longer concentrated around the mean when we go to higher dimensions. Instead of a bell curve, a higher dimensional Gaussian resembles a uniform distribution on a surface of a hypersphere; most of the mass is on the shell of the hypersphere. This can cause inefficiencies when sampling in high dimensions [78]. The random walk Metropolis algorithm tends to perform poorly when sampling in high dimensions; the Hamiltonian Monte Carlo tends to perform better.

### *5.8. GANs vs. VAEs*

VAEs and GANs use generative models to generate new data. GANs tend to be better at generating images that are perceived by humans to be good quality; however, they do not model the density very well with respect to the likelihood criterion [27]. VAEs are the opposite; they tend to have blurry images but model the density very well with respect to the likelihood criterion. The VAE is more stable to train than the GAN.

## 6. Variations of the VAE

There are many ways to extend the VAE models. You can change the prior, the posterior/variational posterior, regularize the posterior, and change the architecture. Changing the architecture includes changing the layers to RNNs/LSTMs/CNN layers, and use other Divergence measures instead of KL Divergence. Many of these variations will often include

convolutional layers, even if not explicitly stated. In this section, we will refer to the original VAE as the vanilla VAE.

### *6.1. VAE Using an STL Estimator*

For the VAE, we can decompose the gradient of the lower bound w.r.t $\boldsymbol{\phi}$ as follows [79]:

$$\begin{aligned}\hat{\nabla}_{\mathrm{TD}}(\boldsymbol{\epsilon}, \boldsymbol{\phi}) &= \nabla_{\boldsymbol{\phi}}\left[\log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})) + \log(p_{\boldsymbol{\theta}}(\mathbf{z})) - \log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right] \\ &= \nabla_{\boldsymbol{\phi}}\left[\log(p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) + \log(p_{\boldsymbol{\theta}}(\mathbf{x})) - \log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right] \\ &= \underbrace{\nabla_{\mathbf{z}}\left[\log(p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) - \log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right]\nabla_{\boldsymbol{\phi}} g(\epsilon, \phi, \mathbf{x})}_{\text{path derivative}} - \underbrace{\nabla_{\boldsymbol{\phi}} \log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))}_{\text{score function}}\end{aligned}$$

The score function term can lead to a higher variance than necessary. One way to address this is to drop the score function term, leading to the following term being used instead of the gradient:

$$\nabla_{\mathbf{z}}\left[\log(p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) - \log(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right]\nabla_{\boldsymbol{\phi}} g(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x}) \tag{59}$$

This does not affect the bias of the estimator because the expectation of the score function is 0. In some cases, dropping it can actually increase the variance if the score function is correlated with the other terms.

We call it the STL estimator because the paper that invented this new estimator was titled "Sticking the Landing: Simple, Lower-Variance Gradient Estimators for Variational Inference".

### *6.2. ρ-VAE*

Instead of an isotropic Gaussian approximate posterior, we use an AR(1) Gaussian distribution [80], so this is a posterior variant. Note by autoregressive Gaussian that we are referring to a traditional autoregressive model. The covariance matrix of the AR(1) process is given by

$$\boldsymbol{C}_{(\rho,s)} = \mathrm{s}\begin{bmatrix} 1 & \rho & \rho^2 & \rho^3 & \cdots & \rho^{d-1} \\ \rho & 1 & \rho & \rho^2 & \cdots & \rho^{d-2} \\ \rho^2 & \rho & 1 & \rho & \cdots & \rho^{d-3} \\ \rho^3 & \rho^2 & \rho & 1 & \cdots & \rho^{d-4} \\ \vdots & & & \ddots & \ddots & \vdots \\ \rho^{d-1} & \cdots & \rho^3 & \rho^2 & \rho & 1 \end{bmatrix} \tag{60}$$

The $\rho$ parameter is a scalar parameter controlling the correlation, so it is between $-1$ and 1; $\mathrm{s} > 0$ is a scalar scaling parameter. The subscript $(\rho, \mathrm{s})$ denotes that that the covariance matrix is dependent on $\rho$ and s.

The vanilla VAE encoder outputs $\mu$ and $\sigma^2$ (or $\log(\sigma^2)$); the encoder in the $\rho$-VAE outputs $\mu$, $\rho$, and s. The determinant for this matrix is

$$\det\left(\mathbf{C}_{(\rho,s)}\right) = \mathrm{s}^{\mathrm{d}}\left(1-\rho^2\right)^{\mathrm{d}-1} \tag{61}$$

The regularization term in the loss function can be formulated as:

$$\mathbb{D}_{KL}\left(\mathcal{N}\left(\boldsymbol{\mu}^{(i)}, \boldsymbol{C}_{(\rho,s)}\right)\|\mathcal{N}(\mathbf{0}, \boldsymbol{I}_{\mathrm{d}})\right) = \frac{1}{2}\left[\left\|\boldsymbol{\mu}^{(i)}\right\|_2^2 + \mathrm{d}(\mathrm{s} - 1 - \log(\mathrm{s})) - (\mathrm{d}-1)\log\left(1-\rho^2\right)\right]$$

We can take the Cholesky decomposition of the covariance matrix; from there, we get the following lower triangular matrix:

$$\tilde{C}_{(\rho,s)} = \frac{1}{\sqrt{s}} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \rho & \sqrt{1-\rho 2} & 0 & 0 & \cdots & 0 \\ \rho^2 & \rho\sqrt{1-\rho 2} & \sqrt{1-\rho 2} & 0 & \cdots & 0 \\ \rho^3 & \rho^2\sqrt{1-\rho 2} & \rho\sqrt{1-\rho 2} & \sqrt{1-\rho 2} & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \vdots \\ \rho^d & \cdots & \rho^3\sqrt{1-\rho 2} & \rho^2\sqrt{1-\rho 2} & \rho\sqrt{1-\rho 2} & \sqrt{1-\rho 2} \end{bmatrix}$$

$\mathbf{z}^{(i)} = \boldsymbol{\mu}^{(i)} + \tilde{\mathbf{C}}^{(i)}_{(\rho,s)}\boldsymbol{\epsilon}$, can be used to generate the latent codes. $z^{(i)}$ is d-dimensional latent code associated with the $i$th input; $\epsilon$ is a d-dimensional vector sampled from a multivariate standard normal distribution, and $\tilde{\mathbf{C}}^{(i)}_{(\rho,s)}$ is a d x d lower triangular matrix for the $i$th input. Variations of the $\rho$-VAE include the $\rho-\beta$-VAE and the INFO-$\beta$-VAE.

### *6.3. Importance Weighted Autoencoder (IWAE)*

#### 6.3.1. Importance Sampling

Importance sampling is a Monte Carlo variance reduction method [81], where you have the following integral to estimate

$$\mathrm{A} = \int_{\mathrm{range}(x)} \mathrm{g(x)} f(\mathrm{x}) d\mathrm{x} = \mathbb{E}_f[\mathrm{g(x)}]$$

where $\mathrm{range}(x) \subseteq \mathbb{R}^{\mathrm{d}}$ is bounded , $\mathrm{g} : \mathrm{range}(x) \to \mathbb{R}^{\mathrm{d}}$ is bounded and integrable; $f$: PDF of a random variable $x \in \mathrm{range}(x); f \geq 0$ on $\mathrm{range}(x), f = 0$ outside of $\mathrm{range}(x)$, and $\int f(\mathrm{x})d\mathrm{x} = 1$. We choose probability distribution function $\gamma$ on $\mathrm{range}(x); \gamma \neq 0$ on $\mathrm{range}(x)$; this $\gamma$ is called the importance function.

$\int_{\mathrm{range}(x)} \mathrm{g(x)} f(\mathrm{x}) d\mathrm{x} \to \int_{\mathrm{range}(x)} \frac{\mathrm{g(x)} f(\mathrm{x})}{\gamma(\mathrm{x})} \gamma(\mathrm{x}) d\mathrm{x} = \mathbb{E}_\gamma \left[ \frac{\mathrm{g}(y) f(y)}{\gamma(y)} \right] = \mathrm{J}$, where $y \sim \gamma$

The importance sampling Monte Carlo estimator becomes

$$\widehat{J} = J_{\mathrm{N}} = \frac{1}{\mathrm{N}} \sum_{k=1}^{\mathrm{N}} \frac{\mathrm{g}(y_k) f(y_k)}{\gamma(y_k)} \tag{62}$$

The algorithm is as follows:

(1) Generate i.i.d sequence $\{y_1, \cdots, y_{\mathrm{N}}\} \sim \gamma$.
(2) Plug into Equation (62).

This is an unbiased estimator.

#### 6.3.2. Importance Sampling for a Latent Variable Model

If we are trying to train a latent variable model to perform inference, with random vectors $\boldsymbol{z}$ and $\boldsymbol{x}$, we can use importance sampling in training the likelihood. If our $f$ is our prior distribution $p_{\boldsymbol{z}}(\mathbf{z})$, and g is the log conditional likelihood $p_{\boldsymbol{\theta}}(\mathbf{x^{(i)}}|\mathbf{z})$, the expected value we are estimating is

$$\mathbb{E}_\gamma \left[ \frac{p(\boldsymbol{x}|\boldsymbol{y}) p_z(\boldsymbol{y})}{\gamma(\boldsymbol{y})} \right]$$

Then, the importance sampling estimator becomes

$$\mathbf{J}_{\mathrm{N}} = \frac{1}{\mathrm{L}} \sum_{l=1}^{\mathrm{L}} \frac{g\left(\boldsymbol{y}^{(l)}\right) p_z\left(\boldsymbol{y}^{(l)}\right)}{\gamma\left(\boldsymbol{y}^{(l)}\right)}, \boldsymbol{y}^{(l)} \sim \gamma^{(l)}$$

We would use importance sampling here if our $p_{\boldsymbol{z}}(\mathbf{z})$ was difficult to sample from, or was not informative. When training the likelihood given by $\sum_i \log\left(\sum_z p_{\boldsymbol{z}}(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x^{(i)}}|\mathbf{z})\right)$,

$\sum_k p_z(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}^{(\mathbf{i})}|\mathbf{z})$ can be estimated with $\mathrm{J_N}$, so

$$\sum_i \log\left(\sum_l p_z(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x}^{(\mathbf{i})}|\mathbf{z})\right) \approx \sum_i \log\left(\frac{1}{\mathrm{L}}\sum_{l=1}^{\mathrm{L}} \frac{p_{\boldsymbol{\theta}}\left(\boldsymbol{y}^{(i,l)}\right)}{\gamma\left(\boldsymbol{y}^{(i,l)}\right)} p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(\mathbf{i})}|\boldsymbol{y}^{(i,l)}\right)\right), \boldsymbol{y}^{(i,l)} \sim \gamma\left(\mathbf{y}^{(\mathbf{i},\mathbf{l})}\right)$$

We choose our importance function to be the variational posterior, $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

$$\sum_i \log\left(\frac{1}{\mathrm{L}}\sum_{l=1}^{\mathrm{L}} \frac{p_{\mathbf{z}}\left(\boldsymbol{z}^{(i,l)}\right)}{q\left(\boldsymbol{z}^{(i,l)}|\mathbf{x}\right)} p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(\mathbf{i})}|\boldsymbol{z}^{(i,l)}\right)\right), \boldsymbol{z}^{(i,l)} \sim q\left(\mathbf{z}^{(\mathbf{i},\mathbf{l})}\right) \tag{63}$$

The IWAE [82] is a variation that uses importance sampling for weighted estimates of the log probability. There are two important terms:

Term 1: $\sum_i \log\left(\frac{1}{\mathrm{L}}\sum_{l=1}^{\mathrm{L}} \frac{p_z\left(\boldsymbol{z}^{(i,l)}\right)}{q\left(\boldsymbol{z}^{(i,l)}\right)} p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(\mathbf{i})}|\boldsymbol{z}^{(i,l)}\right)\right)$ with $\boldsymbol{z}^{(i,l)} \sim q\left(\mathbf{z}^{(\mathbf{i},\mathbf{l})}\right)$

Term 2: $\min_{\phi} \sum_i \mathbb{D}_{KL}\left(q_{\boldsymbol{\phi}}\left(\mathbf{z}|\mathbf{x}^{(\mathbf{i})}\right) \| p_{\boldsymbol{\theta}}\left(\mathbf{z}|\mathbf{x}^{(\mathbf{i})}\right)\right)$

We want to maximize Term 1–Term 2. Thus, we end up with

$$\mathcal{L}_{IWAE} = \mathbb{E}_{z^{(i,1)},z^{(i,2)},z^{(i,3)}\ldots,z^{(i,l)}\sim q(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{1}{\mathrm{L}}\sum_{l=1}^{\mathrm{L}} \frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}, \boldsymbol{z}^{(i,L)}\right)}{q(\boldsymbol{z}^{(i,l)}|\mathbf{x}^{(\mathbf{i})})}\right)\right] \leq \log\left(p_{\boldsymbol{\theta}}(\mathbf{x}^{(\mathbf{i})})\right) \tag{64}$$

We can do a form of the ELBO by taking L samples of $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

$$\mathcal{L}_{\mathrm{VAE}} = \mathbb{E}_{z^{(i,1)},z^{(i,2)},z^{(i,3)},\ldots,z^{(i,L)}\sim q(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{\mathrm{L}}\sum_{l=1}^{\mathrm{L}} \log\left(\frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(\mathbf{i})}, \boldsymbol{z}^{(i,l)}\right)}{q(\boldsymbol{z}^{(i,l)} \mid \mathbf{x}^{(\mathbf{i})})}\right)\right] \leq \log\left(p_{\boldsymbol{\theta}}(\mathbf{x}^{(\mathbf{i})})\right)$$

Using Jensen's Inequality, we can see that

$$\mathcal{L}_{VAE} \leq \mathcal{L}_{IWAE} \leq \log\left(p_{\boldsymbol{\theta}}(\mathbf{x}^{(\mathbf{i})})\right) \tag{65}$$

The loss for the IWAE forms a tighter bound than the VAE, and, as you increase K, the bound becomes tighter. In the IWAE lower bound, the gradient weighs the datapoint by relative importance. In the VAE lower bound, the weights are equally weighted.

#### 6.3.3. IWAE Variance Reductions

The gradient estimator of the IWAE can still have higher variance than desirable [79,83], due to a hidden score function. To eliminate this problem, you can drop the hidden score function, leading to IWAE-STL [79]. You can also use the reparametrization trick on the hidden score function [84]. This new estimator is called the doubly reparametrized gradient estimator (DReG). This leads to the IWAE-DReG.

### *6.4. Mixture-of-Experts Multimodal VAE (MMVAE)*

The Multimodal VAE (MVAE) [85] and MMVAE model [86] address generative modeling of data across multiple modalities. In this context, examples of multimodal data include images with captions and video data with accompanying audio.

We have M modalities, denoted by $m = 1, \ldots, M$ of the form

$$p_{\boldsymbol{\Theta}}(\mathbf{z}, \mathbf{x}_{1:M}) = p(\mathbf{z}) \prod_{m=1}^{M} p_{\boldsymbol{\theta}_m}(\mathbf{x}_m|\mathbf{z}) \tag{66}$$

where $p_{\theta_m}(\mathbf{x}_m|\mathbf{z})$ are likelihoods; it is parameterized by a decoder. This decoder has parameters $\mathbf{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M\}$.

The true joint posterior is denoted as $p_{\mathbf{\Theta}}(\mathbf{z}|\mathbf{x}_{1:M})$, and the variational joint posterior $q_{\mathbf{\Phi}}(\mathbf{z}|\mathbf{x_{1:M}})$,

$$q_{\mathbf{\Phi}}(\mathbf{z}|\mathbf{x_{1:M}}) = \sum_{m=1}^{M} a_m q_{\boldsymbol{\phi}_m}(\mathbf{z}|\mathbf{x_m}) \tag{67}$$

where $a_m = \frac{1}{M}$ and $q_{\phi_m}(\mathbf{z}|\mathbf{x_m})$ denotes a unimodal posterior.

