2024

23RD ANNUAL

PARIS C. KANELLAKIS

MEMORIAL LECTURE

"Robustly-Reliable Learners for Unreliable Data"

Avrim Blum

Professor and Chief Academic Officer,
Toyota Technical Institute at Chicago (TTIC)

4 PM on April 25 • CIT 368

CSCI 1470/2470
Spring 2024

Ritambhara Singh

April 05, 2024
Friday

Deep Learning

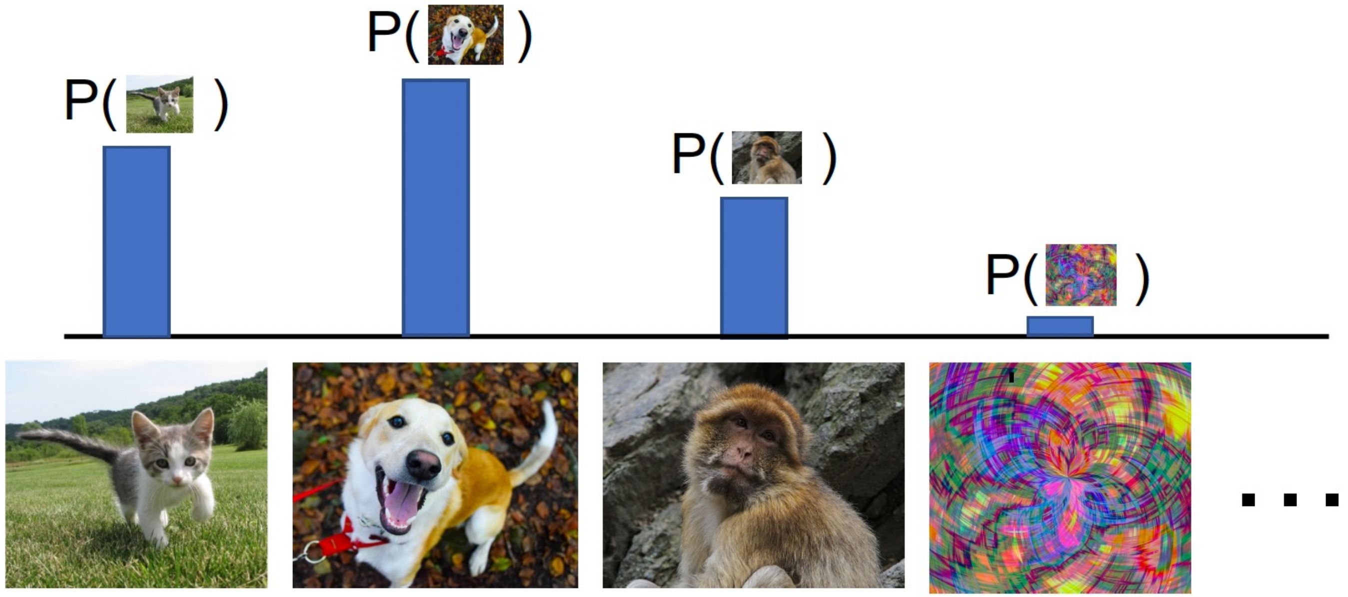
Variational Autoencoders contd.

ChatGPT prompt “minimalist landscape painting of a deep underwater scene with a blue tang fish in the bottom right corner”

Review: Discriminative v/s Generative models

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)



- Generative model: All possible images compete with each other for probability mass

- Model can "reject" unreasonable inputs by assigning them small values

Credit: UMich EECS498

Review: Weighted Combination of Losses

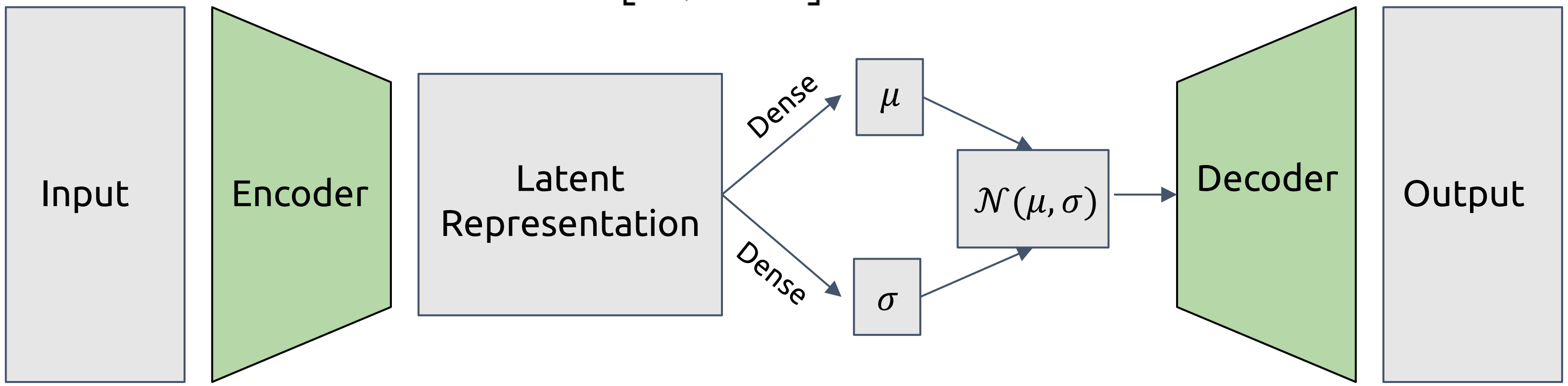
L_1 = loss associated with producing output similar to input

L_2 = loss associated with producing output with some variation to input

$$L = L_1 + \lambda L_2$$

Total Loss:

$$\lambda \in [0, \infty]$$



Today's goal – continue to learn about variational autoencoders (VAEs)

(1) VAE Loss - KL Divergence

(2) Reparameterization trick

(3) Conditional VAE

VAE Losses, Defined

We have seen L_1 before: this is just the autoencoder reconstruction loss

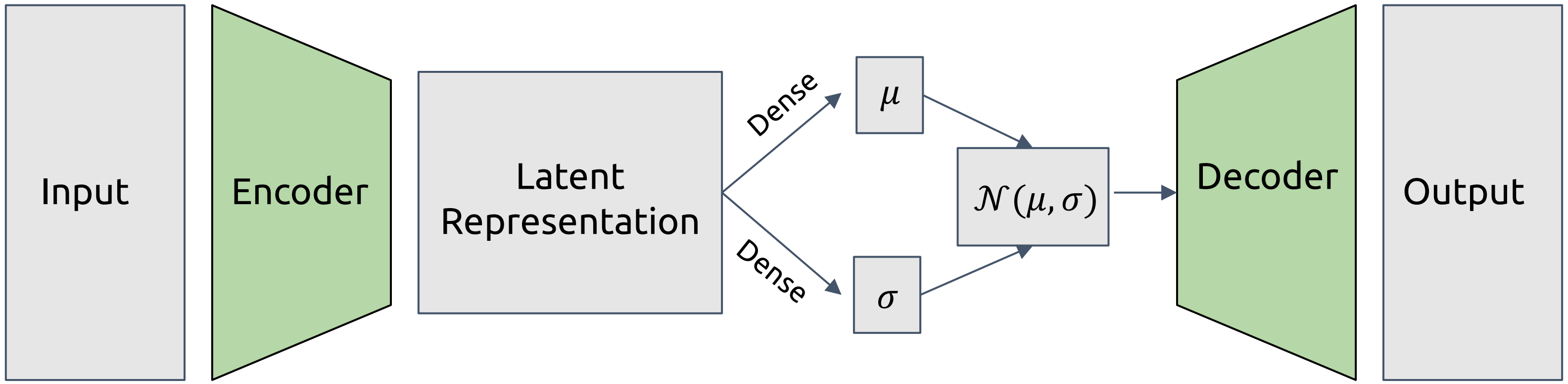
$$L_1(x, \hat{x}) = ||x - \hat{x}||_2^2$$

But with L_2, it's not so clear. How do we measure how much variation our output would have?

$$L_2(??, ??) = ??$$

Defining the L_2 Loss...

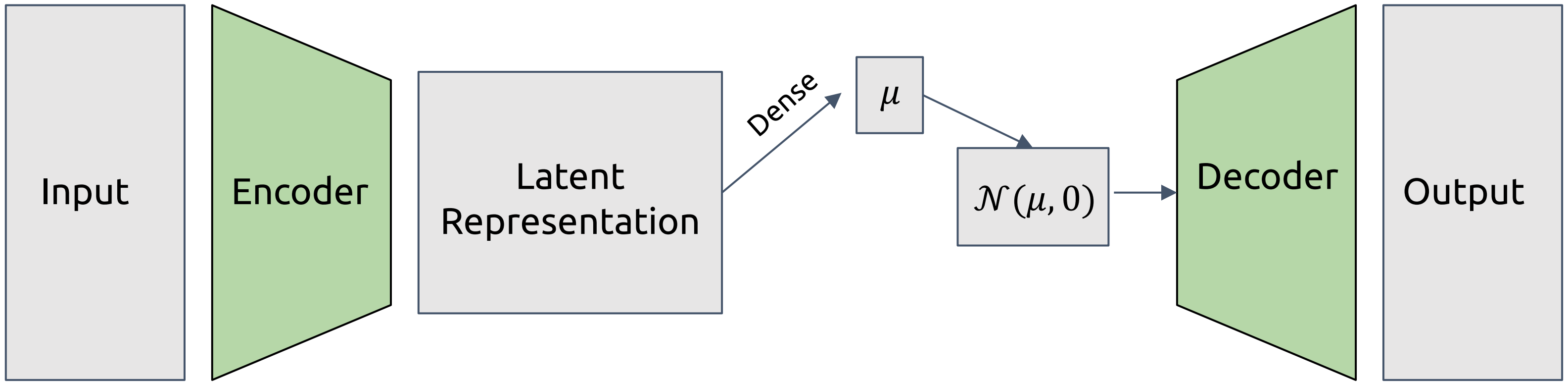
- To get variation, we definitely need a loss that encourages $\sigma > 0$
  - If we don't do this, L_1 will drive σ to zero in an effort to produce the best reconstructions



Defining the L_2 Loss...

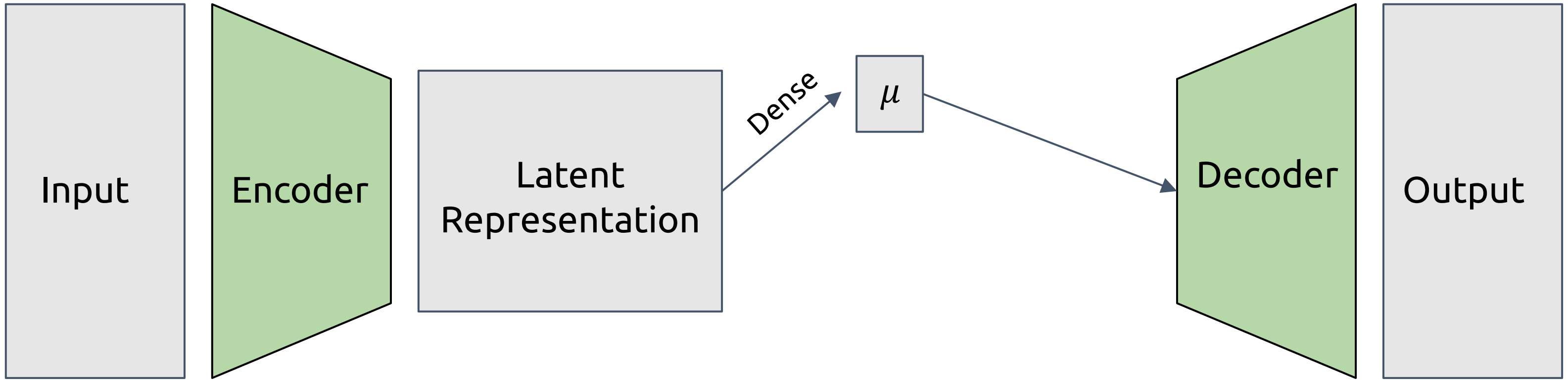
- To get variation, we definitely need a loss that encourages $\sigma > 0$
  - If we don't do this, L_1 will drive σ to zero in an effort to produce the best reconstructions

What's the issue here?



