CSCI 1470

Deep Learning

Eric Ewing

Day 25: Image Generation Day 2: VAEs

Wednesday, 4/2/25

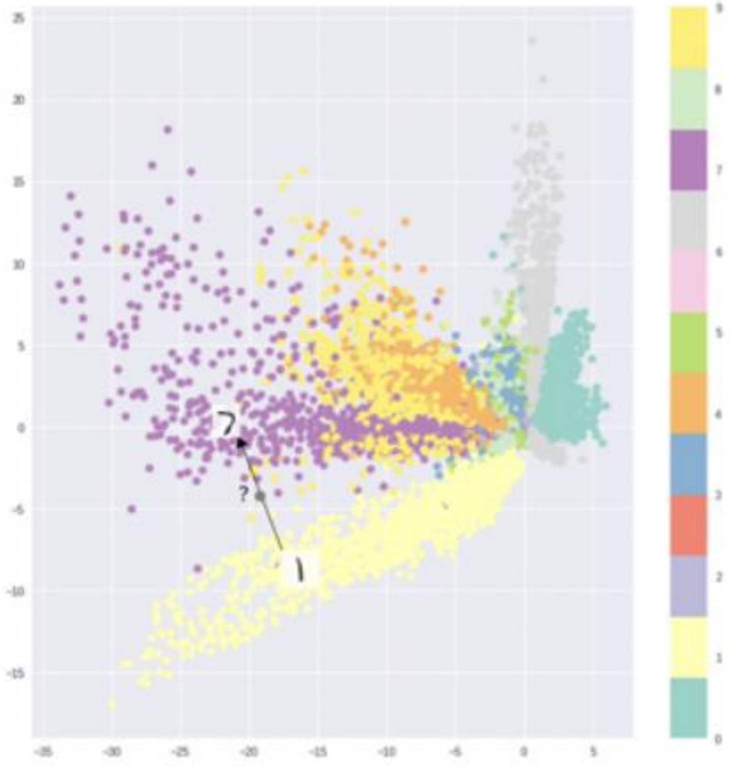
ChatGPT: “Image of space” 4/2/25

Logistics

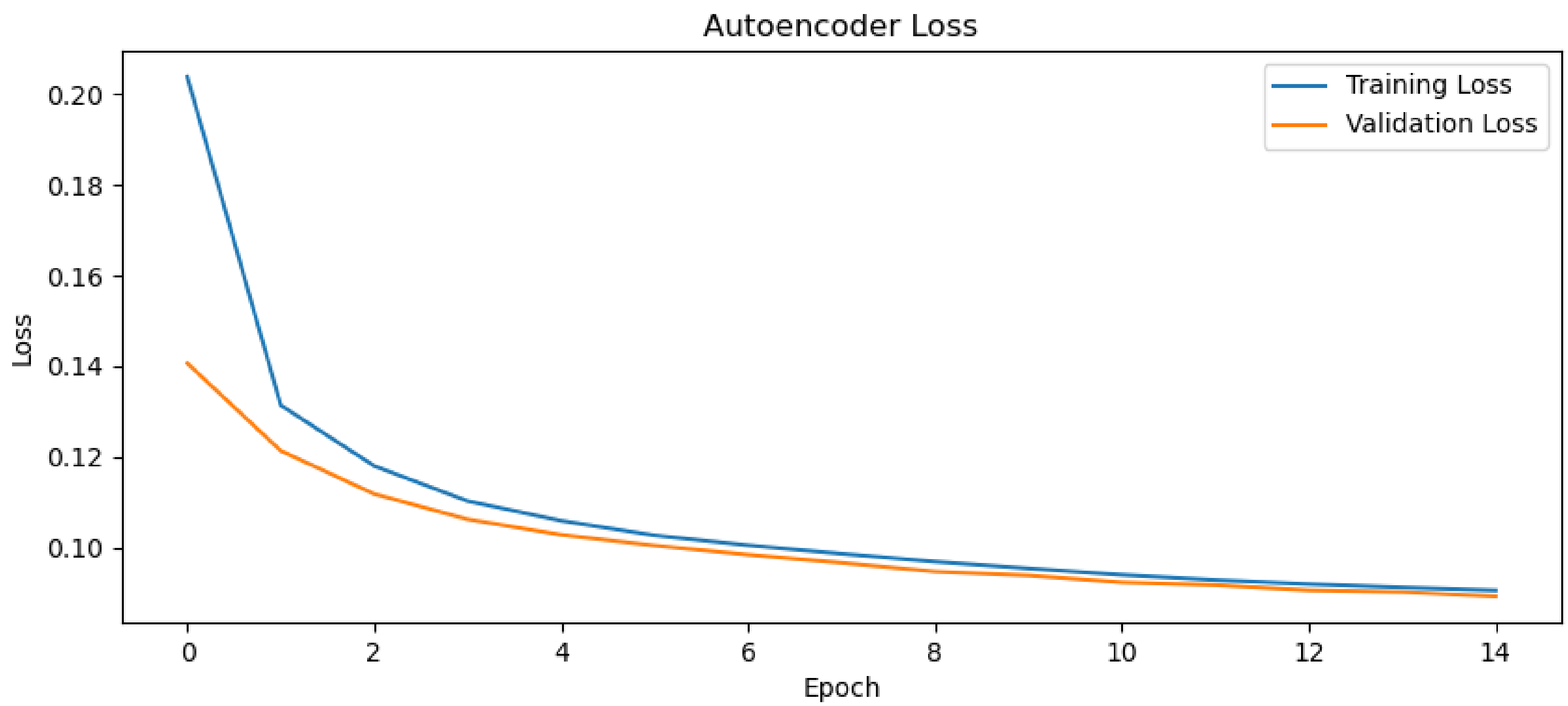
- Weekly Quiz is out
- Make slow and steady progress on your final project!
  - Deep Learning is not something that can all happen at the last minute, data takes time to process, experiments take time to run

Generating Images

- How can we generate a “new” image using a decoder?
- Sample a vector in latent space and send it to the decoder…
- But how do you choose which vector?
- What if you wanted to generate a specific image? How would you find the right vector?

