Variational Auto-Encoders (VAEs): A Review

Hemanth Bharatha Chakravarthy*

December 7, 2022

Contents

1 Introduction

"What I cannot create, I do not understand."

Richard Feynman
Chalkboard quote

Variational Autoencoders provide a relatively principled and interpretable framework for learning deep latent-variable models and corresponding inference models using stochastic gradient descent. Kingma and Welling (2013) introduced the method, utilizing autoencoding for *variational Bayes* to learn directed probabilistic

*Final paper for Statistics 185 (Fall 2022) with Prof. Alex Young at Harvard College, hbharathachakravarthy@gmail.com

models in settings with continuous latent variables with intractable posterior distributions and large N data. Simply put, VAE can be thought of as an autoencoder model which returns a distribution over encodings rather than a single encoded point and whose training process attracts this latent distribution to be close to the standard normal.

Typically, we have N i.i.d samples of d continuous or discrete features, $\vec{x}_1, \ldots, \vec{x}_N \in \mathbb{R}^d$. These construct a data matrix in $\mathbb{R}^{N \times d}$ with its rows as samples, $\mathcal{X} = [\vec{x}_1^T \ \ \vec{x}_2^T \ \ \ldots \ \ \vec{x}_N^T]^T$. We assume that the data are generated by an unobserved, continuous random variable $\mathcal{Z}$ through an (intractable) posterior distribution $p(\mathcal{Z} \mid \mathcal{X})$. We initialize $\mathcal{Z}$ from a prior distribution $p_{\theta*}(\mathcal{Z})$. Then, we decode encoded $\vec{x}_i$ from some conditional distribution $p_{\theta*}(\mathcal{X} \mid \mathcal{Z})$. To simplify, at the vector level, the encoding reduces dimensionality to some $\mathbb{R}^k$ where $k << d$, and sends encoded sample vectors $\vec{z}_i \in \mathbb{R}^k$ to the generator. The decoder is a Bayesian network of the form $p(\vec{x}_i \mid \vec{z}_i)p(\vec{z}_i)$, or $p(\vec{x}_i \mid \vec{z}_l)p(\vec{z}_l \mid \vec{z}_{l-1}) \ldots p(\vec{z}_1 \mid \vec{z}_0)$ when there are multiple latent layers. Inside each conditional, there is a neural network of some depth, such that $\vec{z}_i \mid \vec{x}_i \sim f(\vec{x}_i, \vec{\epsilon})$ where $\vec{\epsilon}$ is a constant vector in $\mathbb{R}^d$ of some noise ϵ. Through its iterative training, the model updates the joint distributions, modeling both $p(\vec{x}_i \mid \vec{z}_i)$ and $p(\vec{z}_i \mid \vec{x}_i)$. (Kingma et al., 2019; Kingma & Welling, 2013)

Broadly, the problem in generative modeling is to learn the joint distribution over all available variables. In VAEs, the encoder learns a posterior distribution of *latent* random variables, and the decoder learns a posterior distribution of meaningful reconstructions from the lower-dimensional encoding. So, VAE can be conceived as two coupled but independently parameterized models: the encoder (or recognition or inference model) and the decoder (or generative model). The encoder is a function of the data $\mathcal{X}$ and the parameter ϕ of the encoder, and is a parametric family of distributions. The encoder is trained to approximate the posterior, and the decoder is trained to approximate the likelihood. Or in other words, the encoder, rooted in Bayesian inference, learns the underlying distribution creating the data so that the decoder can generate new samples from that distribution. (Kristiadi, 2020)

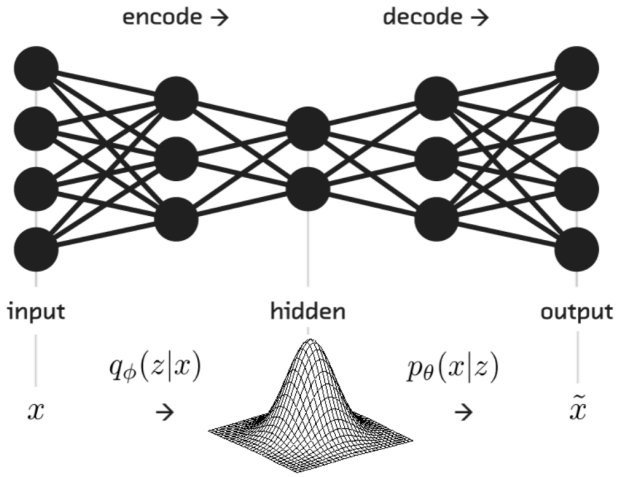

Figure 1: VAE Overview from Shiffman (2016)

The elegance of VAEs is that they simulate the underlying process generating the data. Rather than encode a single approximation, VAEs learn the distribution of approximations. Across iterations, the encoder learns lower-dimensional features, like the curves and edges that constitute a handwritten digit. But once we have constructed these underlying latent spaces, we have recovered the latent manifold and assigned it a coordinate system. Now, it becomes trivial to walk from one point to another along the manifold, running the new samples into the decoder. (Shiffman, 2016) Returning to the handwritten digit example, by picking more samples from the encoded distribution of 4s, we can generate handwritten 4s that the model has never seen

before in its training set.

The loss that is "backpropagated" by VAEs is a sum of reconstruction loss (the distance between data and its encoded-decoded representation) and the *Kullback-Liebler divergence* of the encoder's distribution from the standard normal—$\mathcal{N}(0, \mathbb{I}_k)$—attracting the encoded distributions towards the mean 0 and identity covariance normal distribution. (Kullback & Leibler, 1951; Rocca, 2019) Thus importantly, VAEs offer a method to learn the joint distributions with regularization built into the training process. From pushing the distribution towards mean 0, VAEs attempt to gain local regularization, and from pushing the covariance towards the identity matrix, VAEs attempt to gain global regularization.

This also draws attention to the peculiar position of VAEs as a deep latent-variable model vis-à-vis other dimensionality reduction techniques in unsupervised learning. We can think of a method like PCA as an encoder-decoder setup: the scores are the encodings, and the components (or eigenvectors) are the underlying approximation learned by the encoding. And there is a simple matrix multiplication operation to decode data with the components matrix and the encoded scores. If we do not cut components to attain a lower-dimensional approximation, we can reconstruct data without loss. Even if there is loss in UL methods, these methods are still typically attempting to learn a different representation of the data, achieved through transformations, like in PCA, instead of learning a stochastic function of input variables. That is, VAEs make the assumption that latent spaces are close to normal, and stochastically learn the underlying latent distribution generating the data. This perhaps makes it more similar to clustering methods that make (often Gaussian) parametric assumptions.

2 Method[1]

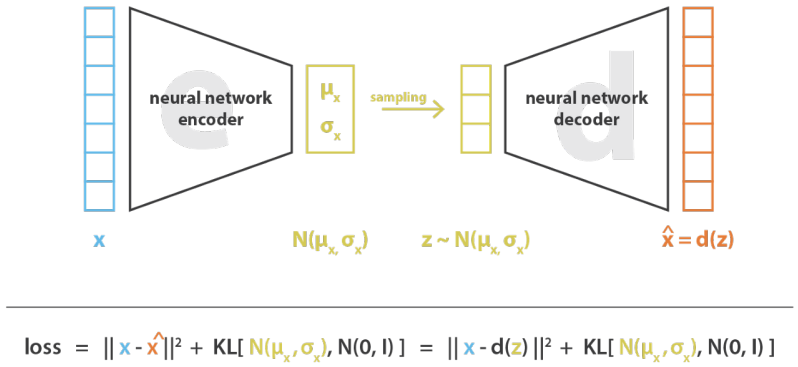

Figure 2: VAE overview from Rocca (2019)

2.1 Setting

Again, we have a data matrix with its rows each as one the N i.i.d samples ($\vec{x}_1, \ldots, \vec{x}_N \in \mathbb{R}^d$) of d continuous or discrete features, $\mathcal{X} = [\vec{x}_1^T \quad \vec{x}_2^T \quad \ldots \quad \vec{x}_N^T]^T \in \mathbb{R}^{N \times d}$. Equivalently, we will notate this as $\mathcal{X} = \{ \vec{x}_i^{(j)} \}_{i=1}^N$ where $\vec{x}_i^{(j)}$ is the jth component of $\vec{x}_i$. We assume that there is some continuous latent random variable, $\mathcal{Z}$, controlling the process generating $\mathcal{X}$. $\mathcal{Z}$ is unobserved and lives in some $\mathbb{R}^{N \times k}$ where $k << d$, such that the rows of Z are the latent variables (or lower-dimensional "features") of corresponding

[1] A plurality of the formalization in sections 2 to 2.5 is adapted from the original VAE paper (Kingma & Welling, 2013) and Chapters 1 and 2 of *An Introduction to Variational Autoencoders* (Kingma et al., 2019).

data: $\mathcal{Z} = [\vec{z}_1^T \quad \vec{z}_2^T \quad \dots \quad \vec{z}_N^T]^T = \{ \ \vec{z}_i^{(j)} \ \}_{i=1}^N$ where $\vec{z}_i \in \mathbb{R}^k$. This process consists of two steps: first, a $\vec{z}_i$ is initialized from some prior distribution $p_{\theta*}(\mathcal{Z})$, and second, the corresponding sample $\vec{x}_i$ is generated from some conditional distribution $p_{\theta*}(\mathcal{X} \mid \mathcal{Z})$. This prior $p_{\theta*}(\mathcal{Z})$ and its likelihood $p_{\theta*}(\mathcal{X} \mid \mathcal{Z})$ are both parametrized towards some unknown true parameter θ, and accordingly, we assume that they come from parametric families $p_\theta(\mathcal{Z})$ and $p_\theta(\mathcal{X} \mid \mathcal{Z})$. Further, we assume that their PDFs are differentiable almost everywhere with respect to θ and $\mathcal{Z}$.

If we wish to model the data, we wish to find $p_\theta(\mathcal{X}) = \int p_\theta(\mathcal{X} \mid \mathcal{Z}) p_\theta(\mathcal{Z}) \, d\mathcal{Z}$. Here, VAEs build an algorithm more general than the *Expectation-Maximation (EM)* algorithm that is relatively robust to *intractability* and large datasets. VAEs create three efficiency gains: first, to estimate θ (which can be part of the objective, for instance, if studying some natural process); second, to approximate the posterior encoding of the latent variable $\mathcal{Z}$ given the data $\mathcal{X}$ and the choice of parameter θ; third, to approximate marginal inference of the data $\mathcal{X}$ which is useful anywhere where a prior over data is important, for example, in inpainting or denoising in computer vision.