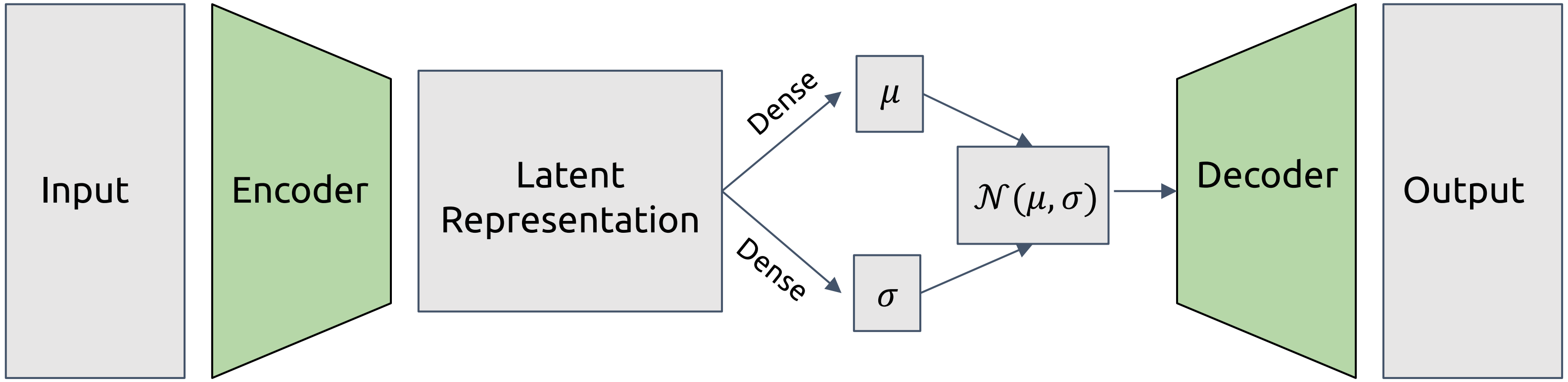
Defining the L_2 Loss...

- To get variation, we definitely need a loss that encourages $\sigma > 0$
  - If we don't do this, L_1 will drive σ to zero in an effort to produce the best reconstructions
  - Behaves the same as a regular autoencoder!



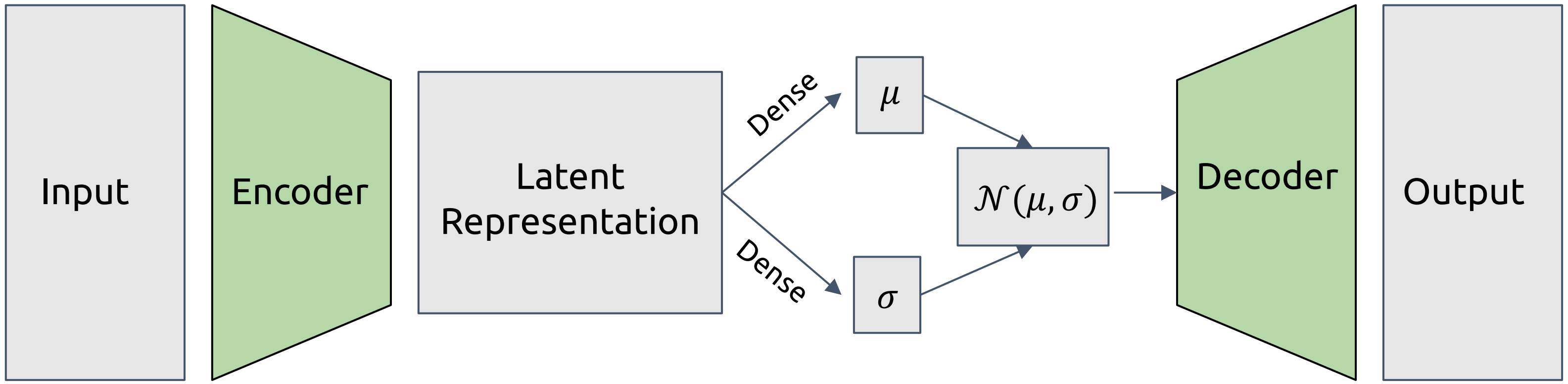
Defining the L_2 Loss...

- To get variation, we definitely need a loss that encourages $\sigma > 0$
  - If we don't do this, L_1 will drive σ to zero in an effort to produce the best reconstructions
- But how big should we encourage σ to be?
- And for that matter, what we do about μ?



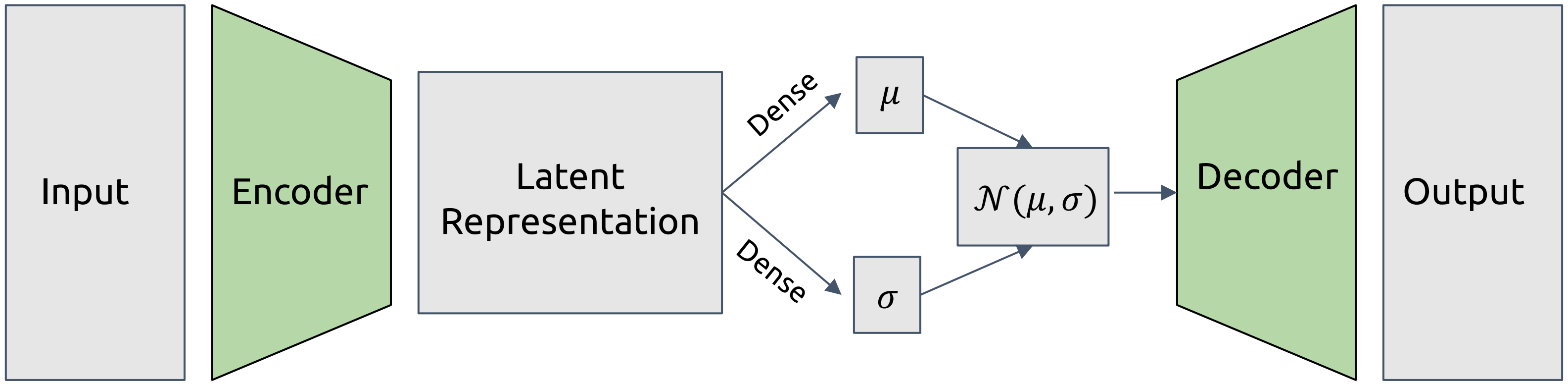
Defining the L_2 Loss…

- **The idea**: make $\mathcal{N}(\mu, \sigma)$ close to $\mathcal{N}(0, 1)$
  - Obviously, we can't perfectly satisfy this for every input (otherwise every input would produce the same set of outputs → terrible reconstruction!)
  - But, we'll see later that having some light pressure to make $\mathcal{N}(\mu, \sigma)$ close to $\mathcal{N}(0, 1)$ will have some beneficial properties



Defining the L_2 Loss...

- Wait...but ***how*** do we make $\mathcal{N}(\mu, \sigma)$ close to $\mathcal{N}(0, 1)$?
- More generally: how do measure the difference between two probability distributions?



Kullback–Leibler (KL) Divergence

Measures the difference between any two probability distributions

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

What this says:

- "Everywhere that p has probability density..."
- "...the difference between p and q should be small"
  - Difference in log probabilities (remember that $\log\left(\frac{a}{b}\right) = \log(a) - \log(b)$)

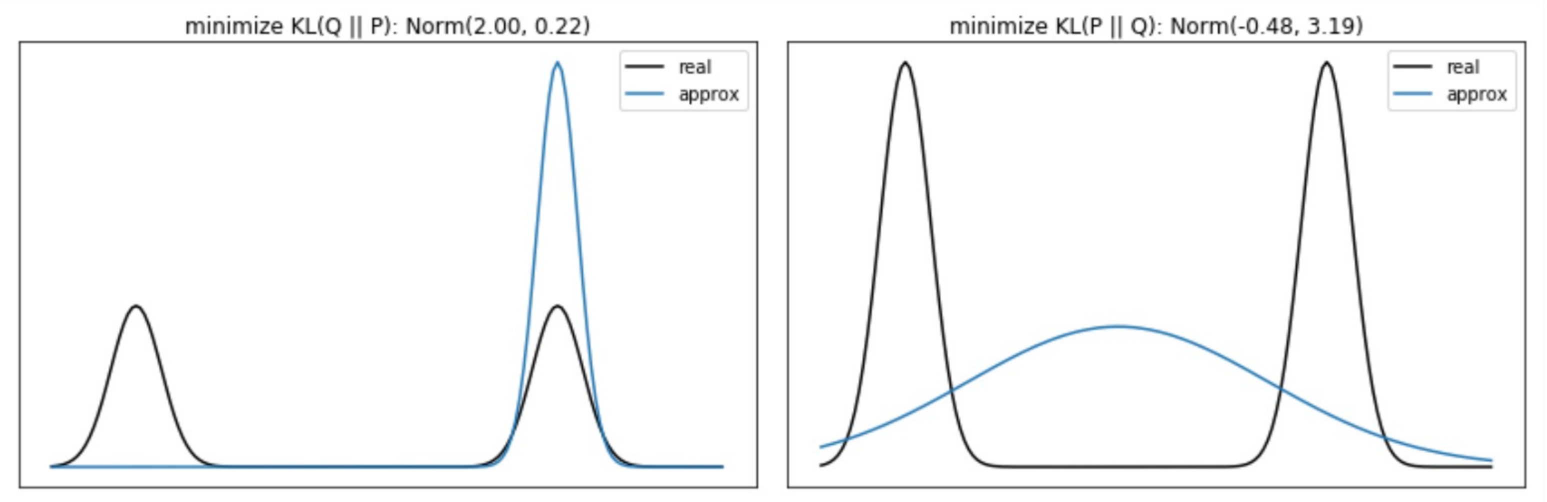
More on KL Divergence: https://jessicastringham.net/2018/12/27/KL-Divergence/

Kullback–Leibler (KL) Divergence

Measures the difference between any two probability distributions

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

- Note that this is not symmetric: $D_{KL}(P||Q) \neq D_{KL}(Q||P)$



Kullback–Leibler (KL) Divergence

- Expensive to compute, in general (no closed form, have to numerically approximate the integral)
- But! There is a closed form for Gaussians:

$$D_{KL}(\mathcal{N}(\mu, \sigma^2)||\mathcal{N}(0, 1)) = \frac{1}{2}\sum_{i=1}^{k}(\mu_i^2 + \sigma_i^2 - \ln \sigma_i^2 - 1)$$

k is the dimensionality of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ (e.g. k = 100 when $\mu \in \mathbb{R}^{100}$)

We won't derive the equation above, but let's convince ourselves it behaves how we expect it to behave

Kullback–Leibler (KL) Divergence

- Expensive to compute, in general (no closed form, have to numerically approximate the integral)
- But! There is a closed form for Gaussians:

$$D_{KL}(\mathcal{N}(\mu,\sigma^2)||\mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\mu_i^2 + \sigma_i^2 - \ln\sigma_i^2 - 1)$$

Derive the expression for (1) $\boldsymbol{\sigma}$=1 and (2) μ=0

KL Divergence for Two Gaussians

$$D_{KL}(\mathcal{N}(\mu, \sigma^2)||\mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\mu_i^2 + \sigma_i^2 - \ln \sigma_i^2 - 1)$$

Let’s take the case $\boldsymbol{\sigma}$ = 1

$$D_{KL}(\mathcal{N}(\mu, 1)||\mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\mu_i^2 + 1^2 - \ln(1) - 1)$$

$$= \frac{1}{2}\sum_{i=1}^{k}\mu_i^2$$

The expression is minimized $\boldsymbol{\mu}$ = 0 (which is what we want!)