We plug this into the $\mathcal{L}_{IWAE}$ to get

$$\mathcal{L}_{IWAE}^{\text{MoE}} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{\mathbf{z}^{1:K} \sim q_{\mathbf{\Phi}}(z|\mathbf{x}_{1:M})} \left[ \log \left( \sum_{k=1}^{K} \frac{1}{K} \frac{p_{\mathbf{\Theta}}\left(\mathbf{z}_m^k, \mathbf{x}_{1:M}\right)}{q_{\mathbf{\Phi}}\left(\mathbf{z}_m^k|\mathbf{x}_{1:M}\right)} \right) \right] \tag{68}$$

### *6.5. VR-α Autoencoder and VRmax Autoencoder*

We can also derive a variational lower bound for Rényi's $\alpha$-Divergence, called variational Rényi $(VR)$ bound [87]. We approximate the exact posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ for $\alpha > 0$.

$$\min_{q(z)} \mathbb{D}_{\alpha}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}\left(\mathbf{z}|\mathbf{x}^{(\mathbf{i})}\right)\right)$$

$$\max_{q \in \mathcal{Q}} \mathcal{L}_{\alpha}(q; \mathbf{x}) = \max_{q \in \mathcal{Q}} \log(p_{\boldsymbol{\theta}}(\mathbf{x})) - \mathbb{D}_{\alpha}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right)$$

when $\alpha \neq 1$, it is equivalent to

$$\mathcal{L}_{\alpha}(q; \mathbf{x}) := \frac{1}{1-\alpha} \log \mathbb{E}_q \left[ \left( \frac{p(z, \mathbf{x})}{q(\boldsymbol{z} \mid \mathbf{x})} \right)^{1-\alpha} \right] \tag{69}$$

This definition can be extended to $\alpha \leq 0$, The VR-$\alpha$ Autoencoder minimizes this VR bound. The VRmax Autoencoder is an autoencoder in the case of the Renyi Divergence where $\alpha = -\infty$.

IWAE can also be seen as the case of the Renyi Divergence when $\alpha = 0$ and L $< \infty$; L is the sample size of the Monte Carlo estimator. When $\alpha$ = 1, the VR-$\alpha$ Autoencoder becomes the vanilla VAE.

### *6.6. INFOVAE*

The INFOVAE [64], also known as MMD-VAE, is a posterior regularizing variant. This leads to better disentangled representations. However, the INFOVAE still has the blurred images generation problem.

The term $q_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$ is the posterior to $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is the posterior to $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.

The Divergence between $q$ and $p$, $\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z})\|p(\mathbf{z}))$, is multiplied by $\lambda$, a scaling parameter. A mutual information between $x$ and $z$ under $q$, denoted by $\mathbb{I}_q(x; z)$, is also added, and scaled by parameter $\alpha$ (this $\alpha$ is different from the $\alpha$ in Renyi Entropy and Divergence). This gives us the following loss function:

$$\begin{aligned} \mathcal{L}_{\text{InfoVAE}} = -\lambda \mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\mathbf{x})\|p(\mathbf{z})) - \mathbb{E}_{q(z)}\left[\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{x}|\boldsymbol{z})\|p_{\boldsymbol{\theta}}(\mathbf{x}|\boldsymbol{z}))\right] \\ + \alpha \mathbb{I}_q(\boldsymbol{x}; \boldsymbol{z}) \end{aligned} \tag{70}$$

This objective function cannot be optimized directly; an equivalent form is

$$\begin{aligned} \mathcal{L}_{\text{InfoVAE}} \equiv \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}))] - (1-\alpha)\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) - \\ (\alpha + \lambda - 1)\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z})\|p(\mathbf{z})) \end{aligned} \tag{71}$$

One typical architecture configuration for the INFOVAE involves using a DCGAN for both the encoder and the decoder.

### *6.7. β-VAE*

The $\beta$-VAE [65] is a posterior regularizing variant. We weight the regularizing term by $\beta$, so the ELBO is modified to:

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(z|x)}[\log(p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}))] - \beta \mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_z(\mathbf{z})) \tag{72}$$

$\beta$ is typically greater than 1. The correct choice of $\beta$ creates more disentangled latent representation. However, the balancing issue comes in to play; there is a trade-off between reconstruction fidelity and the disentanglement of the latent code [67]. In [67,88,89], the $\beta$-VAE's ability to disentangle has been analyzed. In [67], the authors explore disentanglement through the information bottleneck perspective and propose modifications to increase the disentanglement capabilities of the $\beta$-VAE.

### *6.8. PixelVAE*

The PixelVAE [60] is a VAE based model with a decoder based on the PixelCNN. Since the PixelCNN is a neural autoregressive model, the decoder of the PixelVAE is a neural autoregressive decoder.

The encoder and decoder both have convolutional layers. The convolutional layers use strided convolutions in the encoder for downsampling. The convolutions in the decoder and are transposed for upsampling.

Typically, a VAE decoder models each output dimension independently, so they use factorizable distributions. In the PixelVAE, a conditional PixelCNN is used in the decoder. The decoder is modeled by:

$$p(\mathbf{x}|\mathbf{z}) = \prod_i p(\mathbf{x}_i|\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{z}) \tag{73}$$

We model the distribution of $x$ as the product of the distributions of each dimension of $x$, denoted by $x_i$, conditioned by $z$ and all of previous dimensions. The variable $z$ is the latent variable.

The PixelCNN is great at capturing details but does not have a latent code. The VAE is great at learning latent representations and capturing a global structure; it is not great at capturing details. The PixelVAE has the positives of both models; it has a latent representation, and is great at capturing global structure and small details. It can also have latent codes that are more compressed than the vanilla VAE. Figure 7 shows the architecture of the PixelVAE.

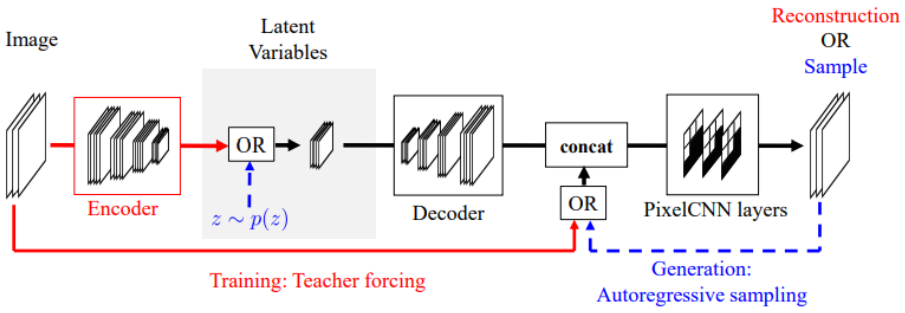



**Figure 7.** PixelVAE architecture. Image taken from [60].

The performance of VAEs can be improved by creating a hierarchy of random latent variables through stacking VAEs. This idea can also be applied to the PixelVAE.

The PixelVAE++ algorithm uses PixelCNN++ instead of PixelCNN in the decoder [90]. It also uses a discrete latent variables with a Restricted Boltzmann Machine prior.

### *6.9. HyperSpherical VAE/S-VAE*

The vanilla VAE often fails to model data whose latent structure is hyperspherical. The soap bubble effect and the gravity origin effect are also a problem with Gaussian priors in the VAE. The HyperSpherical VAE [77] attempts to deal with these problems.

The von Mises Fisher (vmF) distribution is parameterized by $\mu \in \Re^m$ and $\kappa \in R_{\geq 0}; \mu$ is the mean direction, and $\kappa$ is the concentration around the mean. The PDF of a vmF distribution for random vector $z \in \Re^m$:

$$\begin{aligned} q(\mathbf{z}; \mu, \kappa) &= \mathcal{C}_m(\kappa) \exp\left(\kappa \mu^T \mathbf{z}\right) \\ \mathcal{C}_m(\kappa) &= \frac{\kappa^{m/2-1}}{(2\pi)^{m/2} \mathcal{I}_{m/2-1}(\kappa)} \end{aligned} \tag{74}$$

$\mathcal{I}_v$ represents a modified Bessel function of the first kind at order v.

The hyperspherical VAE uses the vMF as a the variational posterior. The primary advantage of this is the ability to use a uniform distribution as the prior. The KL Divergence term $\mathbb{D}_{KL}\left(\text{vMF}(\mu, \kappa) \| U\left(S^{m-1}\right)\right)$ to be optimized is:

$$\kappa \frac{\mathcal{I}_{m/2}(k)}{\mathcal{I}_{m/2-1}(k)} + \log(\mathcal{C}_m(\kappa)) - \log\left(\frac{2\left(\pi^{m/2}\right)}{\Gamma(m/2)}\right)^{-1} \tag{75}$$

The KL term does not depend on $\mu$, this parameter is only optimized in the reconstruction term. The gradient with respect to the $\kappa$ is

$$\begin{aligned} \nabla_\kappa \mathbb{D}_{KL}\left(\text{vMF}(\mu, \kappa) \| U\left(S^{m-1}\right)\right) = \frac{1}{2} k \Big( & \frac{\mathcal{I}_{m/2+1}(k)}{\mathcal{I}_{m/2-1}(k)} \\ & - \frac{\mathcal{I}_{m/2}(k)(\mathcal{I}_{m/2-2}(k) + \mathcal{I}_{m/2}(k))}{\mathcal{I}_{m/2-1}(k)^2} + 1 \Big) \end{aligned} \tag{76}$$

The sampling procedure for the vmF can be found in [91]. The N-Transformation reparameterization trick can be used to extend the reparameterization trick to more distributions [92]; it is used to reparameterize vmF sampling.

### *6.10. δ-VAE*

$\mathbb{D}_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$ is also called the rate term. The $\delta$-VAE [71] attempts to prevent posterior inference by preventing the rate term from going to 0 by using a lower bound. They address the posterior collapse problem with structural constraints so that the KL Divergence between the posterior and prior is lower bounded by design. This can be achieved by choosing families of distributions for $p_{\boldsymbol{\theta}}(\mathbf{z})$ and $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ such that

$$\min_{\boldsymbol{\theta}, \boldsymbol{\phi}} \mathbb{D}_{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})\right) \geq \delta \tag{77}$$

The committed rate of the model is denoted by $\delta$. One way to do so is to select from a family of Gaussian distributions with variances that are fixed but different.

### *6.11. Conditional Variational Autoencoder*

The conditional VAE [93] is a type of deep conditional generative model (CGM). In a deep CGM, there are three types of variables: $\boldsymbol{x}$ denotes the input variables, $\boldsymbol{y}$ denotes the output variables, and $\boldsymbol{z}$ denotes the latent variables. The approximate posterior is is $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, \mathbf{y})$. The conditional prior is $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, and the conditional likelihood is $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}, \mathbf{z})$.

After $\mathbf{x}$ is observed, $\mathbf{z}$ is sampled from $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Then, $\mathbf{y}$ is generated from $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}, \mathbf{z})$. The variational lower bound of the deep CGM is

$$\log(p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})) \geq -\mathbb{D}_{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, \mathbf{y}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}[\log(p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}, \mathbf{z}))] \tag{78}$$

For the CVAE, where L is the number of samples, $\mathbf{z}^{(l)} = g_{\phi}\left(\mathbf{x}, \mathbf{y}, \boldsymbol{\epsilon}^{(l)}\right), \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, the lower bound estimator is

$$\widetilde{\mathcal{L}}_{\mathrm{CVAE}}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}, \boldsymbol{\phi}) = -\mathbb{D}_{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, \mathbf{y}) \| p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right) + \frac{1}{\mathrm{L}} \sum_{l=1}^{\mathrm{L}} \log(p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}, \mathbf{z}^{(l)})) \tag{79}$$

The encoder is $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, \mathbf{y})$, the conditional prior is $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, and the decoder is $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}, \mathbf{z})$.

### *6.12. VAE-GAN*

The VAE-GAN architecture [56] is influenced by both the VAE and the GAN; the decoder is also the generator, and the reconstruction loss term is replaced by a discriminator.

As shown in Figure 8, there you have the same VAE structure, but the sample coming out of the VAE is fed into a discriminator, along with the original training data.

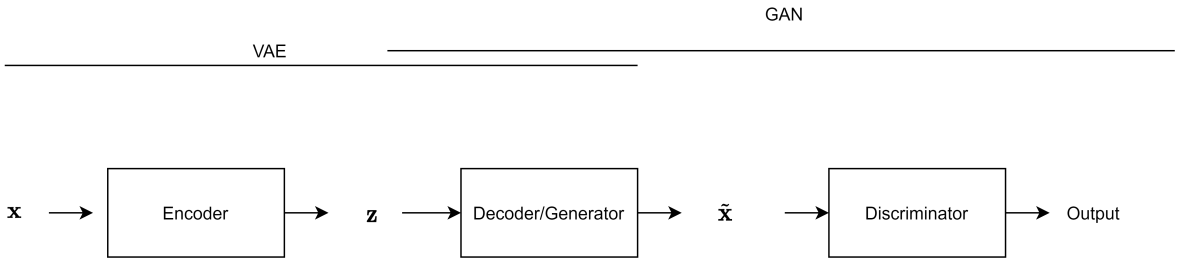



**Figure 8.** Diagram of the VAE-GAN architecture. x is the input to the encoder, which outputs latent variable z, which goes into the decoder/generator. This is fed into the discriminator. The decoder/generator is part of both the VAE and GAN.

In this model, $\boldsymbol{z}$ is the output of the encoder, denoted $z \sim \mathrm{Enc}(\mathbf{x}) = q(\mathbf{z}|\mathbf{x})$, and $\tilde{\boldsymbol{x}}$ is the output of the decoder, denoted by $\tilde{x} \sim \mathrm{Dec}(\mathbf{z}) = p(\mathbf{x}|\mathbf{z})$. $\mathrm{Dis}_l(\mathbf{x})$ denotes the representation of the $l$ th layer of the discriminator.

The likelihood of the lth layer of the discriminator can be given by

$$p(\mathrm{Dis}_l(\mathbf{x})|\mathbf{z}) = \mathcal{N}(\mathrm{Dis}_l(\mathbf{x}); \mathrm{Dis}_l(\tilde{\mathbf{x}}), \boldsymbol{I})$$

It is a Gaussian distribution parametrized with mean $\mathrm{Dis}_l(\tilde{\mathbf{x}})$ and the identity matrix $\boldsymbol{I}$ as its covariance. The likelihood loss for $\mathrm{Dis}_l(\mathbf{x})$ can be calculated as

$$\mathcal{L}_{\mathrm{llike}}^{\mathrm{Dis}\,l} = -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log(p(\mathrm{Dis}_l(\mathbf{x})|\mathbf{z}))]$$

The loss of the GAN is typically given by $\mathcal{L}_{\mathrm{GAN}} = \log(\mathrm{Dis}(\mathbf{x})) + \log(1 - \mathrm{Dis}(\mathrm{Gen}(\mathbf{z})))$. Since the generator and decoder are the same for the VAE-GAN, it can be rewritten as

$$\mathcal{L}_{\mathrm{GAN}} = \log(\mathrm{Dis}(\mathbf{x})) + \log(1 - \mathrm{Dis}(\mathrm{Dec}(\mathbf{z})))$$

The overall loss used for training the VAE-GAN is

$$\mathcal{L} = \mathbb{D}_{KL}(q(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) + \mathcal{L}_{\mathrm{llike}}^{\mathrm{Dis}\,l} + \mathcal{L}_{\mathrm{GAN}} \tag{80}$$

There are multiple practical considerations regarding training the VAE-GAN. The first consideration is to limit propagation of error signals to only certain networks. $\boldsymbol{\theta}_{\mathrm{Enc}}, \boldsymbol{\theta}_{\mathrm{Dec}}, \boldsymbol{\theta}_{\mathrm{Dis}}$ denote the parameters of each network.

