Variational Autoencoders

Presented by:
Jason Yu and Rajshree Daulatabad

Topics Covered

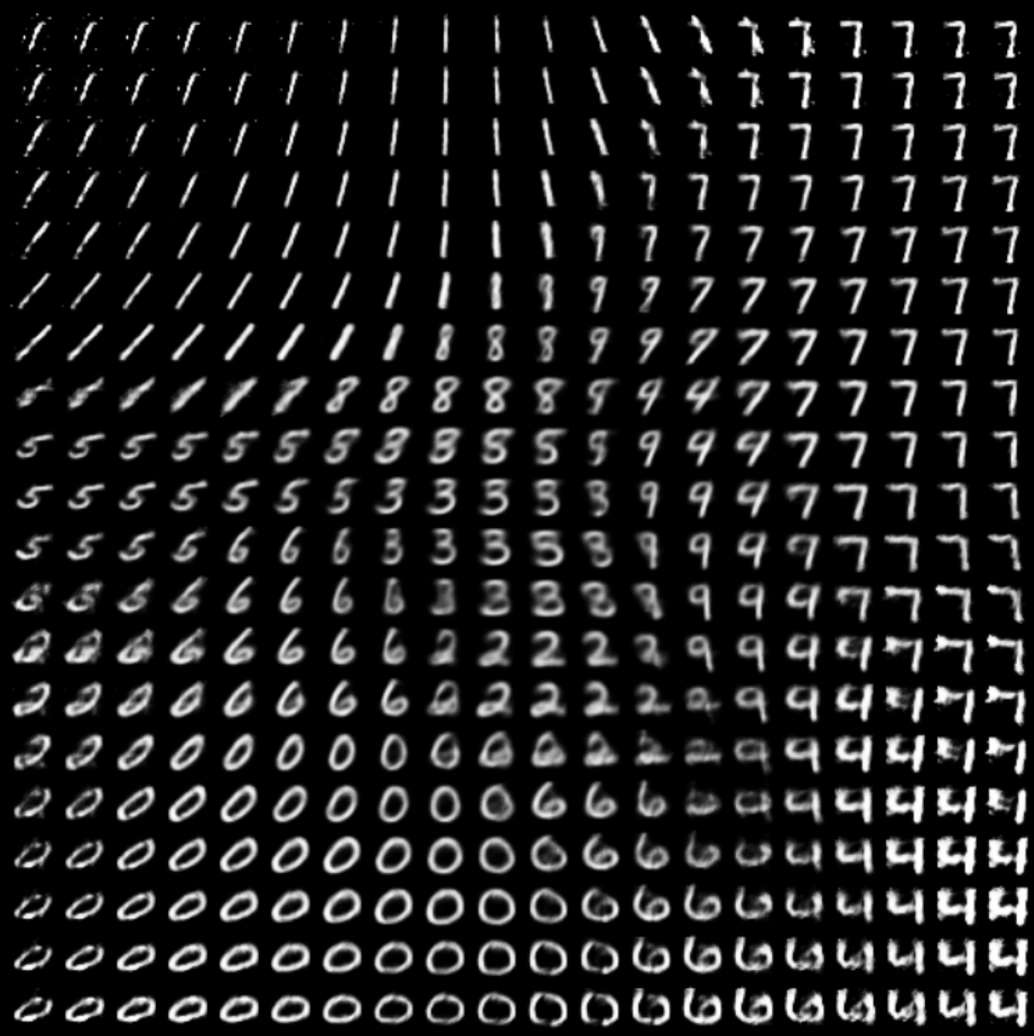
- Before we dive into VAEs - Some General Concepts
- VAE Implementation details and the Math
- Intuitively Understanding VAE
- VAE Applications & Examples
- VAE Advantages and Limitations

Overview & Terminology

Unsupervised Learning

Generative Model

Maximum Likelihood Estimation



Representation or Feature Learning

Probabilistic Model

Kullback-Liebler Divergence

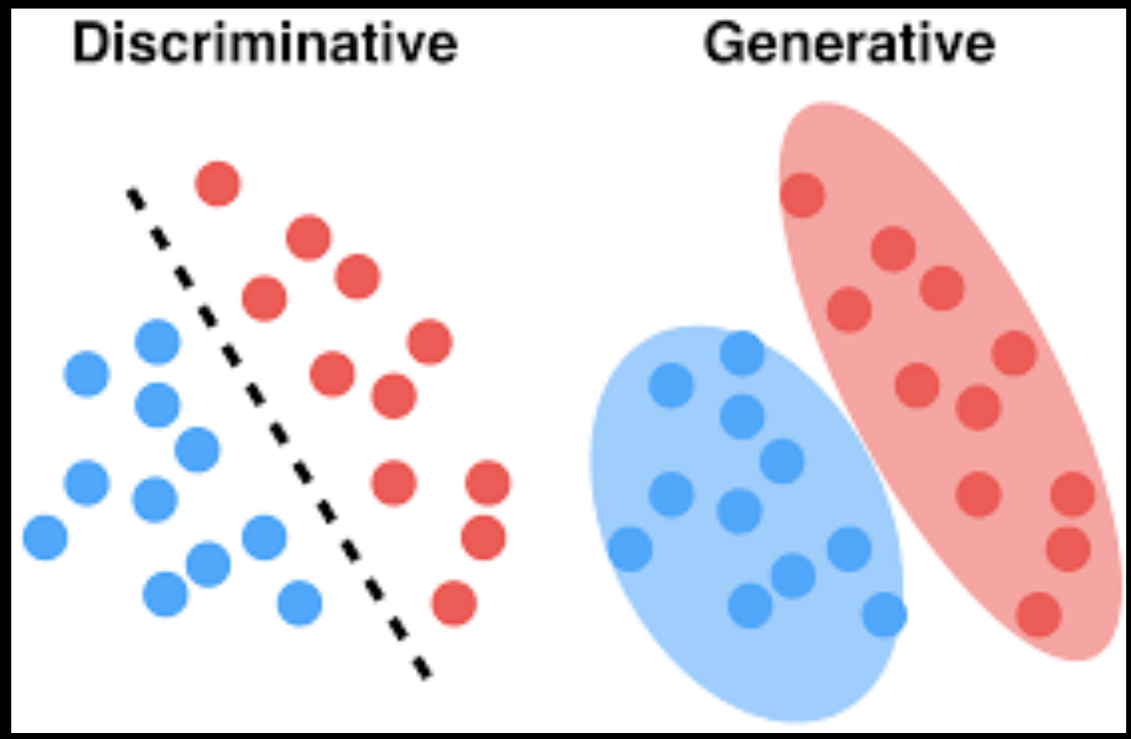
Generative Model vs Discriminative model

- **Discriminative models** learn the (hard or soft) **boundary** between classes.

- **Discriminative classifier** model the posterior p(y|x) directly, or learn a direct map from inputs x to the class labels.

- **Generative models** model the **distribution** of individual classes.

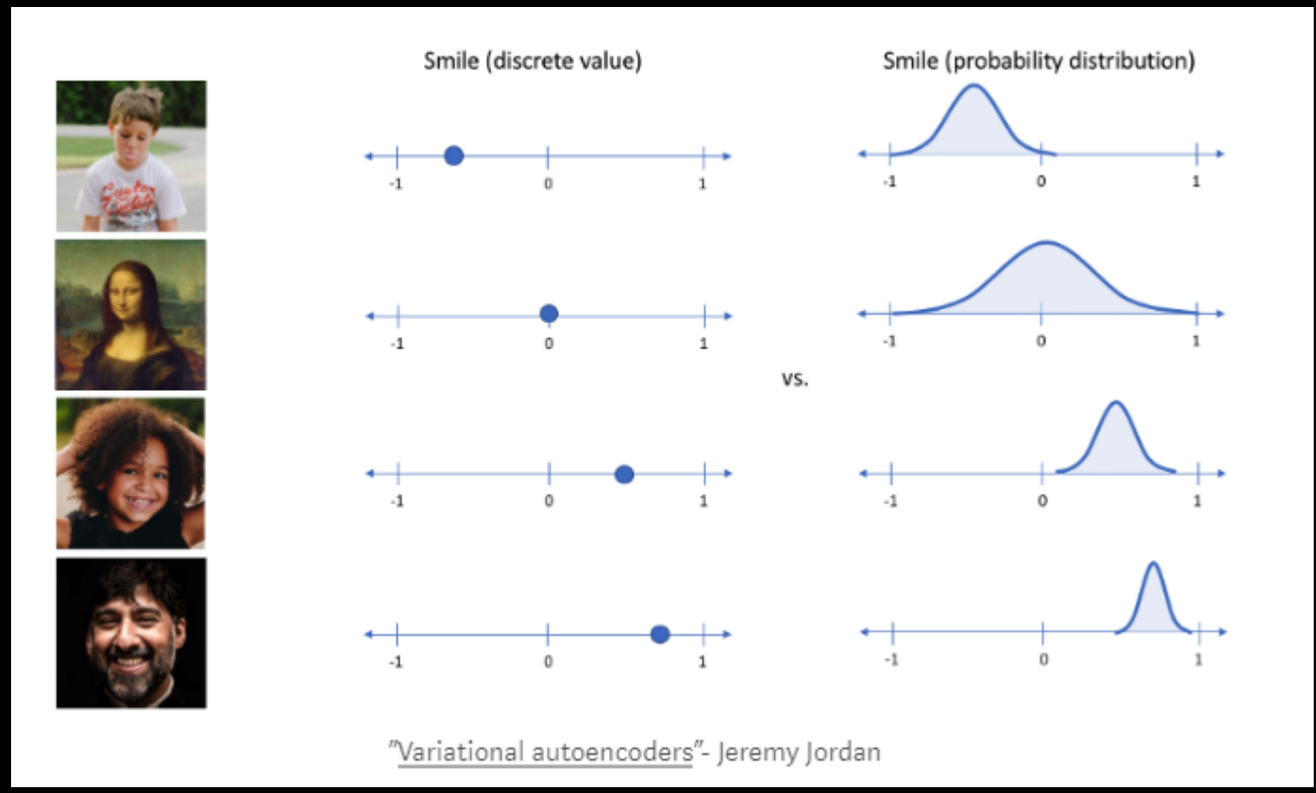
- **Generative classifiers** learn a model of the joint probability , p(x,y), of the inputs x and the label y, and make their predictions by using Bayes rules to calculate P(y|x) and then picking the most likely label y.



Probabilistic Model

The textbook definition of a VAE is that it “provides *probabilistic* descriptions of observations in latent spaces.”

In plain English, this means VAEs store latent attributes as probability distributions.