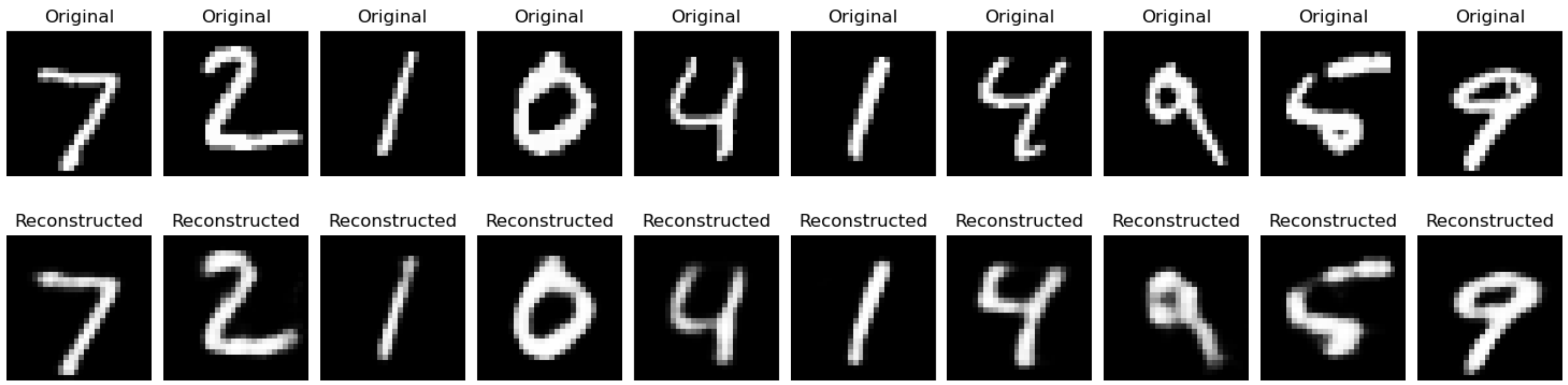


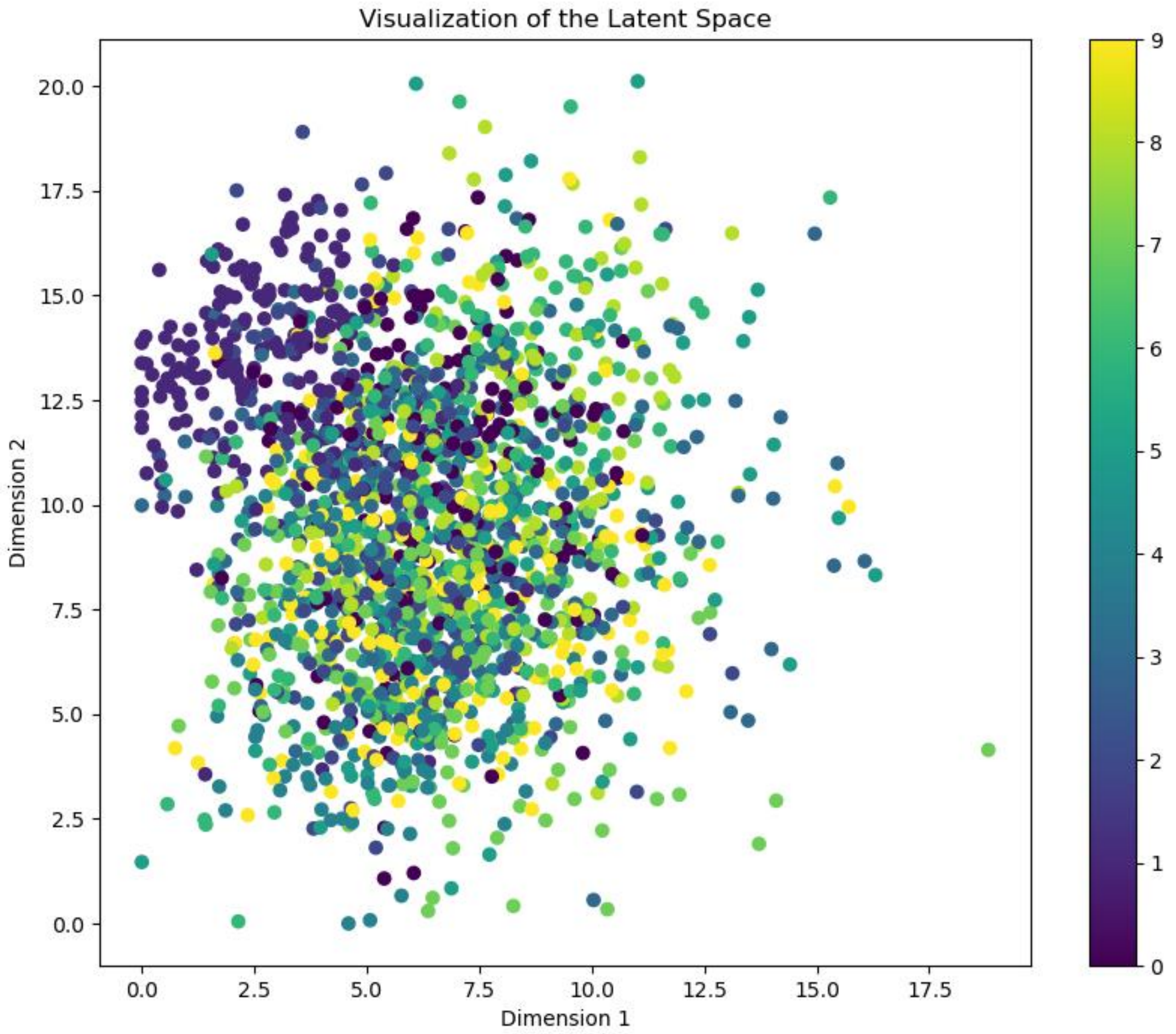
(Encoding size = 32)



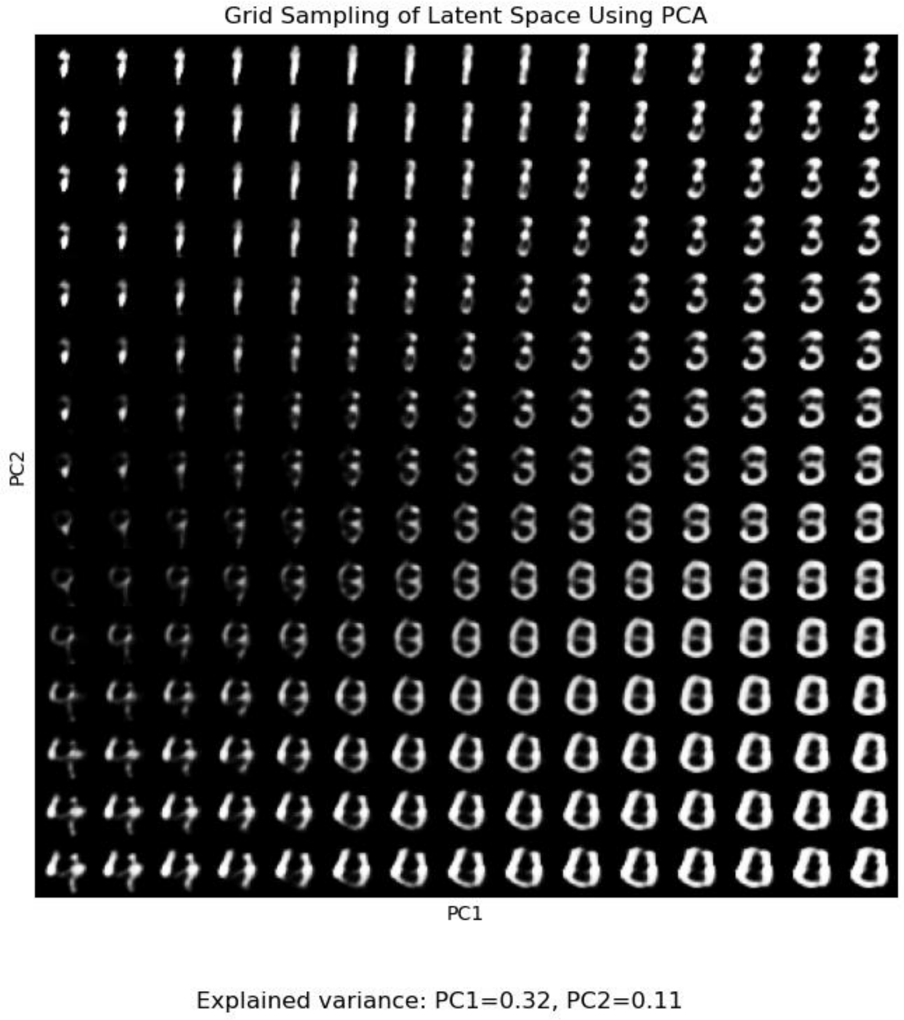
Is this a good autoencoder?

Why is loss alone (even with validation loss) not enough to tell us?





Grid sample latent space and pass to encoder



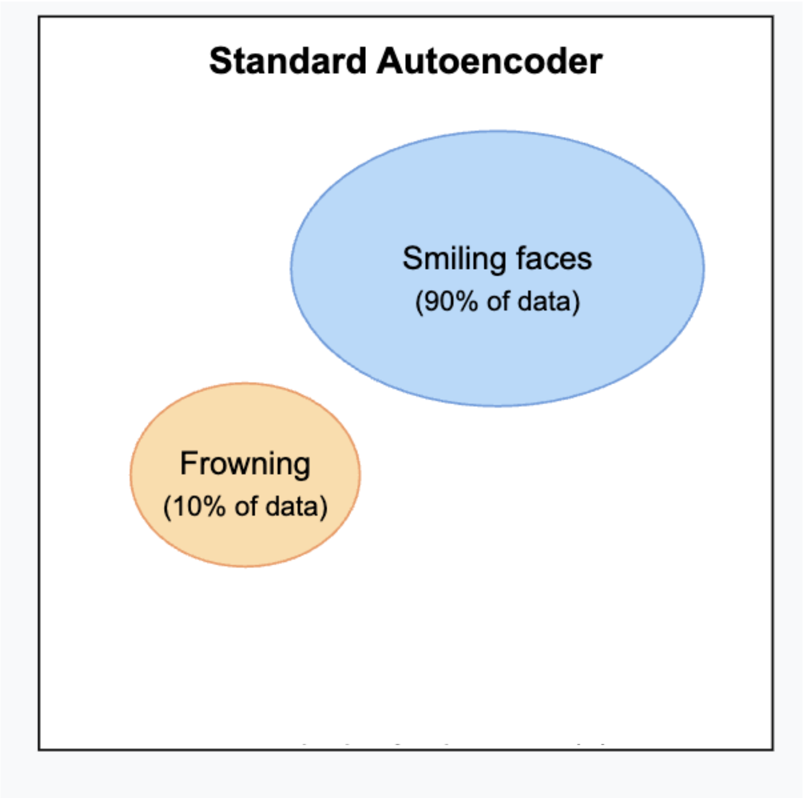
Autoencoders can generate, but are not generative

Recall:

- Discriminative models learn P(y | X)
- Generative models learn P(X)

When we randomly sample, we may get some “invalid” outputs. A generative model could assign these invalid outputs a low probability P(X)

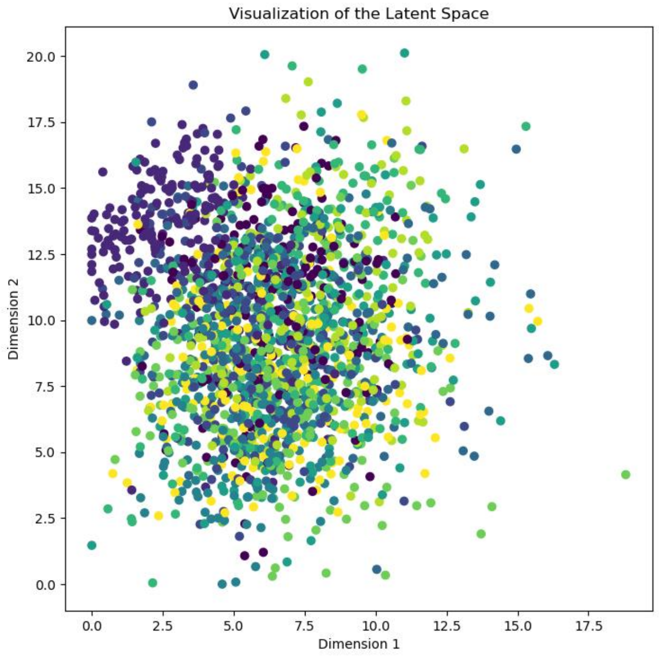
Nothing constrains the latent space of an autoencoder to represent probability distributions

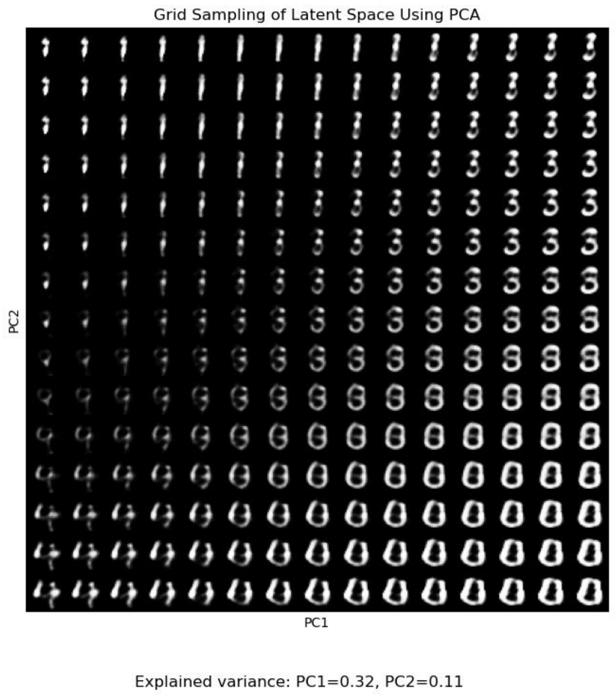


Issues with Autoencoders

- Vectors close together in latent space may not produce similar outputs
- Tend to overfit data (struggle to produce "new" outputs)

How to address issues with overfitting outputs? Try to learn more *variation* in outputs.