The second consideration is to weigh the error signals that the decoder receives. The decoder receives these signals from both $\mathcal{L}_{\mathrm{llike}}^{\mathrm{Dis}\,l}$ and $\mathcal{L}_{\mathrm{GAN}}$,

The parameter $\eta$ is used as a weighting factor, and the update of the decoders parameters looks like:

$$\boldsymbol{\theta}_{\mathrm{Dec}} \leftarrow -\nabla_{\boldsymbol{\theta}_{\mathrm{Dec}}}\left(\eta \mathcal{L}_{\mathrm{llike}}^{\mathrm{Dis}_l} - \mathcal{L}_{\mathrm{GAN}}\right)$$

Empirically, the VAE-GAN performs better if the discriminator input includes samples from both $p(\mathbf{z})$ and $q(\mathbf{z}|\mathbf{x})$. Therefore, the GAN loss can be rewritten as:

$$\begin{aligned}\mathcal{L}_{\text{GAN}} = &\log(\text{Dis}(\mathbf{x})) + \log(1 - \text{Dis}(\text{Dec}(\mathbf{z}))) \\ &+ \log(1 - \text{Dis}(\text{Dec}(\text{Enc}(\mathbf{x}))))\end{aligned}$$

Algorithm 3 shows the VAE-GAN training algorithm given practical considerations, and Figure 9 shows the architecture given these modifications.

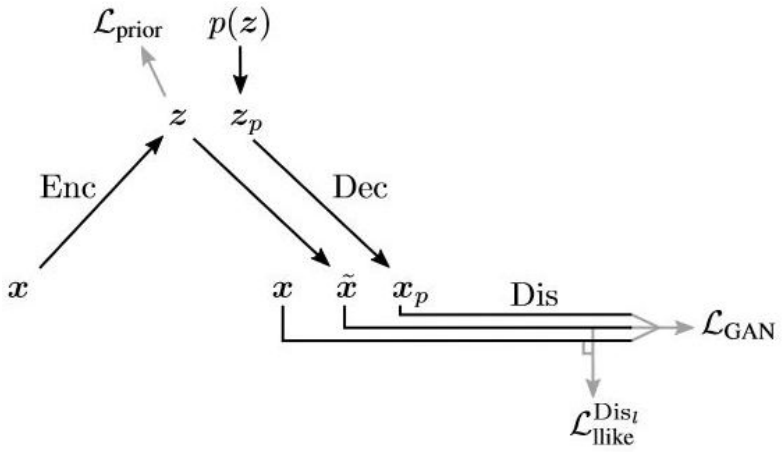



**Figure 9.** Diagram of the VAE-GAN training, taken from [56].

**Algorithm 3** VAE-GAN training

$\boldsymbol{\theta}_{\text{Enc}}, \boldsymbol{\theta}_{\text{Dec}}, \boldsymbol{\theta}_{\text{Dis}} \leftarrow$ initialize network parameters for encoder, decoder, and discriminator networks
**for do** # of training iterations:
$\mathcal{X}^{(M)} \leftarrow$ random mini-batch
$\mathcal{Z}^{(M)} \leftarrow \text{Enc}(\mathcal{X}^{(M)})$
$\mathcal{L}_{\text{prior}} \leftarrow \mathbb{D}_{KL}(q(\mathcal{Z}^{(M)}|\mathcal{X}^{(M)}) \| p(\mathcal{Z}^{(M)})$
$\tilde{\mathcal{X}}^{(M)} \leftarrow \text{Dec}(\mathcal{Z}^{(M)})$
$\mathcal{L}_{\text{Llike}}^{\text{Disl}} \leftarrow -\mathbb{E}_{q(\mathcal{Z}^{(M)}|\mathcal{X}^{(M)})}\left[p\left(\text{Dis}_l(\mathcal{X}^{(M)})|\mathcal{Z}^{(M)}\right)\right]$
$\mathcal{Z}_p^{(M)} \leftarrow$ samples from prior $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$
$\mathcal{X}_p^{(M)} \leftarrow \text{Dec}\left(\mathcal{Z}_p^{(M)}\right)$
$\mathcal{L}_{\text{GAN}} \leftarrow \log(\text{Dis}(\mathcal{X}^{(M)})) + \log(1 - \text{Dis}(\tilde{\mathcal{X}}^{(M)}))$
$+ \log\left(1 - \text{Dis}\left(\mathcal{X}_p^{(M)}\right)\right)$
Update the network parameters with their stochastic gradients:
$\boldsymbol{\theta}_{\text{Enc}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Enc}}}\left(\mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l}\right)$
$\boldsymbol{\theta}_{\text{Dec}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dec}}}\left(\eta\mathcal{L}_{\text{llike}}^{\text{Dis}_l} - \mathcal{L}_{\text{GAN}}\right)$
$\boldsymbol{\theta}_{\text{Dis}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dis}}}\mathcal{L}_{\text{GAN}}$
**end for**

Extensions of the VAE-GAN include the Zero-VAE-GAN [94], F-VAEGAN-D2 [95], 3DVAE-GAN [96], and Hierarchical Patch VAE-GAN [97].

### *6.13. Adversarial Autoencoders (AAE)*

The adversarial autoencoder is another architecture that takes inspiration from both the VAE and GAN [98]. We denote x as the input and $\mathbf{z}$ as the latent code of the autoencoder. Then, $p(\mathbf{z})$ is the prior probability distribution over the latent code, $q(\mathbf{z}|\mathbf{x})$ is the probability

distribution for the encoder, $p(\mathbf{x}|\mathbf{z})$ is the distribution for the decoder, $p_{\mathrm{d}}(\mathbf{x})$ denotes the data generating distribution, and $p(\mathbf{x})$ is the model distribution. The encoder has an aggregated posterior distribution defined as

$$q(\mathbf{z}) = \int_{\mathbf{x}} q(\mathbf{z}|\mathbf{x}) p_{\mathrm{d}}(\mathbf{x}) d\mathbf{x}$$

An adversarial network is connected to the latent code. From there, we sample from the aggregated posterior and the prior, and input both into the discriminator. The discriminator tries to distinguish whether or not the z is from the prior, which means it is real, or if it is from the aggregrated variational posterior, which is fake. This matches the prior with the aggregrated variational posterior, which has a regularizing effect on the autoencoder. The encoder can also be considered the generator of the adversarial net because it is generating the latent code. The autoencoder part of the AAE tries to minimize the reconstruction error.

The AE and the adversarial network are trained in two phases. The first phase is the reconstruction phase. This is where the autoencoder is trained on minimizing the reconstruction loss. The second phase is the regularization phase. In this phase, the discriminative network is trained to discriminate between the real samples and the fake ones. Then, the generator (the encoder of the AE) is also trained to fool the discriminator better. Both of these steps use minibatch SGD.

There is a broad choice of functions for the approximate posterior. Some common choices are a deterministic function, a Gaussian probability distribution, or a universal approximator of the posterior.

Figure 10 shows the architecture of the AAE. We can adjust the architecture of the AAE to do supervised learning, semi supervised learning, unsupervised clustering, and dimensionality reduction.

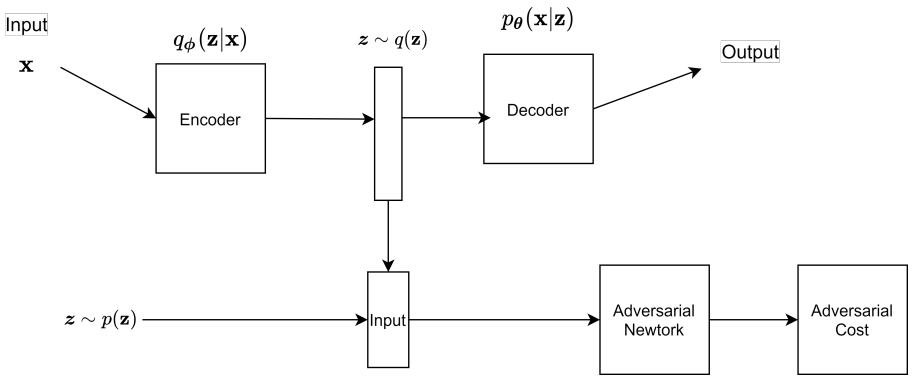



**Figure 10.** Adversarial Autoencoder architecture. The top part is the autoencoder, while the encoder and the bottom part constitute the GAN.

### *6.14. Information-Theoretic Learning Autoencoder*

The Information-Theoretic Learning Autoencoder (ITL-AE) [99] is similar to the VAE, with both and encoder and decoder layers. There are two main differences. One is that it does not use the reparameterization trick.

The second difference is that, instead of using the KL Divergence, it uses alternate Divergence measures, like the CS Divergence and the Euclidean Divergence; these Divergences are estimated through kernel density estimation (KDE) [100].

$$\text{cost} = \mathbb{L}(\mathrm{x}, \tilde{\mathrm{x}}) + \lambda R(Enc, p) \tag{81}$$

where $\mathbb{L}$ is the reconstruction loss, R is the regularization term, typically the Euclidean or CS Divergence. The chosen prior is $p$, $Enc$ is the encoder, and $\lambda$ controls the magnitude of the regularization.

If we want to estimate the QIP $\mathbb{V}$ for $p(\mathbf{x})$ using KDE, it is given by the formula

$$\hat{\mathbb{V}}_p = \frac{1}{\mathrm{N}^2} \sum_{j=1}^{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} \mathbb{G}\left(\mathbf{x}_i | \mathbf{x}_j, \sigma^2\right) \tag{82}$$

where there are N datapoints, and a Gaussian kernel $\mathbb{G}$ with kernel bandwidth $\sigma^2$. Minimizing the QIP is equivalent to maximizing the quadratic Entropy.

To get the kernel density estimator for the CS-Divergence:

$$\hat{\mathbb{D}}_{CS}(q;p) = \log\left(\frac{\hat{\mathbb{V}}_q \hat{\mathbb{V}}_p}{\hat{\mathbb{V}}_c^2}\right) \tag{83}$$

$\hat{\mathbb{V}}_q$ is the QIP estimator for PDF $q(\mathbf{x})$, $\hat{\mathbb{V}}_p$ is the QIP estimator for PDF $p(\mathbf{x})$, $\hat{\mathbb{V}}_c$ the cross information potential estimator, given by

$$\hat{\mathbb{V}}_c = \frac{1}{\mathrm{N}_q \mathrm{N}_p} \sum_{j=1}^{\mathrm{N}_p} \sum_{k=1}^{\mathrm{N}_q} \mathbb{G}\left(\mathbf{x}_{q_k} | \mathbf{x}_{p_j}, 2\sigma^2\right) \tag{84}$$

where $N_q$ is the number of observations for distribution $q(\mathbf{x})$, $N_p$ is the number of observations for distribution $p(\mathrm{x})$, $\mathbb{G}\left(\mathbf{x}_{q_k} | \mathbf{x}_{p_j}, 2\sigma^2\right)$ is a Gaussian kernel between points $\mathbf{x}_{q_k}$ and $\mathbf{x}_{p_j}$, with a kernel bandwidth $2\sigma^2$.

KDE for Euclidean Divergence estimator is given by

$$\hat{\mathbb{D}}_{ED}(q;p) = \hat{\mathbb{V}}_q + \hat{\mathbb{V}}_p - 2\hat{\mathbb{V}}_c \tag{85}$$

We can choose $q$ as the approximation distribution, and $p$ as the prior distribution. When we try to minimize the information potential with respect to $q$, the samples that are generated from $q$ would be spread out; when we try to maximize the cross information potential with respect to $q$, the samples from $q$ and $p$ are closer together. Thus, there is tension between these two effects.

The authors of [99] experimented with three different priors: Laplacian distribution, 2D Swissroll, and a Gaussian distribution, and experimented on MNIST data generation. The Euclidean Divergence did not perform as well as the CS Divergence when the data became high dimensional; high dimensionality also means the batch size has to be larger for the ITL-AE.

### *6.15. Other Important Variations*

Important architectures we have not covered include

(1) The VRNN and VRAE: The VRNN [101] is a RNN with a VAE in each layer. There is also the Variational Recurrent Auto-Encoder (VRAE) [102].
(2) VaDe and GMVAE: Both methods use Gaussian Mixture Models; the specific use case is for clustering and generation [75,76].
(3) VQ-VAE: This method combines Vector Quantization (VQ) with the variational autoencoder [74]. Both the posterior and prior are categorical distributions, and the latent code is discrete. An extension of the VQ-VAE is the VQ-VAE2 [103]. These methods are comparable to GANs in terms of image fidelity.
(4) VAE-IAF: This uses inverse autoregressive flow with the VAE [58].
(5) Wasserstein Auto-Encoder (WAE):
The WAE minimizes a form of the Wasserstein distance between the model PDF and the target PDF [104].
(6) 2-Stage VAE

The 2-Stage VAE [61] addresses multiple problems: image blurriness and the balancing issue. It also can tackle the problem of a mismatch between the aggregrate posterior and

expected prior. It trains two different VAEs sequentially. The first VAE learns how to sample from the variational posterior without matching $q(\mathrm{z}) = p(\mathrm{z})$. The second VAE attempts to sample from the true $q(\mathrm{z})$ without using $p(\mathrm{z})$.

## 7. Applications

VAEs are typically used for generating data, including images and audio; another common application is for dimensionality reduction. There are many example applications we could have chosen. However, we decided to focus on three: financial, speech source separation, and biosignal applications. The financial applications for VAEs is a new area of research with a huge potential for innovation. Source separation has long been an important problem in the signal processing community, so it is important to survey how the VAE performs in this application. Innovations in biosignal research has a great potential for positive impact for patients with disabilities and disorders; VAEs can help improve the performance of classifiers in biosignal applications through denoising and data augmentation.

### *7.1. Financial Applications*

One application is described in [105], where the $\beta$ -VAE is used to complete volatility surfaces and generate synthetic volatility surfaces for options (in the context of finance). Volatility is the standard deviation of the return on an asset. In finance, options are a contract between two parties that "*gives the holder the right to trade in the future at a previously agreed price but takes away the obligation*" [106]. This is for the simple options; there are more complex options. A volatility surface is a volatility function based on moneyness and time to maturity. For moneyness, delta is used; in the context of finance, delta is the derivative of an option price with respect to the underlying asset.

We sample N points from the volatility surface. There are two types of methods to generate volatility surfaces with the VAE: the grid-based approach and pointwise approach. For the grid-based approach, the input to the encoder is the N grid point surface; this surface is flattened into an N point vector. The encoder outputs z which has d dimensions. The decoder uses z to reconstruct the original grid points. Figures 11 and 12 show the architecture for the encoder and decoder for this approach.

For the pointwise approach, the input to the encoder is the N grid point surface, which is then flattened into an N point vector. The encoder outputs z, which is d dimensions. The input to the decoder is z along with moneyness K and maturity T. The output of the decoder is 1 point on the volatility surface. We obtain all the points using batch inference. Figures 13 and 14 show the architecture for the encoder and decoder for this approach.

In the experiments, each volatility surface was a 40 points grid, with eight times to maturity and five deltas. Six currency pairs were used in this experiment. Only the pointwise approach was used. For completing surfaces, it was compared with the Heston Model; this algorithm predicts the surface faster than the Heston Model. In some cases, it outperforms the Heston Model. Table 1 shows the results from the paper comparing the Heston Model with a Variational Autoencoder.

The experiments also generated new volatility surfaces. One main use of generating these surfaces is for data augmentation to create more observations for deep learning models (Maxime Bergeron, Private Communications).