2.2 Encoder-Decoder

The problem occurs when the integral of the marginal likelihood—$p_\theta(\mathcal{X}) = \int p_\theta(\mathcal{Z}) p_\theta(\mathcal{X} \mid \mathcal{Z}) \, d\mathcal{Z}$—is intractable to compute or differentiate. But this expression on the right-hand side is equivalent to marginalizing $\mathcal{Z}$ from the joint distribution $p_\theta(\mathcal{X}, \mathcal{Z})$ and so, knowing the joint distribution is equivalent to knowing both $p_\theta(\mathcal{X} \mid \mathcal{Z})$ and $p_\theta(\mathcal{Z})$. So VAE seeks to infer $p_\theta(\mathcal{Z})$ from $p_\theta(\mathcal{Z} \mid \mathcal{X})$. To evade intractability thus, VAE utilizes the encoder to approximate the posterior distribution with optimized variational parameters ϕ, such that:

$$\mathbf{q}_\phi(\mathcal{Z} \mid \mathcal{X}) \approx \mathbf{p}_\theta(\mathcal{Z} \mid \mathcal{X})$$

Thus the encoder discovers the likely latent variables from the data matrix. The prior over the latent random variable is typically assumed to be multivariate normal $\mathcal{Z} \sim \mathcal{N}(0, \mathbb{I}_k)$. Similarly, $p_\theta(\mathcal{X} \mid \mathcal{Z})$ is assumed to be multivariate normal (in the case of real-valued data but it would be a Bernoulli distribution in the case of binary data). And as above, $q_\phi(\mathcal{Z} \mid \mathcal{X})$, which we will estimate, is a multivariate normal with diagonal covariance of the form $\mathcal{N}(\vec{\mu}, \vec{\sigma}^2 \odot \mathbb{I}_k)$ where $\odot$ indicates element-wise multiplication. So, the encoder estimates $2k$ parameters when choosing a k-dimensional approximation for its latent space, estimating $\vec{\mu}$ and each of the elements of the diagonal covariance matrix, $\sigma^{(1)}, \dots, \sigma^{(k)}$, and consequently estimating the latent distribution.

The encoder can be almost any neural network such as a feed-forward model or a convolutional network of any depth and complexity. For example, we will assume it is an *MLP* (multi-layer perceptron). Let layers be $l \in \{1, \dots, L\}$ and let the encoding as of layer l be $\mathcal{Z}^{(l)}$. So:

$$q_\phi(\mathcal{Z} \mid \mathcal{X}) = q_\phi(\mathcal{Z}^{(1)}, \dots, \mathcal{Z}^{(L)} \mid \mathcal{X})$$

For a practical estimator of the lower bound and its derivatives with respect to its parameters, we can reparametrize $\hat{\mathcal{Z}} \sim q_\phi(\mathcal{Z} \mid \mathcal{X})$ using a differentiable transformation $g_\phi(\boldsymbol{\epsilon}, \mathcal{X})$ such that $\hat{\mathcal{Z}} = g_\phi(\boldsymbol{\epsilon}, \mathcal{X})$ with $\boldsymbol{\epsilon} \sim p(\epsilon)$. Consequently, given a matrix of noise $\boldsymbol{\epsilon}$, the encoder is a stochastic function of the data $\mathcal{X}$ and the noise $\boldsymbol{\epsilon}^{(1)}, \dots, \boldsymbol{\epsilon}^{(L)} \sim \mathcal{N}(0, \mathbb{I}_k)$ i.i.d:

$$(\vec{\mu}, \vec{\sigma}) = \text{EncoderNeuralNet}_\phi^L(\mathcal{X}, \boldsymbol{\epsilon})$$

The encoder acts like an approximate inverse of the decoder according to Bayes rule. The decoder is a neural network that takes the latent variable $\mathcal{Z}$ as input and outputs the reconstructed data $\hat{\mathcal{X}}$. The decoder is a deterministic function of the latent variable $\mathcal{Z}$:

$$p_\theta(\vec{x}_i \mid \vec{z}_i) = p_\theta(\vec{x}_i \mid \vec{z}_i^{(1)}, \dots, \vec{z}_i^{(L)}) \sim \mathcal{N}(\vec{\mu}_i, \vec{\sigma}_i^{\,2} \odot \mathbb{I}_k)$$

The decoder too can be almost any kind of neutral network with any depth and complexity:

$$\hat{\mathcal{X}} = \text{DecoderNeuralNet}_\theta^L(\mathcal{Z})$$

2.3 Evidence Lower-Bound (ELBO)

For any choice of inference model $q_\phi(\mathcal{Z} \mid \mathcal{X})$ and choice of parameter ϕ, we have:

$$\begin{aligned}
\log p_\theta(\mathcal{X}) &= \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\log p_\theta(\mathcal{X})\right] \\
&= \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\log\left[\frac{p_\theta(\mathcal{X},\mathcal{Z})}{p_\theta(\mathcal{Z}\mid\mathcal{X})}\right]\right] \\
&= \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\log\left[\frac{p_\theta(\mathcal{X},\mathcal{Z})}{q_\phi(\mathcal{Z}\mid\mathcal{X})}\frac{q_\phi(\mathcal{Z}\mid\mathcal{X})}{p_\theta(\mathcal{Z}\mid\mathcal{X})}\right]\right] \\
&= \underbrace{\mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\log\left[\frac{p_\theta(\mathcal{X},\mathcal{Z})}{q_\phi(\mathcal{Z}\mid\mathcal{X})}\right]\right]}_{\substack{=\mathcal{L}_{\theta,\phi}(\mathcal{X}) \\ \text{(ELBO)}}} + \underbrace{\mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\log\left[\frac{q_\phi(\mathcal{Z}\mid\mathcal{X})}{p_\theta(\mathcal{Z}\mid\mathcal{X})}\right]\right]}_{=D_{KL}(q_\phi(\mathcal{Z}|\mathcal{X})\|p_\theta(\mathcal{Z}|\mathcal{X}))}
\end{aligned}$$

The second term on the right-hand side is the Kullback-Leibler (KL) divergence between $q_\phi(\mathcal{Z} \mid \mathcal{X})$ and $p_\theta(\mathcal{Z} \mid \mathcal{X})$, which is non-negative: $D_{KL}\left(q_\phi(\mathcal{Z} \mid \mathcal{X})\|p_\theta(\mathcal{Z} \mid \mathcal{X})\right) \geq 0$. The KL-divergence is 0 if, and only if, $q_\phi(\mathcal{Z} \mid \mathcal{X})$ equals the true posterior distribution.

The first term on the right-hand side is the variational lower bound or the ELBO:

$$\mathcal{L}_{\theta,\phi}(\mathcal{X}) = \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\log p_\theta(\mathcal{X},\mathcal{Z}) - \log q_\phi(\mathcal{Z}\mid\mathcal{X})\right]$$

Due to the non-negativity of the KL divergence, the ELBO is a lower bound on the log-likelihood of the data.

$$\begin{aligned}
\mathcal{L}_{\theta,\phi}(\mathcal{X}) &= \log p_\theta(\mathcal{X}) - D_{KL}\left(q_\phi(\mathcal{Z}\mid\mathcal{X})\|p_\theta(\mathcal{Z}\mid\mathcal{X})\right) \\
&\leq \log p_\theta(\mathcal{X})
\end{aligned}$$

So, interestingly, the KL divergence determines both the defined divergence of the approximate posterior from the true posterior and the gap between the ELBO and the marginal (log) likelihood, known as the *tightness of the bound*. Returning to the problem set up in 2: we began seeking to learn the model generating our data, $p_\theta(\mathcal{X}) = \int p_\theta(\mathcal{X} \mid \mathcal{Z})p_\theta(\mathcal{Z})\, d\mathcal{Z}$ which is maximized by maximizing $\log p_\theta(\mathcal{X})$. VAE arranges a compromise, where under some error $D_{KL}(q_\phi(\mathcal{Z} \mid \mathcal{X}) \,||\, p_\theta(\mathcal{Z} \mid \mathcal{X}))$, we estimate a lower bound on $\log p_\theta(\mathcal{X})$ which is, in practice, "good enough" given intractability.

When both the prior and the approximate posterior are Gaussian, the KL-divergence can be computed analytically, and the ELBO can be maximized using gradient descent:

$$\mathcal{L}\left(\theta,\phi;\vec{x}_i\right) \approx \frac{1}{2}\sum_{j=1}^{k}\left(1+\log\left(\left(\vec{\sigma}_j^{2(i)}\right)^2\right)-\left(\vec{\mu}_j^{(i)}\right)^2-\left(\vec{\sigma}_j^{2(i)}\right)^2\right)+\frac{1}{L}\sum_{l=1}^{L}\log p_\theta\left(\vec{x}_i \mid \vec{z}_i^{(l)}\right)$$

where we reparametrize (discussed later in 2.4)

$$\vec{z}_i^{(l)} = \vec{\mu}_i + \vec{\sigma}_i^2 \odot \boldsymbol{\epsilon}^{(l)} \quad \text{and} \quad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbb{I}_k)$$

2.4 Stochastic Gradient Descent

Coming out of the encoder, we have estimates of $q_\phi(\mathcal{Z} \mid \mathcal{X})$ that maps our data $\mathcal{X}$ to latent space and we have $\mathcal{Z}$ the latent variable. From the decoder, we have $p_\theta(\mathcal{X} \mid \mathcal{Z})$ that maps our latent variable $\mathcal{Z}$ to the data

space. This is familiar to those who have used autoencoders: we have an encoder network q_ϕ, the encoded representation $\mathcal{Z}$, and the decoder network p_θ.

From the discussion above, we see that maximizing the ELBO $\mathcal{L}_{\theta,\phi}(\mathcal{X})$ will approximately maximize the marginal likelihood $\log p_\theta(\mathcal{X})$, making the decoder better, and will minimize the KL-divergence of the approximate posterior from the true posterior, making the encoder better. So, the VAE optimization function is just the ELBO, again: $\mathcal{L}_{\theta,\phi}(\mathcal{X}) = \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\log p_\theta(\mathcal{X},\mathcal{Z}) - \log q_\phi(\mathcal{Z} \mid \mathcal{X})\right]$.