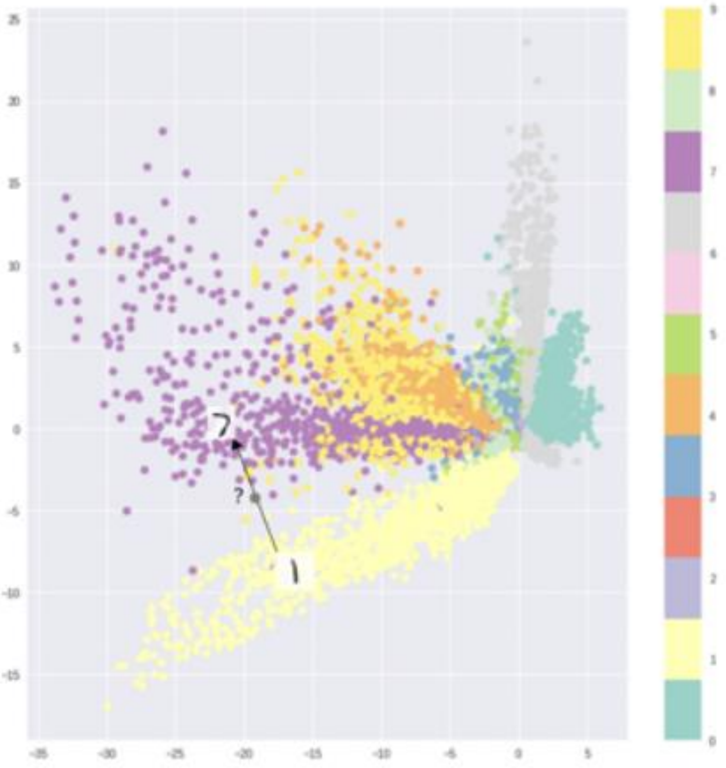




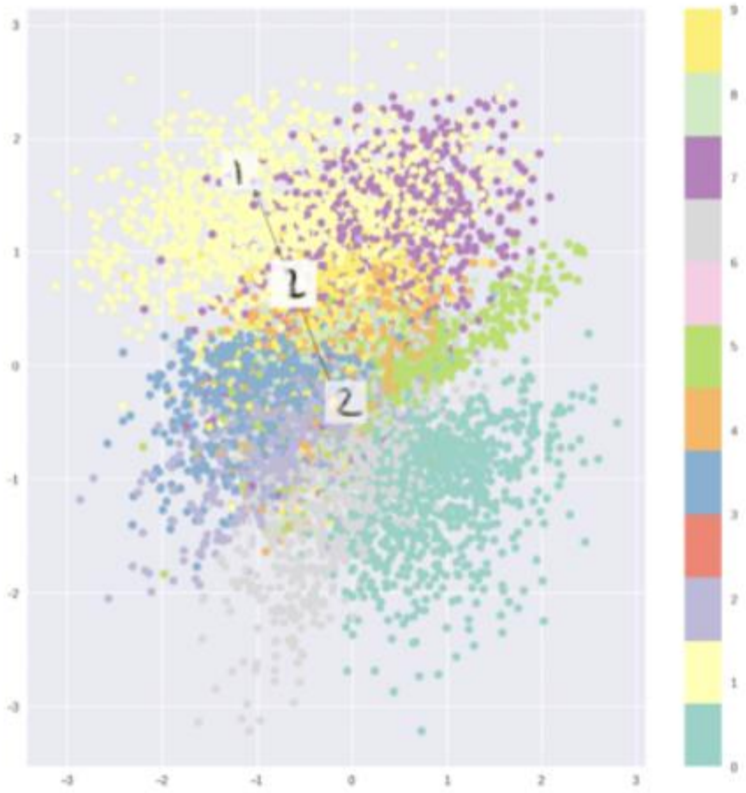
Issues with Autoencoders

What might a better latent space look like for generation?

Autoencoder



Variational Autoencoder



Variational Autoencoders

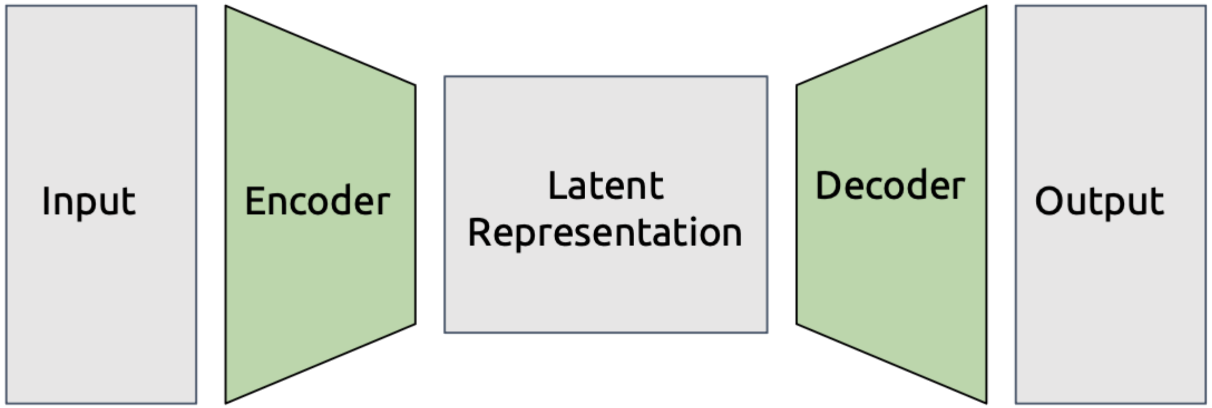
Autoencoder’s goal: Reconstruct the original input

Variational Autoencoder’s goal: Generate a new output that resembles the input

Building up the VAE Architecture

If we were to describe an autoencoder functionally:

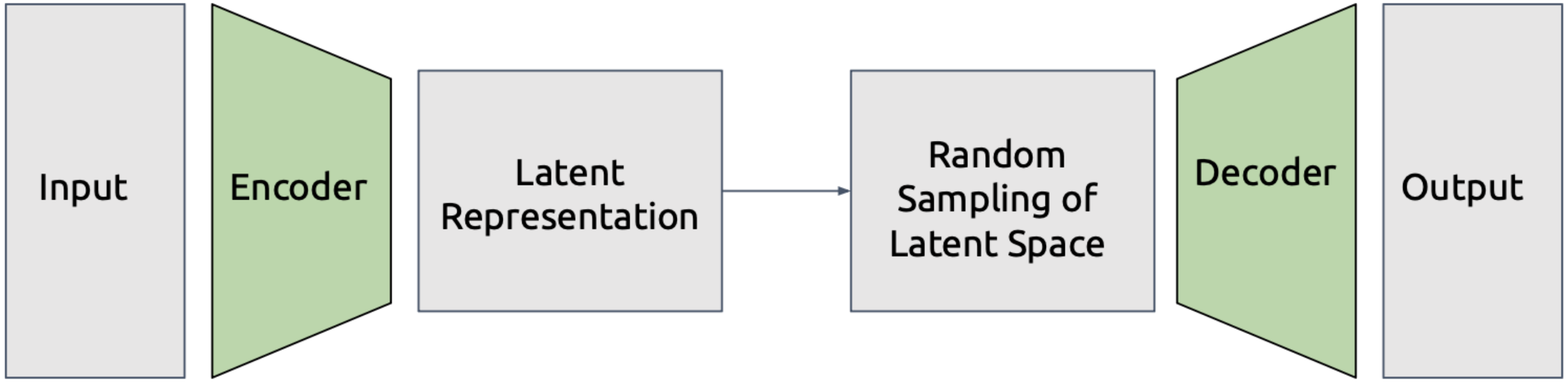
$$\mathtt{Output} = \mathtt{Decoder}(\underbrace{\mathtt{Encoder}(\mathtt{Input})}_{\text{Latent Representation}})$$