**Table 1.** Results from [105] comparing the Heston Model with the VAE for six currency pairs. The units are in Mean Absolute Error (MAE), and the VAE has a latent code size of 4.

| Currency Pair | Heston Model | VAE Model |
|---|---|---|
| AUD/USD | 56.6 | 33.6 |
| USD/CAD | 35.3 | 32.5 |
| EUR/USD | 32.2 | 30.9 |
| GBP/USD | 47.5 | 34.0 |
| USD/JPY | 58.4 | 38.2 |
| USD/MXN | 92.2 | 56.7 |

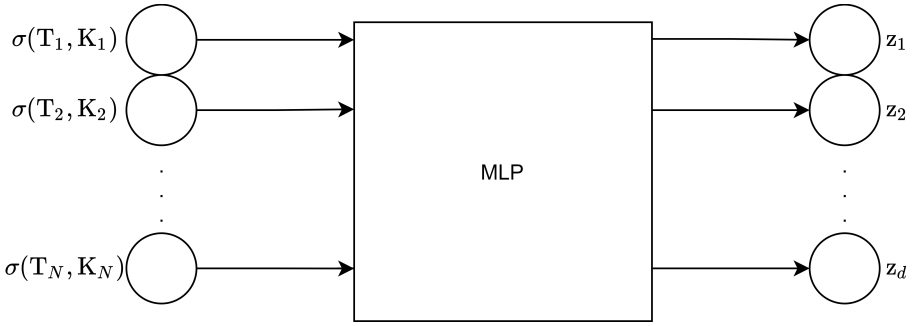



**Figure 11.** Encoder for grid-based approach.

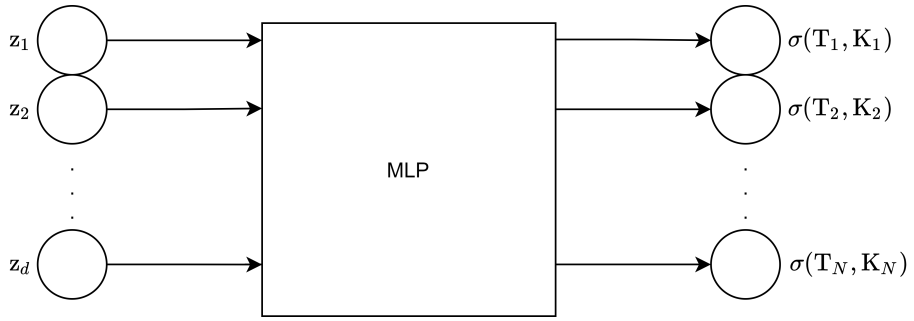



**Figure 12.** Decoder for grid-based approach.

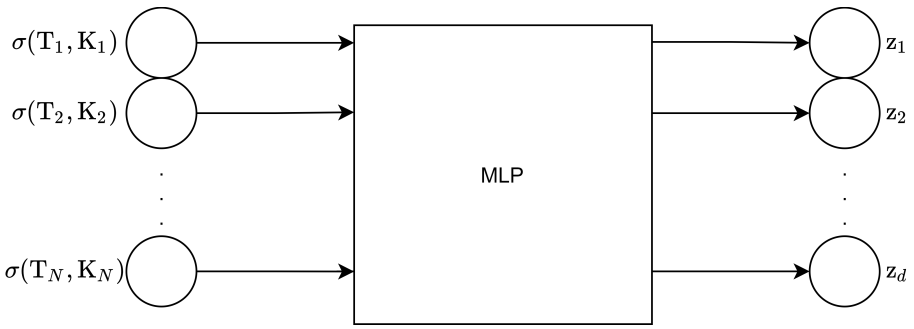



**Figure 13.** Encoder for pointwise approach.

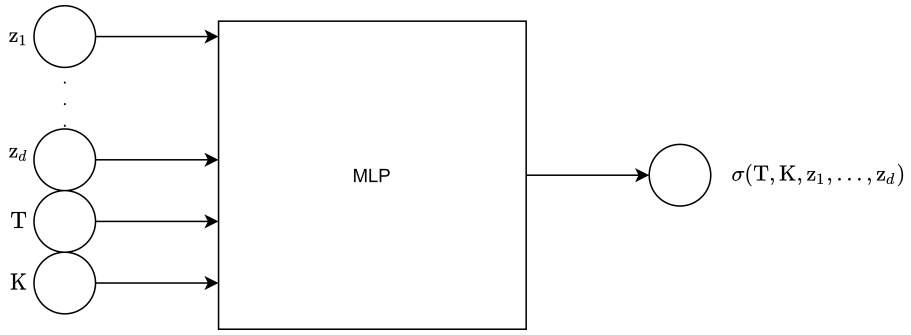



**Figure 14.** Decoder for pointwise approach.

In [107], the $\beta$-VAE was used in conjunction with continuous time stochastic differential equations models to generate arbitrage-free implied volatility (IV) surfaces. SDE models that were tested included Lévy additive processes and time-varying regime switching models.

The method, shown in the chart in Figure 15, has the following steps:

(1) Use historical market data to fit the arbitrage-free SDE model, get collection SDE model parameters.
(2) Train VAE model using on the SDE model parameters.
(3) Sample from the latent space of the VAE model using a KDE approach.
(4) Get a collection of the SDE model parameters by decoding the samples.
(5) Use the SDE model with parameters to get arbitrage-free surfaces.

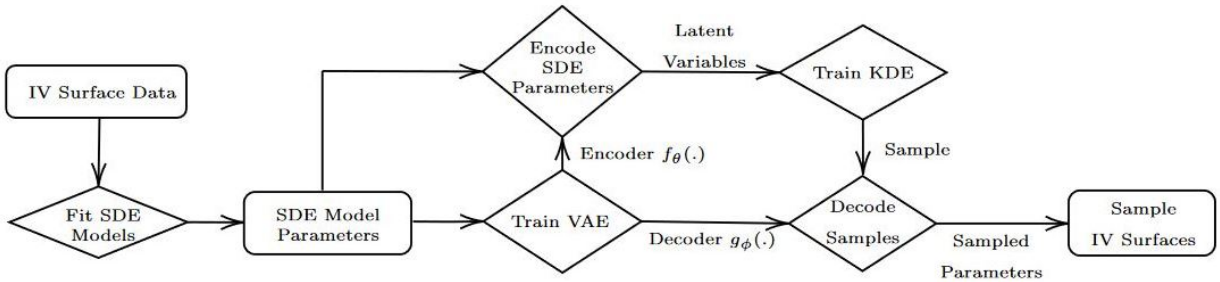



**Figure 15.** Hybrid model diagram, taken from [107].

In [108], LSTM and LightGBM were used to predict the hourly directions of eight banking stocks listed in the BIST 30 Index from 2011 and 2015. The first three years were used in the training set, and the last 2 years were used in the test set. The first experiment used the stock features as the input to the models. The second experiment first put the stock features through a VAE for dimensionality reduction before inputting it into the models. The results found that they performed similarly, though the VAE filtered input uses 16.67% less features. The 3rd experiment involved adding features from other stocks into the first and second experiments, to account for effects from other stocks.

In [109], a deep learning framework was used for multi-step-ahead prediction of the stock closing price. The input features were market open price, market high price, market low price, market close price, and market volume price. This framework used the LSTM-VAE to remove noise, then combined these reconstructed features with original features; these were the input to a stacked LSTM Autoencoder, which outputted a prediction.

In [110], they looked at the index tracking performance of various autoencoder models, including the sparse AE, contractive AE, stacked AE, DAE, and VAE. These were used to find the relationships between stocks and construct tracking portfolios. The results were then compared to conventional methods, and results showed that, in order for the deep learning methods to perform better, there needed to be a higher number of stocks in the tracking portfolio.

### *7.2. Speech Source Separation Applications*

Deep learning has been applied to various aspects of speech processing, speech recognition, speech and speaker identification and such applications [111]. In this subsection, we focus on speech source separation applications using variational autoencoders.

If you have N signals, $s_i(t)$, you can have mixed signal $y(t) = \sum_{i=1}^{N} s_i(t)$; the goal of signal/source separation is to retrieve an estimate of each $s_i(t)$, $\hat{s}_i(t)$. When it is unsupervised, it is known as blind source separation. Figure 16 shows two speech signals, Signal 1 and Signal 2 mixed to create a mixed signal.

Signal to Distortion Ratio (SDR), Signal to Artifact Ratio (SAR), Signal to Interference Ratio (SIR), Signal to Noise Ratio (SNR), and Perceptual Evaluation of Speech Quality (PESQ) are common measures to evaluate speech source separation [112,113]. SDR, SAR, SIR, and SNR are all typically measured in decibels (dB). We will drop the $i$ subscript of a signal estimate $\hat{s}_i(t)$ for simplicity in the following formulas. Using the BSSEval toolbox, we can decompose our estimate of a signal as

$$\hat{s}(t) = s_{\text{target}}(t) + e_{\text{interf}}(t) + e_{\text{noise}}(t) + e_{\text{artif}}(t)$$

$s_{\text{target}}(t)$ is a deformed version of $s_i(t)$, $e_{\text{artif}}(t)$ is an artifact term, $e_{\text{interf}}(t)$ denotes the deformation of the signals due to interference from the unwanted signals, $e_{\text{noise}}(t)$ is a deformation of the perturbating noise.

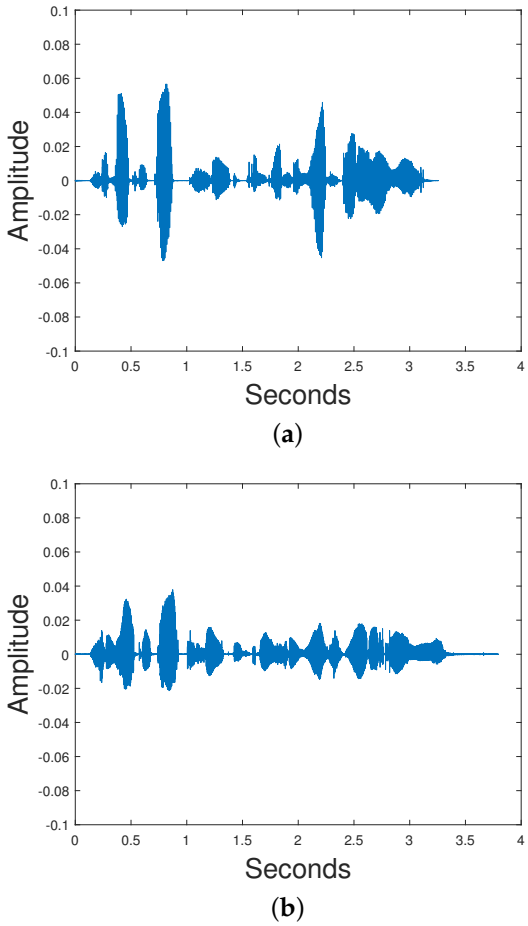
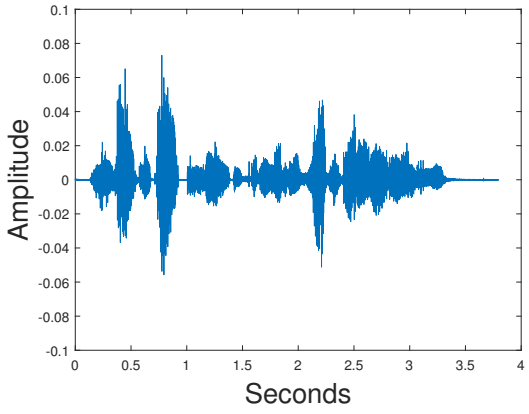



**Figure 16.** *Cont.*




(**c**)

**Figure 16.** Signal 1 and Signal 2 are mixed together to create the mixed signal. (**a**) Signal 1, (**b**) Signal 2, (**c**) Mixed Signal.

The SDR is then given by:

$$\text{SDR} = 10\log_{10}\frac{\|s_{\text{dist}}\|^2}{\|e_{\text{interf}} + e_{\text{noise}} + e_{\text{artif}}\|^2} \tag{86}$$

The SIR is given by:

$$\text{SIR} = 10\log_{10}\frac{\|s_{\text{dist}}\|^2}{\|e_{\text{interf}}\|^2} \tag{87}$$

the SNR is given by:

$$\text{SNR} = 10\log_{10}\frac{\|s_{\text{dist}} + e_{\text{interf}}\|^2}{\|e_{\text{noise}}\|^2} \tag{88}$$

and the SAR is:

$$\text{SAR} = 10\log_{10}\frac{\|s_{\text{dist}} + e_{\text{interf}} + e_{\text{noise}}\|^2}{\|e_{\text{artif}}\|^2} \tag{89}$$

Spectrograms can be used to view a signal's time-frequency representation. The amplitude and frequency information are represented by color intensity indicating the amplitude of the frequency. A common way of generating spectrograms is to take the Short Time Fourier Transform (STFT) of the signal. Two important parameters for the STFT are the window size and overlap size. The spectrogram is found by taking the square of the magnitude of the STFT especially for deep learning algorithms involving speech. Typically, in practice, the spectrogram is often normalized before being fed into a neural network [111]. Alternative inputs include log spectrograms and mel spectrograms.

In [114], the VAE was compared with NNMF, GAN, Gaussian WGAN, including Autoencoding WGANs (AE WGAN), and Autoencoders trained with maximum likelihood (MLAE) in the task of blind monoaural source separation. This VAE did not use convolutional layers. The TIMIT data set was used [115]. Each training set had a female and male speech signal mixed together. There is a VAE for each speaker; so, for a mixed signal that mixes a female and male speaker, we need two VAEs. The input to the VAE is a normalized spectrogram. The training label is the ground truth spectrogram of the original signal that we are trying to obtain. The signal is reconstructed via the Wiener filtering equation:

$$\hat{x}_k(t) = \text{iSTFT}\left(\frac{\hat{S}_k}{\hat{S}_m + \hat{S}_f} \odot S_{\text{mixed}} \odot e^{\text{i}(\text{ phase })}\right), \quad \text{k} \in \{\text{m}, \text{f}\} \tag{90}$$

$S_{\text{mixed}}$ is the magnitude spectra of the mixed signal, and phase is the phase of the mixture signal. $S_m$ and $S_f$ are the reconstructed estimate of male magnitude spectra and

reconstructed estimate of female magnitude spectra. $\hat{x}_k(t)$ is the reconstructed signal. iSTFT denotes the inverse STFT. $\odot$ denotes element wise multiplication. The experiments show that NNMF and the VAE methods result in similar SDR, SIR and SAR. The Gaussian WGAN has a better SIR and SDR than the NNMF and the VAE. The MLAE has a superior SAR to all the other models. Figure 17 shows the results for the experiments.

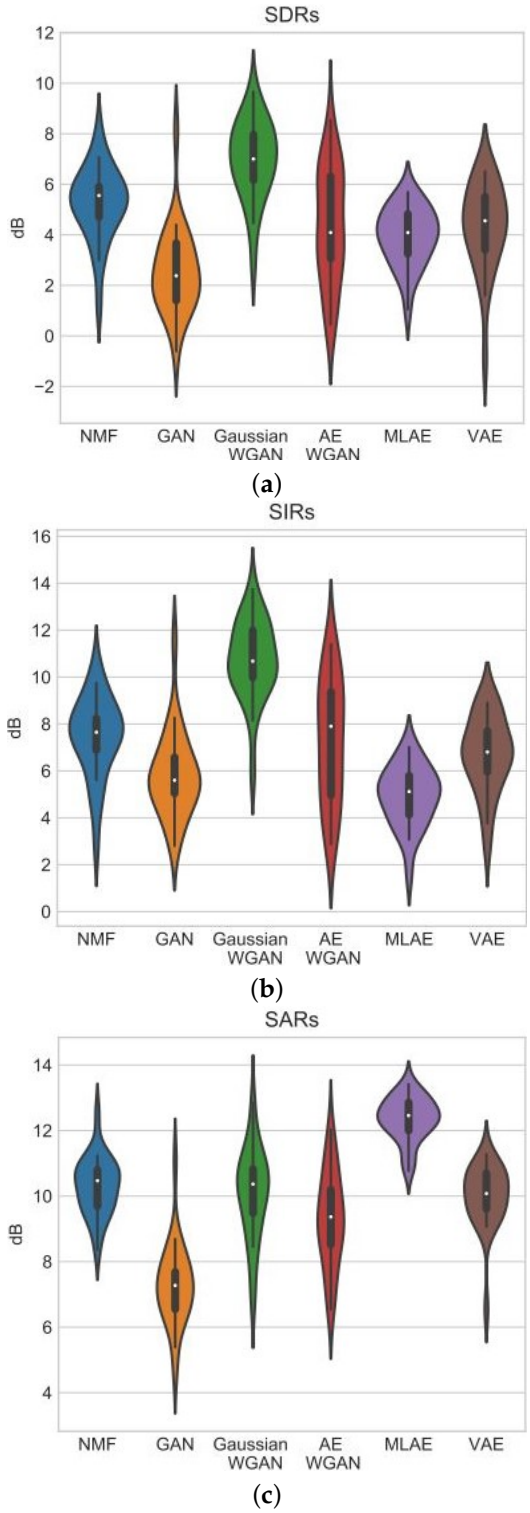



**Figure 17.** The results for the experiments from paper [114]; the graphs are taken directly from this paper. (**a**) These are the results for the SAR. (**b**) These are the results for the SDR. (**c**) These are the results for the SIR.