Specifically, the objective is the sum (or equivalently, the mean) ELBO of the training set:

$$\mathcal{L}_{\theta,\phi}(\mathcal{X}) = \sum_{i=1}^{N} \mathcal{L}_{\theta,\phi}(\vec{x}_i)$$

Now, given $\hat{\mathcal{Z}} = g_\phi(\boldsymbol{\epsilon}, \mathcal{X})$ with $\boldsymbol{\epsilon}^{(l)} \sim p(\epsilon)$, we can use Monte Carlo estimation to approximate the expectation of some function f with respect to the approximate posterior $q_\phi(\mathcal{Z} \mid \mathcal{X})$:

$$\begin{aligned}\mathbb{E}_{q_\phi(\mathcal{Z}|\vec{x}_i)}[f(\mathcal{Z})] &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f\left(g_\phi\left(\boldsymbol{\epsilon}, \vec{x}_i\right)\right)\right] \\ &\approx \frac{1}{L}\sum_{l=1}^{L} f\left(g_\phi\left(\boldsymbol{\epsilon}^{(l)}, \vec{x}_i\right)\right)\end{aligned}$$

Applying this, we can approximate the ELBO:

$$\widetilde{\mathcal{L}}^A(\theta, \phi; \vec{x}_i) = \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\vec{x}_i, \vec{z}_i^{(l)}) - \log q_\phi(\vec{z}_i^{(l)} \mid \vec{x}_i)$$

Unbiased gradients of the ELBO with respect to θ can be simply calculated:

$$\begin{aligned}\nabla_\theta \mathcal{L}_{\theta,\phi}(\mathcal{X}) &= \nabla_\theta \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\log p_\theta(\mathcal{X},\mathcal{Z}) - \log q_\phi(\mathcal{Z} \mid \mathcal{X})\right] \\ &= \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\nabla_\theta\left(\log p_\theta(\mathcal{X},\mathcal{Z}) - \log q_\phi(\mathcal{Z} \mid \mathcal{X})\right)\right] \\ &\approx \nabla_\theta\left(\log p_\theta(\mathcal{X},\mathcal{Z}) - \log q_\phi(\mathcal{Z} \mid \mathcal{X})\right) \qquad \ldots \text{from the Monte Carlo estimation above} \\ &= \nabla_\theta\left(\log p_\theta(\mathcal{X},\mathcal{Z})\right)\end{aligned}$$

However, unbiased gradients of the ELBO with respect to ϕ are not so simple since the expectation is with respect to the approximate posterior $q_\phi(\mathcal{Z} \mid \mathcal{X})$, which is itself a function ϕ, which leads us to the *reparametrization trick* discussed previously at the end of Section 2.2.

$$\begin{aligned}\nabla_\phi \mathcal{L}_{\theta,\phi}(\mathcal{X}) &= \nabla_\phi \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\log p_\theta(\mathcal{X},\mathcal{Z}) - \log q_\phi(\mathcal{Z} \mid \mathcal{X})\right] \\ &\neq \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}\left[\nabla_\phi\left(\log p_\theta(\mathcal{X},\mathcal{Z}) - \log q_\phi(\mathcal{Z} \mid \mathcal{X})\right)\right]\end{aligned}$$

Returning to the trick, we reparametrized our continuous latent random variable, $\hat{\mathcal{Z}} \sim q_\phi(\mathcal{Z} \mid \mathcal{X})$ using a differentiable transformation $g_\phi(\boldsymbol{\epsilon}, \mathcal{X})$ such that $\hat{\mathcal{Z}} = g_\phi(\boldsymbol{\epsilon}, \mathcal{X})$ with $\boldsymbol{\epsilon} \sim p(\epsilon)$. Consequently, given a matrix of noise $\boldsymbol{\epsilon}$, the encoder is a stochastic function of the data $\mathcal{X}$ and noise. The requirement of differentiability presupposes that our model is deterministic—given any input, it returns the same output again for the same set of parameters. To maintain this deterministic nature but arrive at a practical estimator, we inject noise from an auxilliary random variable $\boldsymbol{\epsilon}$. We need the reparameterization in order to backpropagate through a random node, and the "trick" part of it is that randomness is treated as an input to the model rather than something that occurs "naturally" within it, allowing us to evade differentiating with respect to the sampling. In other words, rather than sampling $\mathcal{Z}$ directly from $q_\phi(\mathcal{Z} \mid \mathcal{X})$, we generate Gaussian noise $\boldsymbol{\epsilon}$ from $\mathcal{N}(0, \mathbb{I}_k)$ and compute:

$$\vec{z}_i^{(l)} = \vec{\mu}_i + \vec{\sigma}_i^2 \odot \mathbb{I}_k \quad \forall \;\; i \in \{1, \ldots, k\}$$

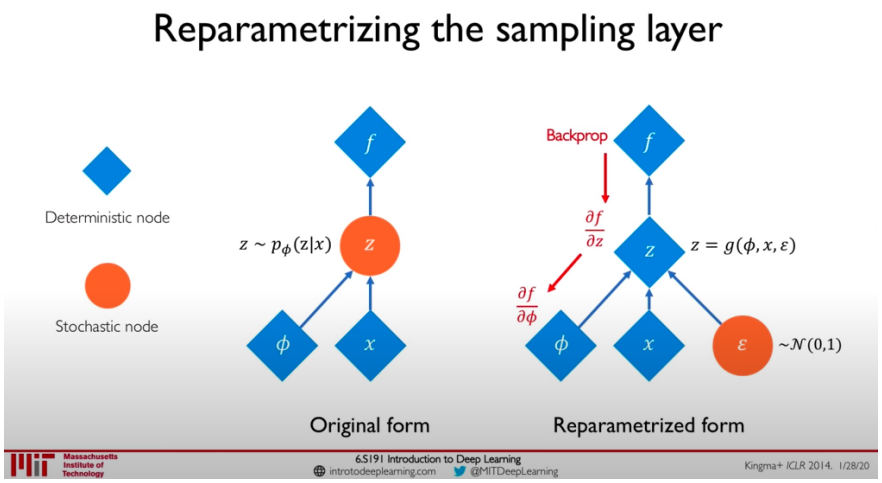

Figure 3: Reparametrization, from deterministic to stochastic function a la Benjaminson (2020)

Thus, we can rewrite the Monte Carlo estimation to be differentiable with respect to ϕ, and iterate the gradient descent algorithm:

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}[f(\mathcal{Z})] &= \nabla_\phi \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(\mathcal{Z})] \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\phi}} f(\mathcal{Z})\right] \\ &\approx \nabla_\phi f(\mathcal{Z})\end{aligned}$$

2.5 Algorithm and Summary

Algorithm 1 Variational Autoencoder

Input: $\mathcal{X} \leftarrow$ Training set
Output: $q_\phi(\mathcal{Z} \mid \mathcal{X}) \leftarrow$ Encoder; $p_\theta(\mathcal{X} \mid \mathcal{Z}) \leftarrow$ Decoder

$\theta, \phi \leftarrow$ Initialize parameters
repeat for all batches $\{b = 1, \ldots, B\}$
  $\mathcal{X}_b \leftarrow$ Sample a minibatch of dimensions $M \times d$ from $\mathcal{X}$
  $\boldsymbol{\epsilon} \leftarrow$ Sample of dimensions $M \times k$ from $p(\boldsymbol{\epsilon})$
  $g \leftarrow \nabla_{\theta,\phi} \mathcal{L}_b(\theta, \phi; \mathcal{X}_b, \boldsymbol{\epsilon})$
  Gradient descent step with step-size α: $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}}{\partial \theta}$ and $\phi \leftarrow \phi - \alpha \frac{\partial \mathcal{L}}{\partial \phi}$
until convergence
return θ, ϕ

VAE synthesizes deep learning and variational Bayes methods. Simply put, given a data matrix with N i.i.d samples in $\mathbb{R}^d$, the encoder learns the generative distribution by producing mean and standard deviation vectors in $\mathbb{R}^k$, sampling from which produces a bottleneck vector $\vec{z}_i$ in $\mathbb{R}^k$. The decoder then produces a reconstructed vector $\hat{x}_i$ in $\mathbb{R}^d$. VAE has an innate *two for one* quality in its optimization function: it improves its generative model by maximizing the marginal likelihood and because the equivalently minimizes KL-divergence in reconstruction loss, it improves in performing dimensionality reduction (or approximation or classification). Further, maximizing the expectation of the marginal likelihood is a maximum likelihood estimation problem, which is a well-studied problem across methods like SVM, logistic regression, and neural networks. (Shiffman, 2016)

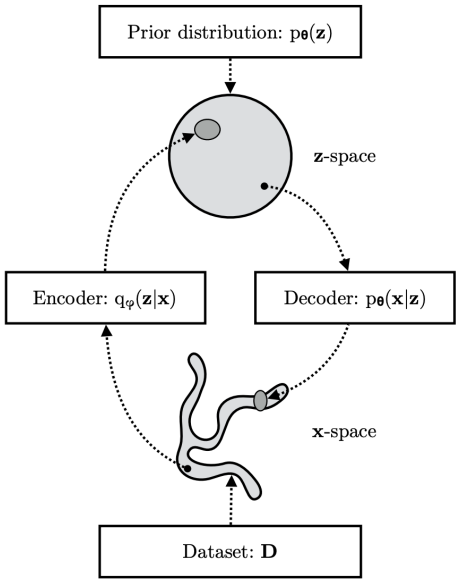

Figure 4: VAE training process from Kingma et al. (2019)

2.6 $\beta-$VAEs or Disentangled VAEs

β-VAE is a type of variational autoencoder that seeks to learn relatively interpretable and factorizable *disentangled latent variables*. (Higgins, Matthey, et al., 2017) It modifies VAEs with an adjustable hyperparameter β that balances latent channel capacity and the reconstruction error. It is a more general form of VAE, where $\beta = 1$ is the standard VAE. Higgins et al. (2017) show that $\beta > 1$ encourages disentangled representations and outperforms VAE. For example, Figure 5 compares models trained on the `celebA` dataset. The grids are the results of manipulating latent variables to interpret the features they contain. The β-VAE below is trained with $\beta = 250$, and the comparison finds that while β-VAE and InfoGAN ed to isolate features like azimuth, emotion, and hairstyle, the standard VAE learned an entangled representation where, for instance, azimuth is confused with emotion, presence of glasses, and gender.