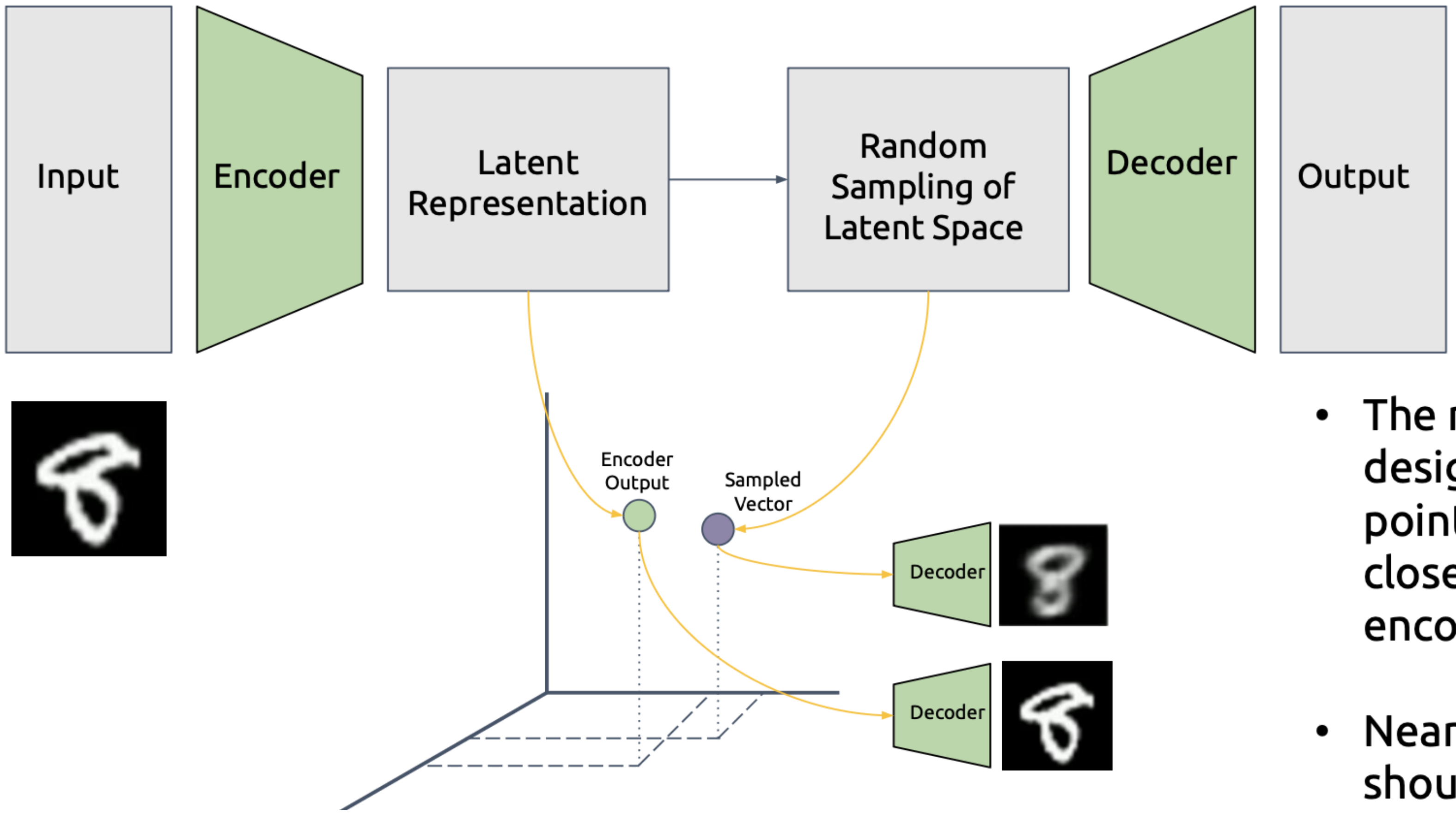
Building up the VAE Architecture

For variational autoencoders, we also do a random sampling operation at the bottleneck

```
Output = Decoder(random_sample(Encoder(Input)))
```



How does random sampling in latent space lead to variation?



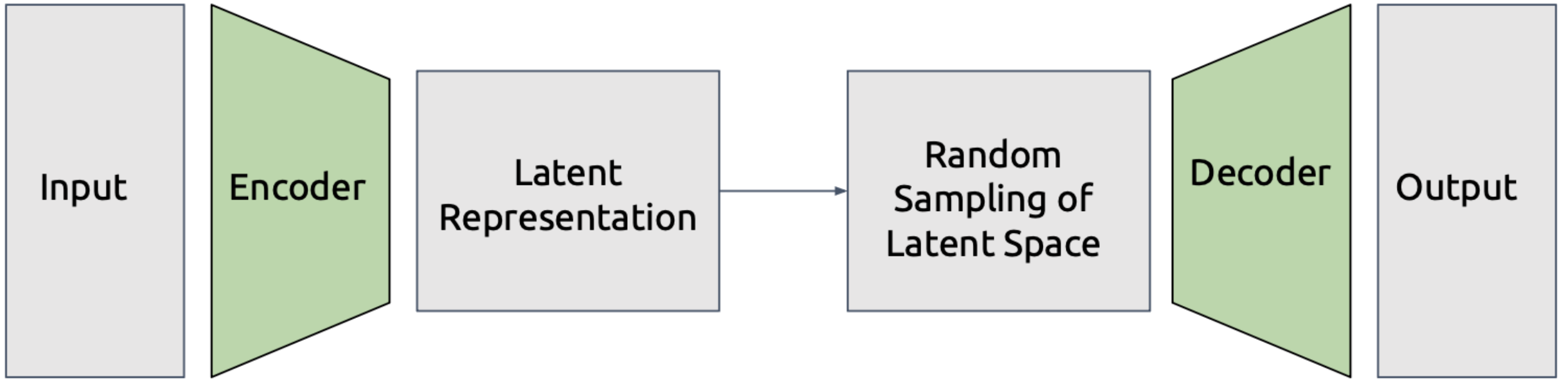
- The random sampling should be designed to produce random points in latent space that are close to the output of the encoder
- Nearby points in the latent space should decode to similar images

How should **`random_sample`** be defined?

`Output = Decoder(`**`random_sample`**`(Encoder(Input)))`

- We want the sample to be close to the encoder output
- One option: sample from a Gaussian centered at `Encoder(Input)`

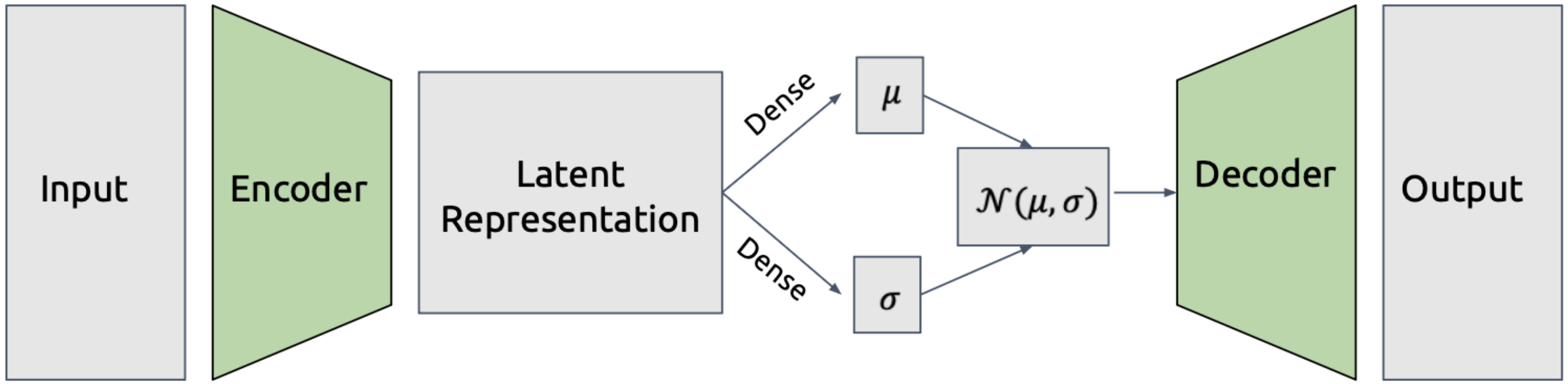
What can we modify?



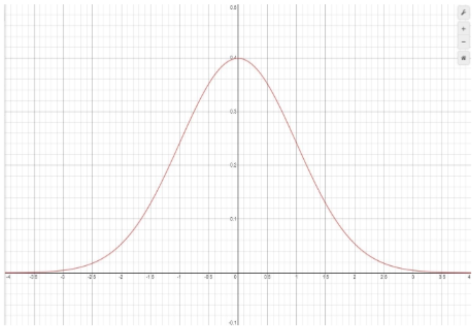
How should `random_sample` be defined?

`Output = Decoder(`**`random_sample`**`(Encoder(Input)))`

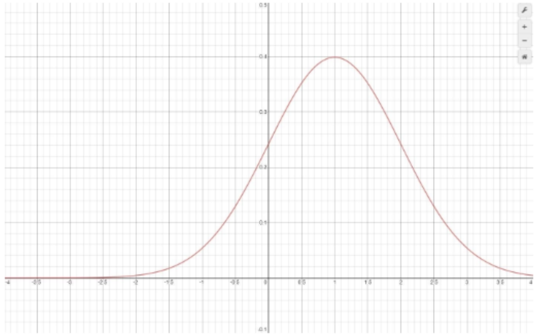
- We want the sample to be close to the encoder output
- One option: sample from a Gaussian centered at `Encoder(Input)`
- Use two dense layers to convert the encoder output into the mean and standard deviation of the Gaussian



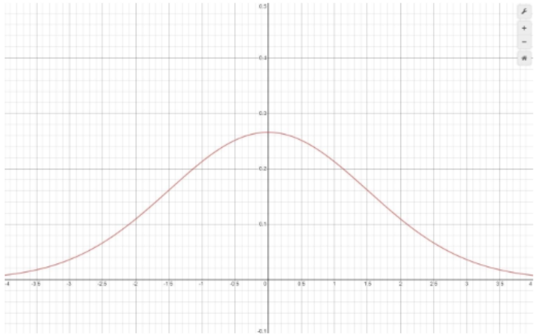
How should `random_sample` be defined?