In [116], the researchers used two data sets, TIMIT and VIVOS. TIMIT is a speech corpus for American English; it has 8 dialects and 630 speakers. Each speaker speaks 10 sentences. In the experiment, they used all eight dialects. Background noise was also used, particularly trumpet sounds, traffic sounds and water sounds.

The architecture of the algorithm involved taking the STFT of the mixed signal, feeding it into a complex data based VAE, using a Chebyshev bandpass filter on the output, followed by an iSTFT to get the final reconstructed signal (Figure 18). The metrics used were SIR and SDR. There are two ways to implement a VAE for this STFT based method, to account for the complex input. One is to assume the real and imaginary part of the input are independent, there will be a VAE for the real and imaginary part respectively, and the output of the two VAEs will be rejoined again before being put into an inverse STFT (Hao Dao, Private Communications).

There were four cases: one dialect vs. one background, many dialects vs. one background, one dialect vs. many backgrounds, and many dialects vs. many backgrounds. In one dialect vs. one background, one dialect was chosen at a time, with 10 people randomly chosen, with 100 utterances total; for each dialect, the utterances were mixed with trumpet sounds and Gaussian noise. The results were shown to be good but not stable. In many dialects vs. one background, 100 utterances were chosen and mixed with trumpet sounds and Gaussian noise. The results were shown to be better than the previous case, possibly due to a different data distribution. In one dialect vs. many backgrounds, the speech data were mixed in four ways: with each background sound or all three of them. The difference in performance between the background sound was not huge, though there was a reduction when all three mixed with the speech signal. Experiments were also run with VIVOS, a speech corpus for Vietnamese language; the performance was slightly lower. They also indicate that performance depends on depth size and code size. They also compare with the ICA, regular VAE, filter banks, and wavelets, and find that their model has better SDR, SIR, and PESQ. The exact results are shown in Table 2.

**Table 2.** Results from [116], comparing the paper's approach with other methods.

| Group | Model | SDR (dB) | SIR (dB) | PESQ |
|---|---|---|---|---|
| 1 | Wavelet | 7.56 | 16.22 | - |
| | Time-Frequency filter bank | 9.47 | 1.09 | - |
| 2 | ICA | 5.98 | 11.92 | - |
| | VAE | 9.47 | - | 2.37 |
| 3 | VAE | 9.47 | - | 2.35 |
| Paper Approach | VAE | 9.08 | 14.76 | 2.02 |
| | BPF | 6.87 | 10.01 | 0.97 |
| | VAE + BPF | 12.99 | 15.02 | 2.41 |

In [117], source separation is achieved through class information instead of the source signal themselves using the $\beta$-VAE; this $\beta$-VAE had convolutional layers.

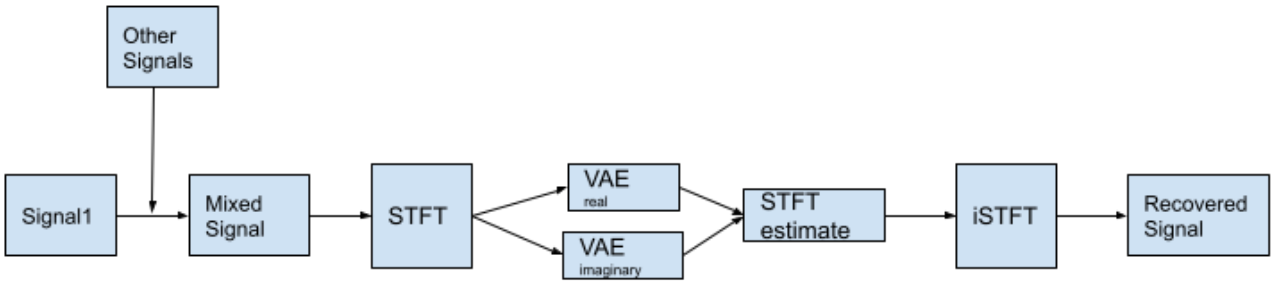



**Figure 18.** This is the architecture used by [116].

In [118], Convolutional Denoising Autoencoders (CDAEs) were used for monoaural source separation. Given the trained CDAEs, the magnitude spectrogram of the mixed signal is passed through all the trained CDAEs. The output of the CDAE of source $i$ is the estimate $\tilde{\mathbf{S}}_i$ of the spectrogram of source $i$. The CDAE performs better than the MLP at separating drums but is similar in separating the other components.

In [119], the multichannel conditional VAE (MCVAE) method was developed and used for semi-blind source separation with the Voice Conversion Challenge (VCC) 2018 data set. This technique has also been used with supervised determined source separation [120].

The generalized MCVAE is used to deal with multichannel audio source separation that has underdetermined conditions [121,122]. While the MVAE has good performance in source separation, the computational complexity is high, and it does not have a high source classification accuracy. The Fast MCVAE has been developed to deal with these issues [123]. The MCVAE does not perform as well under reverberant conditions, and ref. [124] works on extending the MCVAE to deal with this problem.

In [125], variational RNNs (VRNNs) were used for speech separation on the TIMIT dataset, achieving superior results over the RNN, NNMF, and DNN for SDR, SIR, and SAR.

Autoencoders and VAEs are also used for speech enhancement [126–130]. The goal of speech enhancement is to increase the quality of a speech signal, often by removing noise.

### *7.3. BioSignal Applications*

VAEs can also be applied to biosignals, such as electrocardiogram (ECG) signals, electroencephalography (EEG) signals, and electromyography (EMG) signals.

#### 7.3.1. ECG Related Applications

ECG machines measure the electrical signals from the heart; the signal recorded in an ECG machine is known as an ECG signal. The typical ECG has 12 leads; six on the arms/legs are called limb leads, and the six on the torso are called precordial leads. ECG waves can be defined as a "*positive or negative deflection from baseline that indicates a specific electrical event*" [131]. The common ECG waves are the P wave, Q wave, R wave, S wave, T wave, and U wave. A typical ECG waveform is shown in Figure 19. The frequencies of an ECG signal are in the 0–300 Hz range, though most of the information is available in the 0.5–150 Hz range [132].

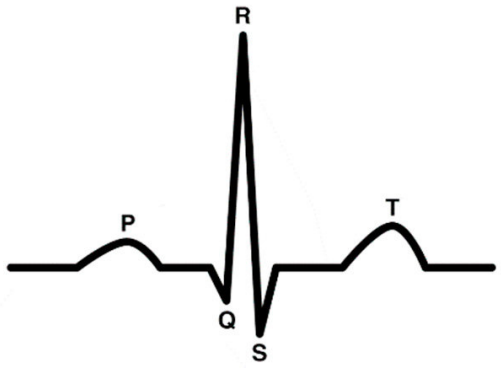



**Figure 19.** ECG waveform with P, Q, R, S, and T waves shown, taken from [133].

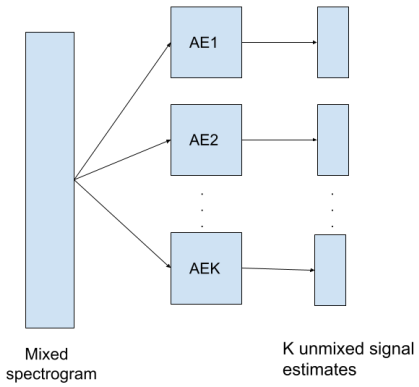
Using ECGs, doctors can detect serious diseases by identifying distortions in the signal. One very important application is measuring the ECG signal of a fetus when a woman is pregnant; this is to help detect any heart problem that the fetus has. There are invasive and noninvasive methods of measuring this; due to side effects of the invasive methods, the noninvasive method is preferred. However, the mother's ECG (MECG) signal is mixed with the baby's ECG (FECG) signal, along with external respiratory noise. Thus, the two signals need to be separated, which is the blind source separation problem. The traditional methods to deal with fetal source separation have been ICA methods and adaptive filters

such as LMS and RLS. For GANS, VAEs and AEs, it is more difficult to train the models due to the fact that we do not have a ground truth for the FECG signals. This problem can be solved by generating synthetic FECG and MECG signals from libraries such as signalz [134] and FECGSYN toolbox [135]. The average range for the beats per minute (BPM) of a pregnant woman is 80–90. The average range for the BPM of a fetus is 120–160. The MECG signal amplitude is also 2–10 times larger than the amplitude of a FECG signal. These are the important parameters needed to generate the synthetic ECG signals necessary.

While the VAE itself has not been used for fetal source separation, a similar method, the cross adversarial source separation framework (CASS), has been used. CASS is a method mixing AE and GANs for source separation tasks [136]. For each mixture component, there is an autoencoder and discriminator. The mixed signal goes into each autoencoder, whose output goes into a discriminator. Typically, each AE and GAN pair is trained independently. For each pair, the $i$th signal is separated, and the rest of the components in the mixture are treated as noise. Cross adversarial training is used to share information between each of those components. This was done by letting the $i$th discriminator to reject samples from the other components. Figure 20 shows the architectures. In their paper, the authors used two components in CASS for this particular problem. They generated synthetic FECG and MECG signals, mixed them, and added noise to simulate periodic respiratory noise. This noise consisted of random sinusoidal signals with varying frequencies and varying amplitudes. The synthetic data were converted into spectrograms before being inputted to the networks. The results are shown in Table 3. The CASS is superior to the AE model, but the CASS with cross adversarial training is inferior to CASS with training each component independently for the MECG.

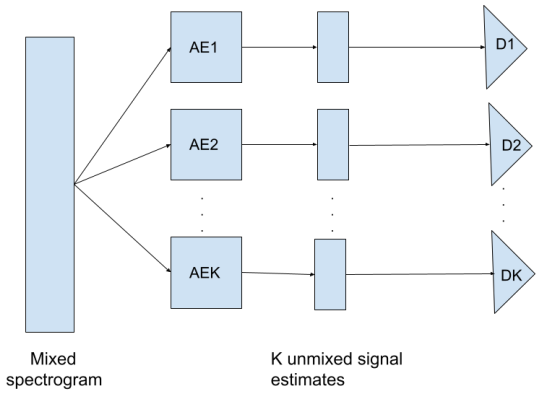
**Table 3.** Results from [136]. $L_p$ norm errors of different frameworks after training. The three frameworks were the baseline AE design, CASS (training each component independently), and CASS with cross adversarial training.

| **Maternal Method** | $L_1$ **Error** | $L_2$ **Error** | $L_\infty$ **Error** |
|---|---|---|---|
| Baseline AE | 0.45158 | 0.53502 | 0.85077 |
| CASS | 0.40672 | 0.47942 | 0.77370 |
| CASS with Cross Training | 0.40994 | 0.48082 | 0.77911 |
| **Fetal Method** | $L_1$ **Error** | $L_2$ **Error** | $L_\infty$ **Error** |
| Baseline AE | 0.49818 | 0.57435 | 0.80708 |
| CASS | 0.37387 | 0.46627 | 0.75402 |
| CASS with Cross Training | 0.37218 | 0.45848 | 0.74462 |

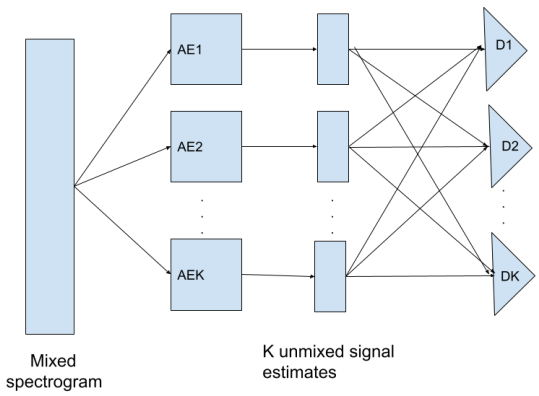



(**a**)

**Figure 20.** *Cont.*




(**b**)




(**c**)

**Figure 20.** The architectures for the experiments for each model from [136]. (**a**) This is source separation using K autoencoders. (**b**) This is the CASS architecture, where each component is trained independently. (**c**) This is the CASS architecture using cross adversarial training.

Detecting distortions in ECG signals are difficult to find due to noise from disturbances, such baseline wandering, muscle shaking, and electrode movement. The VAE has been used to distinguish these ECG signals under noise conditions [137]. They used three data sets: AHA ECG database, the APNEA ECG database, and CHFDB ECG database. From these datasets, they obtained 30,000 ECG signals.

To evaluate how well the model denoised the ECG signals, noise was added to the ECG signal data. This included AWGN, salt and pepper noise, and Poisson noise (also known as shot noise). Sinusoidal signals with different amplitudes were also added to the signal to imitate baseline wandering noise. First, the ECG signal is preprocessed; this is done by using an algorithm to split the waves in segments according to the cardiac cycle. After these steps are completed, the new data are inputted into a VAE. The results showed that the VAE is as robust in the noise scenarios presented.

Morphological diagnosis, which is "*A diagnosis based on predominant lesion(s) in the tissue(s)*" [138], is one use of ECGs. Human experts typically perform better at ECG morphological recognition than deep learning methods, due to the fact that there are an insufficient amount of positive samples. In [139], a pipeline is used that involves the VQ-VAE to generate new positive samples for data augmentation purposes. A classifier was then trained using this additional synthetic data to identify ten ECG morphological abnormalities, which resulted in an increase in the F1 score for the classifier. These ten abnormalities are myocardial infarction (MI), left bundle branch block (LBBB), right bundle branch block (RBBB), left anterior fascicular block (LAFB), left atrial enlargement (LAE), right atrial

enlargement (RAE), left ventricular hypertrophy (LVH), right ventricular hypertrophy (RVH), I ° atrial ventricular block (IAVB), and pre-excitation syndrome (WPW).

Myocardial infarctions are also known as heart attacks, which can often lead to death. They need to be rapidly diagnosed to prevent deaths. Conventional methods are not very reliable and also perform poorly when applied to 6 lead ECG. In [140], a deep learning algorithm was used to detect myocardial infarction using a 6 lead ECG. They found that using a deep learning algorithm with VAE for 6-lead ECG performed better than the traditional rule-based interpretation. 425,066 ECGs from 292,152 patients were used in the study.

Deep learning can be used to classify the type of beat in an ECG signal. However, due to the black box nature of deep learning algorithms and their complexity, they are not easily adopted into clinical practice. Autoencoders have been used to reduce the complexity; the neural networks models would then use a lower dimensional embedding for the data. This solution still has problems with interpretability due to interactions between components of the embeddings. The $\beta$-VAE can be used to disentangle these interactions between components, leading to an interpretable and explainable beat embedding; this was done in [141]. They used the $\beta$-VAE to create interpretable embedding with the MIT-BIH Arrhythmia dataset. VAEs can also be used to generating an ECG signal for one cardiac cycle [142]. This is useful for data augmentation purposes. This method is relatively simple but cannot generate whole ECG signals.

In electrocardiographic imaging, recreating the heart's electrical activity runs into numerical difficulties when using body surface potentials. A method using generative neural nets based on CVAEs have been used to tackle this problem [143].