KL Divergence for Two Gaussians

$$D_{KL}(\mathcal{N}(\mu, \sigma^2)||\mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\mu_i^2 + \sigma_i^2 - \ln \sigma_i^2 - 1)$$

Let's take the case $\boldsymbol{\mu} = 0$

$$D_{KL}(\mathcal{N}(0, \sigma^2)||\mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\sigma_i^2 - \ln(\sigma_i^2) - 1)$$

This expression is minimized when $\boldsymbol{\sigma} = 1$ (which is also what we want!)

The Final VAE Loss Function

We now have all the tools necessary to construct our loss function.

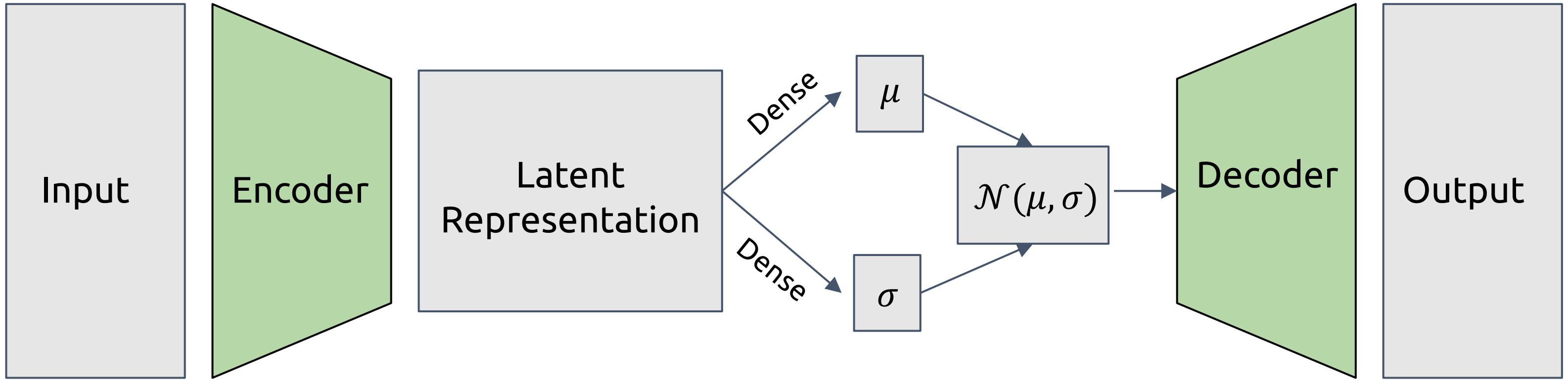
$$L = L_1 + \lambda L_2 \qquad\qquad \lambda \in [0, \infty]$$

Which turns into this:

$$L = ||x - \hat{x}||_2^2 + \lambda D_{KL}(\mathcal{N}(\mu, \sigma), \mathcal{N}(0, 1))$$

Putting it all together

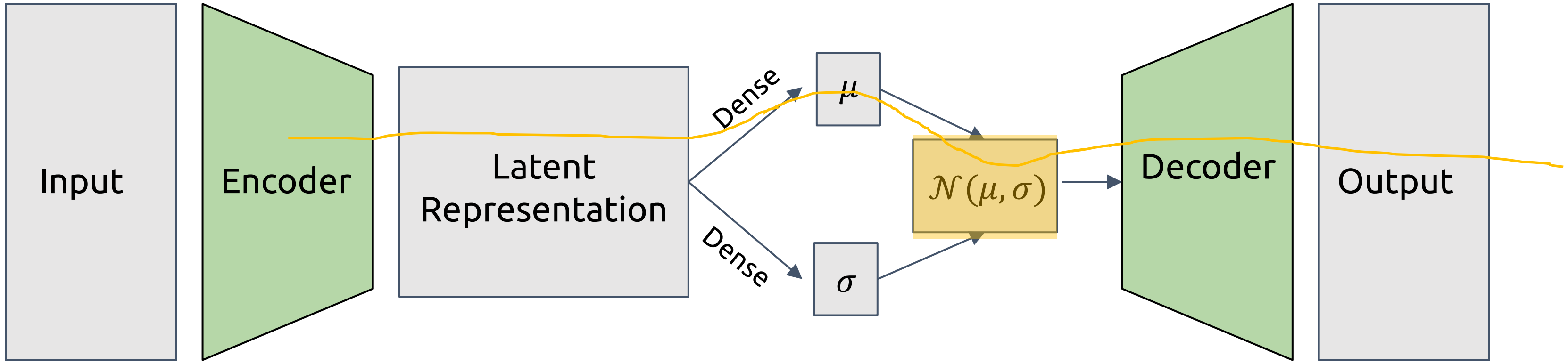
$$L = ||x - \hat{x}||_2^2 + \lambda D_{KL}(\mathcal{N}(\mu, \sigma), \mathcal{N}(0, 1))$$



Ah, but there's a catch:

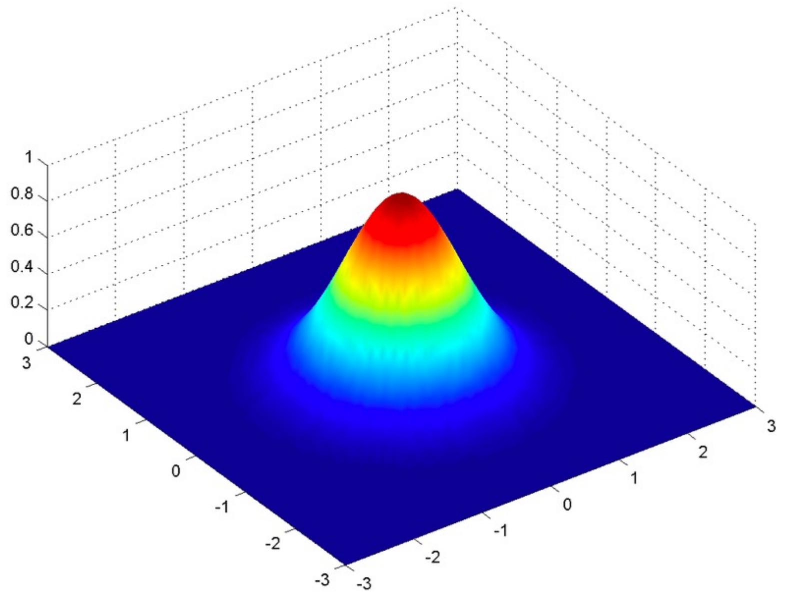
Can anyone guess?

- To update the weights of the encoder, we have to backprop through a random sampling operation
- Sampling a random value seems not differentiable...



Remember our sampling strategy for Gaussian?

- The Gaussian Distribution
  - $p(x \mid \mu, \sigma) = \mathcal{N}(\mu, \sigma)(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
  - Sampling:
    - [Sample from the unit normal distribution]() $\rightarrow\ r \sim \mathcal{N}(0, 1)$
    - Return $\mu + r\sigma$



The Reparameterization Trick

A nice property of Gaussian distributions: if we sample $z \sim \mathcal{N}(\mu, \sigma)$ we can rewrite it as:

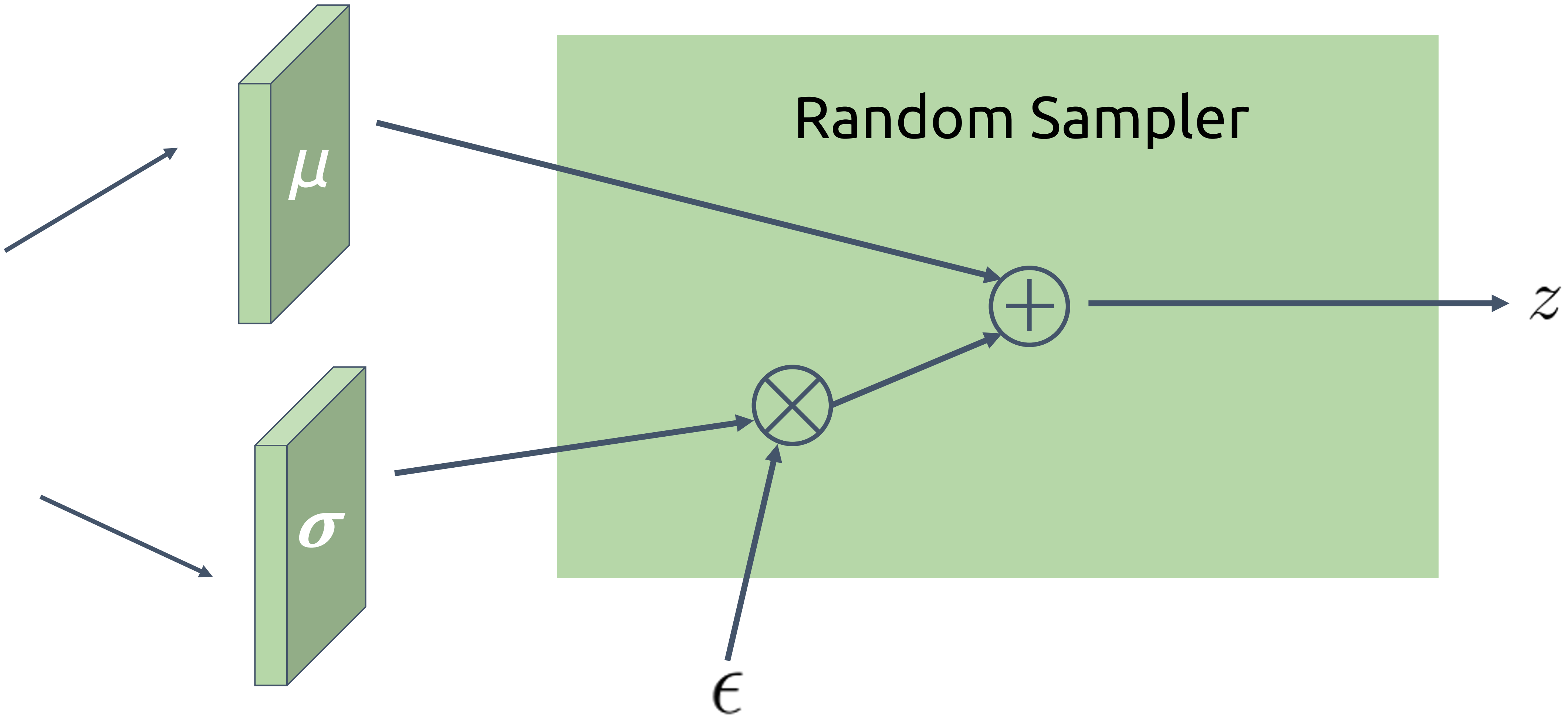
$$z = \mu + \epsilon \cdot \sigma$$

Where

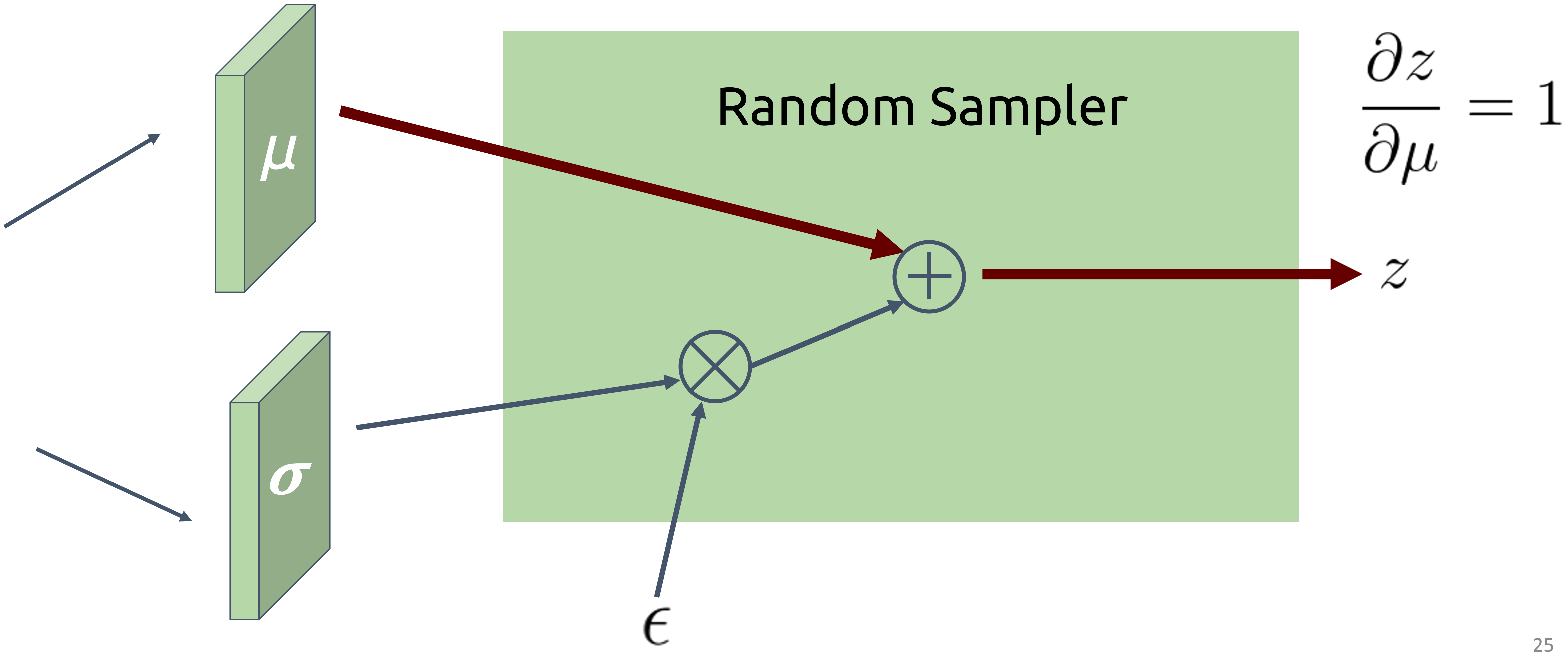
$$\epsilon \sim \mathcal{N}(0, 1)$$

- The random sampling no longer depends on learnable parameters
- This allows us to do backpropagation

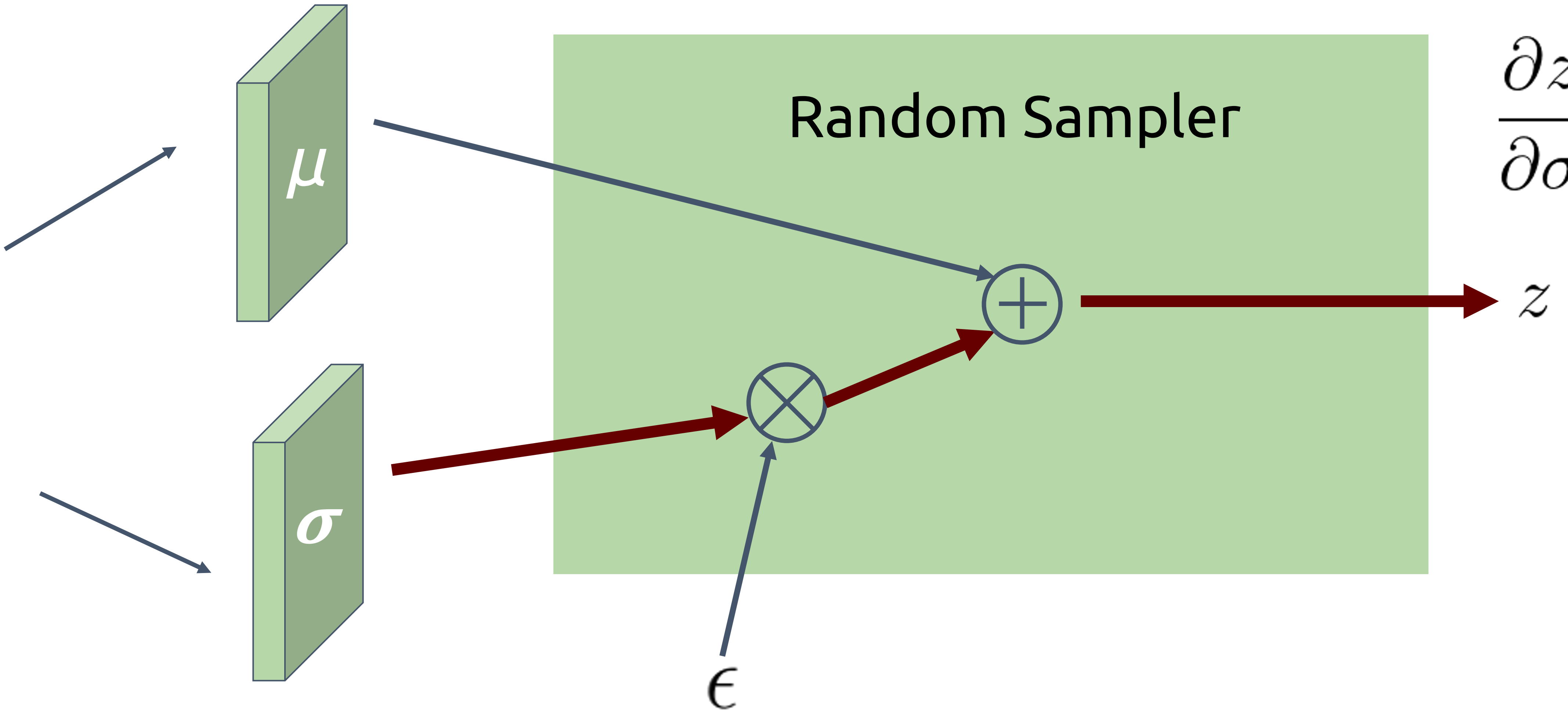
Random Sampler with Reparameterization Trick



Random Sampler with Reparameterization Trick

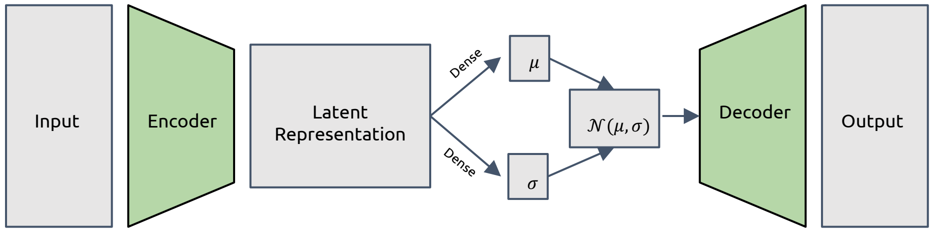


Random Sampler with Reparameterization Trick



$$\frac{\partial z}{\partial \sigma} = \epsilon$$

One more practical detail



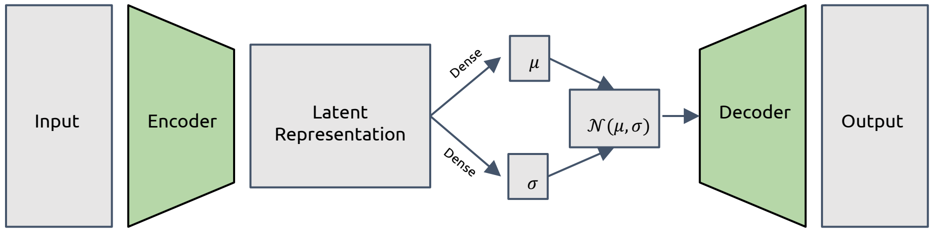
Let's again consider our sampling operation

$$z \sim \mathcal{N}(\mu, \sigma)$$

$$\mu_i \in [-\infty, \infty] \qquad \sigma_i \in [0, \infty]$$

- Nothing prevents the neural network from outputting ***negative*** values for the standard deviation.
- Instead of predicting $\boldsymbol{\sigma}$, we will instead predict $\log(\sigma^2)$. This ensures that every $\sigma_i \in [0, \infty]$
  - i.e. just treat the output of the Dense layer as if it is $\log(\sigma^2)$

One more practical detail



Let's again consider our sampling operation

Any questions?

$$z \sim \mathcal{N}(\mu, \sigma)$$

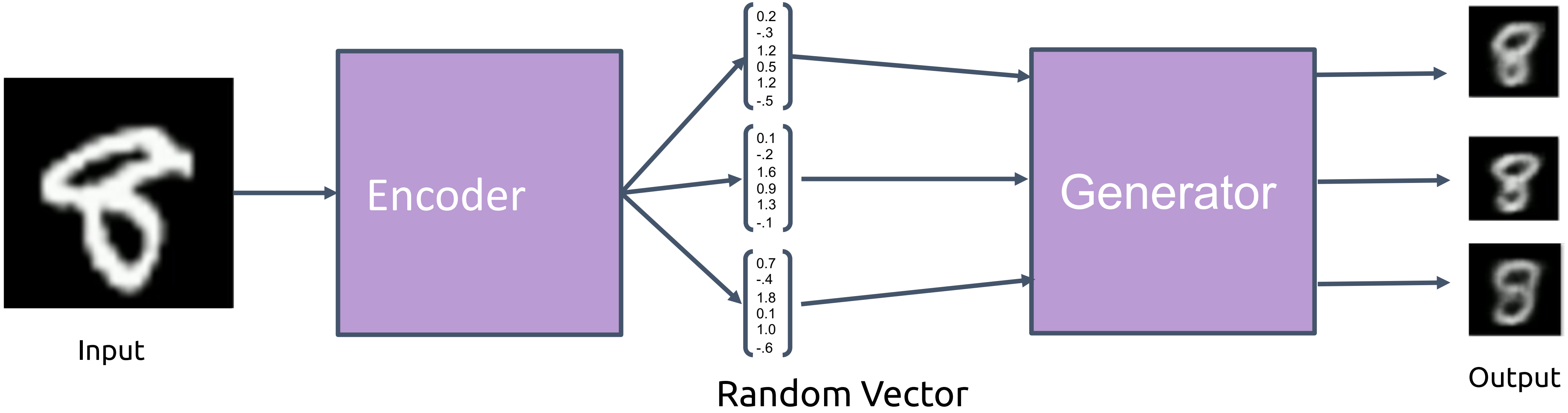
$$\mu_i \in [-\infty, \infty] \qquad \sigma_i \in [0, \infty]$$

- Instead of predicting $\boldsymbol{\sigma}$, we will instead predict $\log(\sigma^2)$. This ensures that every $\sigma_i \in [0, \infty]$
  - i.e. just treat the output of the Dense layer as if it is $\log(\sigma^2)$

$$\boxed{D_{KL}(\mathcal{N}(\mu,\sigma^2)||\mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\mu_i^2 + \sigma_i^2 - \ln \sigma_i^2 - 1)}$$

Sampling from a VAE

- We can use a trained VAE to generate random variants of an input data point...