We can easily adapt the VAE algorithm to β-VAE by adding a hyperparameter β to the loss function, and then optimizing the loss function with respect to the parameters θ and ϕ. We can use the Kuhn-Tucker conditions to write this in one line:

$$\mathcal{L}(\theta, \phi, \beta; \mathcal{X}, \boldsymbol{\epsilon}) = \mathbb{E}_{q_\phi(\mathcal{Z}|\mathcal{X})}[\log p_\theta(\mathcal{X} \mid \mathcal{Z})] - \beta \left[D_{KL}(q_\phi(\mathcal{Z} \mid \mathcal{X}) \,||\, p(\mathcal{Z})) - \boldsymbol{\epsilon}\right]$$

where β is the regularization coefficient that constrains the capacity of the latent channel by adjusting the weight on the KL-divergence term, subject to error $\boldsymbol{\epsilon}$. As the weight on the KL-divergence increases, we maximize the marginal likelihood by making the bound tighter. And because of the prior that $p(\mathcal{Z})$ is normally distributed, β puts a constraint on the capacity of the latent channel and places implicit pressure on the learned posterior to consist of independent features.

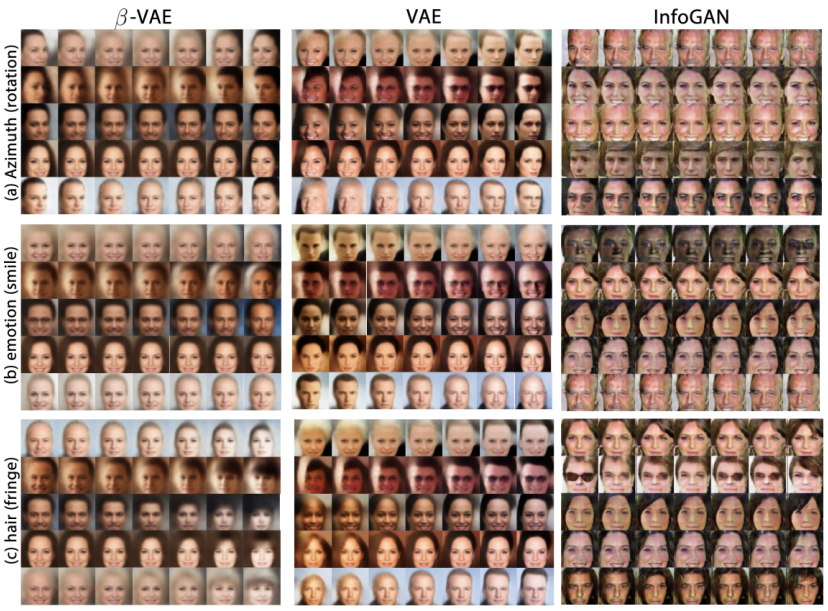

Figure 5: Beta-VAE from Higgins, Matthey, et al. (2017)

2.7 Ongoing Work, GANs, and Zero-Shot RL Transfer

Since its inception, VAEs have been drawn upon including to dynamic models (see Johnson et al., 2016), models with attention (see Gregor et al., 2014), models with multiple levels of stochastic latent variables (see Kingma et al., 2016), and so forth. An interesting early result is Liu and Inkpen (2015), who use stacked *denoising autoencoders* (DAEs) to predict user location from social media posts. VAEs also appears meaningfully supplementary to the active area of work in GANs (as summarized in Goodfellow et al., 2020). An exciting area of work has been hybrid VAE-GANs: the generator is a VAE, and by sampling from the latent space, we can generate real and imagined samples to train the discriminator.

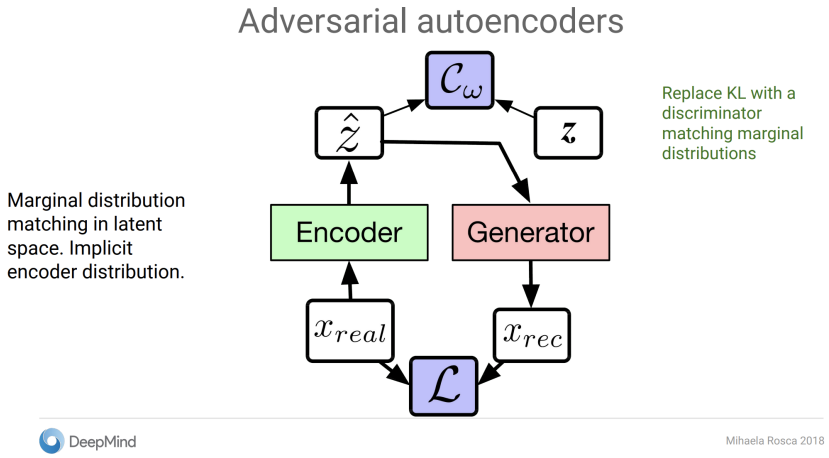

Figure 6: VAE-GAN from Rosca et al. (2018)

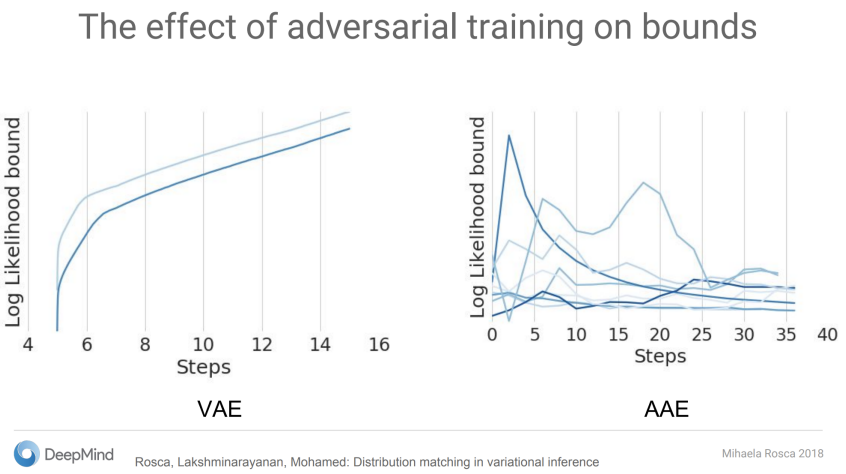

Figure 7: Training loss for VAE vs. VAE-GAN from Rosca et al. (2018)

Another interesting area of ongoing work is in reinforcement learning models with sparse rewards. Higgins et al. (2017b) introduced 'DARLA,' a system for zero-shot domain transfer in reinforcement learning. It attempts to develop agents who learn robust policies in the source domain that generalize well to target domains with minimal finetuning. DARLA learns a disentangled latent space representation that is shared between the source and target domains using β-variational autoencoders. Thus, the agent can "see" the new domain before entering by studying the underlying stochastic process generating the data. Given, a source domain D_S and a target domain D_T. Between the two domains, the reward and transition functions are approximately equal, the set of available policies (or actions) is identical, and the state space is different. DARLA samples random actions in the source domain and observes information. Then, it pretrains a β-VAE on those observations, with a DAE to finetune. Now, the encoder learns a disentangled representation of the world in the source domain, allowing us to put the agent in the target domain with minimal retraining.

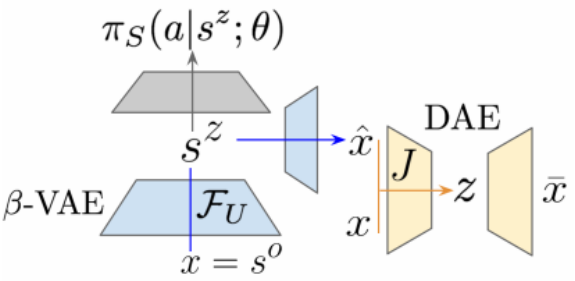

Figure 1. Schematic representation of DARLA. Yellow represents the denoising autoencoder part of the model, blue represents the β-VAE part of the model, and grey represents the policy learning part of the model.

Figure 8: DARLA from Higgins, Pal, et al. (2017b)

3 Discussion

3.1 Motivation: Variational Inference and Regularity

We can motivate VAEs from scratch: to arrive at an encoder-decoder setup, we require a function to *extract* or *select* lower-dimensional features from data, and a function to map (or "project") the encoding back into the data's original dimensionality. A simple place to start is with an interpretable and geometrically-intuitive linear method like PCA, with the approximated (cut-off at lower rank) principle components (or eigenvectors matrix W) modeling the underlying process, the lower-dimensional scores (Y) being the encoded samples, and a decoder process of matrix multiplications with the score and the eigenmatrix to return reconstructed vectors ($\hat{x}_i$) (from the equation $\vec{y}_i = WW^T\vec{x}_i$). But this is visibly insufficient: besides the lack of non-linearity, there might exist a different basis that is lower-dimensional that PCA cannot find when conducting a change of basis to an orthonormal eigenbasis.