#### 7.3.2. EEG Related Applications

EEG machines measure the electrical signals from the scalp; these are used to measure problems in the brain. A RNN based VAE has been used to improve the EEG based speech recognition systems by generating new features from raw EEG data [144]. VAEs have been used to find the latent factors for emotion recognition in multi channel EEG data [145]. These latent factors are then used as input for a sequence based model to predict emotion. Similarly, the bi-lateral variational domain adversarial neural network (BiVDANN), which uses a VAE as part of its architecture, has been used for emotion recognition from EEG signals [146]. Video game to assess cognitive abilities have been developed, with a task performance metric and EEG signals recorded; from this, a deep learning algorithm for detecting task performance from the EEG data has been developed [147]. First, this involves feature extraction, then dimensionality reduction by the VAE. The output of the VAE are used as input to an MLP, which predicts task performance.

#### 7.3.3. EMG Related Applications

An EMG machine "*measures muscle response or electrical activity in response to a nerve's stimulation of the muscle*" [148]. They can be used to find neuromuscular problems. Upper limb prosthetics use myoelectric controllers; however, these controllers are susceptible to interface noise, which reduces performance [149]. Latent representations of muscle activation patterns have been found using supervised denoising VAE. These representations were robust to noise in single EMG channels. Latent space based deep learning classifiers have outperformed conventional LDA based classifiers.

Brain-Machine Interfaces (BMIs) can be used in paralyzed patients to help return their voluntary movement; they can do this by extracting information from neural signals. However, the interface has performance issues with this method. Using latent representations through methods like the VAE has improved the performance of the BMI [150].

## 8. Experiments

### *8.1. Experiment Setup and Data*

We focused specifically on speech source separation experiments. We used the TIMIT dataset for the input [115]. Two types of speakers, male and female, were used; each of the recordings was 2–4 s. We normalized the speech signals, then combined male signals with female signals to create 90 mixed signals. Eighty signals were used for training, 10 were used for testing. We used three models: the VAE, ITL-AE, and $\beta$-VAE. For all three models, we used fully connected layers.

First, we will go over the VAE setup. There is a VAE for each speaker; thus, for a mixed signal that mixes a female and male speaker, we need two VAEs. The input to the VAE is a normalized spectrogram. The training label is the ground truth spectrogram of the original signal that we are trying to obtain. The signal is reconstructed via the Wiener filtering equation, from Equation (90). The metrics used SDR, SIR, and SAR, evaluated using the BSS Eval Toolbox [112]. This setup is repeated for the $\beta$-VAE and ITL-AE.

### *8.2. Results*

#### 8.2.1. Hyperparameter Tuning

Normalizing the spectrogram was necessary for the loss function to not explode. We also found that the Gaussian distribution that generates $\epsilon$ had to have a standard deviation of 0.01. Parameters that we varied include window and overlap size for the STFT, latent variable code size, and batch size.

Figure 21 shows the results of varying the window size and overlap size with violin plots. The encoding layers had the value [M, 256, 256, 128], and the decoding layers had the value [256, 256, M], with M being the number of spectral frames, which depends on the choice of the STFT parameters. Using a 64 ms window size fared worse than a 32 ms window and a 16 ms window. Using 64 ms window with 32 ms overlap size resulted in a better SAR than using 64 ms window with 16 ms overlap size. Using a 16 ms window size with 8 ms overlap size had the best overall results by a wide margin. Using a 16 ms window size with 4 ms overlap size had worse results than using a 16 ms window size with 8 ms overlap size. When the window size went down to 8 ms, the results became worse than 16 ms window size.

We varied the latent code size d. The encoder layers are [129-128-d], and the decoding layers are [128-129], with the window size being 256 ms and overlap size is 128 ms. The experiment found no clear difference in SIR, SDR, and SAR. We also experiment between the batch sizes in 1, 17, 34, 70. For batch sizes 17, 34, and 70, there was no clear difference between varying the batch size. Using batch size 1 increased the time while not changing the performance.

For the $\beta$-VAE, $\beta$ was a hyperparameter to be tuned. For $\beta$ >1, there is no discernible difference between the various $\beta$ 's.

For the ITL-AE, the hyperparameter that we varied was the latent code size. We varied the latent code size in the range [32, 64, 128, 256]. There was no discernible difference between the various latent codes.

#### 8.2.2. Final Results

In Figure 22, we compare the results from the three different models using violin plots. The VAE and $\beta$-VAE have similar results. The ITL-AE has a worse SAR, similar SDR, and a better SIR.

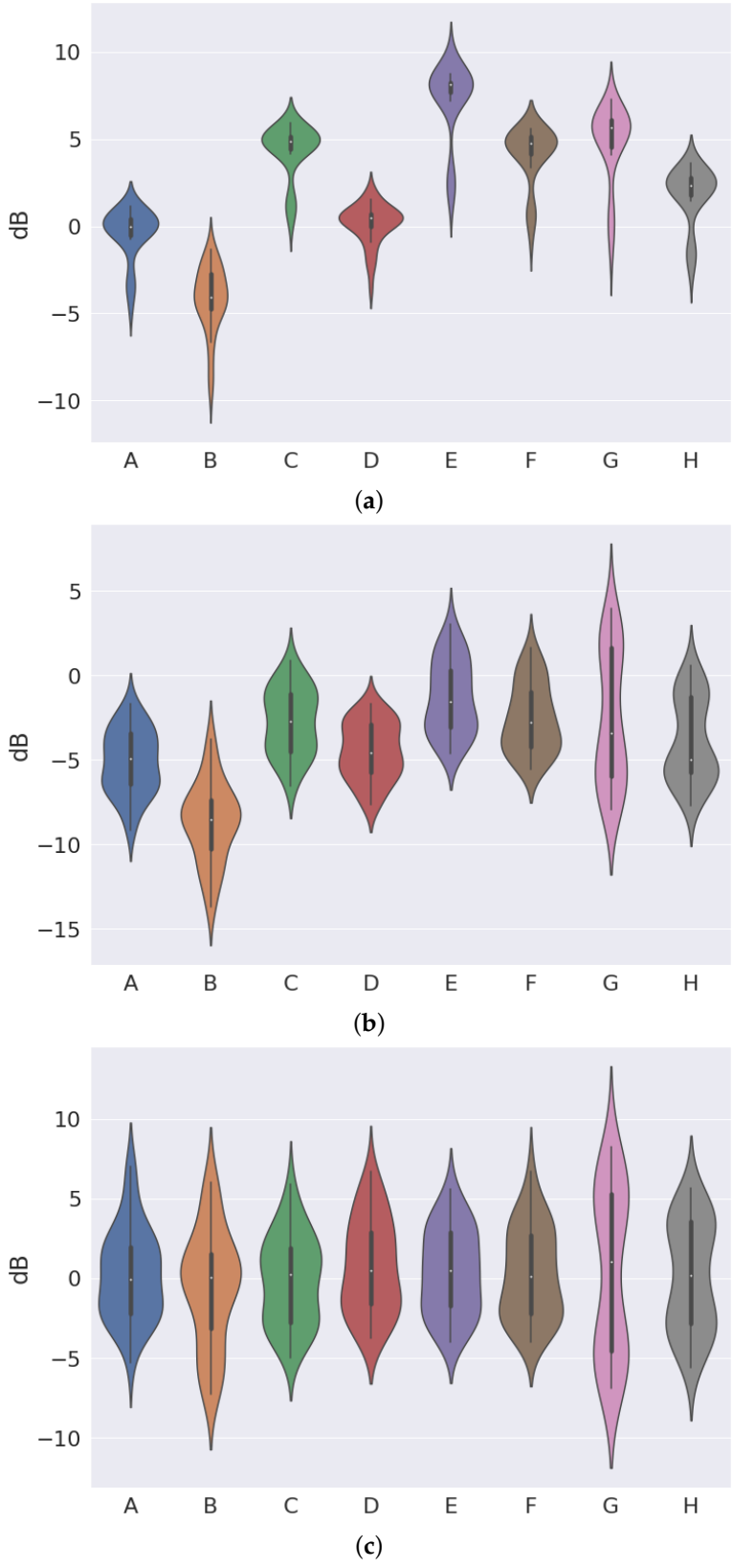



**Figure 21.** These are the results of varying the resolution and window size, in dB. For (window size, overlap size) pairs, A is (64 ms, 16 ms), B is (64 ms, 32 ms), C is (32 ms, 8 ms), D is (32 ms, 16 ms), E is (16 ms, 8 ms), F is (16 ms, 4 ms), G is (8 ms, 4 ms) and H is (8 ms, 2 ms) (**a**) These are the results for the SAR. (**b**) These are the results for the SDR. (**c**) These are the results for the SIR.

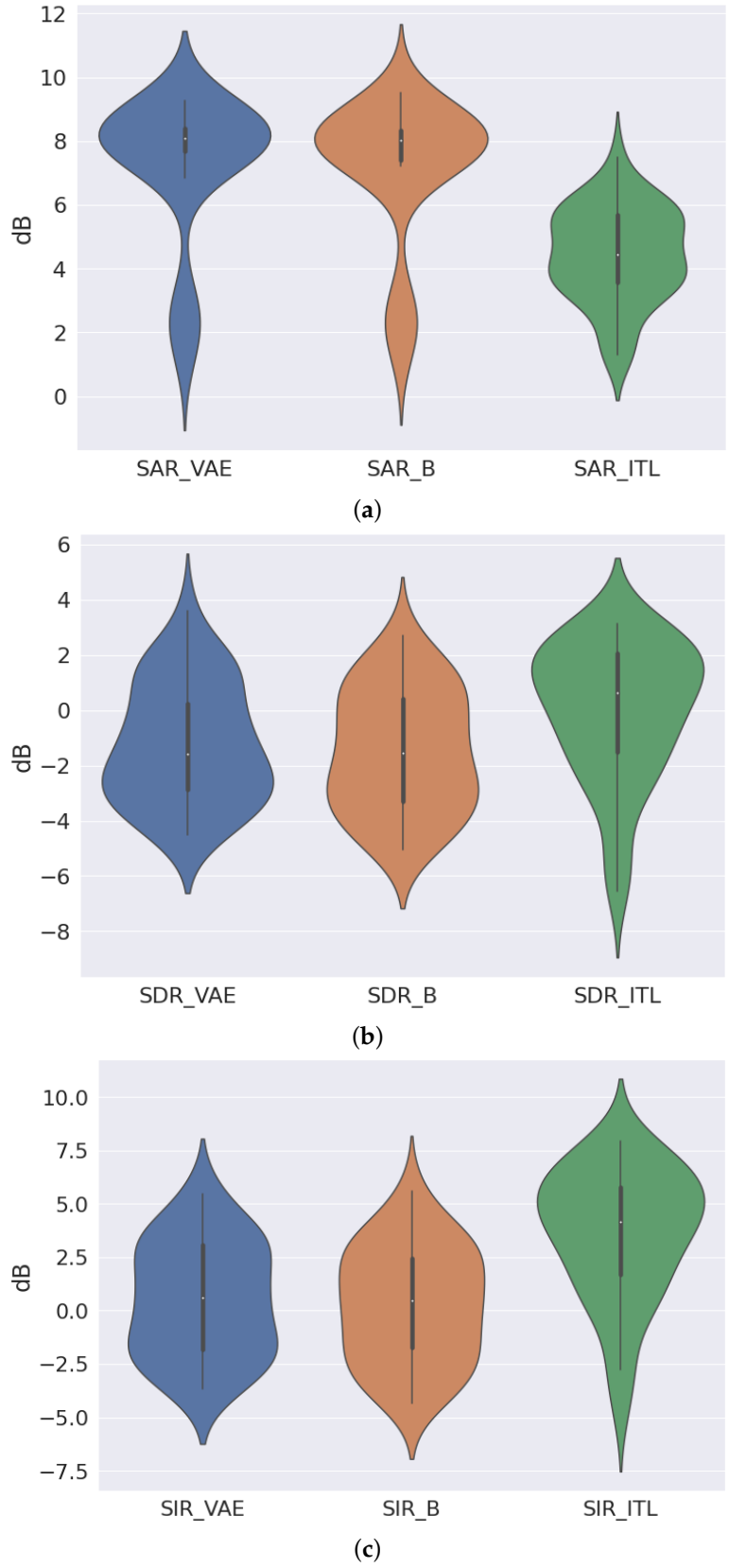



**Figure 22.** The results for the experiments for each model are shown here, in dB. (**a**) These are the results for the SAR. (**b**) These are the results for the SDR. (**c**) These are the results for the SIR.

## 9. Conclusions

In this paper, first, we provided a detailed tutorial on the original variational autoencoder model. After that, we outlined the major problems of the vanilla VAE and the latest research on resolving these issues. These problems include posterior collapse, variance loss/image blurriness, disentanglement, the soap bubble effect, and the balancing issue.

Then, we comprehensively surveyed many important variations of the VAE. We can organize most of the variations into the following four groups:

(1) Regularizing Posterior Variant:We can regularize the posterior distribution to improve the disentanglement, like with the $\beta$-VAE and INFOVAE.
(2) Prior/Posterior variants: We can change the prior or posterior, like with the hyperspherical VAE, $\rho$-VAE, and the VQ-VAE. The hyperspherical VAE uses a vmF for the posterior and a uniform for the prior, which makes it superior for a hyperspherical latent space. The $\rho$-VAE uses an AR(1) Gaussian process as its posterior, which leads to superior results over a vanilla VAE in terms of the loss function.
(3) Architectural Changes: There are many potential architectural changes. The VAE-GAN and Adversarial Autoencoder take inspiration from both the VAE and GAN, and as a result mitigate the downsides of both frameworks. Combining the VAE framework with neural autoregressive models has created more flexible inference networks; the PixelVAE combines the PixelCNN with the VAE, allowing it to capture both small details and global structure. The conditional VAE uses a conditional likelihood instead of likelihood; it has been key to the MCVAE method for source separation. The MMVAE is better at dealing with data that have multiple modalities.
(4) Other Variations: Variance reduction of the VAE has been achieved through the STL estimator. Methods such as the IWAE use importance sampling to achieve a tighter lower bound for the loss function; variations of the IWAE using a DreG estimator or STL estimator have also reduced variance.

The applications of VAEs for generating images, speech/audio and text data are well known and well studied. In our article, we decided to focus on the less well known applications that VAEs can be used for, specifically in signal processing/time series analysis. In finance, we highlighted the use of the VAE in generation and completion of volatility surfaces for options, along with dimensionality reduction use cases. In the speech source separation subsection, we summarized the research on using the VAE framework for source separation in speech signals. We reviewed the use of the VAE framework for dimensionality reduction, generating disentangled interpretable features, and data generation for biosignals such as EEG, ECG, and EMG signals.

Some of the future potential areas of research for VAEs are:

(1) Disentanglement Representations: While many VAE extensions are state of the art for disentanglement, there is still the problem that they do not learn the disentangled representation in a truly unsupervised manner.
(2) Data Generation in Finance: VAEs are relatively unused in finance applications compared to other fields. For generating data, the VAE framework has been studied thoroughly and has had amazing results for natural images and audio data. Research into generating finance related data, such as volatility surfaces, is relatively unexplored, as we have only found two papers on this topic.
(3) Source Separation for Biosignals and Images: While speech source separation using the VAE framework has been heavily explored, the literature on biosignal source separation and image source separation with VAE is sparse. We see these as strong candidates for exploring the powerful capabilities of the VAE.

**Author Contributions:** Conceptualization, A.S., T.O.; methodology, A.S.; software, A.S.; validation, A.S.; formal analysis, A.S., T.O.; investigation, A.S.; resources, A.S., T.O.; data curation, A.S.; writing—original draft preparation, A.S., T.O.; writing—review and editing, A.S., T.O.; visualization, A.S.; supervision, T.O.; project administration, T.O.; funding acquisition, T.O. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** The TIMIT dataset is available in a publicly accessible repository.