Maximum Likelihood Estimation (MLE)

Maximum likelihood estimation (MLE) is a method of estimating the parameters of a statistical model, given observations. MLE attempts to find the parameter values that maximize the likelihood function, given the observations

Traditional MLE Approach

- We are given a finite sample from a data distribution

$$X = \{x | x \sim p_{\text{data}}(x)\}, |X| = n$$

- We construct a parametric model $p_{\text{model}}(x; \theta)$ for the distribution, and build a likelihood

$$\mathcal{L}(\theta; X) = \prod_{x \in X} p_{\text{model}}(x; \theta)$$

- In practice, we optimize through MCMC or other means, and obtain

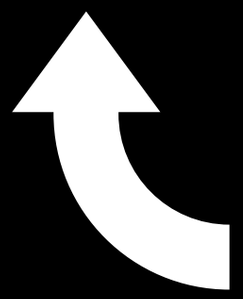
$$\theta_{\text{opt}} = \arg\min_{\theta} \{-\ln \mathcal{L}(\theta; X)\}$$

KL Divergence

- Kullback-Leibler (KL) divergence measures how “different” two probability distributions are.
- KL-divergence is better *not* to be interpreted as a "distance measure" between distributions, but rather as **a measure of entropy increase due to the use of an approximation to the true distribution rather than the true distribution itself.**

Variational

~~Autoencoders~~



Implementation Detail

~~Neural Networks~~

Probabilistic Models

Discriminative Models

$$P(Y|X)$$

Generative Models

$$P(X,Y)$$

Probabilistic Models

Discriminative Models

$P($ **“Cat”** $|$  $)$

Generative Models

$$P(X,Y)$$

Probabilistic Models

Discriminative Models

$P($ **“Cat”** $|$  $)$

Generative Models

$P($  , **“Cat”** $)$

Probabilistic Models

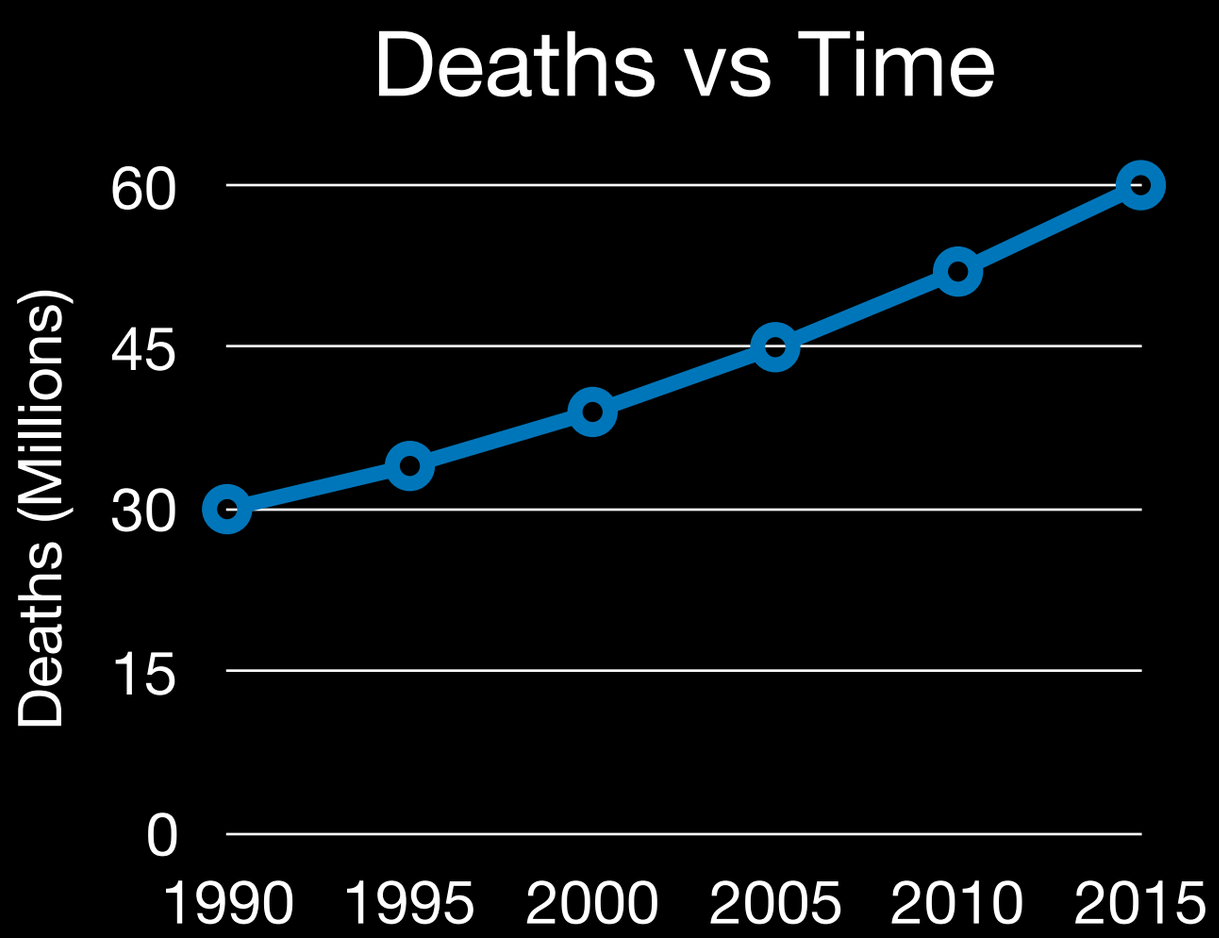
Discriminative Models

$P($ **“Cat”** $|$  $)$

Generative Models

$P($  $)$

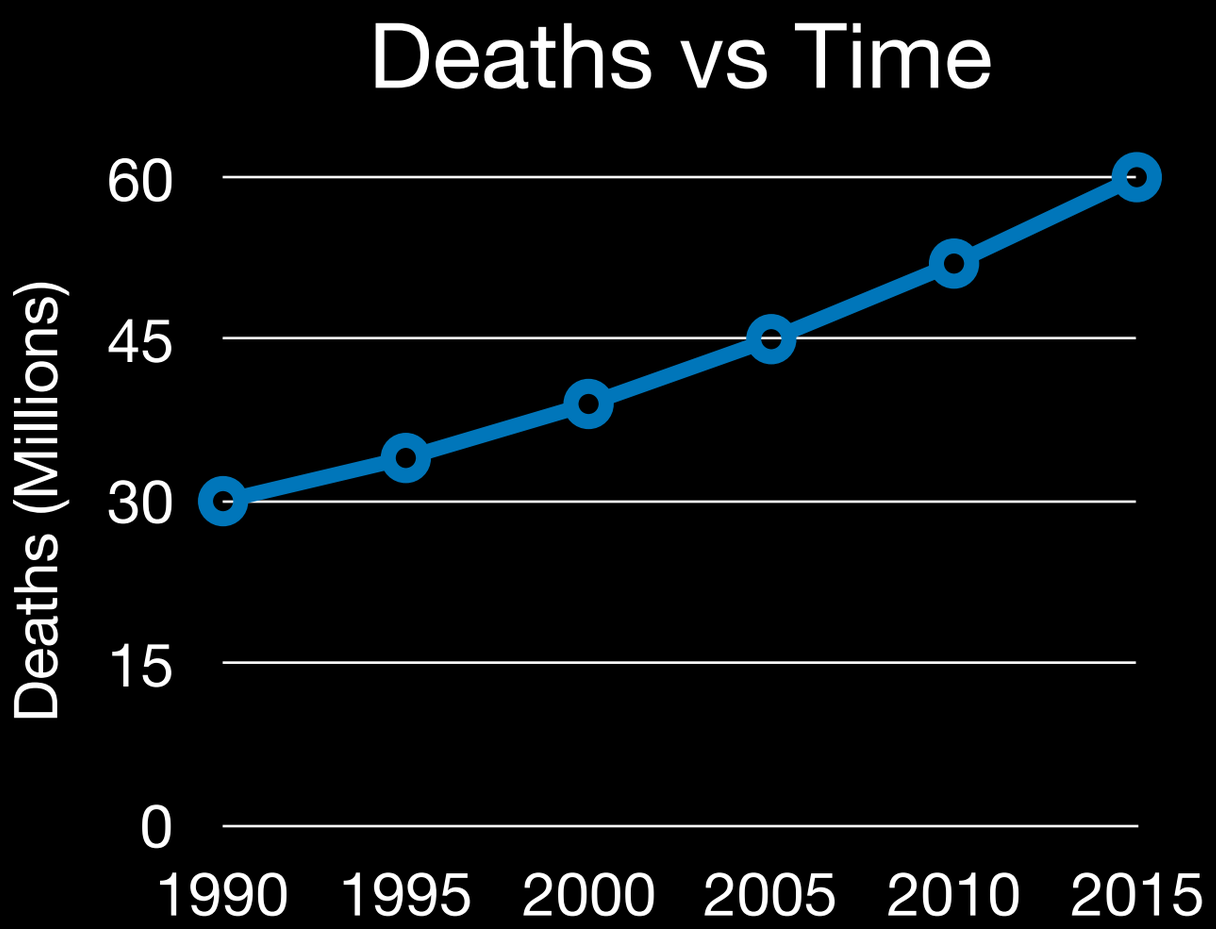
Latent Variables



x

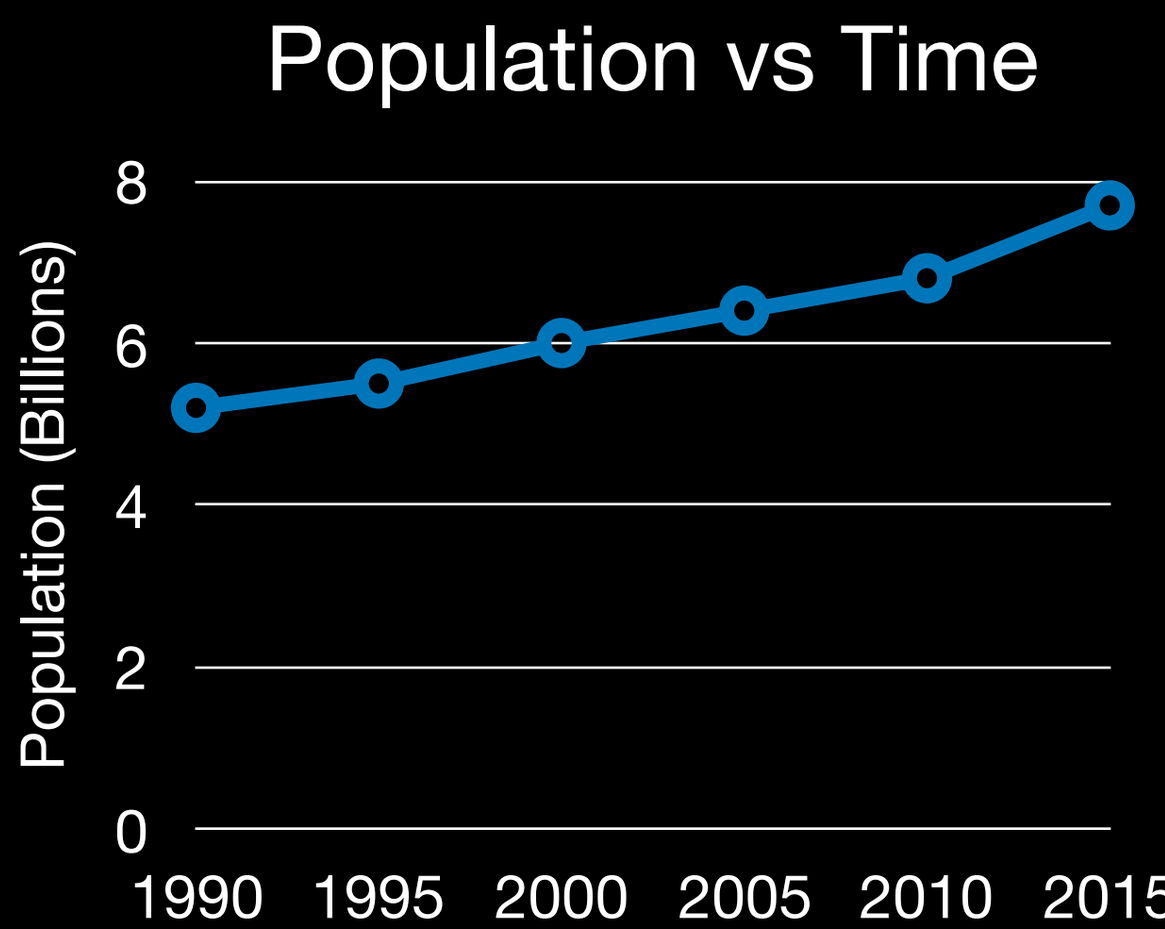
Observed Variable

Latent Variables



x

Observed Variable



z

Latent Variable

Latent Variables

x

Observed Variable

Latent Variables

Glasses

No Glasses

x

Observed Variable

z

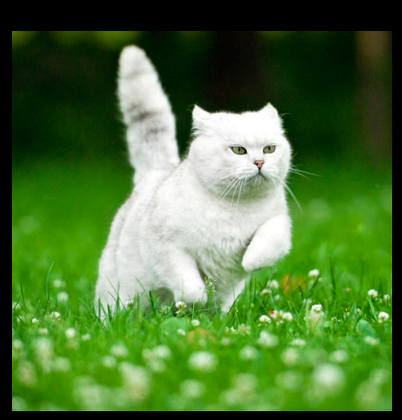
Latent Variable

Theory
And
Intuition

Task

$$D = \{d_0, d_1, ..., d_n\}$$

$$X \to D$$

$$D = \{$$    $$\}$$

$$\hat{x} =$$ 

$$X \to \hat{x} \qquad \hat{x} \notin D$$

Modeling Data

$$X$$

Modeling Data

$$P(X)$$

Modeling Data

$$P(X) = \int P(X, z) \, dz$$

Modeling Data

$$P(X) = \int P(X|z)\,P(z)\,dz$$

Modeling Data

How is

$$P(X) = \int P(X|z)\,P(z)\,dz$$

defined?