This leads to the Autoencoder, with neural networks modeling the encoding and decoding through iterative optimization of loss between the *reconstruction* (encoded-decoded data) and the original data. A linear (and thus single-layer) autoencoder might already outperform PCA by learning a "truer"—that is, lower dimension—linear subspace that the data lives on. And this scales up when we design a non-linear encoder-decoder setup with greater depth and complexity in terms of layers. However, autoencoders struggle with overfitting and often do not produce latent spaces with the "good properties" that allow us to use the de-dimensioned data for inference, generation, compression, classification, or other downstream functions. (Steck, 2020)

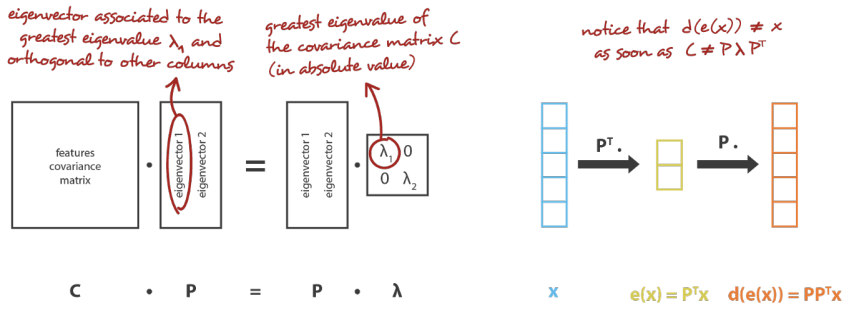

Figure 9: PCA as an encoder-decoder setup a la Rocca (2019)

Ideally, we want two properties of regularity in the latent space of our encoder: first, a notion of *continuity*, where two close points in the latent space have close decoded reconstructions; second, a notion of *completeness* wherein each sampled point from the latent distribution should produce some meaningful output when decoded. (Barrett et al., 2022; Perl et al., 2020) These properties would create a sort-of manifold over the information encoded in the latent space. However, when we build deeper autoencoders, the encoder becomes "infinitely" able to reduce dimensionality. For instance, the encoder could flatten the entire data onto the real number line as the decoder learns to rebuild the data from just a point in R (perhaps even notoriously memorizing the training set and decoding its "indices"). But even if we force the autoencoder to return a distribution of points rather than a single point, without a well-defined regularization term, the model can "learn to cheat" by effectively ignoring the returning-distributions part of the optimand. That is, a generic autoencoder returning distributions can return ones that are *punctual*—having tiny variance—or return distributions with very different means, defeating the notion of closeness in the latent space (and essentially still producing a punctual distribution). This motivates the architecture of VAEs where we reward learning distributions close to the standard normal, thus producing distributions with regular covariance matrices and means while ensuring sufficient variance to yield a smooth latent space. Thus, we both construct a practical

estimator by making Gaussian parametric assumptions whereas estimating the true posterior distribution is intractable and we build an estimator that is locally and globally regularized, allowing the encodings to be more *interpretable*. Simply put, if we add a new data point between the means of two encoded distributions generated from different training sets, the new point should be somewhere between the first and second datasets as they both must have seen this point somewhere in their shared latent spaces. A continuous and complete distribution ensures that any sampled point in the latent space has "meaning", allowing us to use the decoder like a *generator* in adversarial models like *generative adversarial networks* (GANs): by sampling a random point in the latent space and decoding it, the decoder becomes generative. The illustrations below show the difference between a simple autoencoder and a VAE, with the latter having a regularized latent space that allows for a more meaningful encoder empowered with variational Bayes inference.

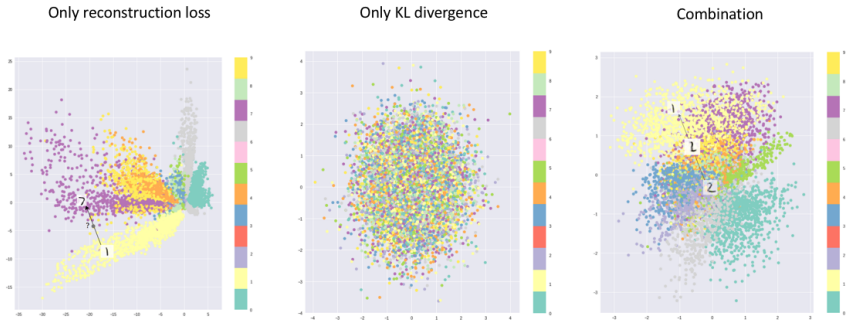

Figure 10: Model encodings a la Jordan (2018). L-R: simple autoencoder, KL-divergence model, VAE.

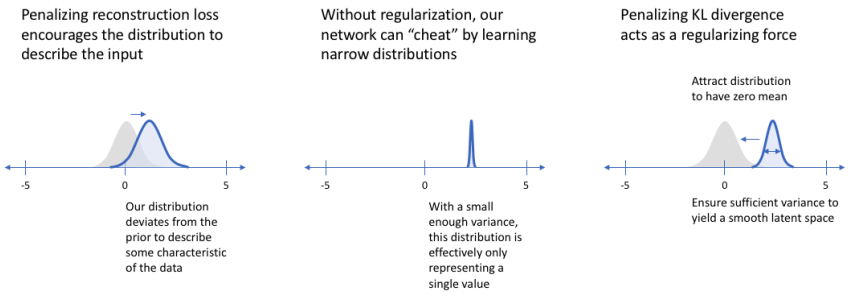

Figure 11: Disincentivizing "cheating" a la Jordan (2018)

3.2 Strengths

Importantly, VAEs do not make the common simplifying assumptions about the marginal or posterior probabilities. Further, the gradient estimated in VAE is an unbiased estimator of the exact single-data-point ELBO gradient; when averaged over noise $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ this gradient equals the single-data-point ELBO gradient. So, VAE makes fairly minimal parametric assumptions about its PDFs and their differentiability. Conversely thus, VAEs build a generalized algorithm that works efficiently even when facing intractable marginal likelihoods, large datasets, and high-dimensional data. Particularly, in the event where we have so much data that batch optimization is too costly, we want to use small minibatches to train epochs. Sampling-based solutions to estimating the marginal likelihood are not efficient, and the EM algorithm is not applicable when the posterior is intractable. Similarly, the Monte Carlo EM algorithm is too slow for large datasets. Thus, VAEs can be efficient and scalable in their setting.

An important benefit with VAEs, especially compared to other powerful generative models such as BERT-like transformers, is their interpretability, and this is especially so in the case of β-VAE's disentangled encoded representations. For one, there is elegant mathematical intuition underpinning the generation in VAEs, and this gives us the confidence that we have learned a lower-dimensional process that generates the data. Although θ and ϕ are unobserved in the blackbox, we arrive at a latent space that represents the underlying distribution. And by exploring and manipulating the latent subspace of a particular example, sampling and decoding new points from it, we can easily interpret what features that latent subspace controls. And to get here, we make fairly minimal and robust parametric assumptions that are common to a wide variety of model classes in machine learning. This is in contrast to the blackbox of transformers, where we have no idea what the model is doing, and we have to rely on the model's ability to generate text to interpret its behavior. Granted, transformer models can be studied by viewing individual *atoms* and their activations, but VAEs appear to offer a more systematic apparatus for interpretation.

3.3 Weaknesses and Guidelines

Method	Train Speed	Sample Speed	Num. Params.	Resolution Scaling	Free-form Jacobian	Exact Density	FID	NLL (in BPD)
Generative Adversarial Networks								
DCGAN [182]	★★★★★	★★★★★	★★★★★	★★★☆☆	✓	✗	37.11	-
ProGAN [114]	★★★★☆	★★★★★	★★★★☆	★★★★★	✓	✗	15.52	-
BigGAN [19]	★★★★☆	★★★★★	★★★★★	★★★★☆	✓	✗	14.73	-
StyleGAN2 + ADA [115]	★☆☆☆☆	★★★★★	★★★☆☆	★★★★★	✓	✗	2.42	-
Energy Based Models								
IGEBM [46]	★★★☆☆	★★☆☆☆	★★★★★	★★★☆☆	✓	✗	37.9	-
Denoising Diffusion [87]	★★★★★	★★☆☆☆	★★★☆☆	★★★★☆	✓	(✓)	3.17	$\leq$ 3.75
DDPM++ Continuous [206]	★★★★★	★★☆☆☆	★★★☆☆	★★★★☆	✓	(✓)	2.20	-
Flow Contrastive (EBM) [55]	★☆☆☆☆	★★☆☆☆	★★☆☆☆	★★☆☆☆	✓	✗	37.30	$\approx$ 3.27
VAEBM [247]	★★☆☆☆	★★★★☆	★☆☆☆☆	★★★★☆	✓	✗	12.19	-
Variational Autoencoders								
Convolutional VAE [123]	★★★★★	★★★★★	★★★★★	★★☆☆☆	✓	(✓)	106.37	$\leq$ 4.54
Variational Lossy AE [29]	★★★☆☆	★☆☆☆☆	★☆☆☆☆	★★☆☆☆	✗	(✓)	-	$\leq$ 2.95
VQ-VAE [184], [235]	★★☆☆☆	★★★☆☆	★☆☆☆☆	★★★★★	✗	(✓)	-	$\leq$ 4.67
VD-VAE [31]	★☆☆☆☆	★★★★★	★★★☆☆	★★★★★	✓	(✓)	-	$\leq$ 2.87
Autoregressive Models								
PixelRNN [234]	★★☆☆☆	★☆☆☆☆	★☆☆☆☆	★★★☆☆	✗	✓	-	3.00
Gated PixelCNN [233]	★★★☆☆	★☆☆☆☆	★☆☆☆☆	★★★☆☆	✗	✓	65.93	3.03
PixelIQN [173]	★★★☆☆	★☆☆☆☆	★☆☆☆☆	★★★☆☆	✗	✓	49.46	-
Sparse Trans. + DistAug [32], [110]	★★★★☆	★☆☆☆☆	★★★☆☆	★★★☆☆	✗	✓	14.74	2.66
Normalizing Flows								
RealNVP [43]	★★☆☆☆	★★★★★	★★★★★	★★★☆☆	✗	✓	-	3.49
GLOW [124]	★☆☆☆☆	★★★★★	★★☆☆☆	★★★★☆	✗	✓	45.99	3.35
FFJORD [62]	★☆☆☆☆	★★★☆☆	★★★★★	★★☆☆☆	✓	(✓)	-	3.40
Residual Flow [26]	★★☆☆☆	★★★★☆	★★★★☆	★★★★☆	✓	(✓)	46.37	3.28

Figure 12: Comparison with other deep generative models a la Bond-Taylor et al. (2021)

In the early 2010s, the most exciting methods in generative modeling were often VAEs, GANs, and autoregressive models (AR). Unfortunately, VAEs have fallen well out of the state-of-the-art in generative modeling since, especially since the emergence of BERT-like transformer models and diffusion models. It appears part of the anecdotal experience that the elegant mathematical intuition underpinning VAEs did not translate into empirical success over time (for example, see Shu (2022) for an OpenAI engineering's account of the seeming mathematical promise of VAEs against diffusion models at the time). For instance, in the `CIFAR-10` benchmark, we see VAEs outperformed by GANs and diffusion models among other classes of models in reducing *FID* or the Fréchet Inception Distance, a metric for measuring the distance between two distributions of images. This is not to say that VAEs are not useful: rather, VAEs are still useful in the context of disentangled representations, and they are still useful in the context of learning a latent space that is interpretable and can be manipulated. However, they are not among the most powerful generative models in the current state-of-the-art. Similarly, in the `celebA` benchmark they are outperformed by Diffusion Style-GANs, INDM, Soft diffusion, and other diffusion models, and NCP-VAE is ranked ninth with a FID roughly 4-times that of the benchmark leader.