**Acknowledgments:** We would like to acknowledge Maxime Bergeron, Sohrab Ferdowsi, and Hao Dao for helpful discussions.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Goodfellow, I.J.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
2. Kingma, D.P.; Welling, M. Auto-Encoding Variational Bayes. *arXiv* **2014**, arXiv:1312.6114.
3. Wei, R.; Garcia, C.; El-Sayed, A.; Peterson, V.; Mahmood, A. Variations in Variational Autoencoders—A Comparative Evaluation. *IEEE Access* **2020**, *8*, 153651–153670. [CrossRef]
4. Asperti, A.; Evangelista, D.; Piccolomini, E.L. A survey on Variational Autoencoders from a GreenAI perspective. *arXiv* **2021**, arXiv:2103.01071.
5. Cox, D.R. The Regression Analysis of Binary Sequences. *J. R. Stat. Soc. Ser. b-Methodol.* **1958**, *20*, 215–232. [CrossRef]
6. E Silva, D.G.; Fantinato, D.G.; Canuto, J.C.; Duarte, L.T.; Neves, A.; Suyama, R.; Montalvão, J.; de Faissol Attux, R. An Introduction to Information Theoretic Learning, Part I: Foundations. *J. Commun. Inf. Syst.* **2016**, *31*. [CrossRef]
7. Ogunfunmi, T.; Deb, M. On the PDF Estimation for Information Theoretic Learning for Neural Networks. In Proceedings of the 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Honolulu, HI, USA, 12–15 November 2018; pp. 1215–1221. [CrossRef]
8. Yu, S.; Príncipe, J.C. Understanding Autoencoders with Information Theoretic Concepts. *Neural Netw.* **2019**, *117*, 104–123. [CrossRef]
9. Tapia, N.I.; Est'evez, P.A. On the Information Plane of Autoencoders. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; pp. 1–8.
10. Tishby, N.; Pereira, F.C.; Bialek, W. The information bottleneck method. *arXiv* **2000**, arXiv:physics/0004057.
11. Cover, T.M.; Thomas, J.A. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*; Wiley-Interscience: Hoboken, NJ, USA, 2006; pp. 39–69.
12. Murphy, K.P. *Machine Learning: A Probabilistic Perspective*; MIT Press: Cambridge, MA, USA, 2012; pp. 733–734.
13. Jaoude, A.A. *The Monte Carlo Methods—Recent Advances, New Perspectives and Applications*; InfoTech Publishers: Irving, TX, USA, 2021.
14. Gobet, E. *Monte-Carlo Methods and Stochastic Processes: From Linear to Non-Linear*; CRC Press: Boca Raton, FL, USA, 2016; p. 10.
15. Li, B.*Math 214:Computational Stochastics: Lecture 1*; University of California: San Diego, CA, USA, 2021.
16. Chollet, F. Building Autoencoders in Keras. Available online: https://blog.keras.io/building-autoencoders-in-keras.html (accessed on 31 October 2021).
17. Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; Manzagol, P.A. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.* **2010**, *11*, 3371–3408.
18. Fleuret, F. Denoising Autoencoders. Available online: https://fleuret.org/dlc/materials/dlc-slides-7-3-denoising-autoencoders.pdf (accessed on 31 October 2021).
19. Barber, D. *Bayesian Reasoning and Machine Learning*; Cambridge University Press: Cambridge, UK, 2012.
20. Kingma, D.P. Variational Inference & Deep Learning: A New Synthesis. Ph.D. Thesis, University of Amsterdam, Amsterdam, The Netherlands, 2017.
21. Atanasov, N. *ECE 276A Sensing & Estimation in Robotics Lecture 4: Supervised Learning*; University of California: San Diego, CA, USA, 2020.
22. Mnih, A. Modern Latent Variable Models and Variational Inference, UCL x DeepMind. Available online: https://storage.googleapis.com/deepmind-media/UCLxDeepMind_2020/L11%20-%20UCLxDeepMind%20DL2020.pdf (accessed on 31 October 2021).
23. Vasconcelos, N. Mixture Density Estimation. Available online: http://www.svcl.ucsd.edu/courses/ece271A/handouts/mixtures.pdf (accessed on 31 October 2021).
24. Ogunfunmi, T.; Deb, M.K. Markov Chain Monte Carlo in a Dynamical System of Information Theoretic Particles. In *The Monte Carlo Methods—Recent Advances, New Perspectives and Applications*; Intechopen: London, UK, 2021.
25. Blei, D.M.; Kucukelbir, A.; McAuliffe, J.D. Variational Inference: A Review for Statisticians. *J. Am. Stat. Assoc.* **2016**, *112*, 859–877. [CrossRef]
26. Abbeel, P. *CS294-158-SP20 Deep Unsupervised Learning Lecture 4 Latent Variable Models—Variational AutoEncoder (VAE)*; University of California: Berkeley, CA, USA, 2020. Available online: https://drive.google.com/file/d/1JV-Rsz1MECZWWtvdXjxnt03HOxiGWPYy/view (accessed on 31 October 2021).
27. Kingma, D.P.; Welling, M. An Introduction to Variational Autoencoders. *Found. Trends Mach. Learn.* **2019**, *12*, 307–392. [CrossRef]
28. Bond-Taylor, S.; Leach, A.; Long, Y.; Willcocks, C.G. Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**. [CrossRef]

29. Abbeel, P. *CS294-158-SP20 Deep Unsupervised Learning Lecture 3 Likelihood Models: Flow Models*; University of California: Berkeley, CA, USA, 2020. Available online: https://drive.google.com/file/d/1j-3ErOVr8gPLEbN6J4jBeO84I7CqQdde/view (accessed on 31 October 2021).
30. Haykin, S. *Adaptive Filter Theory*, 4th ed.; Pearson: Albuquerque, NM, USA, 2002.
31. Abbeel, P. *CS294-158-SP20 Deep Unsupervised Learning Lecture 2 Likelihood Models: Autoregressive Models*; University of California: Berkeley, CA, USA, 2020. Available online: https://drive.google.com/file/d/1sHTVdppBqStzL1G1AHdWQrzHiqNFkzGH/view (accessed on 31 October 2021).
32. Larochelle, H.; Murray, I. The Neural Autoregressive Distribution Estimator. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, USA, 11–13 April 2011.
33. Uria, B.; Côté, M.A.; Gregor, K.; Murray, I.; Larochelle, H. Neural Autoregressive Distribution Estimation. *J. Mach. Learn. Res.* **2016**, *17*, 205:1–205:37.
34. Germain, M.; Gregor, K.; Murray, I.; Larochelle, H. MADE: Masked Autoencoder for Distribution Estimation. In Proceedings of the International Conference on Machine Learning (ICML), Lille, France, 6–11 July 2015.
35. Gregor, K.; Danihelka, I.; Mnih, A.; Blundell, C.; Wierstra, D. Deep AutoRegressive Networks. *arXiv* **2014**, arXiv:1310.8499.
36. van den Oord, A.; Kalchbrenner, N.; Kavukcuoglu, K. Pixel Recurrent Neural Networks. *arXiv* **2016**, arXiv:1601.06759.
37. van den Oord, A.; Kalchbrenner, N.; Espeholt, L.; Kavukcuoglu, K.; Vinyals, O.; Graves, A. Conditional Image Generation with PixelCNN Decoders. In Proceedings of the Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5–10 December 2016.
38. van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.W.; Kavukcuoglu, K. WaveNet: A Generative Model for Raw Audio. *arXiv* **2016**, arXiv:1609.03499.
39. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.C.; Bengio, Y. Generative Adversarial Nets. In Proceedings of the Annual Conference on Neural Information Processing Systems 2014 (NIPS), Montreal, QC, Canada, 8–13 December 2014.
40. Goodfellow, I.J. NIPS 2016 Tutorial: Generative Adversarial Networks. *arXiv* **2017**, arXiv:1701.00160.
41. Chang, D.T. Latent Variable Modeling for Generative Concept Representations and Deep Generative Models. *arXiv* **2018**, arXiv:1812.11856.
42. Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein GAN. *arXiv* **2017**, arXiv:1701.07875.
43. Mirza, M.; Osindero, S. Conditional Generative Adversarial Nets. *arXiv* **2014**, arXiv:1411.1784.
44. Radford, A.; Metz, L.; Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv* **2016**, arXiv:1511.06434.
45. Springenberg, J.T.; Dosovitskiy, A.; Brox, T.; Riedmiller, M.A. Striving for Simplicity: The All Convolutional Net. *arXiv* **2015**, arXiv:1412.6806.
46. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv* **2015**, arXiv:1502.03167.
47. Jiang, W.; Liu, S.; Gao, C.; Cao, J.; He, R.; Feng, J.; Yan, S. PSGAN: Pose and Expression Robust Spatial-Aware GAN for Customizable Makeup Transfer. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 14–19 June 2020; pp. 5193–5201.
48. Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; Abbeel, P. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In Proceedings of the Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5–10 December 2016.
49. Zhu, J.Y.; Park, T.; Isola, P.; Efros, A.A. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 2242–2251.
50. Karras, T.; Laine, S.; Aila, T. A Style-Based Generator Architecture for Generative Adversarial Networks. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 16–19 June 2019; pp. 4396–4405.
51. Zhang, H.; Goodfellow, I.J.; Metaxas, D.N.; Odena, A. Self-Attention Generative Adversarial Networks. In Proceedings of the International Conference on Machine Learning (ICML), Long Beach, CA, USA, 10–15 June 2019.
52. Williams, R.J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* **2004**, *8*, 229–256. [CrossRef]
53. Mohamed, S.; Rosca, M.; Figurnov, M.; Mnih, A. Monte Carlo Gradient Estimation in Machine Learning. *J. Mach. Learn. Res.* **2020**, *21*, 132:1–132:62.
54. Asperti, A. Variance Loss in Variational Autoencoders. *arXiv* **2020**, arXiv:2002.09860.
55. Zhao, S.; Song, J.; Ermon, S. Towards Deeper Understanding of Variational Autoencoding Models. *arXiv* **2017**, arXiv:1702.08658.
56. Larsen, A.B.L.; Sønderby, S.K.; Larochelle, H.; Winther, O. Autoencoding beyond pixels using a learned similarity metric. *arXiv* **2016**, arXiv:1512.09300.
57. Cai, L.; Gao, H.; Ji, S. Multi-Stage Variational Auto-Encoders for Coarse-to-Fine Image Generation. In Proceedings of the SIAM International Conference on Data Mining (SDM), Calgry, AB, Canada, 2–4 May 2019. [CrossRef]

58. Kingma, D.P.; Salimans, T.; Józefowicz, R.; Chen, X.; Sutskever, I.; Welling, M. Improving Variational Autoencoders with Inverse Autoregressive Flow. In Proceedings of the Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5–10 December 2016.
59. Vahdat, A.; Kautz, J. NVAE: A Deep Hierarchical Variational Autoencoder. *arXiv* **2020**, arXiv:2007.03898.
60. Gulrajani, I.; Kumar, K.; Ahmed, F.; Taïga, A.A.; Visin, F.; Vázquez, D.; Courville, A.C. PixelVAE: A Latent Variable Model for Natural Images. *arXiv* **2017**, arXiv:1611.05013.
61. Dai, B.; Wipf, D.P. Diagnosing and Enhancing VAE Models. *arXiv* **2019**, arXiv:1903.05789.
62. Bengio, Y.; Courville, A.C.; Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1798–1828. [CrossRef] [PubMed]
63. Locatello, F.; Bauer, S.; Lucic, M.; Gelly, S.; Schölkopf, B.; Bachem, O. Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations. *arXiv* **2019**, arXiv:1811.12359.
64. Zhao, S.; Song, J.; Ermon, S. InfoVAE: Balancing Learning and Inference in Variational Autoencoders. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI), Honolulu, HI, USA, 27 January–1 February 2019. [CrossRef]
65. Higgins, I.; Matthey, L.; Pal, A.; Burgess, C.P.; Glorot, X.; Botvinick, M.M.; Mohamed, S.; Lerchner, A. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In Proceedings of the International Conference on Learning Representations (ICLR), Toulon, France, 24–26 April 2017.
66. Chen, T.Q.; Li, X.; Grosse, R.B.; Duvenaud, D.K. Isolating Sources of Disentanglement in Variational Autoencoders. In Proceedings of the 2018 Conference on Neural Information Processing Systems (NeurIPS), Montreal, QC, Canada, 3–8 December 2018.
67. Burgess, C.P.; Higgins, I.; Pal, A.; Matthey, L.; Watters, N.; Desjardins, G.; Lerchner, A. Understanding disentangling in $\beta$-VAE. *arXiv* **2018**, arXiv:1804.03599.
68. Kim, H.; Mnih, A. Disentangling by Factorising. *arXiv* **2018**, arXiv:1802.05983.
69. Asperti, A.; Trentin, M. Balancing Reconstruction Error and Kullback–Leibler Divergence in Variational Autoencoders. *IEEE Access* **2020**, *8*, 199440–199448. [CrossRef]
70. Lucas, J.; Tucker, G.; Grosse, R.B.; Norouzi, M. Don't Blame the ELBO! A Linear VAE Perspective on Posterior Collapse. In Proceedings of the 2019 Conference on Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 8–14 December 2019.
71. Razavi, A.; van den Oord, A.; Poole, B.; Vinyals, O. Preventing Posterior Collapse with delta-VAEs. *arXiv* **2019**, arXiv:1901.03416.
72. Tomczak, J.M.; Welling, M. VAE with a VampPrior. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), Lanzarote, Spain, 9–11 April 2018.
73. Yeung, S.; Kannan, A.; Dauphin, Y.; Fei-Fei, L. Tackling Over-pruning in Variational Autoencoders. *arXiv* **2017**, arXiv:1706.03643.
74. van den Oord, A.; Vinyals, O.; Kavukcuoglu, K. Neural Discrete Representation Learning. In Proceedings of the Conference on Neural Information Processing Systems NIPS, Long Beach, CA, USA, 4–9 December 2017.
75. Jiang, Z.; Zheng, Y.; Tan, H.; Tang, B.; Zhou, H. Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Melbourne, Australia, 19–25 August 2017.
76. Dilokthanakul, N.; Mediano, P.A.M.; Garnelo, M.; Lee, M.J.; Salimbeni, H.; Arulkumaran, K.; Shanahan, M. Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders. *arXiv* **2016**, arXiv:1611.02648.
77. Davidson, T.R.; Falorsi, L.; Cao, N.D.; Kipf, T.; Tomczak, J.M. Hyperspherical Variational Auto-Encoders. In Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI), Monterey, CA, USA, 7–9 August 2018.
78. Park, B. The Curse of Dimensionality. 2018. Available online: https://barumpark.com/blog/2018/the-curse-of-dimensionality/ (accessed on 31 October 2021).
79. Roeder, G.; Wu, Y.; Duvenaud, D.K. Sticking the Landing: Simple, Lower-Variance Gradient Estimators for Variational Inference. In Proceedings of the Conference on Neural Information Processing Systems NIPS, Long Beach, CA, USA, 4–9 December 2017.
80. Ferdowsi, S.; Diephuis, M.; Rezaeifar, S.; Voloshynovskiy, S. $\rho$-VAE: Autoregressive parametrization of the VAE encoder. *arXiv* **2019**, arXiv:1909.06236.
81. Li, B. *Math 214:Computational Stochastics: Lecture 12*; University of California: San Diego, CA, USA, 2021.
82. Burda, Y.; Grosse, R.B.; Salakhutdinov, R. Importance Weighted Autoencoders. *arXiv* **2016**, arXiv:1509.00519.
83. Rainforth, T.; Kosiorek, A.R.; Le, T.A.; Maddison, C.J.; Igl, M.; Wood, F.; Teh, Y.W. Tighter Variational Bounds are Not Necessarily Better. In Proceedings of the International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018.
84. Tucker, G.; Lawson, D.; Gu, S.S.; Maddison, C.J. Doubly Reparameterized Gradient Estimators for Monte Carlo Objectives. *arXiv* **2019**, arXiv:1810.04152.
85. Wu, M.; Goodman, N.D. Multimodal Generative Models for Scalable Weakly-Supervised Learning. In Proceedings of the 2018 Conference on Neural Information Processing Systems (NeurIPS), Montreal, QC, Canada, 3–8 December 2018.
86. Shi, Y.; Siddharth, N.; Paige, B.; Torr, P.H.S. Variational Mixture-of-Experts Autoencoders for Multi-Modal Deep Generative Models. *arXiv* **2019**, arXiv:1911.03393.
87. Li, Y.; Turner, R.E. Rényi Divergence Variational Inference. In Proceedings of the Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5–10 December 2016.
88. Mathieu, E.; Rainforth, T.; Narayanaswamy, S.; Teh, Y.W. Disentangling Disentanglement in Variational Autoencoders. In Proceedings of the International Conference on Machine Learning (ICML), Long Beach, CA, USA, 10–15 June 2019.