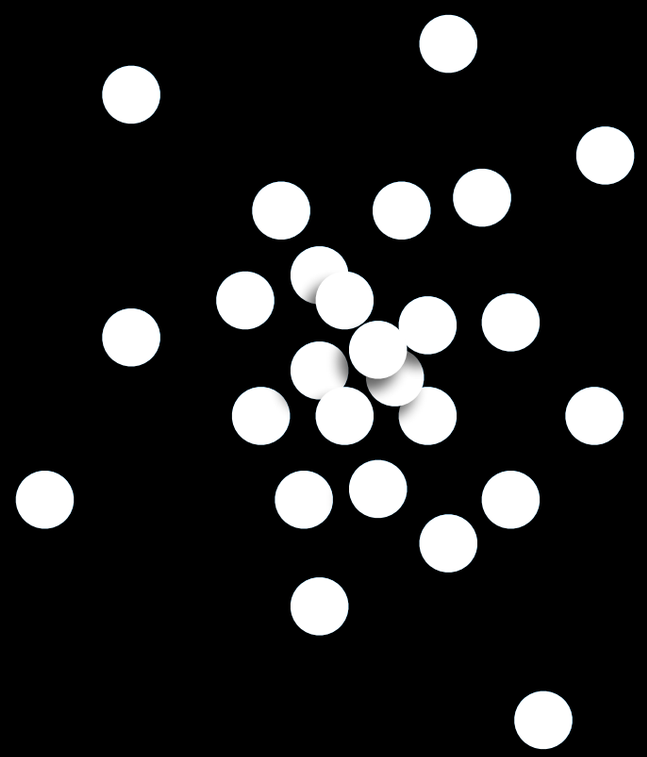
Modeling Data

How is

$$= \int P(X|z)P(z)dz$$

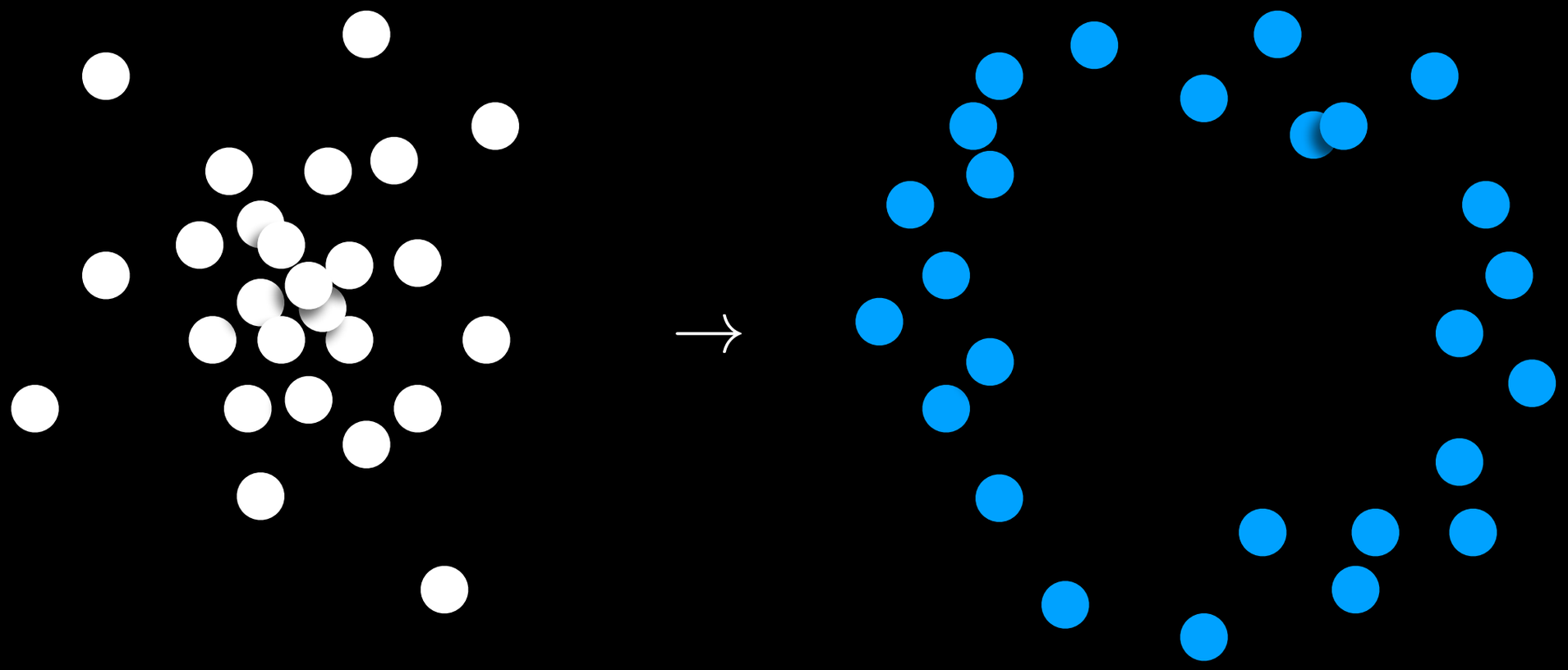
defined?

Transforming Distributions



$$z \leftarrow \mathcal{N}(0, I)$$

Transforming Distributions



$z \leftarrow \mathcal{N}(0, I)$

$f(z)$

Modeling Data

$$P(z)$$

Modeling Data

$$P(z) \sim \mathcal{N}(0, I)$$

Modeling Data

$$P(z) \sim \mathcal{N}(0, I)$$

$$z$$

Modeling Data

$$P(z) \sim \mathcal{N}(0, I)$$

$$P(X|z) \neq f(z)$$

Modeling Data

$$P(z) \sim \mathcal{N}(0, I)$$

$$\mathcal{N}(f(z), \sigma^2 * I)$$

Modeling Data

$$P(z) \sim \mathcal{N}(0, I)$$

$$\mathcal{N}(f(z), \sigma^2 * I)$$

Hyper-parameter

Modeling Data

$$P(z) \sim \mathcal{N}(0, I)$$

$$P(X|z) = \mathcal{N}(f(z), \sigma^2 * I)$$

Modeling Data

$$P(z) \sim \mathcal{N}(0, I)$$

$$P(X|z;\theta) = \mathcal{N}(f(z;\theta), \sigma^2 * I)$$

First Objective

$$\underset{\theta}{\operatorname{argmax}} \sum_{x \in D} P(x; \theta)$$

First Objective

$$\underset{\theta}{\operatorname{argmax}} \sum_{x \in D} log(\, P(x; \theta))$$

First Objective

$$\underset{\theta}{\operatorname{argmax}} \sum_{x \in D} log(\int P(x|z;\theta)P(z)dz\,)$$

First Objective

$$\underset{\theta}{\operatorname{argmax}} \sum_{x \in D} log(\int P(x|z;\theta) P(z) dz)$$

First Objective

$$\underset{\theta}{\operatorname{argmax}} \sum_{x \in D} log(\int P(x|z;\theta) P(z) dz)$$

Intractable!

First Objective

$$\underset{\theta}{\operatorname{argmax}} \sum_{x \in D} log(\int P(x|z;\theta) P(z) dz)$$

Many of these values are close to 0

$$z \leftarrow P(Z|x)$$

First Objective

$$\int P(x|z;\theta)P(z)dz$$

$$\approx \quad \frac{1}{N}\sum_{z \leftarrow P(Z|x)} P(x|z;\theta)P(z)$$

$$z \leftarrow P(Z|x)$$

First Objective

$$\int P(x|z;\theta)P(z)dz$$

$$\approx \quad \frac{1}{N}\sum_{z \leftarrow Q(Z)} P(x|z;\theta)P(z)$$

$$z \leftarrow Q(Z)$$

$$Q(Z) \approx P(Z|x)$$

Second Objective

We want:

$$Q(Z) \approx P(Z|x)$$

Second objective

$$\text{argmin}\ \mathcal{D}[Q(Z)||P(Z|X)]$$

Kullback-Leibler Divergence

$$\mathcal{D}[P||Q]$$

Kullback-Leibler Divergence

$$\int_{-\infty}^{\infty} p(x)\ log\ \frac{p(x)}{q(x)}\ dx$$

Kullback-Leibler Divergence

$$\mathbb{E}_{x \sim P} \left[log \frac{p(x)}{q(x)} \right]$$

Kullback-Leibler Divergence

$$\mathbb{E}_{x\sim P}\left[log\, p(x) - log\, q(x)\right]$$

$$\mathcal{D}[P||Q] = 0,\ P = Q$$

Analysing the Second Objective

$$\mathcal{D}[Q(z)||P(z|X)]$$

Analysing the Second Objective

$$\mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, Q(z) - log\, P(z|X)\right]$$

Analysing the Second Objective

$$\mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, Q(z) - log\, P(X|z) - log\, P(z)\right] - log\, P(X)$$

Analysing the Second Objective

$$log\, P(X) - \mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, P(X|z)\right] - \mathbb{E}_{z\sim Q}\left[\, log\, Q(z) - log\, P(z)\right]$$

Analysing the Second Objective

$$log\, P(X) - \mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, P(X|z)\right] - \mathcal{D}[Q(z)||P(z)]$$

Analysing the Second Objective

Intractable

$$log\, P(X) - \mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, P(X|z)\right] - \mathcal{D}[Q(z)||P(z)]$$

Analysing the Second Objective

$$log\, P(X) - \mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, P(X|z)\right] - \mathcal{D}[Q(z)||P(z)]$$

Analysing the Second Objective

$$log\, P(X) - \mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, P(X|z)\right] - \mathcal{D}[Q(z)||P(z)]$$

Analysing the Second Objective

$$log\, P(X) - \mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, P(X|z)\right] - \mathcal{D}[Q(z)||P(z)]$$

Analysing the Second Objective

$$log\, P(X) - \mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, P(X|z)\right] - \mathcal{D}[Q(z)||P(z)]$$