Sampling from a VAE

... But ultimately, we want to draw random samples from a VAE

$$\begin{bmatrix} 0.2 \\ 0.5 \\ -.3 \\ -.2 \\ 0.8 \end{bmatrix} \xrightarrow{\text{VAE}}$$

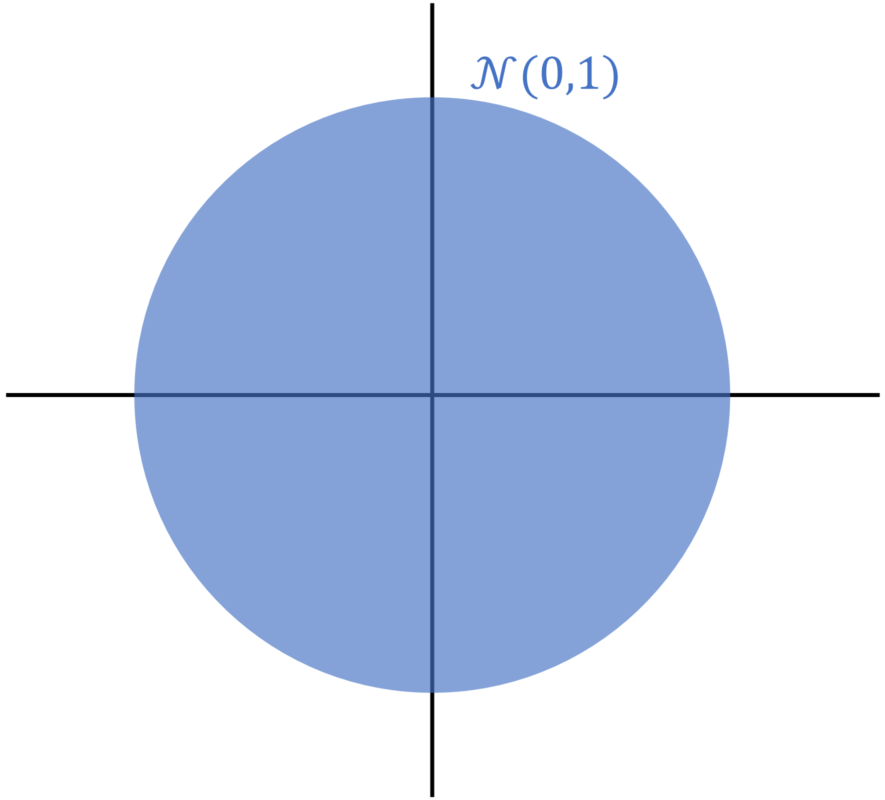
Random Noise

Random Sample

How can we do this?

This is where our particular choice of training loss will pay off

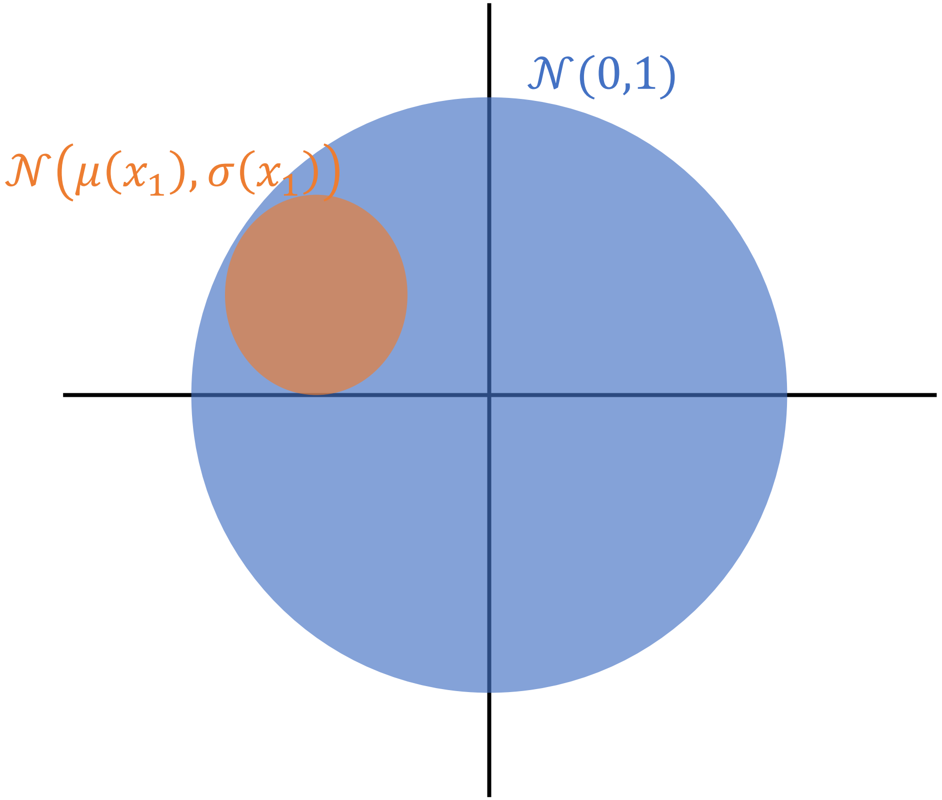
Encoding different points into latent space



Let this circle represent the probability density of a unit Gaussian in latent space

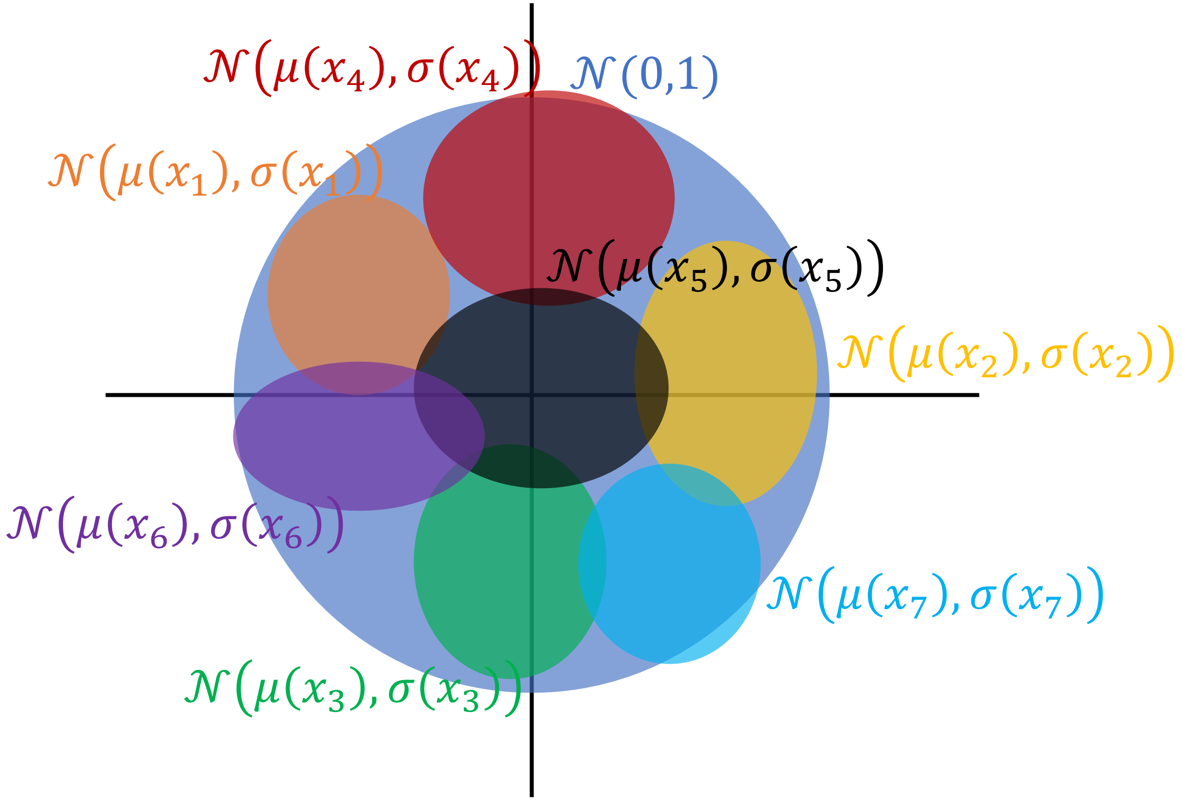
Encoding different points into latent space

Let this circle represent the probability density of the $\mathcal{N}(\mu, \sigma)$ distribution that the encoder predicts given an input data point x_1



Encoding different points into latent space

$$L = ||x - \hat{x}||_2^2 + \lambda D_{KL}(\mathcal{N}(\mu, \sigma), \mathcal{N}(0,1))$$



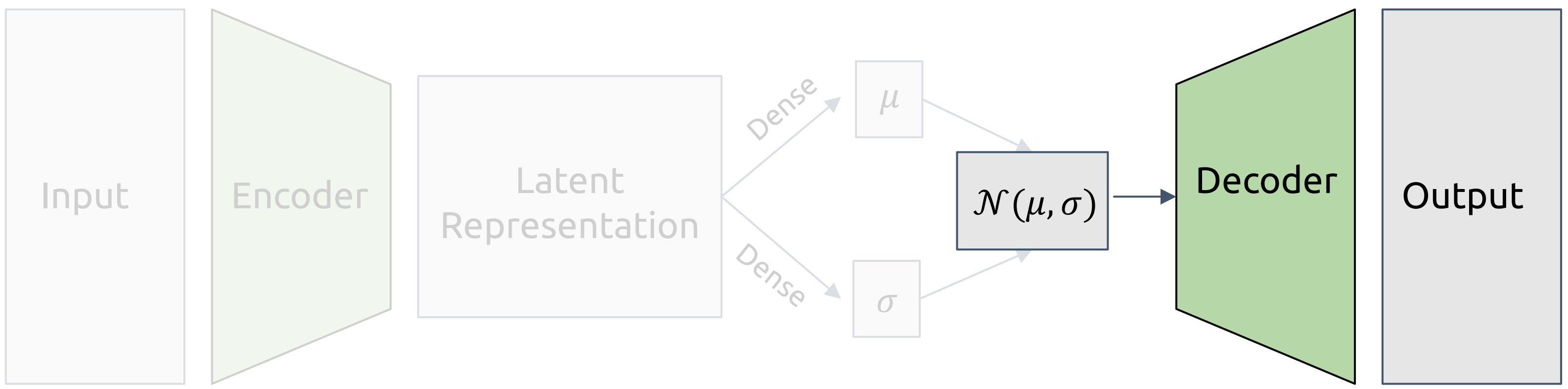
Because of our KL divergence loss, the $\mathcal{N}(\mu, \sigma)$ for any input data point has to be somewhat similar to $\mathcal{N}(0,1)$

So, if we sample a point from $\mathcal{N}(0,1)$, it is very likely to fall within one of these encoded distributions from the training set…

…which the decoder has been trained to reconstruct well!