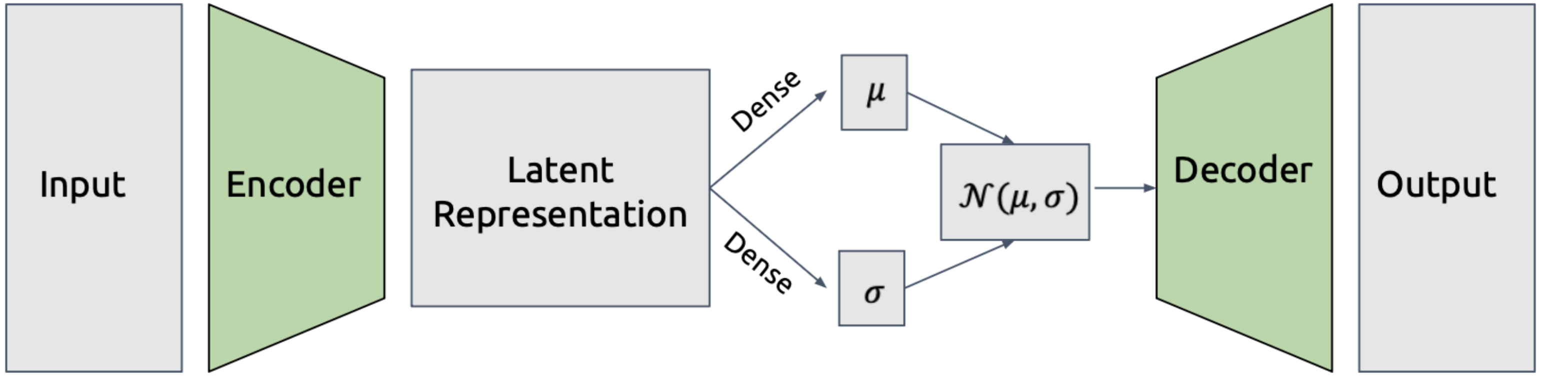
$\mu = 0$
$\sigma = 1$



$\mu = 1$
$\sigma = 1$



$\mu = 0$
$\sigma = 1.5$

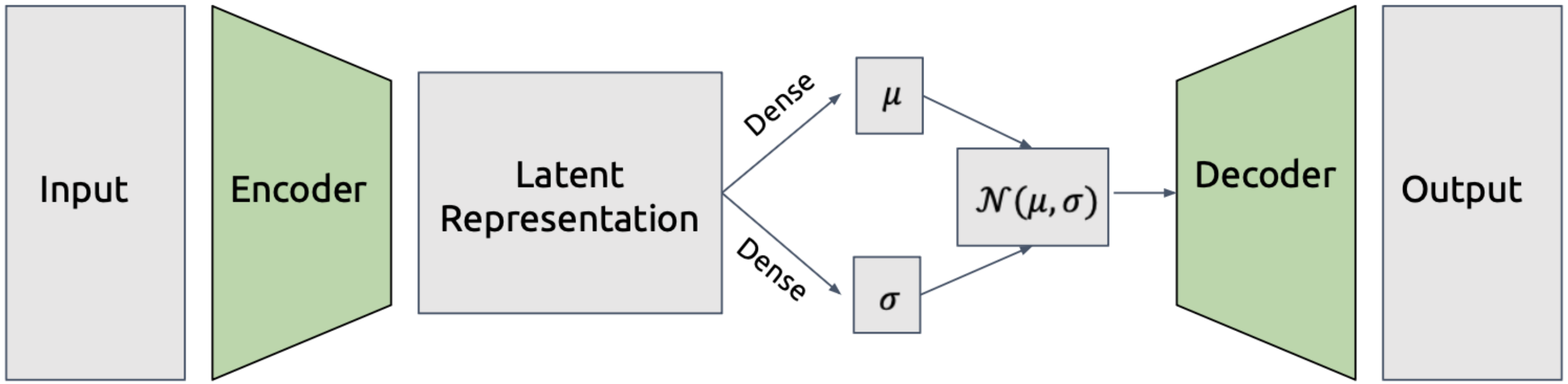


Training a VAE

Two goals:

1. Reproduce an output similar to the input (Input ≈ Output)
2. Have some variation in our output (Input $\neq$ Output)

- Seems like two conflicting goals!
- How do we resolve these two goals?



Weighted Combination of Losses

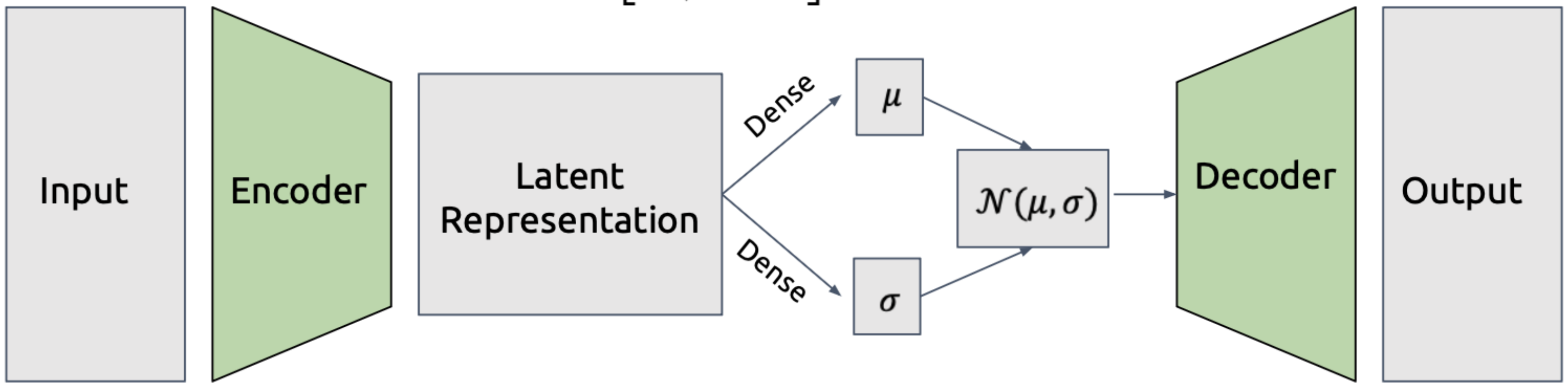
L_1 = loss associated with producing output similar to input

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

$$L = L_1 + \lambda L_2$$

Total Loss:

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



VAE Losses

L_1 is easy, we've seen this before

$$L_1(x, \hat{x}) = \left|\left|x - \hat{x}\right|\right|_2 \text{ (MSE)}$$

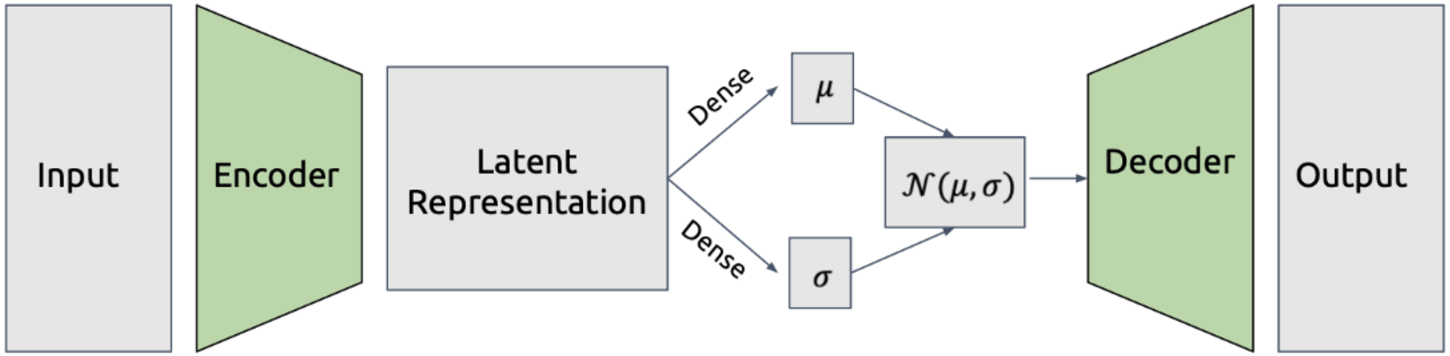
But what is L_2? How do we measure how much variation our output has?

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

Defining the Variation Loss

Whatever our loss function, it needs to encourage $\sigma > 0$, or else the model will force σ to 0 in an effort to create the best recreations possible.