Analysing the Second Objective

$$log\, P(X) - \mathcal{D}[Q(z)||P(z|X)]$$

$$=$$

$$\mathbb{E}_{z\sim Q}\left[log\, P(X|z)\right] - \mathcal{D}[Q(z)||P(z)]$$

Tractable

Conditioning Q

$$\mathcal{D}[Q(z)||P(z|X)]$$

Conditioning Q

$$Q(z)$$

Conditioning Q

$$Q(z|x)$$

Conditioning Q

$$Q(z|x;\theta)$$

Conditioning Q

$$Q(z|x;\theta)$$

$$=$$

$$\mathcal{N}(\mu(x;\theta), \Sigma(x;\theta))$$

Recap

$$\underset{\theta}{\operatorname{argmax}} \sum_{x \in D} log\, P(x; \theta)$$

Recap

$$\underset{\theta}{\operatorname{argmax}} \sum_{x \in D} log\, P(x; \theta)$$

Due to marginalizing out z

Recap

$$\underset{\theta}{\text{argmax}} \sum_{x \in D} log\, P(x; \theta)$$

Due to marginalizing out z

$$\underset{\theta}{\text{argmin}}\, \mathcal{D}[Q(Z|X;\theta)||P(Z|X)]$$

Recap

$$\underset{\theta}{\text{argmax}} \sum_{x \in D} log\, P(x; \theta)$$

Due to marginalizing out z

$$\underset{\theta}{\text{argmin}}\, \mathcal{D}[Q(Z|X;\theta)||P(Z|X)]$$

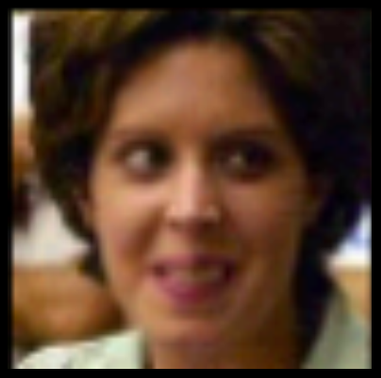
Via analysis of the second objective

$$\mathbb{E}_{z \sim Q}[log\, P(X|z;\theta)] - \mathcal{D}[Q(z|X;\theta)||P(z)]$$

Tractable objective equal to the original objective

“Autoencoder” Training

x



$\mathcal{N}(\mu(x;\theta), \Sigma(x;\theta))$

$Q(z|x)$

“Encoder”

$\rightarrow z$

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

$\mathcal{N}(f(z;\theta), \sigma^2 * I)$

$P(x|z)$

“Decoder”

$\hat{x}$

“Autoencoder” Test

$$P(Z)$$

$\rightarrow z$

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

$$\mathcal{N}(f(z;\theta), \sigma^2 * I)$$

$$P(x|z)$$

“Decoder”

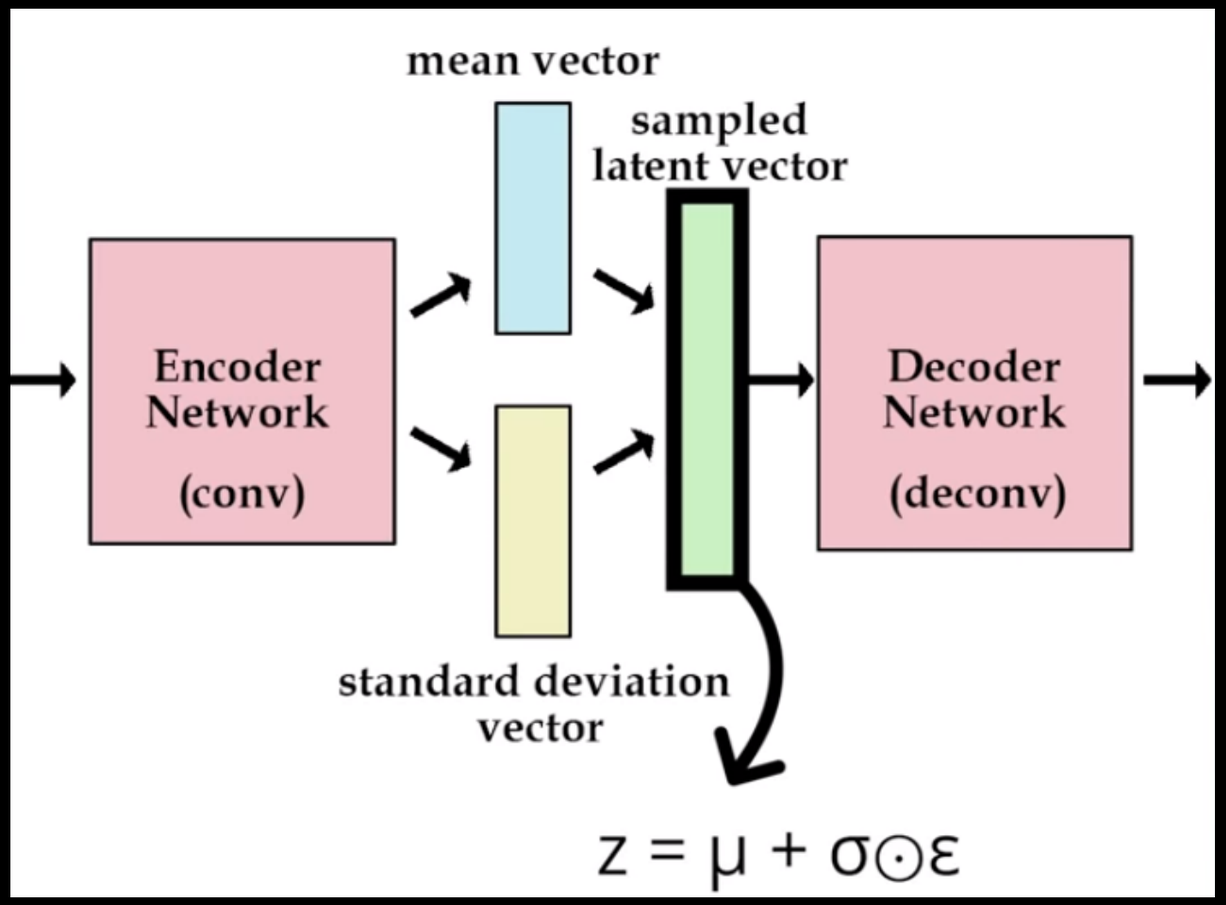
$\hat{x}$

Implementation

Using NN for VAE – but there's a problem!

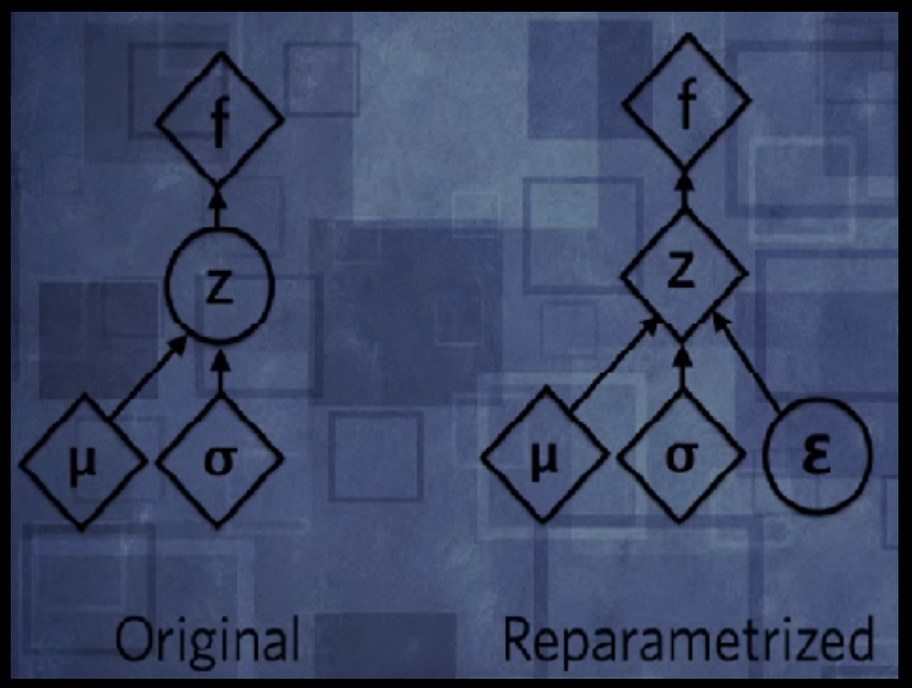
- Imagine Encoder - Q(z|X) is a neural net
- How do we get z? - we could sample z from a Gaussian which parameters are the outputs of the encoder.
- But how do we train without gradient descent? Sampling directly wont do!!