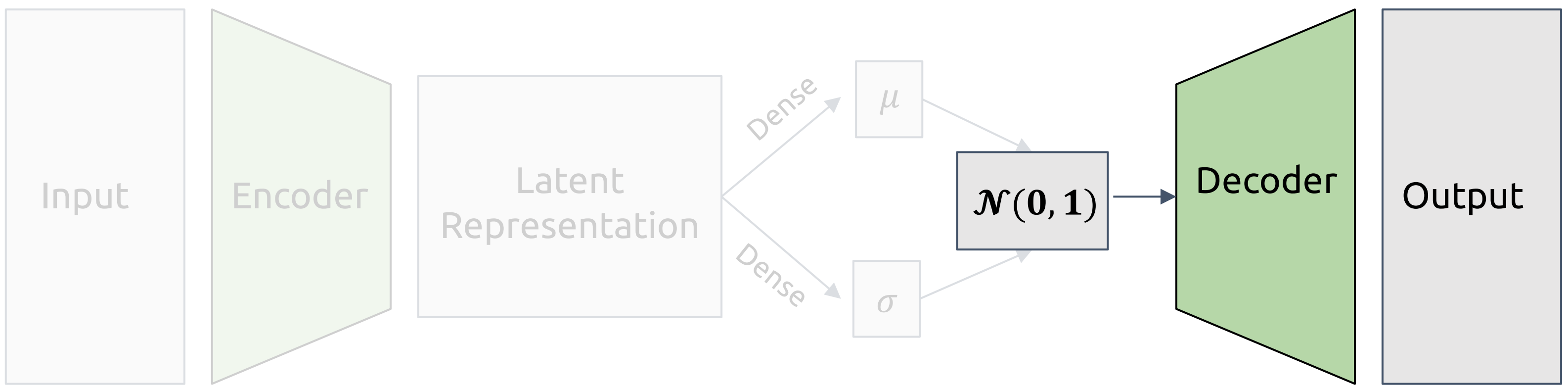
Sampling from a VAE

So what do we do?



- Discard this part of the network…

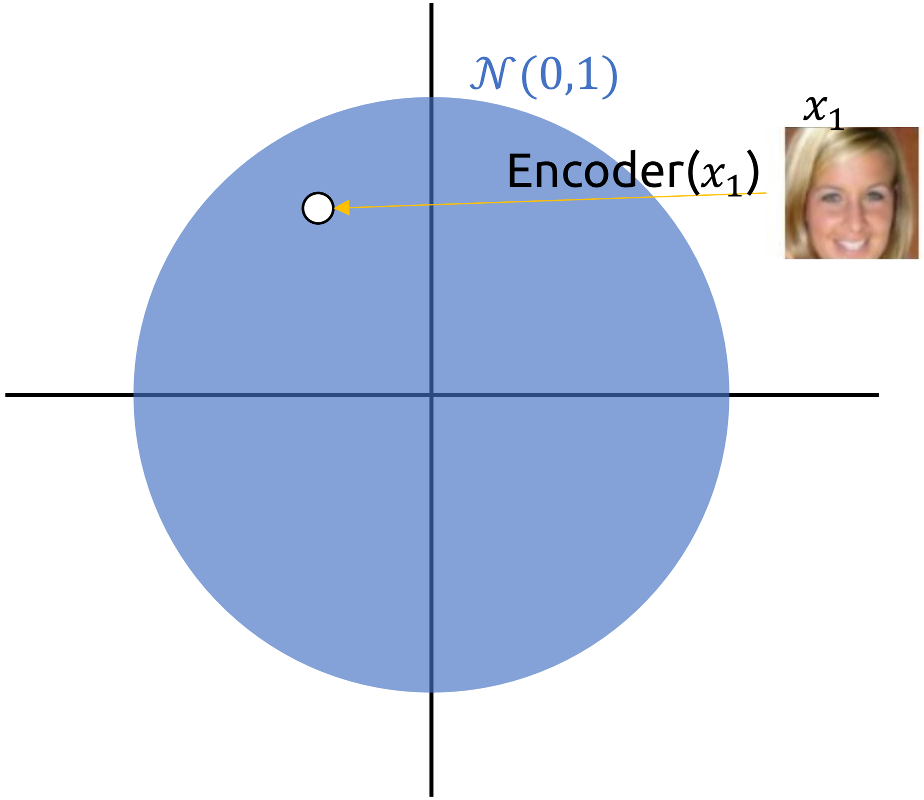
Sampling from a VAE



- Discard this part of the network…
- …and set $(\mu, \sigma) = (0, 1)$

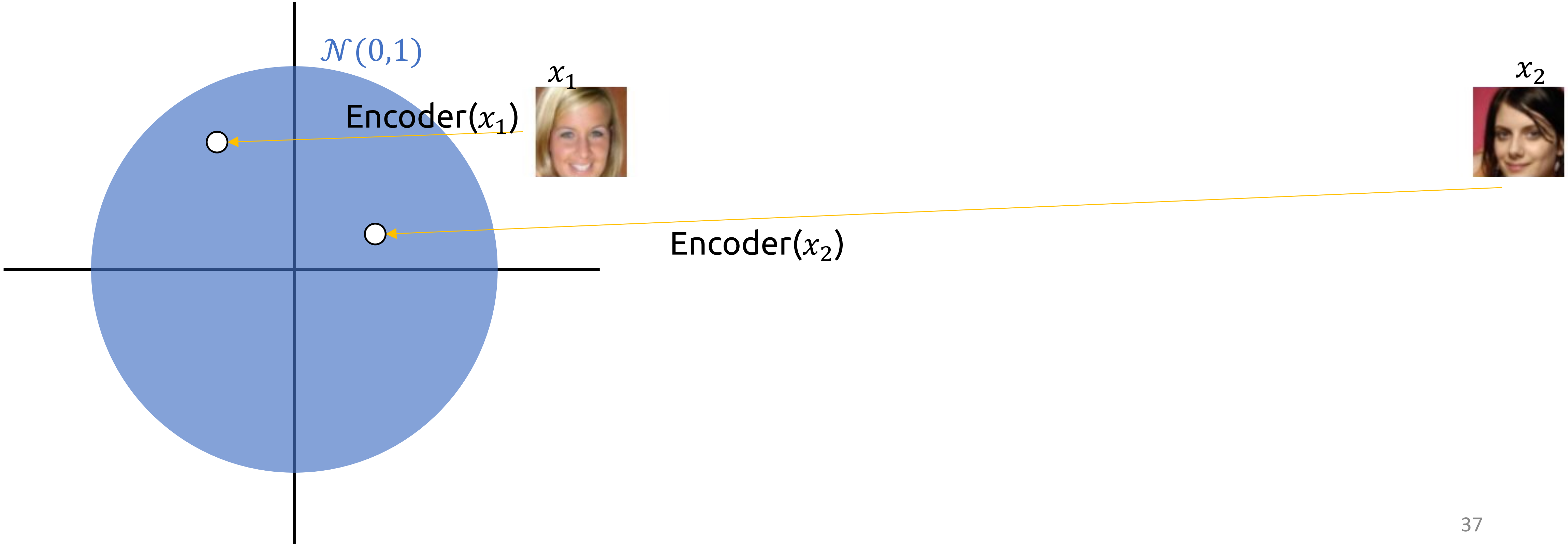
Latent Space Interpolation

- Trace a linear path between two points in latent space, put all points along the path into the decoder



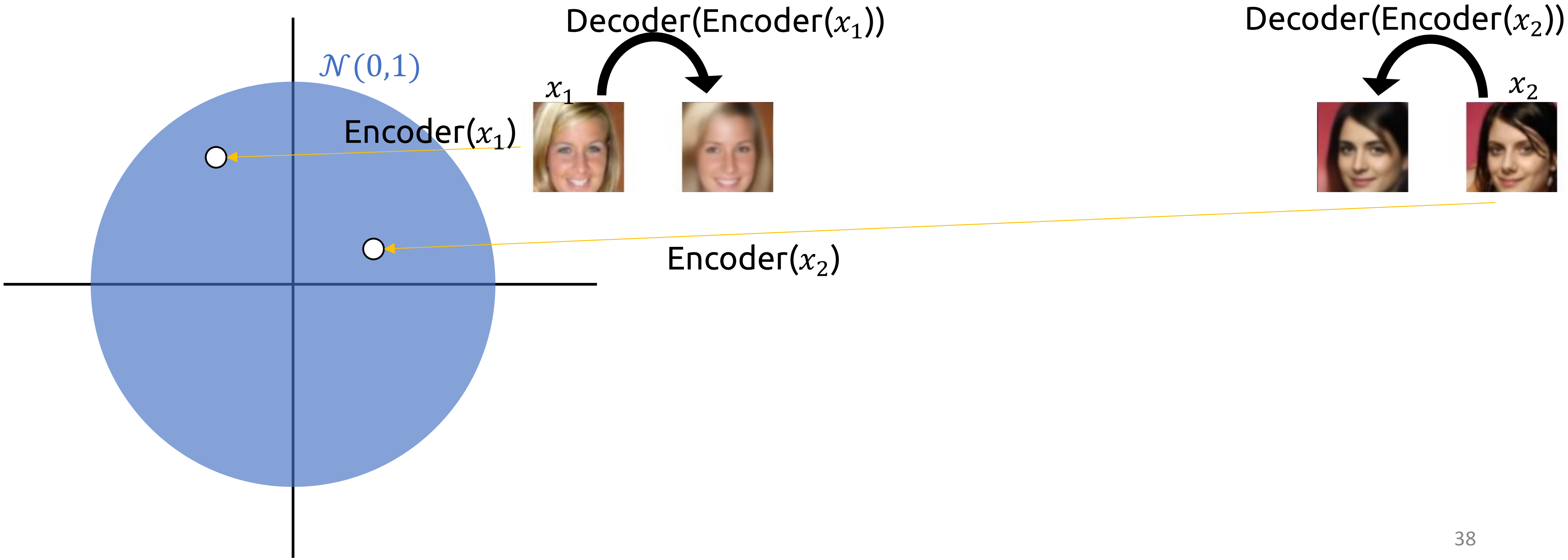
Latent Space Interpolation

- Trace a linear path between two points in latent space, put all points along the path into the decoder



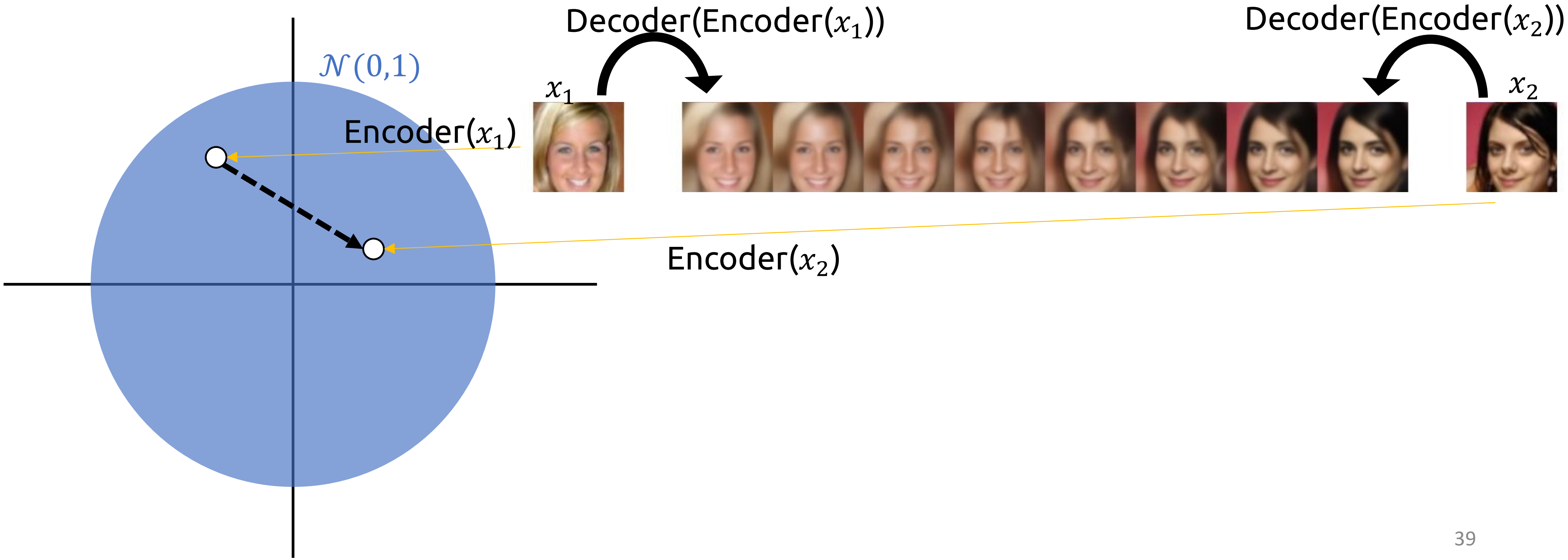
Latent Space Interpolation

- Trace a linear path between two points in latent space, put all points along the path into the decoder