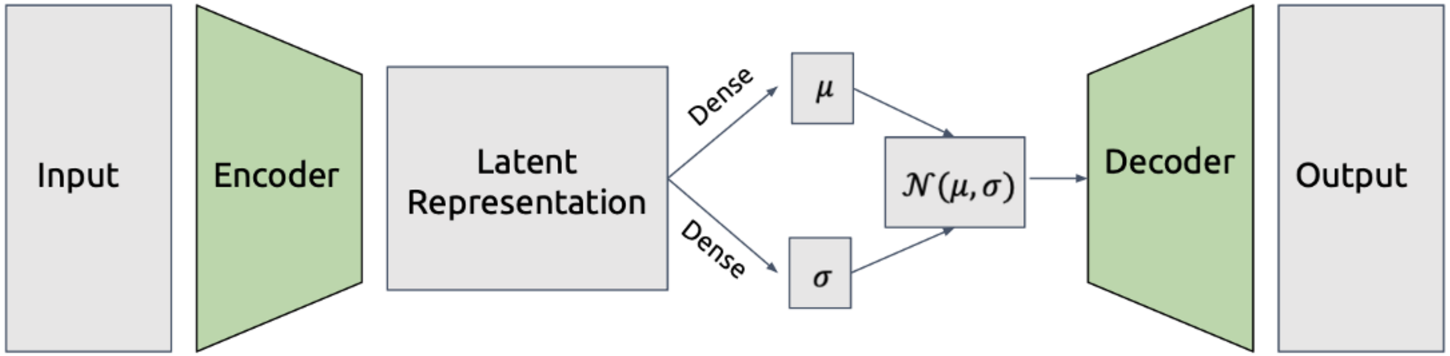
If $\sigma = 0$, then the VAE will behave the same as an autoencoder!



Defining the Variation Loss

Whatever our loss function, it needs to encourage $\sigma > 0$, or else the model will force σ to 0 in an effort to create the best recreations possible.

But it can't be too big… because too much variation will create poor reconstructions.

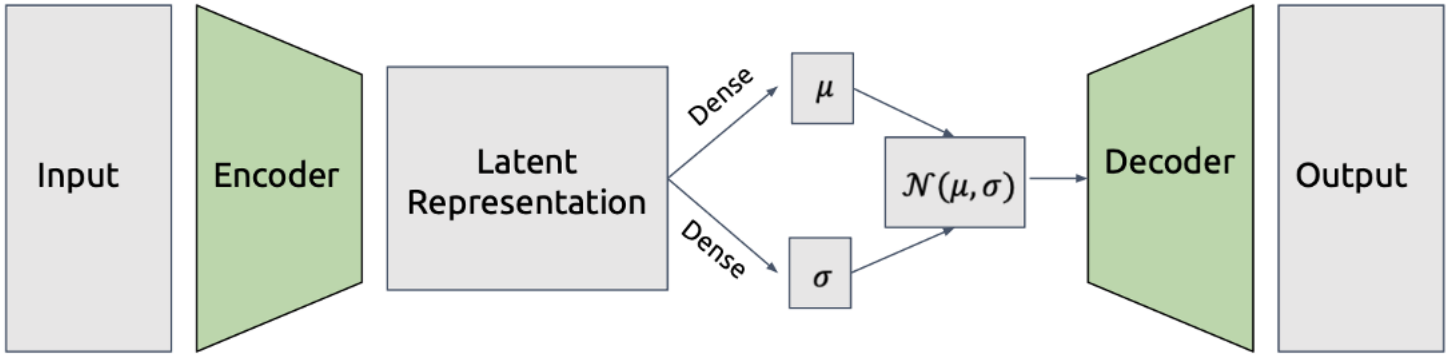


Defining the Variation Loss

Whatever our loss function, it needs to encourage $\sigma > 0$, or else the model will force σ to 0 in an effort to create the best recreations possible.

But it can't be too big… because too much variation will create poor reconstructions.

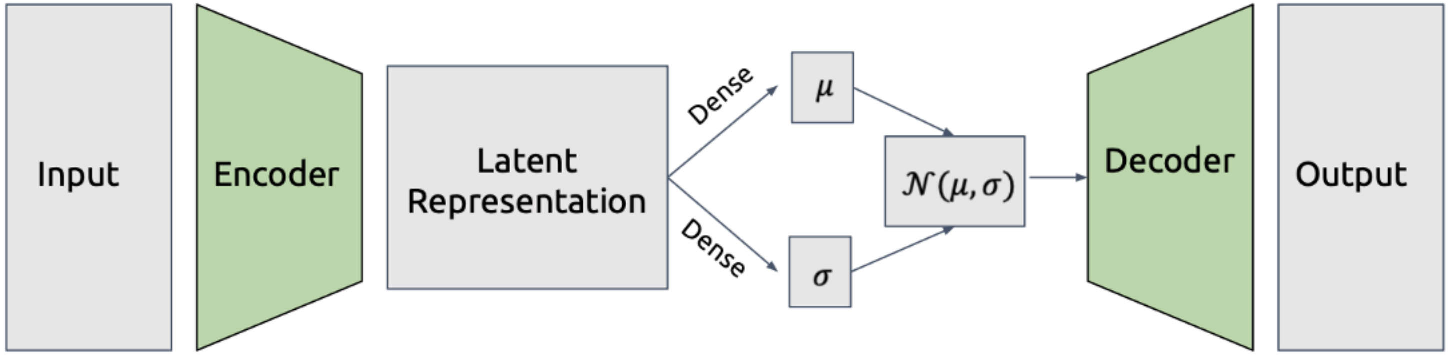
Also, what should μ be?



Defining the Variation Loss

The idea:

- Introduce a *prior* probability function we we want our latent space to look like.
- Encourage $N(\mu, \sigma)$ close to $N(0, 1)$
- (This will have beneficial properties we'll see later)



How do we measure distance between probabilities?

Kullback–Leibler (KL) Divergence

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

What this says:

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

(For most problems in DL, minimizing Cross Entropy minimizes KL-Divergence)

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 $\vec{\mu}, \vec{\sigma}$ (i.e., the size of the encoding)

The Final VAE Loss Function

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

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

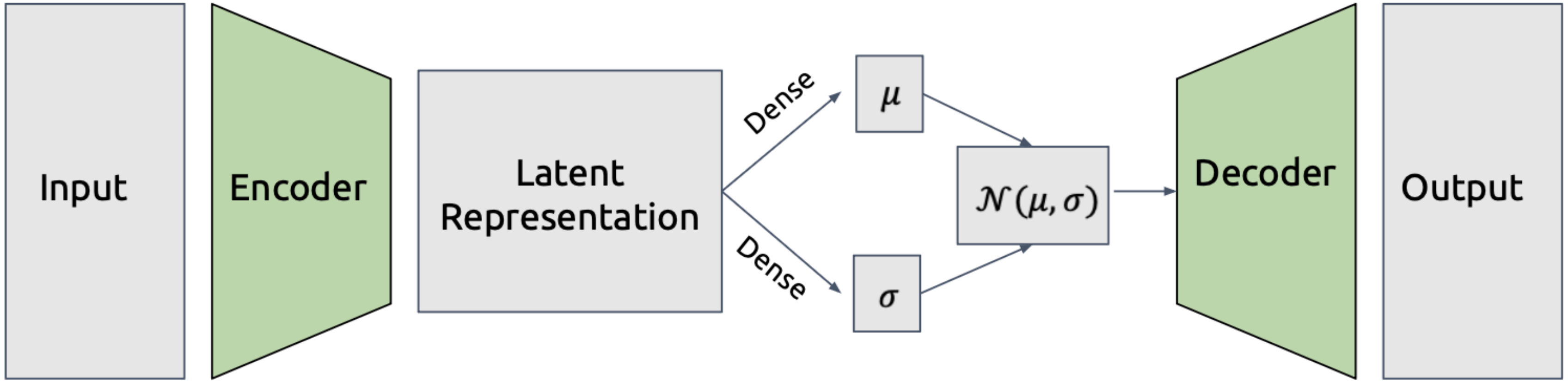
Which turns into this:

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

Any questions?

Putting it all together

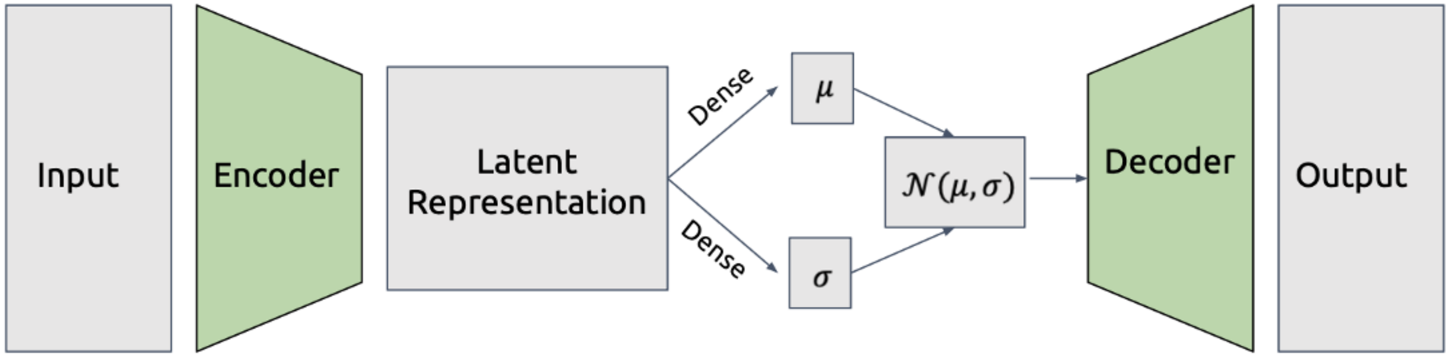
$$L = ||x - \hat{x}||_2^2 + \lambda D_{KL}(\mathcal{N}(\mu, \sigma), \mathcal{N}(0, 1))$$



There's just one issue

How do we take the gradient of a sampling operation?