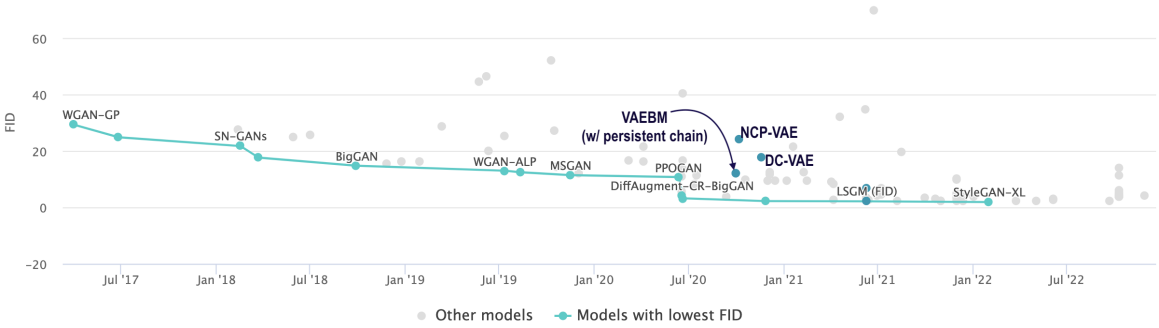

Figure 13: **Benchmark:** Image generation on CIFAR-10, adapted from PapersWithCode

Kingma et al. (2019) disclaim that they find, consistent with results found by others, that the encoder-decoder networks often fail to converge at a desirable stable equilibrium. This is owing to the relatively weak marginal likelihood term initialized at the start of training, $\log p_\theta(\mathcal{Z})$. So, an initially attractive state is where $q_\phi(\mathcal{Z} \mid \mathcal{X}) \approx p_\theta(\mathcal{Z})$, resulting in an undesirable stable equilibrium that it can be heard for the stochastic descent algorithm to escape. Yacoby et al. (2020) point out two prominent pathologies with VAEs backed up by empirical results. Their first theorem shows that VAE trades off generative model quality in return for learning unimportant or simple posteriors. Or formally, that the VAE learns choices of likelihood functions that reconstruction $p_\theta(\mathcal{X})$ poorly, even when learning the ground truth is possible. Their second theorem shows that the ELBO optimand biases the learning of the observation noise variance. That is, the variance $\vec{\sigma}_i^2$ that maximizes the ELBO depends on the approximate posterior $q_\phi(\mathcal{Z} \mid \mathcal{X})$ and thus even if the ground truth θ_{GT} is known and we set $\theta = \theta_G T$, the learned vector $\hat{\sigma}_i^2$ may not equal ground truth $\vec{\sigma}_i^2$ if $q_\phi(\mathcal{Z} \mid \mathcal{X})$ is not the true posterior.

These issues are particularly concerning given the freedom in the *choice of* k, the dimensionality of the latent space. Presumably, we choose k that maximizes the log-likelihood of the data $\log p_\theta(\mathcal{X})$. But, increasing k alleviates the otherwise existent need to weaken the decoder in order to improve the inference model and leads to better approximation of $p(\mathcal{X})$. Thus, the ELBO will favor model mismatch (k larger than the ground truth) and prevent us from learning highly compressed representations when they are available.

Another critical downstream consequence of these pathologies is when there are distinct cohorts in our sample. VAE forces all the cohorts to share the same latent manifold, even though different cohorts might have different distributions (densities) for their characteristics. VAEs also struggle with tasks that require denoising, and DAEs might be more appropriate for such tasks. In a way, VAEs trade-off between generating realistic data and realistic counterfactuals. And they must be trained with careful attention to the dimensionality of the latent space and the depth of the encoder-decoder networks to balance regularity with performance in terms of minimizing reconstruction loss (or equivalently, pushing it toward maximizing the information held in the encoded features).

4 Examples

4.1 Implementation: VAE for Image Generation

We implement variational autoencoders on the `CiFAR-10` dataset, explore its latent space, and generate output. CIFAR-10 is a dataset of ~60,000 32×32 color images in 10 classes, with 6,000 images per class. The 10 classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is commonly used for object recognition and computer vision research. CIFAR-10 has been widely used for

benchmarking various machine learning algorithms. Below is a random sample of 25 images from the dataset.

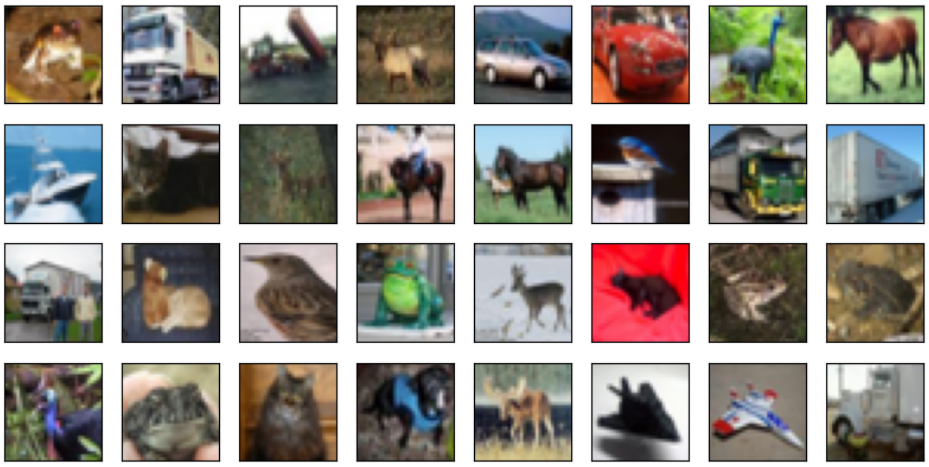

Figure 14: **CIFAR-10:** Random sample of 20

In our case, each sample is a 32×32 image, or a corresponding matrix of pixel densities between 0 and 255, in each of 3 RGB channels, i.e., a sample of our data is a `(32, 32, 3)` array, which can be thought of as a sample vector in $\mathbb{R}^{32 \cdot 32 \cdot 3}$ or $\mathbb{R}^{3072}$. Our training set has 1,563 batches of 32 images each, so we have a data matrix, inheriting previous notation, $\mathcal{X} \in \mathbb{R}^{50016 \times 3072}$. We use a convolutional neural network (CNN) to encode the image into a latent space of dimension $k = 256$, with one hidden layer. We then use a decoder, with a hidden layer of similar but reversed input-output dimensions, to reconstruct the image from the latent space. We use the *Adam optimizer* with a learning rate of 0.001, and the *GeLu function* for non-linearity. We train the model for 100 epochs. The model is implemented in `PyTorch`. The model was trained on a cloud-hosted GPU on Vast AI, using one `RTX 4090` at 101.7 *TFLOPS*, and it took approximately 13 minutes and 9 seconds to train. The model is summarized in Table 1.

Layer (type)	Output Shape	Param #
Linear-1	[-1, 512]	1,573,376
Linear-2	[-1, 256]	131,328
Linear-3	[-1, 256]	131,328
Encoder-4	[-1, 256]	0
Linear-5	[-1, 512]	131,584
Linear-6	[-1, 3072]	1,575,936
Decoder-7	[-1, 3, 32, 32]	0
Total params		3,543,552
Trainable params		3,543,552
Non-trainable params		0
Input size (MB)		0.01
Forward/backward pass size (MB)		0.06
Params size (MB)		13.52
Estimated Total Size (MB)		13.59

Table 1: VAE example for CiFAR-10

4.2 Evaluation

Model evaluation follows: Figure 15 shows the training loss and 16 shows the structural similarity index (SSIM) across epochs. SSIM, or structural similarity index, is a metric that can be used to measure the similarity between two images. In the context of a variational autoencoder, SSIM can be used to measure the similarity between the original input image and the output image produced by the autoencoder. The SSIM value can range from 0 to 1, with a value of 1 indicating that the two images are exactly the same and a value of 0 indicating that they are completely different. The SSIM could also be used as a loss function during training of the autoencoder to encourage the output image to be as similar as possible to the input image. Per the discussion of weaknesses in SGD convergence highlighted by Kingma et al. (2019), we can see that the VAE can get stuck in low-quality local maxima in the training process. This is a product of the intractability of the ELBO, and the fact that the KL divergence is not convex. That is, the variance $\vec{\sigma}_i^2$ that maximizes the ELBO depends on the approximate posterior $q_\phi(\mathcal{Z} \mid \mathcal{X})$ and thus even if the ground truth θ_{GT} is known and we set $\theta = \theta_G T$, the learned vector $\hat{\sigma}_i^2$ may not equal ground truth $\vec{\sigma}_i^2$ if $q_\phi(\mathcal{Z} \mid \mathcal{X})$ is not the true posterior. The SSIM converges at a reasonably high ~0.8 and the SSE plateaus after the first third of epochs.