Latent Space Interpolation

- Trace a linear path between two points in latent space, put all points along the path into the decoder

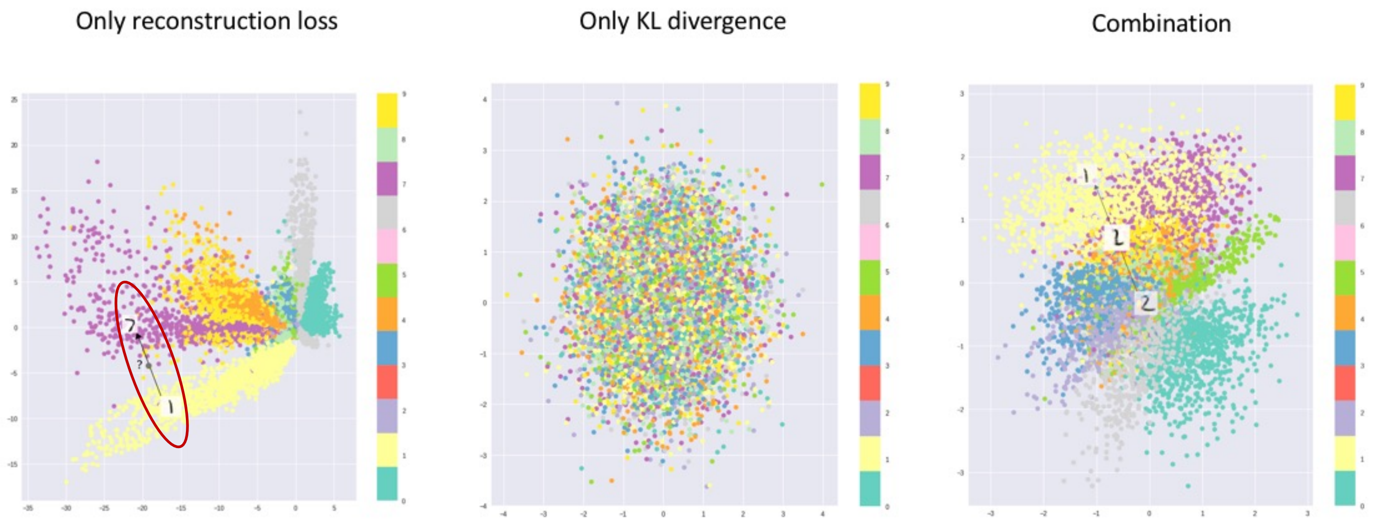


Any questions?

Latent Space Interpolation

- Can also try it with a regular autoencoder
  - Doesn't work as well
  - ***Why not?***
  - The KL divergence loss regularizes the shape of the latent space. Without it, a regular autoencoder might have "empty" pockets of latent space

Linear interpolation has to cross a pocket of empty space (where the behavior of the decoder is not well defined)



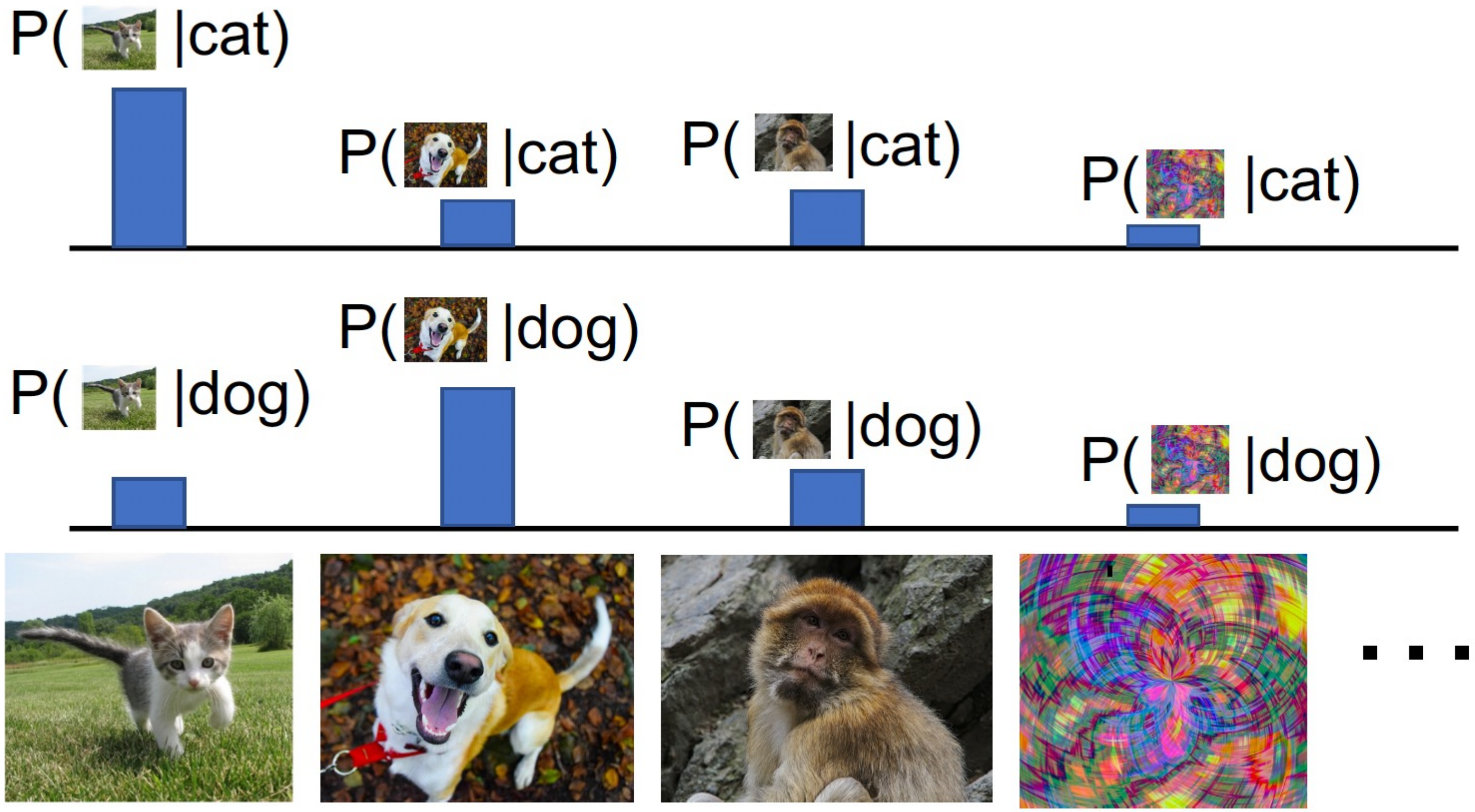
https://www.jeremyjordan.me/variational-autoencoders/

Discriminative vs Generative Models

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

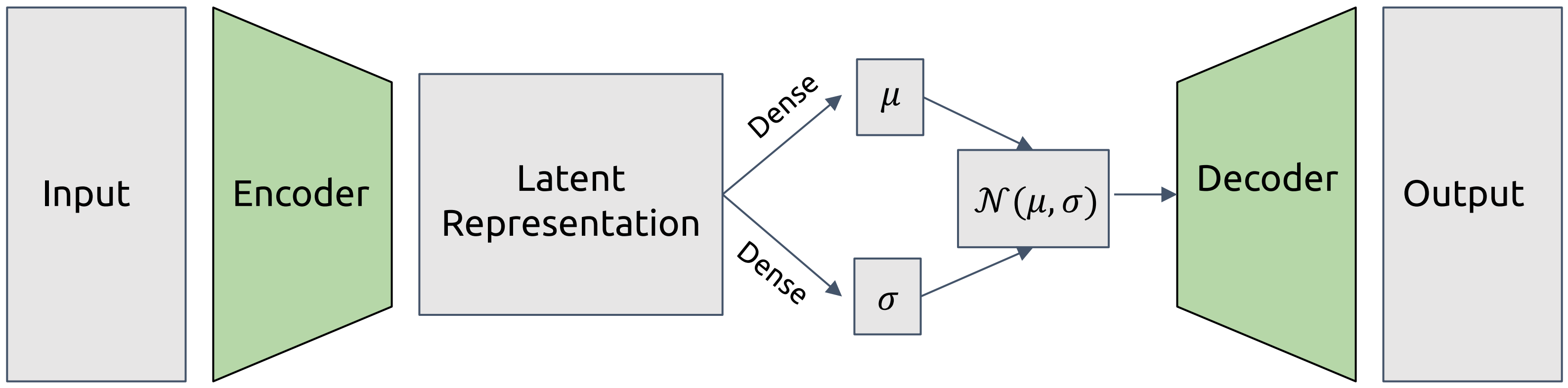


Conditional Generative Model: Each possible label induces a competition among all images

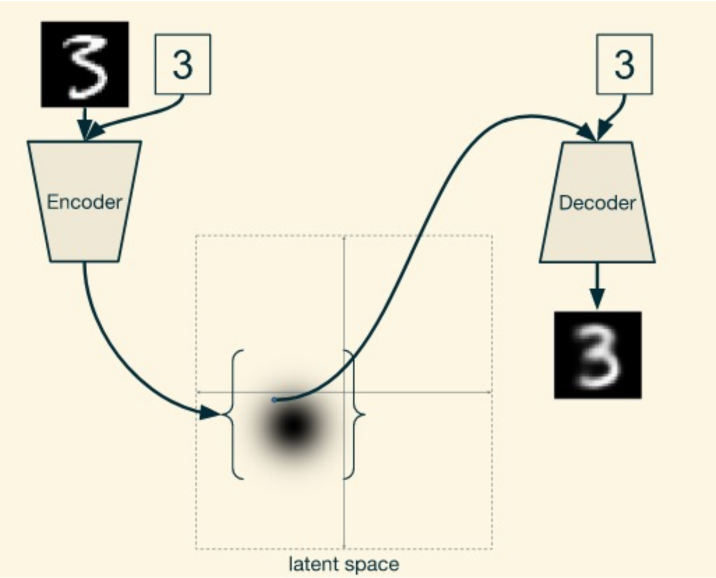
Credit: UMich EECS498

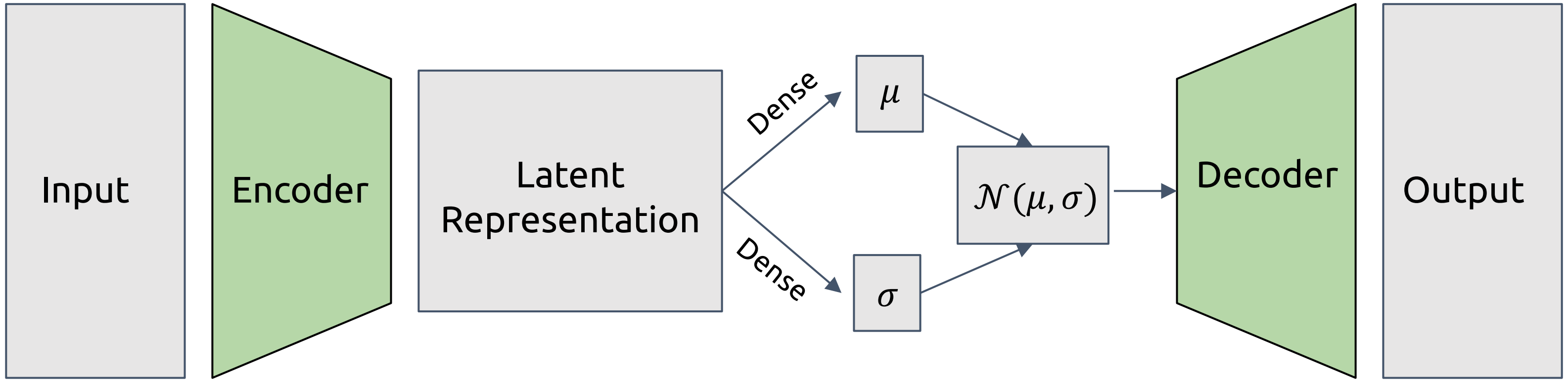
Conditional VAE

Any ideas?