$$z \sim N(\mu, \sigma)$$



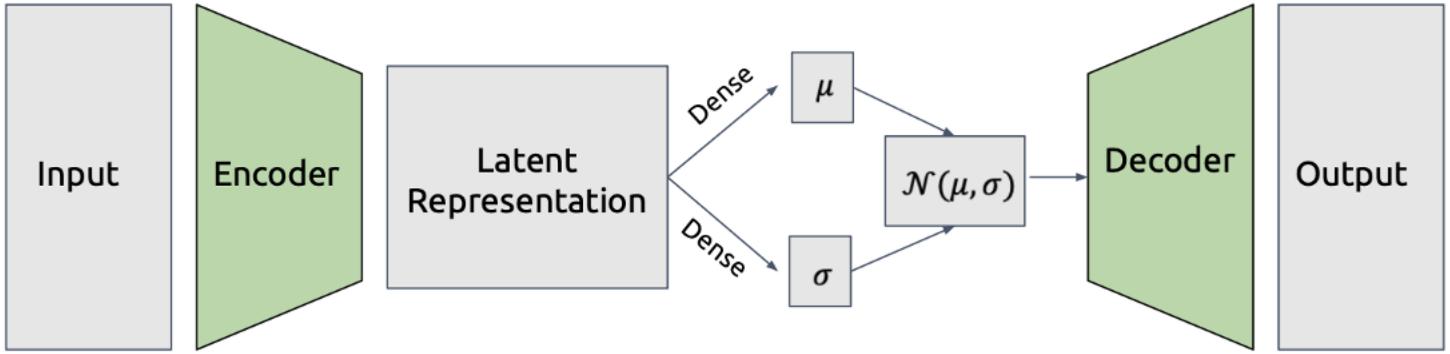
Reparametization Trick

$$z \sim N(\mu, \sigma)$$

Can be rewritten as:

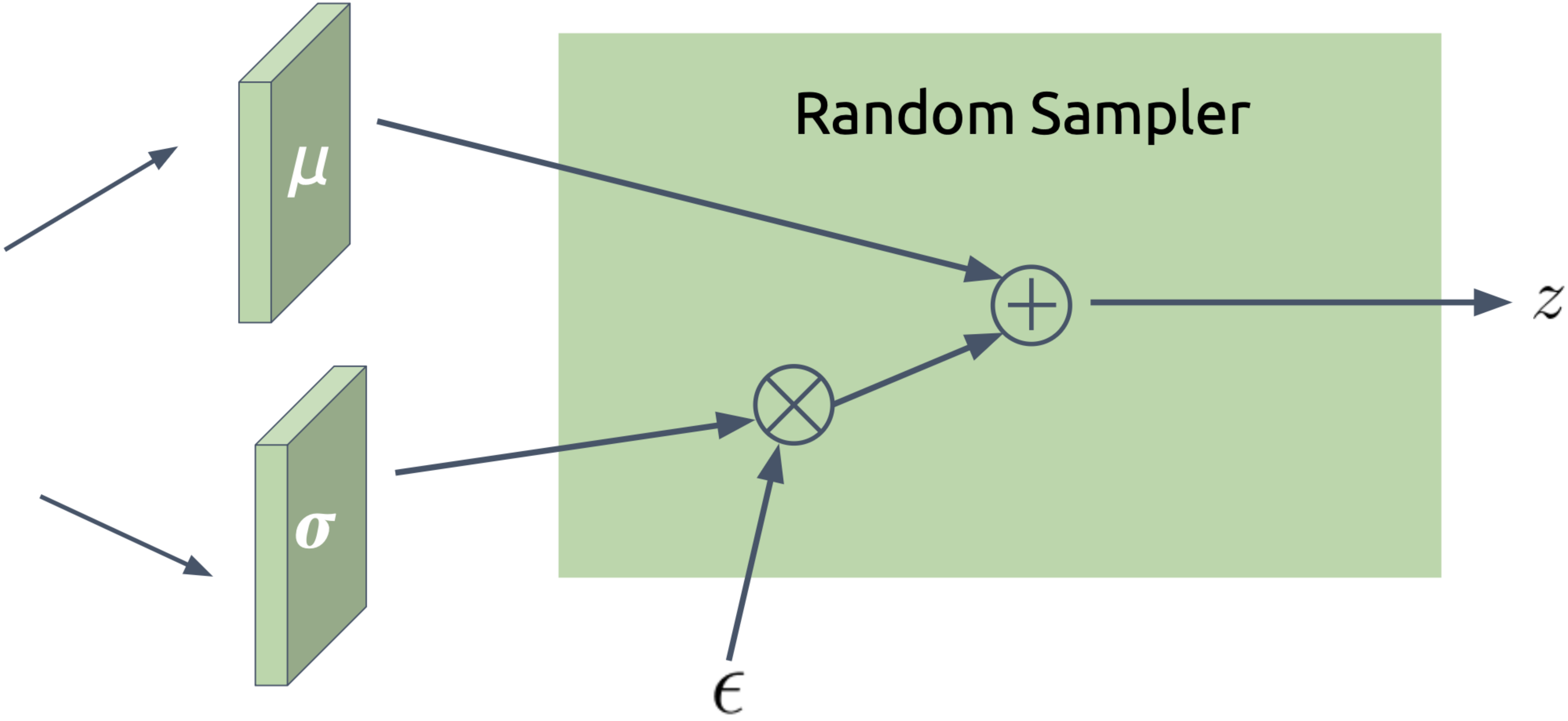
$z = \mu + \epsilon\sigma$, where $\epsilon \sim N(0,1)$

Random sampling operation (ϵ) no longer depends on learnable parameters

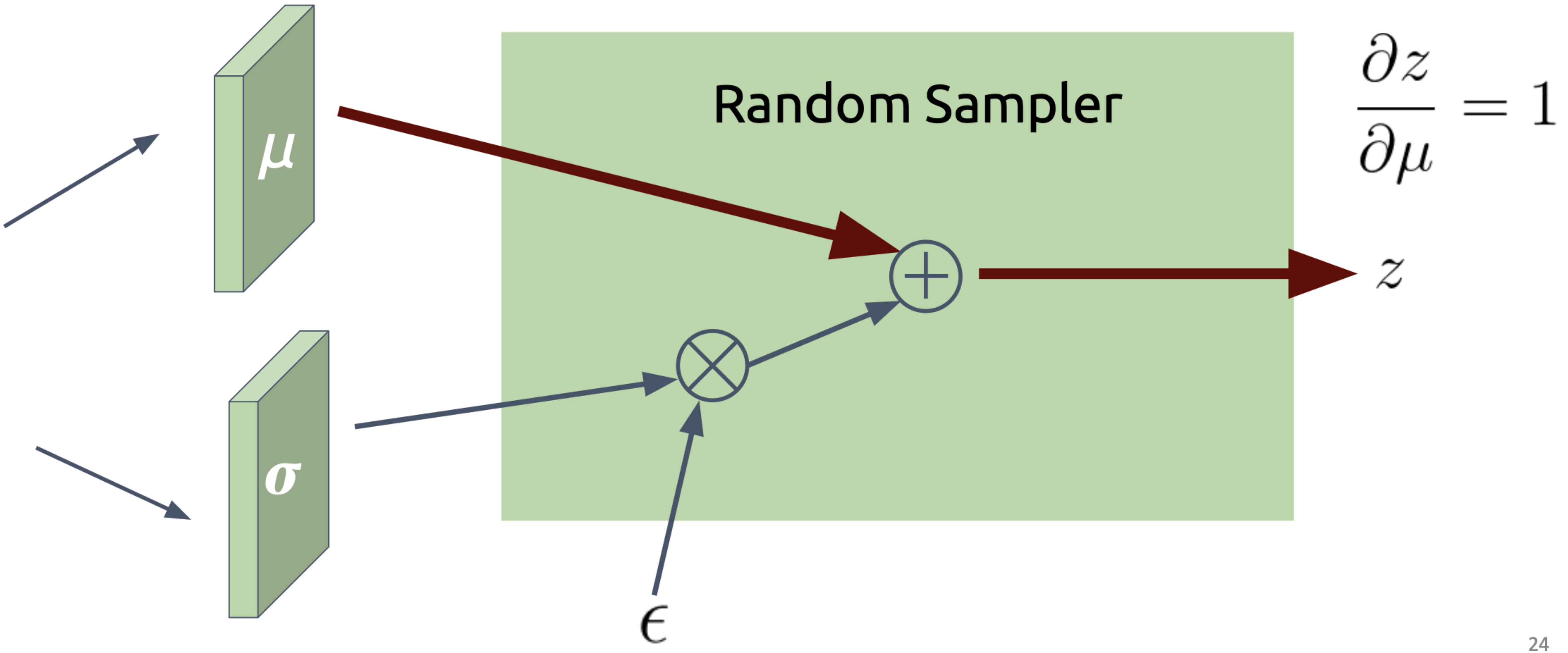


Another explanation of why this is needed: https://gregorygundersen.com/blog/2018/04/29/reparameterization/