89. Sikka, H.D.; Zhong, W.; Yin, J.; Pehlevan, C. A Closer Look at Disentangling in $\beta$-VAE. In Proceedings of the 2019 53rd Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 3–6 November 2019; pp. 888–895.
90. Sadeghi, H.; Andriyash, E.; Vinci, W.; Buffoni, L.; Amin, M.H. PixelVAE++: Improved PixelVAE with Discrete Prior. *arXiv* **2019**, arXiv:1908.09948.
91. Ulrich, G. Computer Generation of Distributions on the M-Sphere. *J. R. Stat. Soc. Ser. C-Appl. Stat.* **1984**, *33*, 158–163. [CrossRef]
92. Naesseth, C.A.; Ruiz, F.J.R.; Linderman, S.W.; Blei, D.M. Reparameterization Gradients through Acceptance-Rejection Sampling Algorithms. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, USA, 20–22 April 2017.
93. Sohn, K.; Lee, H.; Yan, X. Learning Structured Output Representation using Deep Conditional Generative Models. In Proceedings of the Annual Conference on Neural Information Processing Systems 2015 (NIPS), Montreal, QC, Canada, 7–12 December 2015.
94. Gao, R.; Hou, X.; Qin, J.; Chen, J.; Liu, L.; Zhu, F.; Zhang, Z.; Shao, L. Zero-VAE-GAN: Generating Unseen Features for Generalized and Transductive Zero-Shot Learning. *IEEE Trans. Image Process.* **2020**, *29*, 3665–3680. [CrossRef] [PubMed]
95. Xian, Y.; Sharma, S.; Schiele, B.; Akata, Z. F-VAEGAN-D2: A Feature Generating Framework for Any-Shot Learning. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 16–19 June 2019; pp. 10267–10276.
96. Wu, J.; Zhang, C.; Xue, T.; Freeman, B.; Tenenbaum, J.B. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In Proceedings of the Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5–10 December 2016.
97. Gur, S.; Benaim, S.; Wolf, L. Hierarchical Patch VAE-GAN: Generating Diverse Videos from a Single Sample. *arXiv* **2020**, arXiv:2006.12226.
98. Makhzani, A.; Shlens, J.; Jaitly, N.; Goodfellow, I.J. Adversarial Autoencoders. *arXiv* **2015**, arXiv:1511.05644.
99. Santana, E.; Emigh, M.S.; Príncipe, J.C. Information Theoretic-Learning auto-encoder. In Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada, 24–29 July 2016; pp. 3296–3301.
100. Silverman, B.W. *Density Estimation for Statistics and Data Analysis*; Chapman & Hall: London, UK, 1986.
101. Chung, J.; Kastner, K.; Dinh, L.; Goel, K.; Courville, A.C.; Bengio, Y. A Recurrent Latent Variable Model for Sequential Data. In Proceedings of the Annual Conference on Neural Information Processing Systems 2015 (NIPS), Montreal, QC, Canada, 7–12 December 2015.
102. Fabius, O.; van Amersfoort, J.R.; Kingma, D.P. Variational Recurrent Auto-Encoders. *arXiv* **2015**, arXiv:1412.6581.
103. Razavi, A.; van den Oord, A.; Vinyals, O. Generating Diverse High-Fidelity Images with VQ-VAE-2. *arXiv* **2019**, arXiv:1906.00446.
104. Tolstikhin, I.O.; Bousquet, O.; Gelly, S.; Schölkopf, B. Wasserstein Auto-Encoders. *arXiv* **2018**, arXiv:1711.01558.
105. Bergeron, M.; Fung, N.; Poulos, Z.; Hull, J.; Veneris, A.G. Variational Autoencoders: A Hands-Off Approach to Volatility. *Risk Manag. Anal. Financ. Inst.* **2021**. [CrossRef]
106. Wilmott, P. *Paul Wilmott Introduces Quantitative Finance*, 2nd ed.; Wiley-Interscience: Hoboken, NJ, USA, 2007.
107. Ning, B.; Jaimungal, S.; Zhang, X.; Bergeron, M. Arbitrage-Free Implied Volatility Surface Generation with Variational Autoencoders. *arXiv* **2021**, arxiv: 2108.04941.
108. Gunduz, H. An efficient stock market prediction model using hybrid feature reduction method based on variational autoencoders and recursive feature elimination. *Financ. Innov.* **2021**, *7*, 28. [CrossRef]
109. Choudhury, A.R.; Abrishami, S.; Turek, M.; Kumar, P. Enhancing profit from stock transactions using neural networks. *AI Commun.* **2020**, *33*, 75–92. [CrossRef]
110. Zhang, C.; Liang, S.; Lyu, F.; Fang, L. Stock-Index Tracking Optimization Using Auto-Encoders. *Front. Phys.* **2020**, *8*, 388. [CrossRef]
111. Ogunfunmi, T.; Ramachandran, R.P.; Togneri, R.B.; Zhao, Y.; Xia, X. A Primer on Deep Learning Architectures and Applications in Speech Processing. *Circuits Syst. Signal Process.* **2019**, *38*, 3406–3432. [CrossRef]
112. Févotte, C.; Gribonval, R.; Vincent, E. BSS_EVAL Toolbox User Guide—Revision 2.0. 2005. Available online: https://gitlab.inria.fr/bass-db/bss_eval (accessed on 31 October 2021).
113. Rix, A.W.; Beerends, J.G.; Hollier, M.; Hekstra, A.P. Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs. In Proceedings of the 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing, Salt Lake City, UT, USA, 7–11 May 2001; Volume 2, pp. 749–752.
114. Sübakan, Y.C.; Smaragdis, P. Generative Adversarial Source Separation. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15–20 April 2018; pp. 26–30.
115. Garofolo, J.S.; Lamel, L.F.; Fisher, W.M.; Fiscus, J.G.; Pallett, D.S.; Dahlgren, N.L. *DARPA TIMIT Acoustic Phonetic Continuous Speech Corpus CDROM*; Linguistic Data Consortium: Philadelphia, PA, USA, 1993.
116. Do, H.D.; Tran, S.T.; Chau, D.T. Speech Source Separation Using Variational Autoencoder and Bandpass Filter. *IEEE Access* **2020**, *8*, 156219–156231. [CrossRef]
117. Karamatlı, E.; Cemgil, A.T.; Kırbız, S. Weak Label Supervision for Monaural Source Separation Using Non-negative Denoising Variational Autoencoders. In Proceedings of the 2019 27th Signal Processing and Communications Applications Conference (SIU), Sivas, Turkey, 24–26 April 2019; pp. 1–4. [CrossRef]

118. Grais, E.M.; Plumbley, M.D. Single channel audio source separation using convolutional denoising autoencoders. In Proceedings of the 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP), Montreal, QC, Canada, 14–16 November 2017; pp. 1265–1269.
119. Kameoka, H.; Li, L.; Inoue, S.; Makino, S. Semi-blind source separation with multichannel variational autoencoder. *arXiv* **2018**, arXiv:1808.00892.
120. Kameoka, H.; Li, L.; Inoue, S.; Makino, S. Supervised Determined Source Separation with Multichannel Variational Autoencoder. *Neural Comput.* **2019**, *31*, 1891–1914. [CrossRef]
121. Seki, S.; Kameoka, H.; Li, L.; Toda, T.; Takeda, K. Underdetermined Source Separation Based on Generalized Multichannel Variational Autoencoder. *IEEE Access* **2019**, *7*, 168104–168115. [CrossRef]
122. Seki, S.; Kameoka, H.; Li, L.; Toda, T.; Takeda, K. Generalized Multichannel Variational Autoencoder for Underdetermined Source Separation. In Proceedings of the 2019 27th European Signal Processing Conference (EUSIPCO), A Coruna, Spain, 2–6 September 2019; pp. 1–5. [CrossRef]
123. Li, L.; Kameoka, H.; Makino, S. Fast MVAE: Joint Separation and Classification of Mixed Sources Based on Multichannel Variational Autoencoder with Auxiliary Classifier. In Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 2–17 May 2019; pp. 546–550. [CrossRef]
124. Inoue, S.; Kameoka, H.; Li, L.; Seki, S.; Makino, S. Joint Separation and Dereverberation of Reverberant Mixtures with Multichannel Variational Autoencoder. In Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 2–17 May 2019; pp. 96–100. [CrossRef]
125. Chien, J.T.; Kuo, K.T. Variational Recurrent Neural Networks for Speech Separation. In Proceedings of the Conference of the International Speech Communication Association (INTERSPEECH), Stockholm, Sweden, 20–24 August 2017.
126. Girin, L.; Roche, F.; Hueber, T.; Leglaive, S. Notes on the use of variational autoencoders for speech and audio spectrogram modeling. In Proceedings of the DAFx 2019—22nd International Conference on Digital Audio Effects, Birmingham, UK, 2–6 September 2019.
127. Bando, Y.; Sekiguchi, K.; Yoshii, K. Adaptive Neural Speech Enhancement with a Denoising Variational Autoencoder. In Proceedings of the Conference of the International Speech Communication Association (INTERSPEECH), Shanghai, China, 25–29 October 2020.
128. Fang, H.; Carbajal, G.; Wermter, S.; Gerkmann, T. Variational Autoencoder for Speech Enhancement with a Noise-Aware Encoder. In Proceedings of the 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 6–11 June 2021; pp. 676–680.
129. Leglaive, S.; Alameda-Pineda, X.; Girin, L.; Horaud, R. A Recurrent Variational Autoencoder for Speech Enhancement. In Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 371–375.
130. Sadeghi, M.; Leglaive, S.; Alameda-Pineda, X.; Girin, L.; Horaud, R. Audio-visual Speech Enhancement Using Conditional Variational Auto-Encoder. *arXiv* **2019**, arXiv:1908.02590.
131. Introduction to ECG. Available online: https://www.healio.com/cardiology/learn-the-heart/ecg-review/ecg-interpretation-tutorial/introduction-to-the-ecg (accessed on 31 October 2021).
132. Prasad, S.T.; Varadarajan, D.S. ECG Signal Processing Using Digital Signal Processing Techniques. *Int. J. Sci. Eng. Res.* **2013**, *4*, 1624–1628.
133. Isın, A.; Ozdalili, S. Cardiac arrhythmia detection using deep learning. *Procedia Comput. Sci.* **2017**, *120*, 268–275. [CrossRef]
134. Cejnek, M. Signalz: Synthetic Data Generators in Python. 2017. Available online: https://matousc89.github.io/signalz/ (accessed on 14 November 2021).
135. Andreotti, F.; Behar, J.; Zaunseder, S.; Oster, J.; Clifford, G.D.; Oster, J.; Clifford, G.D. An Open-Source Framework for Stress-Testing Non-Invasive Foetal ECG Extraction Algorithms. *Physiol. Meas.* **2016**, *37*, 627–648. [CrossRef]
136. Ong, Y.Z.; Chui, C.K.; Yang, H. CASS: Cross Adversarial Source Separation via Autoencoder. *arXiv* **2019**, arXiv:1905.09877.
137. Chen, S.; Meng, Z.; Zhao, Q. Electrocardiogram Recognization Based on Variational AutoEncoder. In *Machine Learning and Biometrics*; Intechopen: London, UK, 2018. [CrossRef]
138. Introduction to Pathology. Available online: https://www.animalnexus.com.pk/uploads/documents/Pathology.pdf (accessed on 31 October 2021).
139. Liu, H.; Zhao, Z.; Chen, X.; Yu, R.; She, Q. Using the VQ-VAE to improve the recognition of abnormalities in short-duration 12-lead electrocardiogram records. *Comput. Methods Programs Biomed.* **2020**, *196*, 105639. [CrossRef] [PubMed]
140. Cho, Y.H.; myoung Kwon, J.; Kim, K.H.; Medina-Inojosa, J.R.; Jeon, K.H.; Cho, S.; Lee, S.Y.; Park, J.; Oh, B.H. Artificial intelligence algorithm for detecting myocardial infarction using six-lead electrocardiography. *Sci. Rep.* **2020**, *10*, 20495. [CrossRef] [PubMed]
141. Steenkiste, T.V.; Deschrijver, D.; Dhaene, T. Generating an Explainable ECG Beat Space With Variational Auto-Encoders. *arXiv* **2019**, arXiv:1911.04898.
142. Kuznetsov, V.V.; Moskalenko, V.A.; Gribanov, D.V.; Zolotykh, N.Y. Interpretable Feature Generation in ECG Using a Variational Autoencoder. *Front. Genet.* **2021**, *12*, 638191. [CrossRef] [PubMed]
143. Bacoyannis, T.; Krebs, J.; Cedilnik, N.; Cochet, H.; Sermesant, M. Deep Learning Formulation of ECGI for Data-Driven Integration of Spatiotemporal Correlations and Imaging Information. In Proceedings of the Functional Imaging and Modeling of the Heart: 10th International Conference (FIMH), Bordeaux, France, 6–8 June 2019.

144. Krishna, G.; Tran, C.; Carnahan, M.; Tewfik, A.H. Constrained Variational Autoencoder for improving EEG based Speech Recognition Systems. *arXiv* **2020**, arXiv:2006.02902.
145. Li, X.; Zhao, Z.; Song, D.; Zhang, Y.; Pan, J.; Wu, L.; Huo, J.; Niu, C.; Wang, D. Latent Factor Decoding of Multi-Channel EEG for Emotion Recognition Through Autoencoder-Like Neural Networks. *Front. Neurosci.* **2020**, *14*, 87. [CrossRef] [PubMed]
146. Hagad, J.L.; Kimura, T.; ichi Fukui, K.; Numao, M. Learning Subject-Generalized Topographical EEG Embeddings Using Deep Variational Autoencoders and Domain-Adversarial Regularization. *Sensors* **2021**, *21*, 1792. [CrossRef] [PubMed]
147. Vereshchaka, A.; Yang, F.; Suresh, A.; Olokodana, I.L.; Dong, W. Predicting Cognitive Control in Older Adults using Deep Learning and EEG Data. In Proceedings of the 2020 International Conference on Social Computing, Behavioral-Cultural Modeling & Prediction and Behavior Representation in Modeling and Simulation (SBP-BRiMS 2020), Washington, DC, USA, 19–22 October 2020; pp. 19–22.
148. Electromyography (EMG). Available online: https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electromyography-emg (accessed on 31 October 2021).
149. Teh, Y.; Hargrove, L.J. Using Latent Representations of Muscle Activation Patterns to Mitigate Myoelectric Interface Noise. In Proceedings of the 2021 10th International IEEE/EMBS Conference on Neural Engineering (NER), Virtual Event, Italy, 4–6 May 2021; pp. 1148–1151.
150. Farshchian, A.; Gallego, J.A.; Cohen, J.P.; Bengio, Y.; Miller, L.E.; Solla, S.A. Adversarial Domain Adaptation for Stable Brain-Machine Interfaces. *arXiv* **2019**, arXiv:1810.00045.