Reparameterization trick

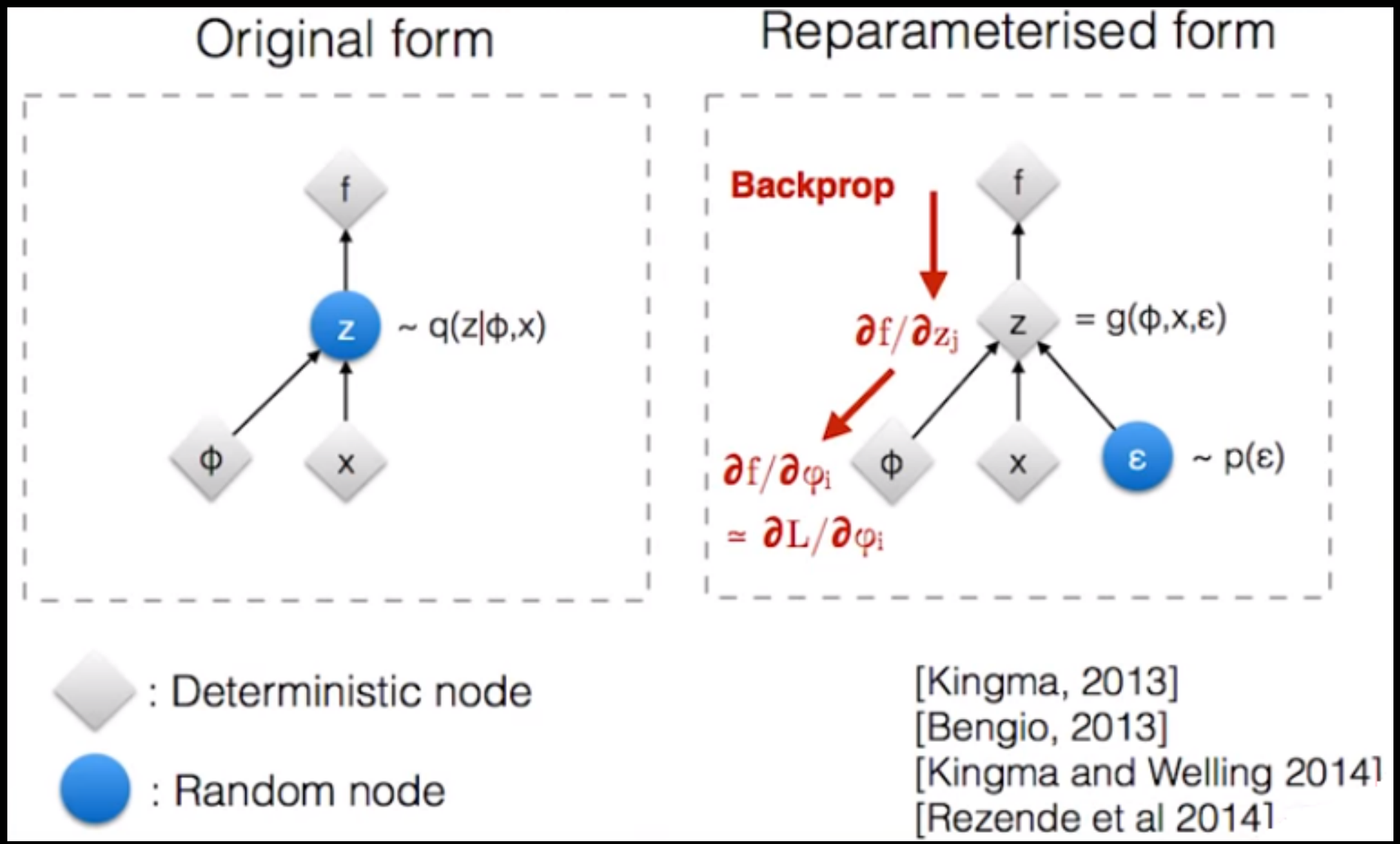


$$z = \mu + \sigma \odot \varepsilon$$

where $\varepsilon \sim \text{Normal}(0,1)$



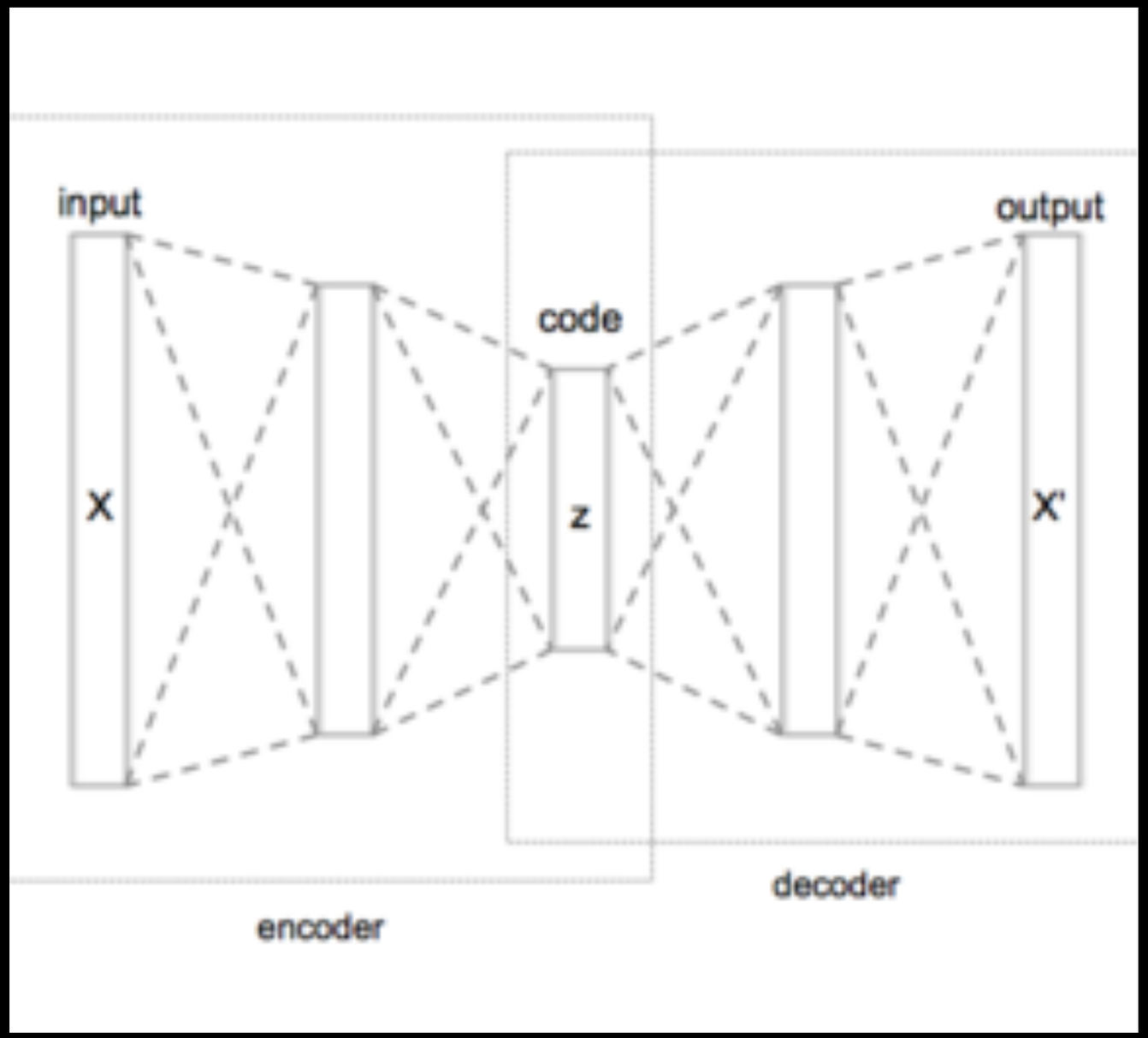
Backpropagation



Intuitively Understanding Variational Autoencoders

Standard Autoencoder

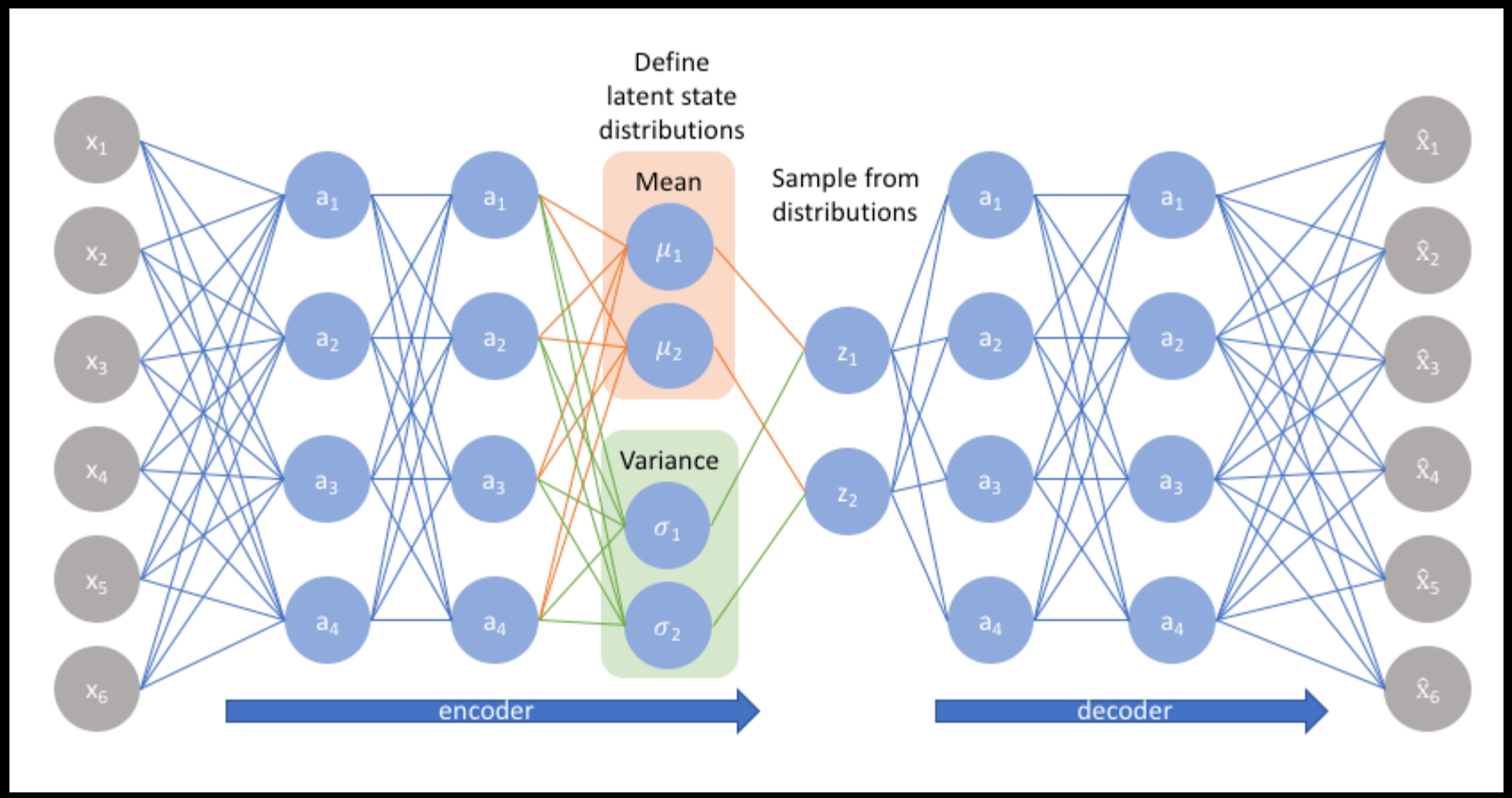
- The entire network is usually trained as a whole. The loss function is usually either the mean-squared error or cross-entropy between the output and the input, known as the *reconstruction loss*, which penalizes the network for creating outputs different from the input.



The problem with standard autoencoders

- Standard autoencoders learn to generate compact representations and reconstruct their inputs well, but asides from a few applications like denoising autoencoders, they are fairly limited.
- The fundamental problem with autoencoders, for generation, is that the latent space they convert their inputs to and where their encoded vectors lie, may not be continuous, or allow easy interpolation.

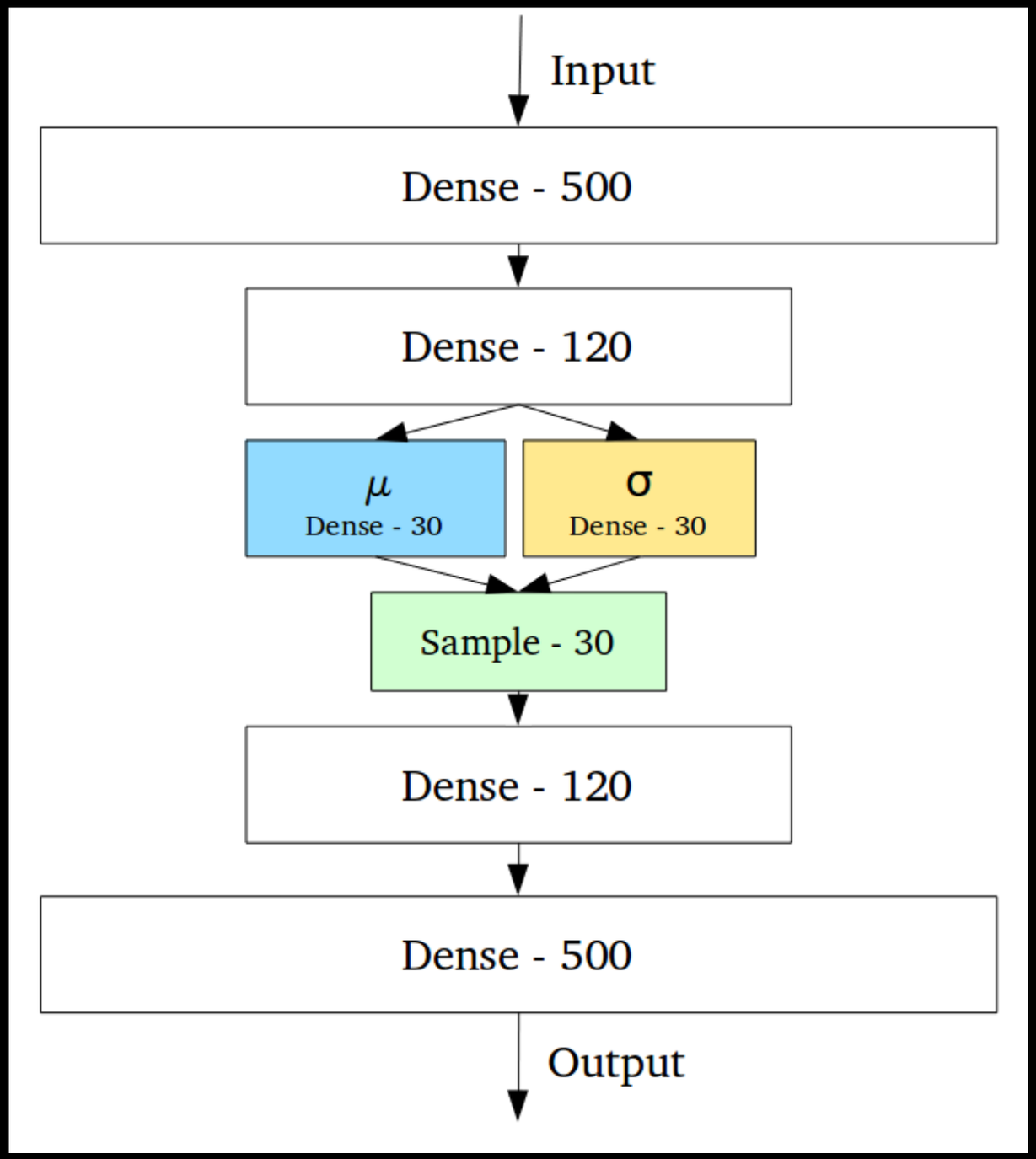
How Variational Autoencoder solves the problem