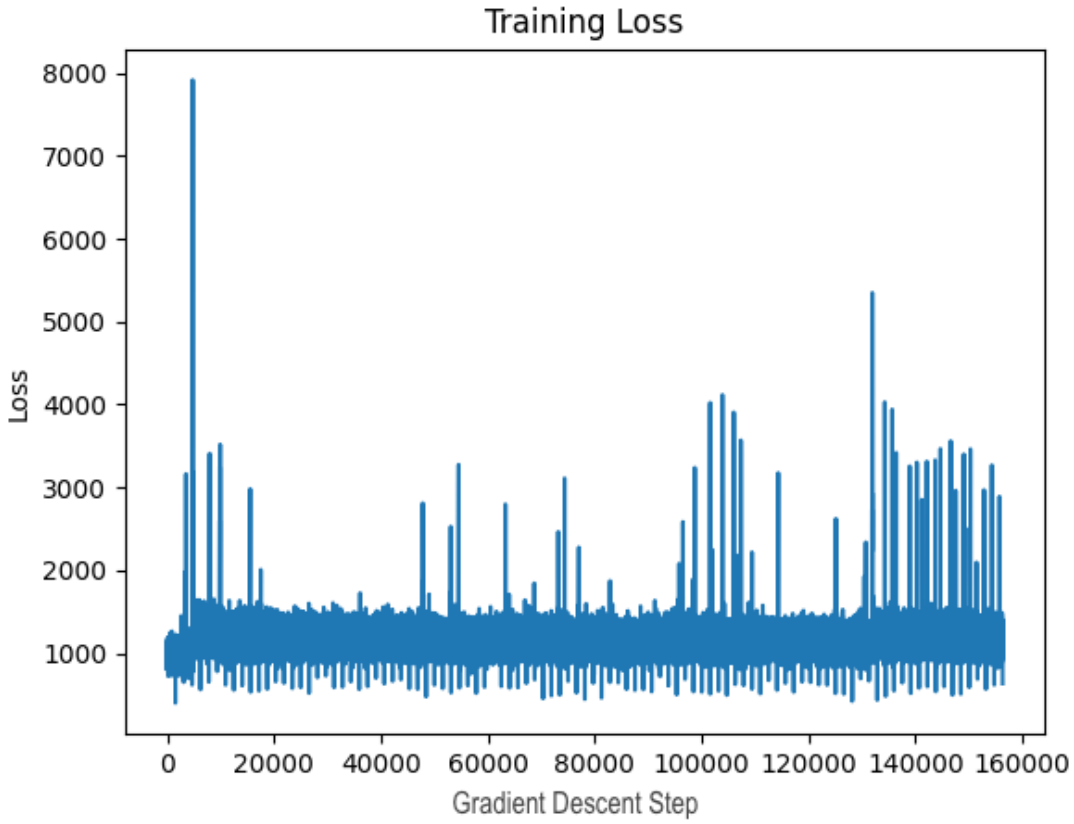

Figure 15: **Training Loss (SSE):** Volatile and plateaus early

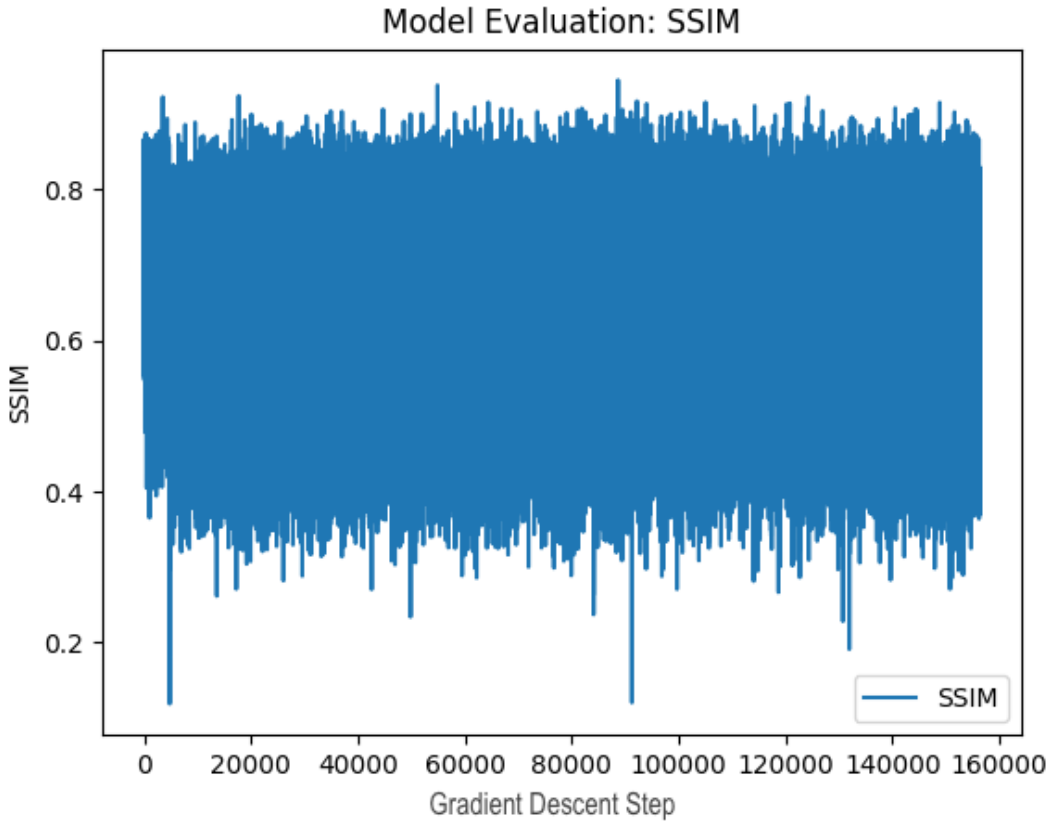

Figure 16: **Training Structural Similarity (SSIM):** The generative model or the posterior finds many undesirable equilibria

When we view the reconstructed data, we find that the model does a pretty good job of recovering the original information. To be fair, these are very low-resolution 32×32 images to begin with, which allows VAEs to perform relatively decently. Still, its reconstructions are significantly blurrier and can have noisy or meaningless patches. The reconstructions of some random draws from the dataset via VAE are shown in Figure 17.

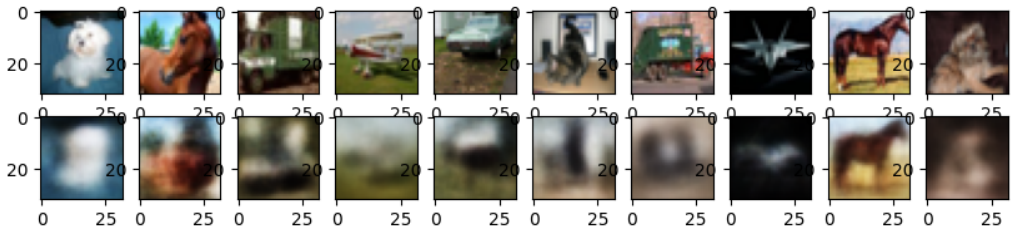

Figure 17: **Reconstructed Data:** With original sample side-by-side

4.2.1 Latent Space

The latent space is visualized by projecting it down to 2D via clustering. We see that the 9 cluster labels ("cat", "boat", etc.) are quite mixed given that the pictures come from open outdoor settings and therefore share distributional characteristics. However, this is also a strong visualization of the pathologies identified above in Section 3.3. Where there are separate cohorts in our dataset, and likely different densities and distributions that generate them, VAE forces them to share the same latent manifold. Here, projected to 2D, we see an ellipse, whose surface is populated fairly uniformly with data belonging to various clusters. So, as shown in Figure 18, multiple cohorts are mapped onto the same latent manifold.

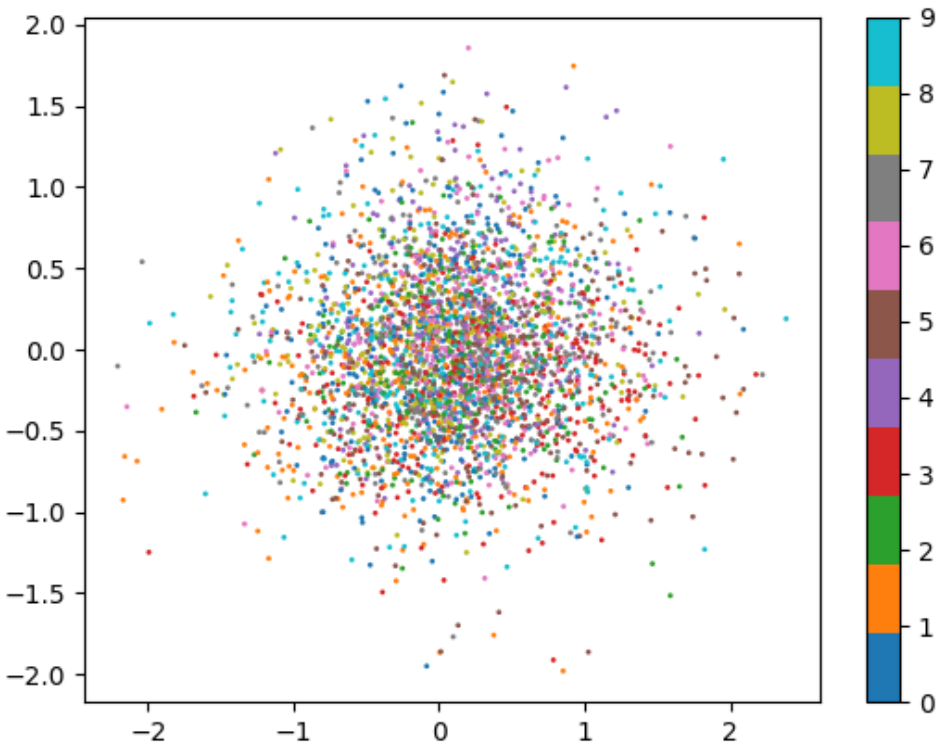

Figure 18: **Latent Space:** Cohorts like cat and boat are forced to share the same latent manifold

4.3 Generating Images from the Latent Space

When we sample completely random points from a normal distribution of the latent dimensionality and decode them, we yield strange images that look similar to the PCA "eigenfaces." There are negative- or colorful-looking patches, curves, and edges, presumably capturing some feature in the generative process. This is shown in Figure 19.