Conditional VAE





https://towardsdatascience.com/understanding-conditional-variational-autoencoders-cd62b4f57bf8

VAE output

https://towardsdatascience.com/what-the-heck-are-vae-gans-17b86023588a

What's the issue here?

Why?

Why are VAE samples blurry?

- Our reconstruction loss is the culprit
- Mean Square Error (MSE) loss looks at each pixel in isolation
- If no pixel is too far from its target value, the loss won't be too bad
- Individual pixels look OK, but larger-scale features in the image aren't recognizable

- ***Solutions?***
  - Let's choose a different reconstruction loss!

Input

VAE reconstruction

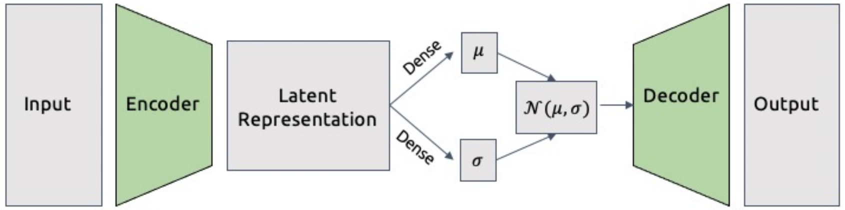
https://towardsdatascience.com/what-the-heck-are-vae-gans-17b86023588a

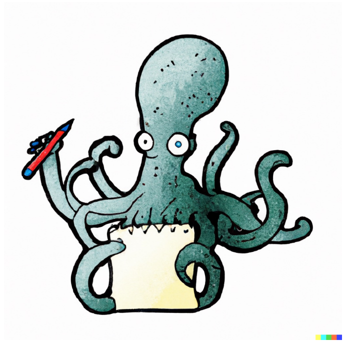
Recap

Variational Autoencoders (VAEs)

- Loss Function
- Reparameterization Trick
- Conditional VAEs

https://towardsdatascience.com/what-the-heck-are-vae-gans-17b860235 88a





Extra Material: Deriving the VAE loss

Full derivation here - https://arxiv.org/pdf/1907.08956.pdf

Variational autoencoder (a generative model)

Unfortunately, z is unknown, so we need to **marginalize** over all possible z:

$$p_\theta(x) = \int p_\theta(x,z)dz = \boxed{\int}\boxed{p_\theta(x|z)}\boxed{p_\theta(z)}\boxed{dz}$$

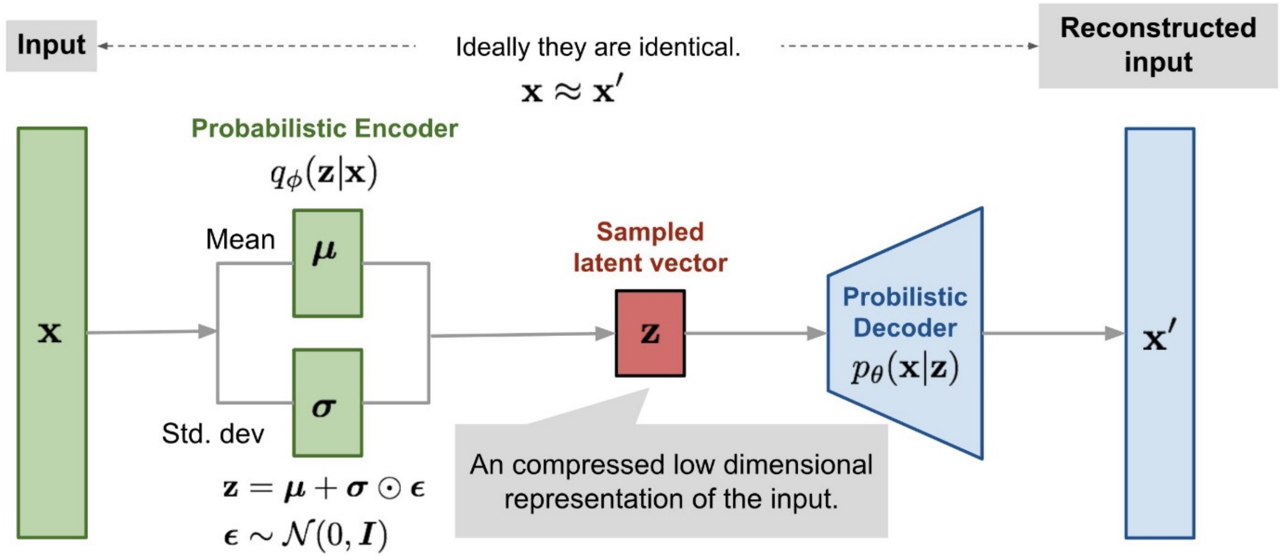
Problem: Impossible to integrate over all z!

How to train this model?

Basic idea: **maximize likelihood of data**

compute with decoder network

we assume Gaussian prior



*Marginalization is a method that requires summing over the possible values of one variable to determine the marginal contribution of another

Credit: UMich EECS498

Variational autoencoder (a generative model)

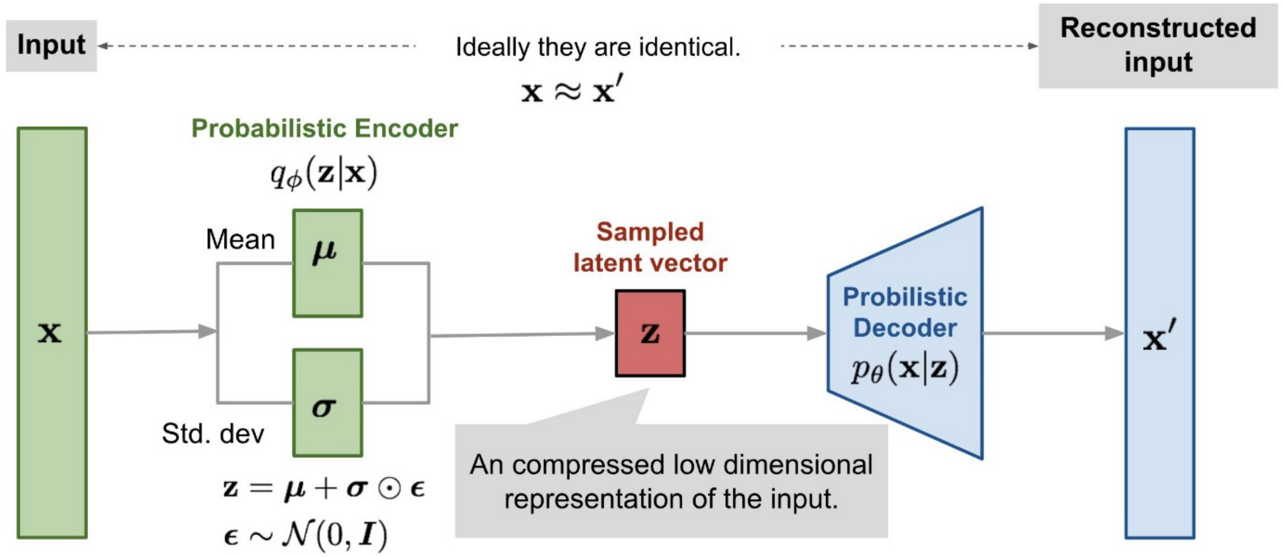
compute with decoder network   we assume Gaussian prior

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)} \approx \frac{p_\theta(x \mid z)p_\theta(z)}{q_\phi(z \mid x)}$$

Train an encoder that learns

$$q_\phi(z \mid x) \approx p_\theta(z \mid x)$$

Idea: Jointly train both encoder and decoder to maximize $p_\theta(x)$!



Recall Bayes Rule:

$$\underbrace{\boxed{P(A|B)}}_{\text{posterior}} = \underbrace{\boxed{P(A)}}_{\text{prior}} \times \frac{\boxed{P(B|A)}\ \text{likelihood}}{\boxed{P(B)}\ \text{marginal}}$$

Credit: UMich EECS498

Variational autoencoder (a generative model)

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)}$$

Bayes' Rule

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)}$$

Take log on each sides

$$= \log \frac{\boxed{p_\theta(x|z)}\boxed{p(z)}\boxed{q_\phi(z|x)}}{\boxed{p_\theta(z|x)}\boxed{q_\phi(z|x)}}$$

Multiply top and bottom by $q_\Phi(z|x)$

$$= \log \boxed{p_\theta(x|z)} - \log \boxed{\frac{q_\phi(z|x)}{p(z)}} + \log \boxed{\frac{q_\phi(z|x)}{p_\theta(z|x)}}$$

Split up using rules for logarithms

Credit: UMich EECS498

Variational autoencoder (a generative model)

$$\log p_\theta(x) = \log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p(z)} + \log \frac{q_\phi(z|x)}{p_\theta(z|x)}$$

We want to maximize the likelihood of the distribution p(x)
So, we reframe the likelihood function by wrapping in expectation w.r.t. z

$$= E_z[\log p_\theta(x|z)] - E_z\left[\log \frac{q_\phi(z|x)}{p(z)}\right] + E_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]$$

$$\log p_\theta(x) = E_{z\sim q_\phi(z|x)}[\log p_\theta(x)]$$

doesn't depend on z

$$= E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right) + D_{KL}(q_\phi(z|x), p_\theta(z|x))$$

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x\in\mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Data reconstruction by the decoder

KL divergence between prior, and samples from the encoder network

KL is >= 0, so dropping this term gives a **lower bound!**

$$\log p_\theta(x) \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

Variational Lower Bound

*Expected value = summation or integration of all possible values of a random variable

Variational autoencoder (a generative model)

Maximum Likelihood Estimation:

$$\log p_\theta(x) \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

Loss:

$$-E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] + D_{KL}\left(q_\phi(z|x), p(z)\right)$$

$$L = ||x - \hat{x}||_2^2 + \lambda D_{KL}(\mathcal{N}(\mu, \sigma), \mathcal{N}(0, 1))$$

[See Deep Learning Book](#)
(Section 5.5)