Variational Autoencoders (VAEs) have one fundamentally unique property that separates them from vanilla autoencoders, and it is this property that makes them so useful for generative modeling: their latent spaces are, *by design,* continuous, allowing easy random sampling and interpolation.

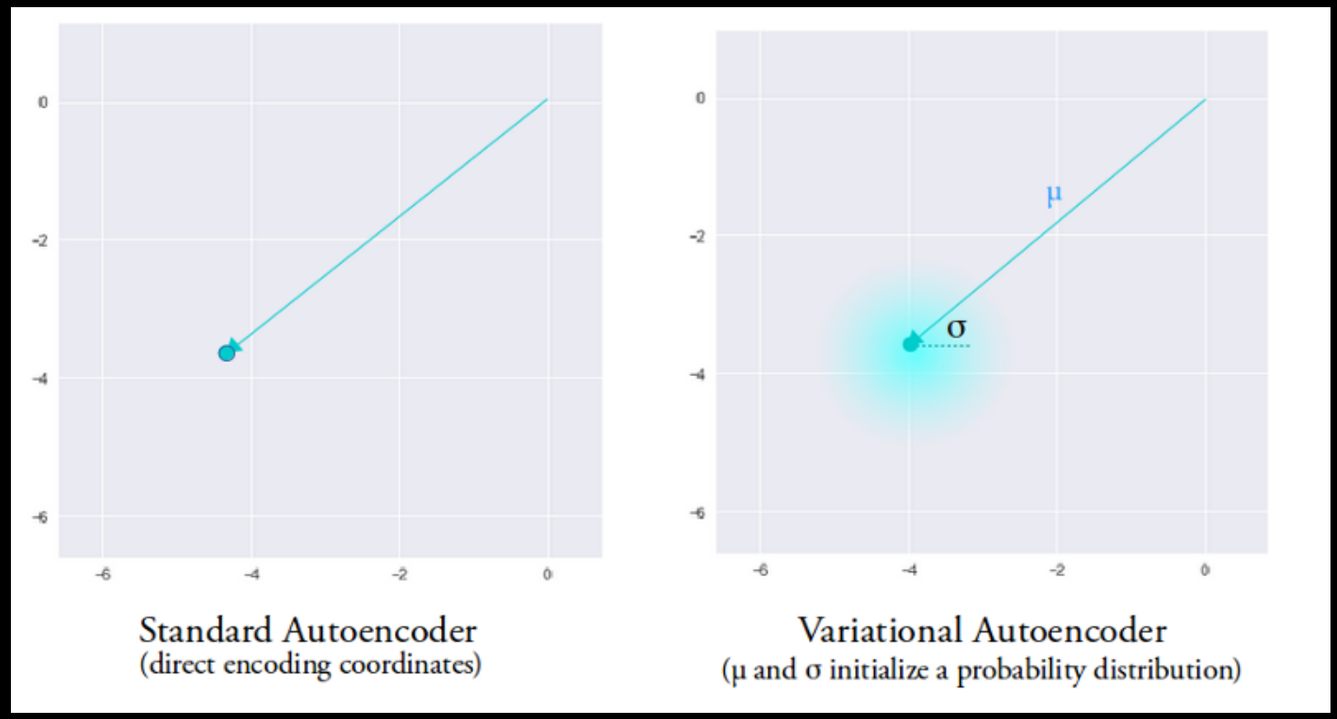
How VAE solves this problem

It achieves this by doing something that seems rather surprising at first: making its encoder not output an encoding vector of size *n,* rather, outputting two vectors of size n: a vector of means**, μ**, and another vector of standard deviations, **σ**.



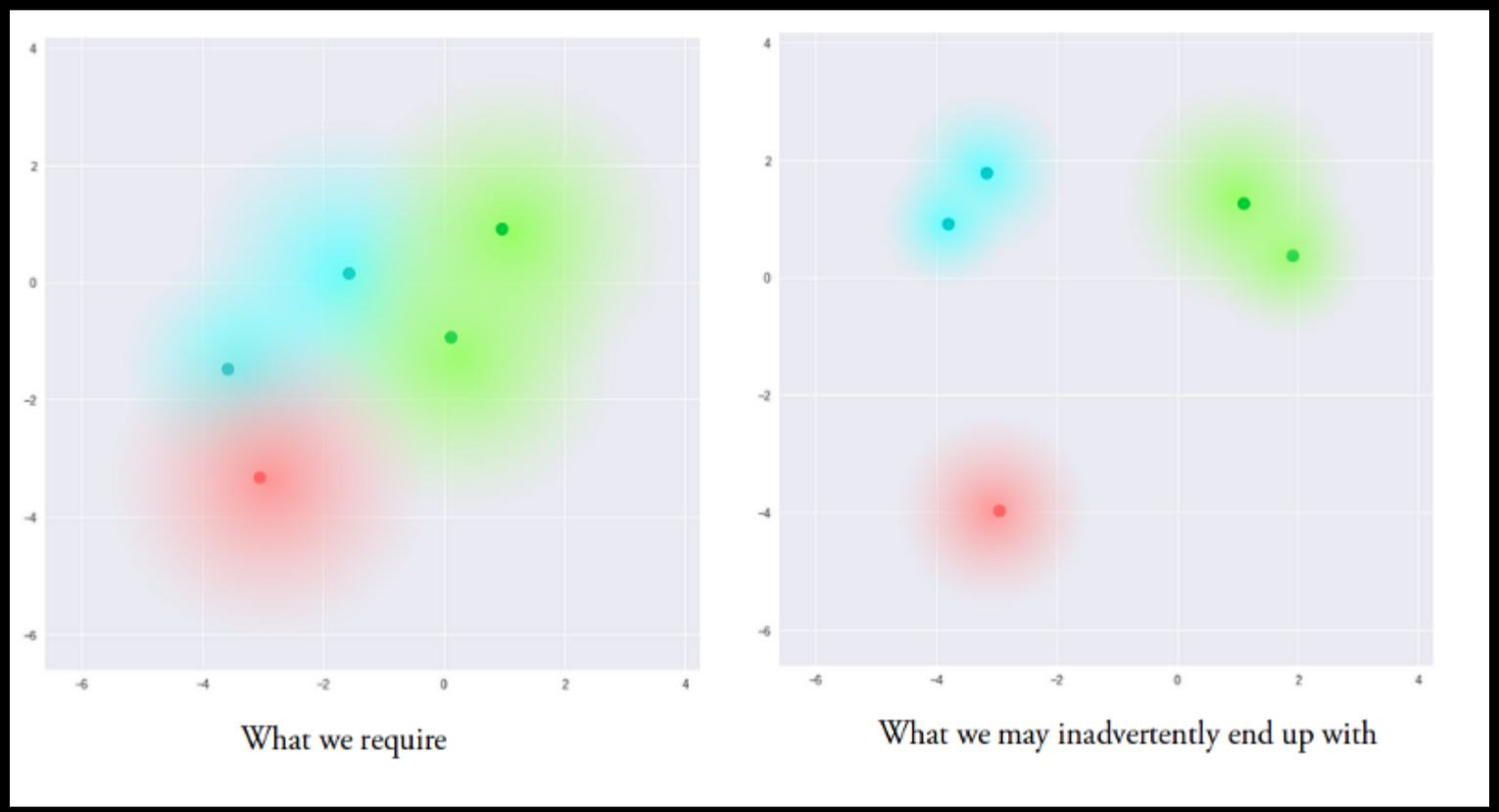
How VAE solves this problem

Intuitively, the mean vector controls where the encoding of an input should be centered around, while the standard deviation controls the “area”, how much from the mean the encoding can vary.



As encodings are generated at random from anywhere inside the “circle” (the distribution), the decoder learns that not only is a single point in latent space referring to a sample of that class, but all nearby points refer to the same as well. This allows the decoder to not just decode single, specific encodings in the latent space (leaving the decodable latent space discontinuous), but ones that slightly vary too, as the decoder is exposed to a range of variations of the encoding of the same input during training.

How VAE solves this problem



What we ideally want are encodings, *all* of which are as close as possible to each other while still being distinct, allowing smooth interpolation, and enabling the construction of *new* samples.

How VAE solves this problem

- In order to force this, we introduce the Kullback–Leibler divergence (KL divergence) into the loss function. The KL divergence between two probability distributions simply measures how much they *diverge* from each other.