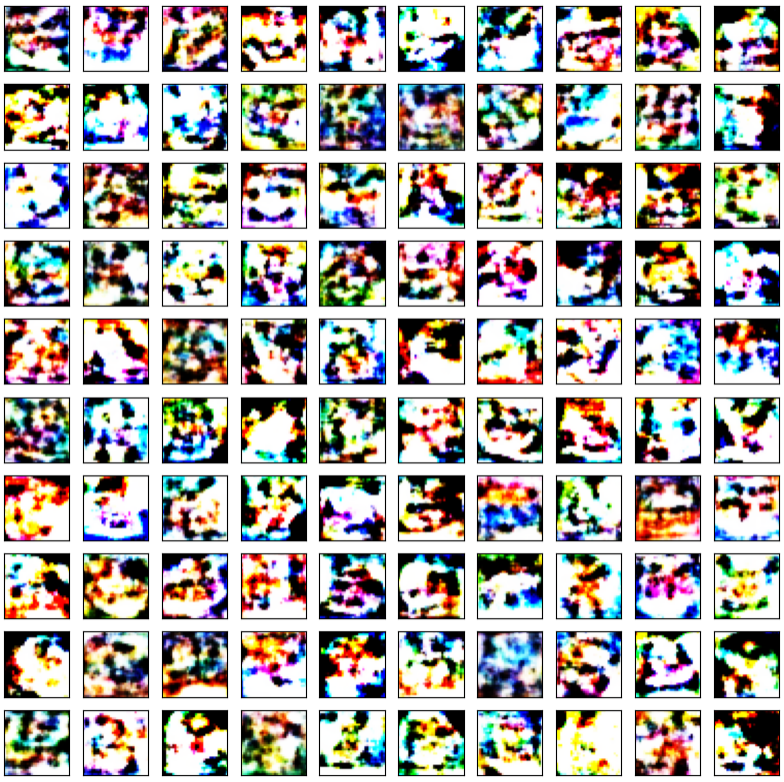

Figure 19: **Latent Space Exploration:** Decoding random points

However, when we constrain the sampled encoding points to be in the neighborhood of the encodings of real data, by adding noise to the real encodings, we get much more realistic images. This is shown in Figure 20. We achieve this by adding small Gaussian noise to the encodings of real data, in `torch` this looks like:

```
# x is a sample from the dataset
z = autoencoder.encoder(x)
# add noise
z = z + torch.randn_like(z)*0.1
x_hat = autoencoder.decoder(z)
```

Figure 20: **Latent Space Exploration:** Decoding random points from the neighborhood of real data

4.4 Code

We define the following model classes using `torch` classes. First, the encoder:

```
class Encoder(nn.Module):
    def __init__(self, latent_dims):
        super().__init__()
        # for a 32x32 image we rewrite this to 1024
        self.linear1 = nn.Linear(3072, 512)
        self.linear2 = nn.Linear(512, latent_dims)
        self.linear3 = nn.Linear(512, latent_dims)
        self.N = torch.distributions.Normal(0, 1)
        self.kl = 0

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = F.gelu(self.linear1(x))
        mu =  self.linear2(x)
        sigma = torch.exp(self.linear3(x))
        z = mu + sigma*self.N.sample(mu.shape)
        self.kl = (sigma**2 + mu**2 - torch.log(sigma) - 1/2).sum()
        return z
```

Second, the decoder:

```
class Decoder(nn.Module):
    def __init__(self, latent_dims):
        super().__init__()
        self.linear1 = nn.Linear(latent_dims, 512)
```

```
        self.linear2 = nn.Linear(512, 3072)

    def forward(self, z):
        z = F.gelu(self.linear1(z))
        z = torch.sigmoid(self.linear2(z))
        return z.reshape((-1, 3, 32, 32))
```

Now composing them together into the VAE:

```
class Autoencoder(nn.Module):
    def __init__(self, latent_dims):
        super().__init__()
        self.encoder = Encoder(latent_dims)
        self.decoder = Decoder(latent_dims)

    def forward(self, x):
        z = self.encoder(x)
        return self.decoder(z)
```

And we train this model as typical for deep learning in torch. We also compute SSIM for each epoch, given data x and labels y: `ssim_values.append(structural_similarity(x[0].cpu().detach().numpy().transpose(1, 2, 0), x_hat[0].cpu().detach().numpy().transpose(1, 2, 0), multichannel=True))`.

The functions to plot latent space is adapted from the MNIST (handwritten digits) VAE tutorial in Alexander de Kleut's blog post. To visualize the latent space:

```
def plot_latent(autoencoder, data, num_batches=100):
    for i, (x, y) in enumerate(data):
        z = autoencoder.encoder(x.to(device))
        z = z.cpu().detach().numpy()
        plt.scatter(z[:, 0], z[:, 1], c=y, s=1, cmap='tab10')
        if i == num_batches:
            break
    plt.colorbar()
```

Second, to explore the latent space, I wrote this function:

```
def latent_space_samples(autoencoder, num_samples=100):
    z = torch.randn(num_samples, latent_dims)
    x_hat = autoencoder.decoder(z.to(device))
    x_hat = x_hat.cpu().detach().numpy()
    fig, ax = plt.subplots(10, 10, figsize=(10, 10))
    for i, axi in enumerate(ax.flat):
        axi.imshow(x_hat[i].transpose(1, 2, 0))
        axi.set(xticks=[], yticks=[])
    plt.savefig(...)
```

Further, the following two functions generate images. The first samples the proximate latent space of real samples and the second shows the exact reconstruction of the real samples:

```
fig, ax = plt.subplots(4, 8, figsize=(10, 5))
for i, axi in enumerate(ax.flat):
    x = data.dataset[i][0].unsqueeze(0).to(device)
    z = autoencoder.encoder(x)
```

```
  # add noise
  z = z + torch.randn_like(z)*0.1
  x_hat = autoencoder.decoder(z)
    axi.imshow(x_hat[0].cpu().detach().numpy().transpose(1, 2, 0))
    axi.set(xticks=[], yticks=[])
plt.show()

def plot_sample():
    with torch.no_grad():
        sample = next(iter(data))
        sample = sample[0].to(device)
        output = autoencoder(sample)
        output = output.cpu()
        sample = sample.cpu()
        fig, ax = plt.subplots(2, 10, figsize=(10, 2))
        for i in range(10):
            ax[0][i].imshow(sample[i].permute(1, 2, 0))
            ax[1][i].imshow(output[i].permute(1, 2, 0))
        plt.show()
```

5 References

Barrett, B., Camuto, A., Willetts, M., & Rainforth, T. (2022). Certifiably robust variational autoencoders. *International Conference on Artificial Intelligence and Statistics*, 3663–3683.

Benjaminson, E. (2020). *The Reparameterization Trick*. https://sassafras13.github.io/ReparamTrick/

Bond-Taylor, S., Leach, A., Long, Y., & Willcocks, C. G. (2021). Deep generative modelling: A comparative review of VAEs, GANs, normalizing flows, energy-based and autoregressive models. *arXiv Preprint arXiv:2103.04922*.

Chen, T. Q., Li, X., Grosse, R. B., & Duvenaud, D. K. (2018). Isolating sources of disentanglement in variational autoencoders. *Neural Information Processing Systems*.

Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. *arXiv Preprint arXiv:1605.08803*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144.

Gregor, K., Danihelka, I., Mnih, A., Blundell, C., & Wierstra, D. (2014). Deep autoregressive networks. *International Conference on Machine Learning*, 1242–1250.

Higgins, I., Matthey, L., Glorot, X., Pal, A., Uria, B., Blundell, C., Mohamed, S., & Lerchner, A. (2016). *Early visual concept learning with unsupervised deep learning*. arXiv. https://doi.org/10.48550/ARXIV.1606.05579

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., & Lerchner, A. (2017). Beta-VAE: Learning basic visual concepts with a constrained variational framework. *International Conference on Learning Representations*. https://openreview.net/forum?id=Sy2fzU9gl

Higgins, I., Pal, A., Rusu, A. A., Matthey, L., Burgess, C. P., Pritzel, A., Botvinick, M. M., Blundell, C., & Lerchner, A. (2017a). DARLA: Improving zero-shot transfer in reinforcement learning. *International Conference on Machine Learning*.

Higgins, I., Pal, A., Rusu, A. A., Matthey, L., Burgess, C. P., Pritzel, A., Botvinick, M., Blundell, C., & Lerchner, A. (2017b). *DARLA: Improving zero-shot transfer in reinforcement learning*. arXiv. https://doi.org/10.48550/ARXIV.1707.08475

Johnson, M. J., Duvenaud, D. K., Wiltschko, A., Adams, R. P., & Datta, S. R. (2016). Composing graphical models with neural networks for structured representations and fast inference. *Advances in Neural Information Processing Systems*, *29*.

Jordan, J. (2018). Introduction to autoencoders. *Jeremy Jordan, Mar*.

Khemakhem, I., Kingma, D. P., & Hyvärinen, A. (2020). Variational autoencoders and nonlinear ICA: A unifying framework. *ArXiv*, *abs/1907.04809*.

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *Advances in Neural Information Processing Systems*, *31*.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, *29*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv Preprint arXiv:1312.6114*.

Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. *ArXiv*, *abs/1906.02691*.

Kingma, D. P., Welling, M., et al. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), 307–392.

Kristiadi, A. (2020). Variational autoencoder: Intuition and implementation. *Agustinus Kristiadi's Blog*.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86.

Li, J., Kang, D., Pei, W., Zhe, X., Zhang, Y., He, Z., & Bao, L. (2021). Audio2Gestures: Generating diverse gestures from speech audio with conditional variational autoencoders. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 11273–11282.

Liang, D., Krishnan, R. G., Hoffman, M. D., & Jebara, T. (2018). Variational autoencoders for collaborative filtering. *Proceedings of the 2018 World Wide Web Conference*.

Liu, J., & Inkpen, D. (2015). Estimating user location in social media with stacked denoising auto-encoders. *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, 201–210. https://doi.org/10.3115/v1/W15-1527

Papamakarios, G., Nalisnick, E. T., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *J. Mach. Learn. Res.*, *22*(57), 1–64.

Perl, Y. S., Bocaccio, H., Pérez-Ipiña, I., Zamberlán, F., Piccinini, J., Laufs, H., Kringelbach, M., Deco, G., & Tagliazucchi, E. (2020). Generative embeddings of brain collective dynamics using variational autoencoders. *Physical Review Letters*, *125*(23), 238101.

Rocca, J. (2019). Understanding variational autoencoders (VAEs). *Data Science*.

Rosca, M., Lakshminarayanan, B., & Mohamed, S. (2018). *Distribution matching in variational inference*. arXiv. https://doi.org/10.48550/ARXIV.1802.06847

Schönfeld, E., Ebrahimi, S., Sinha, S., Darrell, T., & Akata, Z. (2019). Generalized zero- and few-shot learning via aligned variational autoencoders. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8239–8247.

Shiffman, M. (2016). *Under the hood of the variational autoencoder (in prose and code)*. Fast Forward Labs.

Shu, R. (2022). Building on top of black magic. In *Building on Top of Black Magic | Rui Shu*. https://ruishu.io/2022/10/14/building-on-top-of-black-magic/

Steck, H. (2020). Autoencoders that don't overfit towards the identity. *Advances in Neural Information Processing Systems*, *33*, 19598–19608.

Wu, B., Nair, S., Martín-Martín, R., Fei-Fei, L., & Finn, C. (2021). Greedy hierarchical variational autoencoders for large-scale video prediction. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2318–2328.

Yacoby, Y., Pan, W., & Doshi-Velez, F. (2020). Failure modes of variational autoencoders and their effects on downstream tasks. *arXiv Preprint arXiv:2007.07124*.