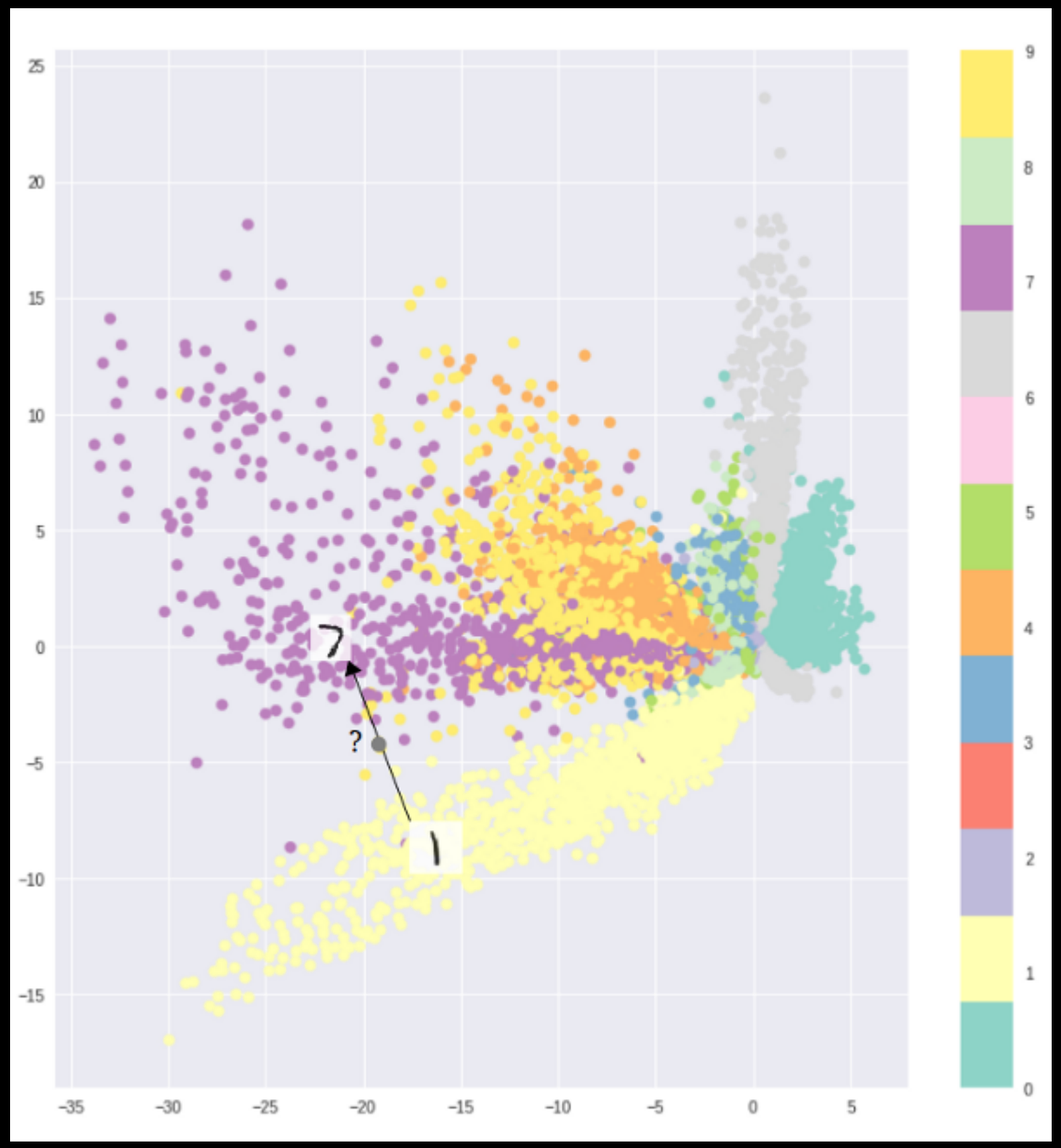
- Intuitively, this loss encourages the encoder to distribute all encodings (for all types of inputs, eg. all MNIST numbers), evenly around the center of the latent space. If it tries to “cheat” by clustering them apart into specific regions, away from the origin, it will be penalized.

VAE Loss function = Reconstruction Loss + KL Divergence loss

Optimizing purely for reconstruction loss

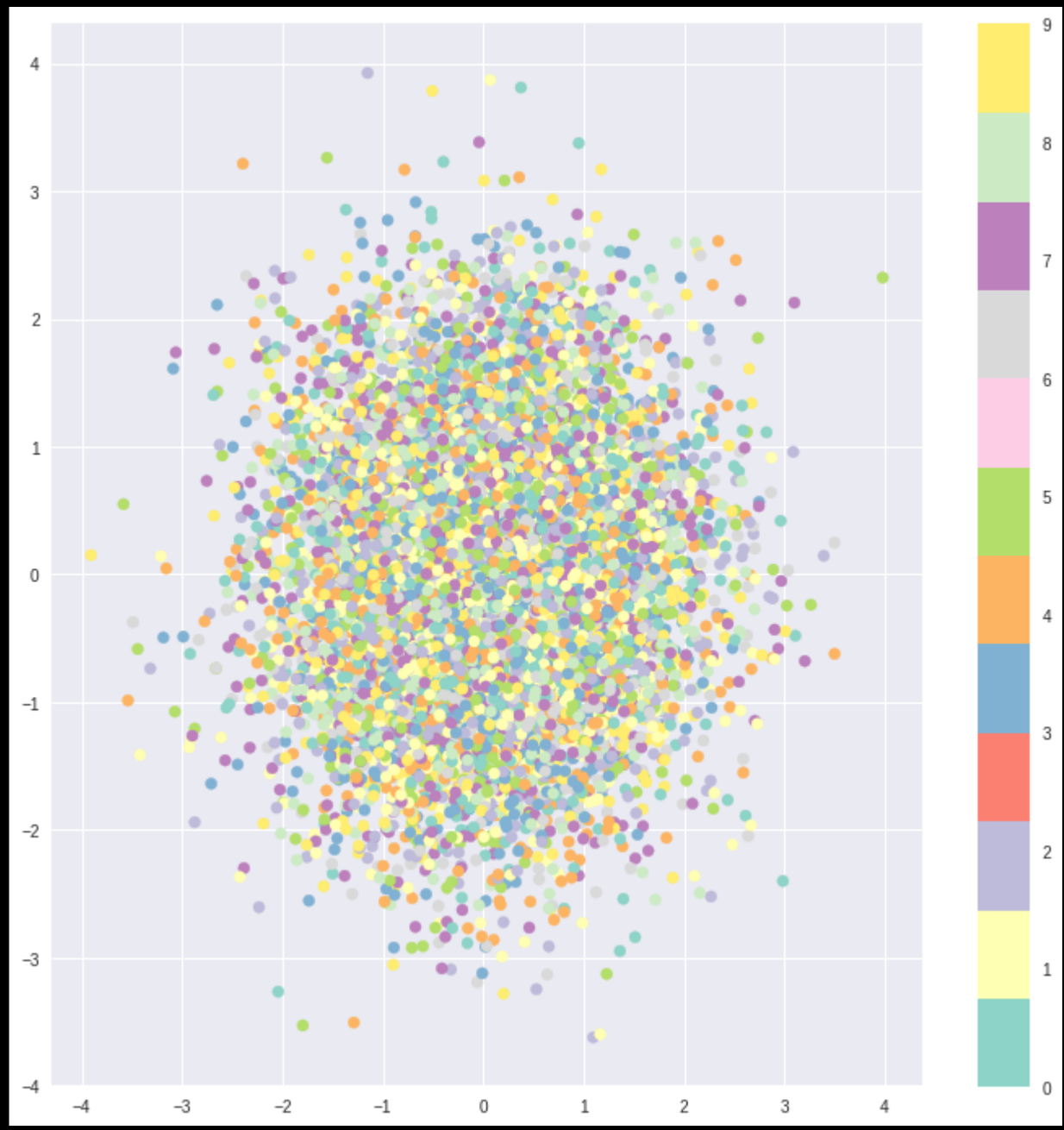
- For example, training an autoencoder on the MNIST dataset, and visualizing the encodings from a 2D latent space reveals the formation of distinct clusters. This makes sense, as distinct encodings for each image type makes it far easier for the decoder to decode them. This is fine if you're just *replicating* the same images.

- But when you're building a *generative* model, you **don't** want to prepare to *replicate* the same image you put in. You want to randomly sample from the latent space, or generate variations on an input image, from a continuous latent space.



Optimizing using pure KL divergence loss

Now, using purely KL loss results in a latent space results in encodings densely placed randomly, near the center of the latent space, with little regard for similarity among nearby encodings. The decoder finds it impossible to decode anything meaningful from this space, simply because there really isn't any meaning.

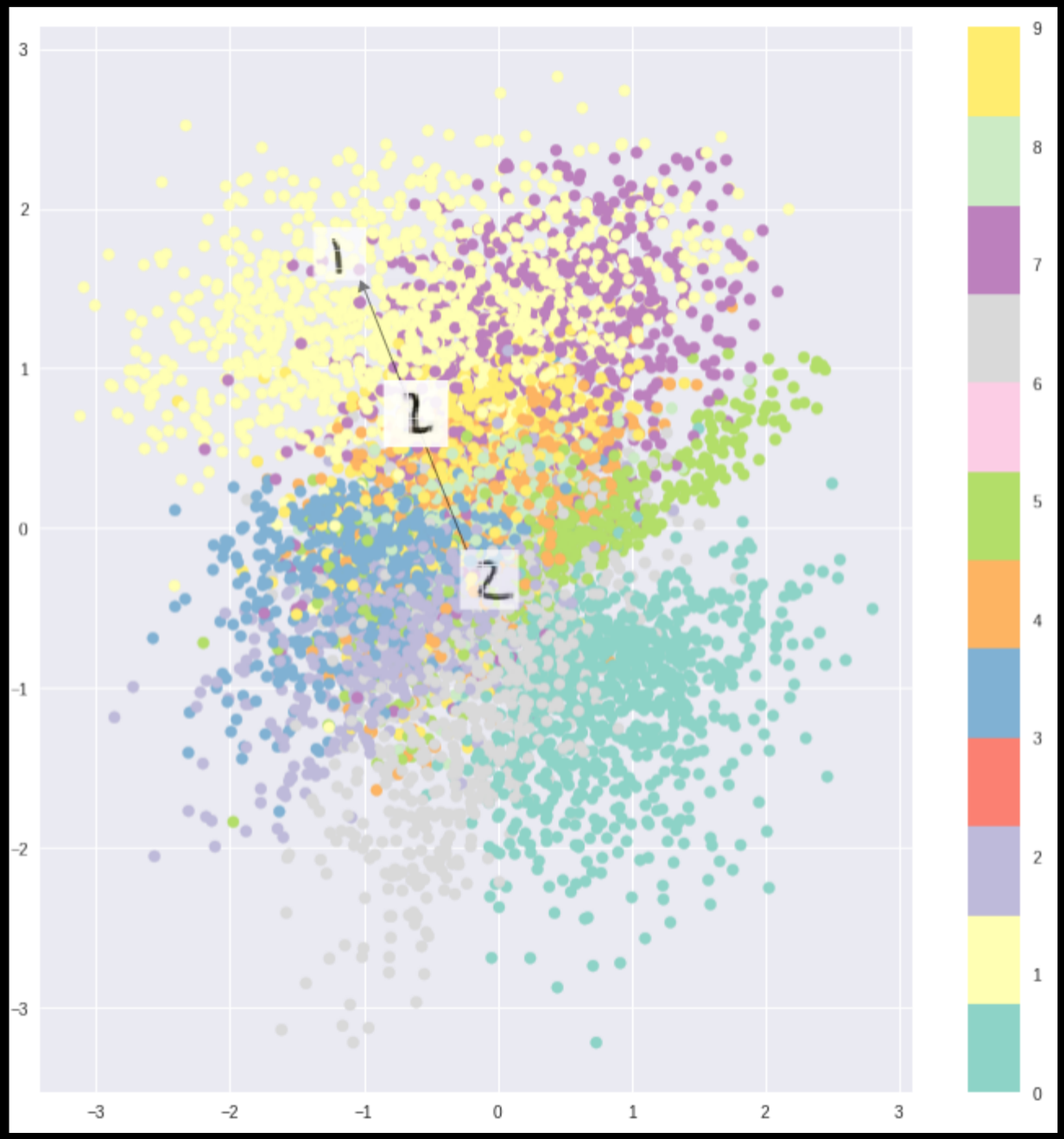


Optimizing using both reconstruction loss and KL divergence loss

- Optimizing the two together, however, results in the generation of a latent space which maintains the similarity of nearby encodings on the *local scale* via clustering, yet *globally,* is very densely packed near the latent space origin (compare the axes with the original).

- Intuitively, this is the equilibrium reached by the *cluster-forming* nature of the reconstruction loss, and the *dense packing* nature of the KL loss, forming distinct clusters the decoder can decode.

- This is great, as it means when randomly generating, if you sample a vector from the same prior distribution of the encoded vectors, $N(\mathbf{0}, \mathbf{I})$, the decoder will successfully decode it. And if you're interpolating, there are no sudden gaps between clusters, but a *smooth mix of features* a decoder can understand.

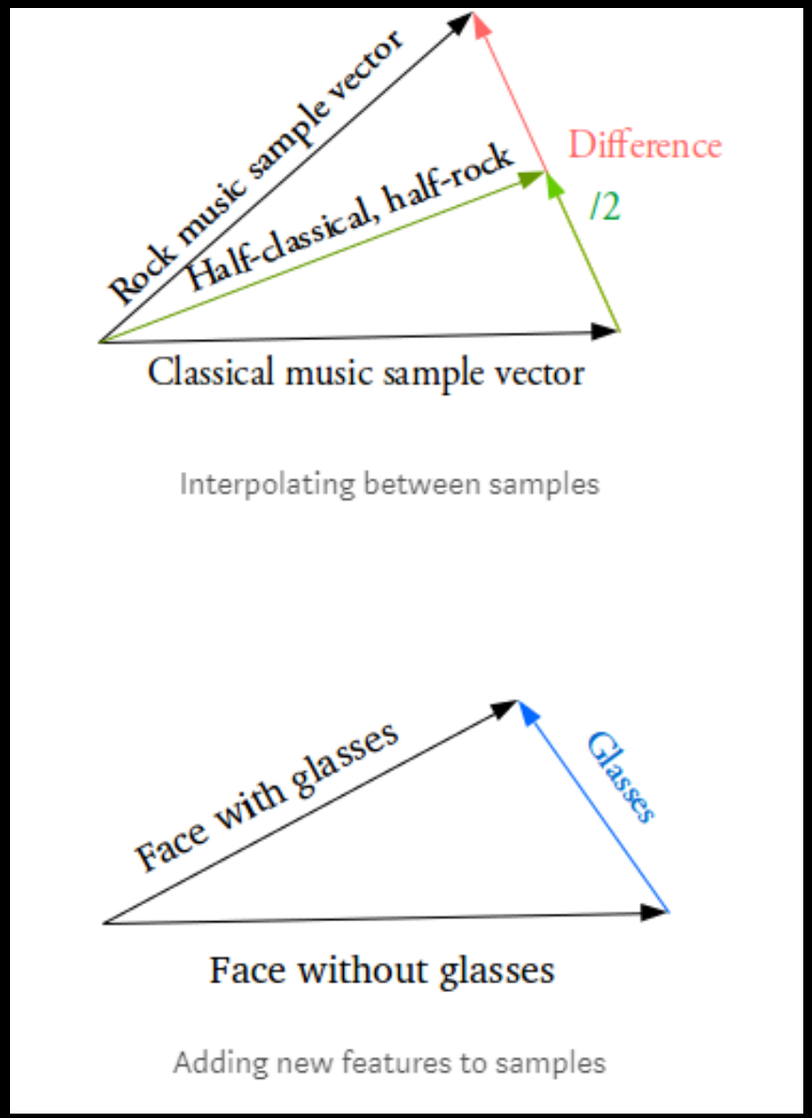


Vector arithmetic

- So how do we actually produce these smooth interpolations we speak of? From here on out, it's simple vector arithmetic in the latent space.

- For example, if you wish to generate a new sample halfway between two samples, just find the difference between their mean (**μ**) vectors, and add half the difference to the original, and then simply decode it.

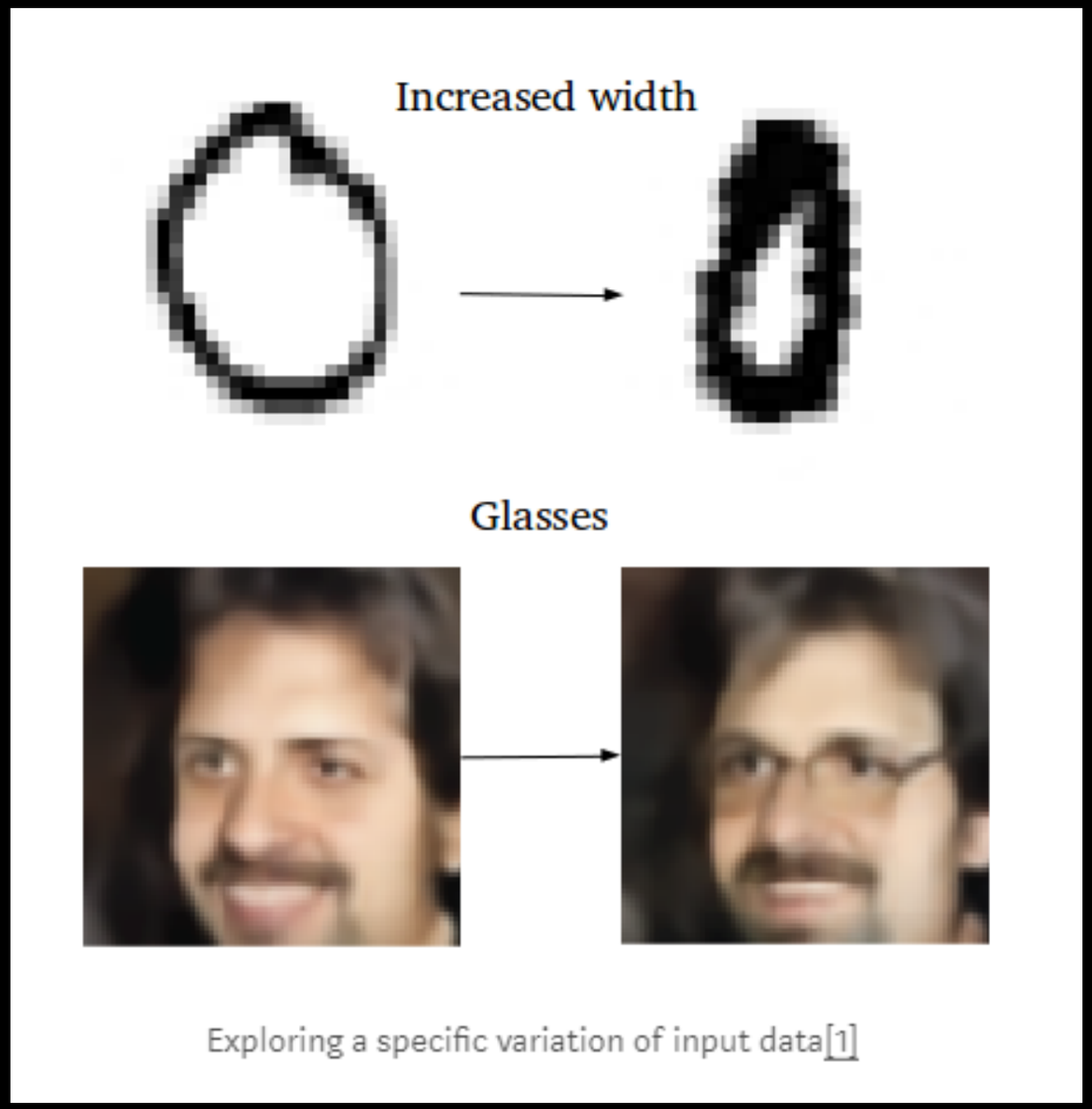
- What about generating *specific features*, such as generating glasses on a face? Find two samples, one with glasses, one without, obtain their encoded vectors from the encoder, and save the difference. Add this new "glasses" vector to any other face image, and decode it.



Advantages of VAEs

While using generative models, one of our goal is to generate a random, new output which looks similar to the training data.

So we are trying to alter or explore variations on data which we already have in random way or in a desired, specific direction. This is where VAEs work better than any other method currently available.



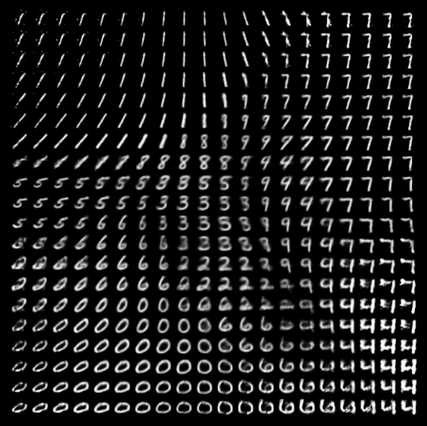
Exploring a specific variation of input data[1]

Applications

Applications

- VAEs work with remarkably diverse types of data, sequential or non-sequential, continuous or discrete, even labelled or completely unlabeled, making them highly powerful generative tools.

- You could indeed, replace the standard fully-connected dense encoder-decoder with a convolutional-deconvolutional encoder-decoder pair, to produce great synthetic human face photos.



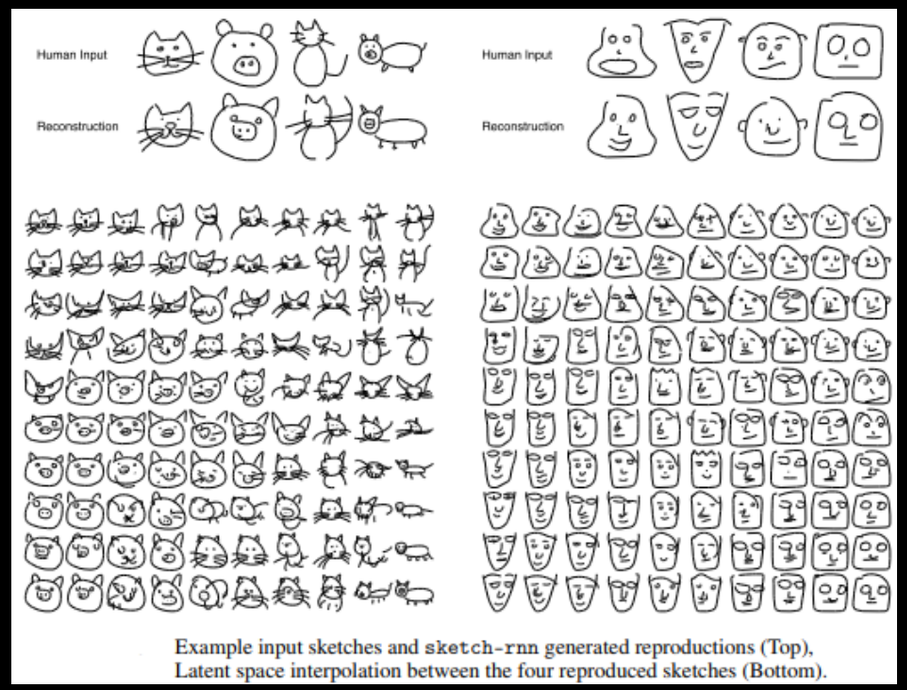
- Producing purely synthetic music - https://magenta.tensorflow.org/music-vae

- Producing synthetic text, speech

- Forecasting from Static Images using Variational Autoencoders

- Producing drawing - https://magenta.tensorflow.org/assets/sketch_rnn_demo/multi_vae.html

- Application in Faster training of Reinforcement Learning



Example input sketches and `sketch-rnn` generated reproductions (Top), Latent space interpolation between the four reproduced sketches (Bottom).

https://www.youtube.com/watch?v=KLLKNgzaDW4

Limitations

Limitations and Open issues

- The choice of approximate posterior distribution is one of the core problems in variational inference. Most applications of variational inference employ simple families of posterior approximations in order to allow for efficient inference, focusing on mean-field or other simple structured approximations. This restriction has a significant impact on the quality of inferences made using variational methods.

- Example - If we use the Gaussian as likelihood for image-generation models, we end up with blurry reconstructions