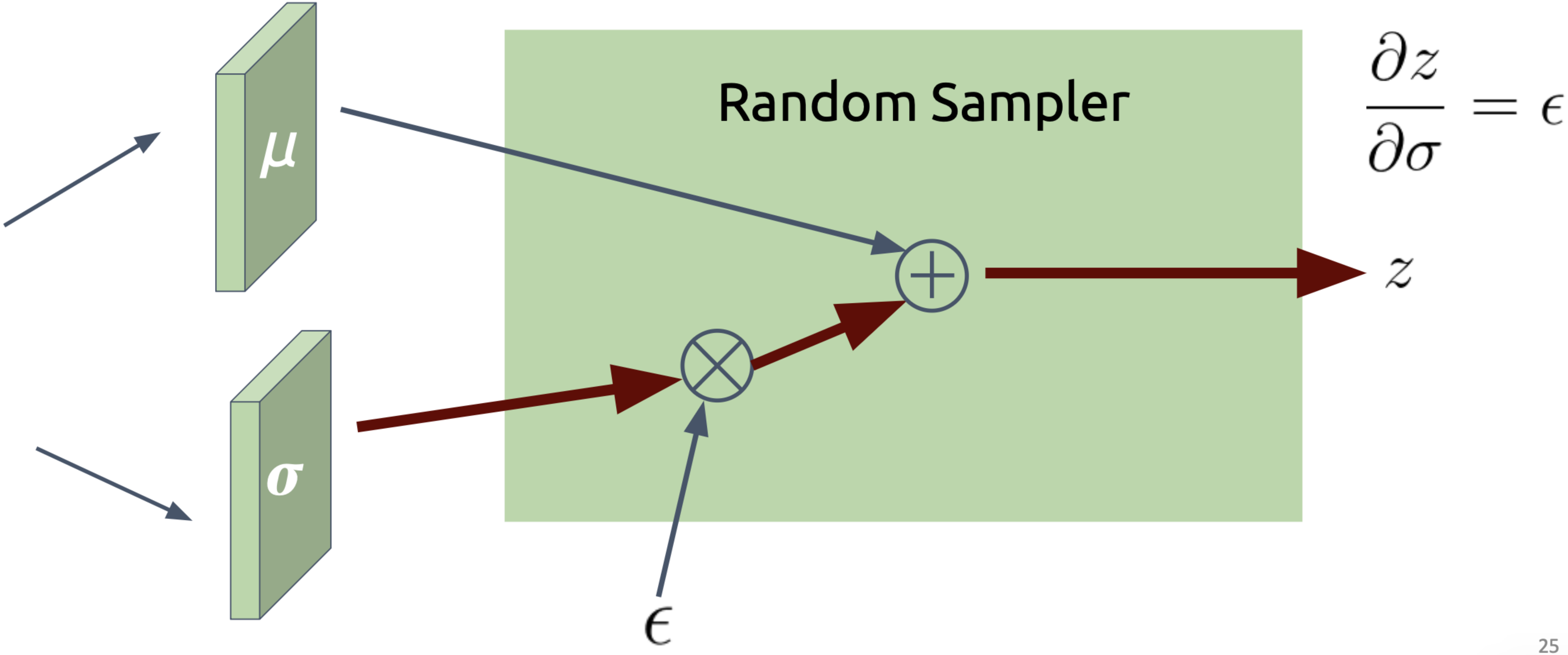
Random Sampler with Reparameterization Trick



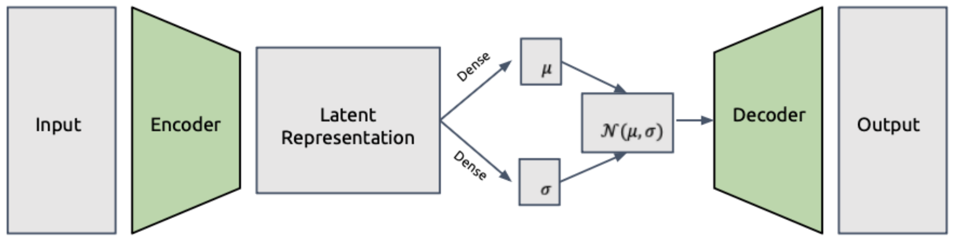
Random Sampler with Reparameterization Trick



Random Sampler with Reparameterization Trick



One more practical detail



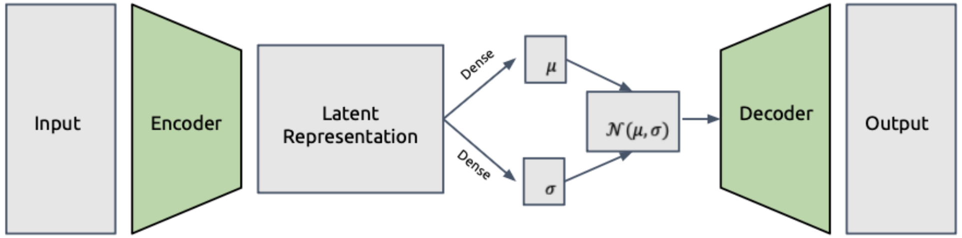
Let’s again consider our sampling operation

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

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

- 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

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

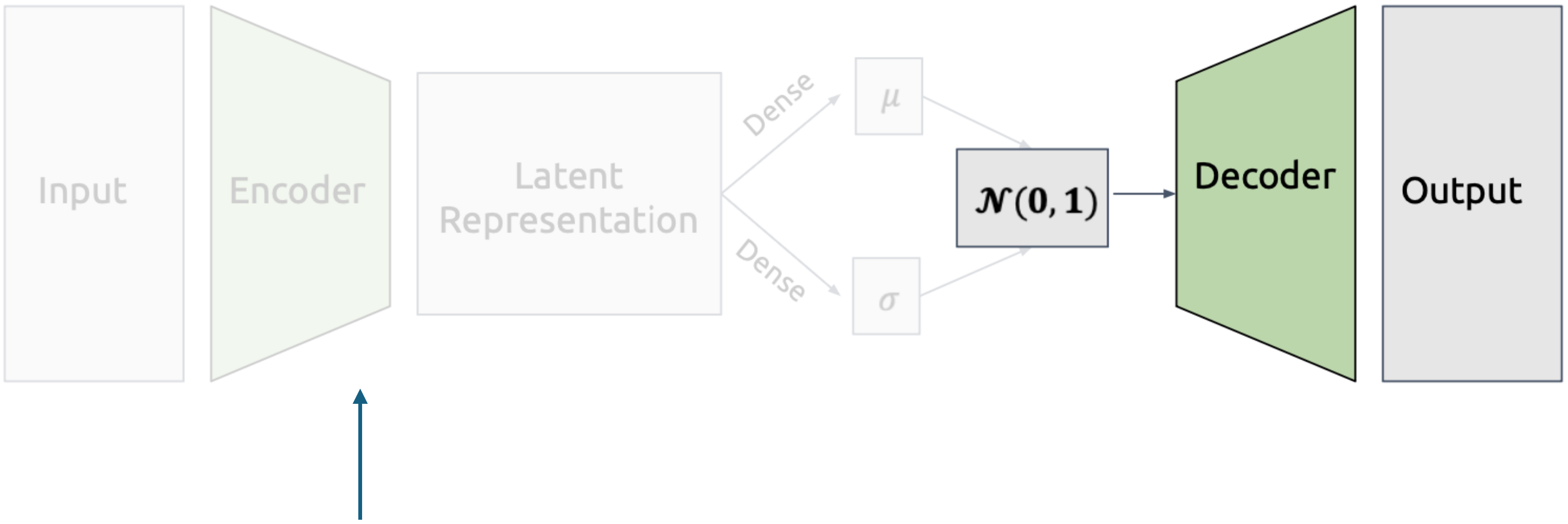
$$\mu_i \in [-\infty, \infty] \qquad \sigma_i \in [0, \infty]$$

Any questions?

- 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 $\log(\sigma^2)$

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

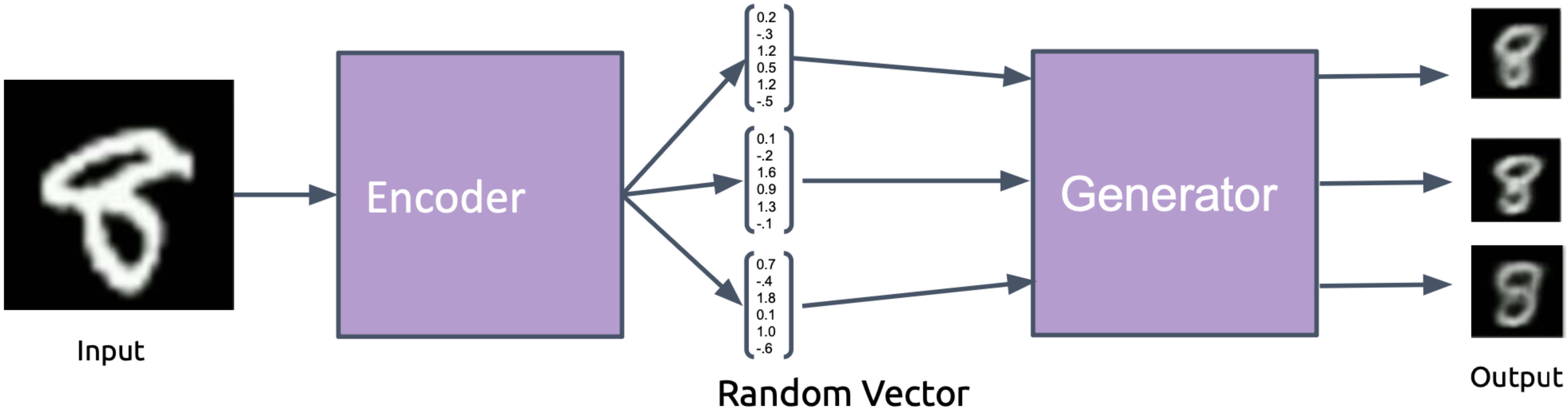
Sampling from a VAE



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

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

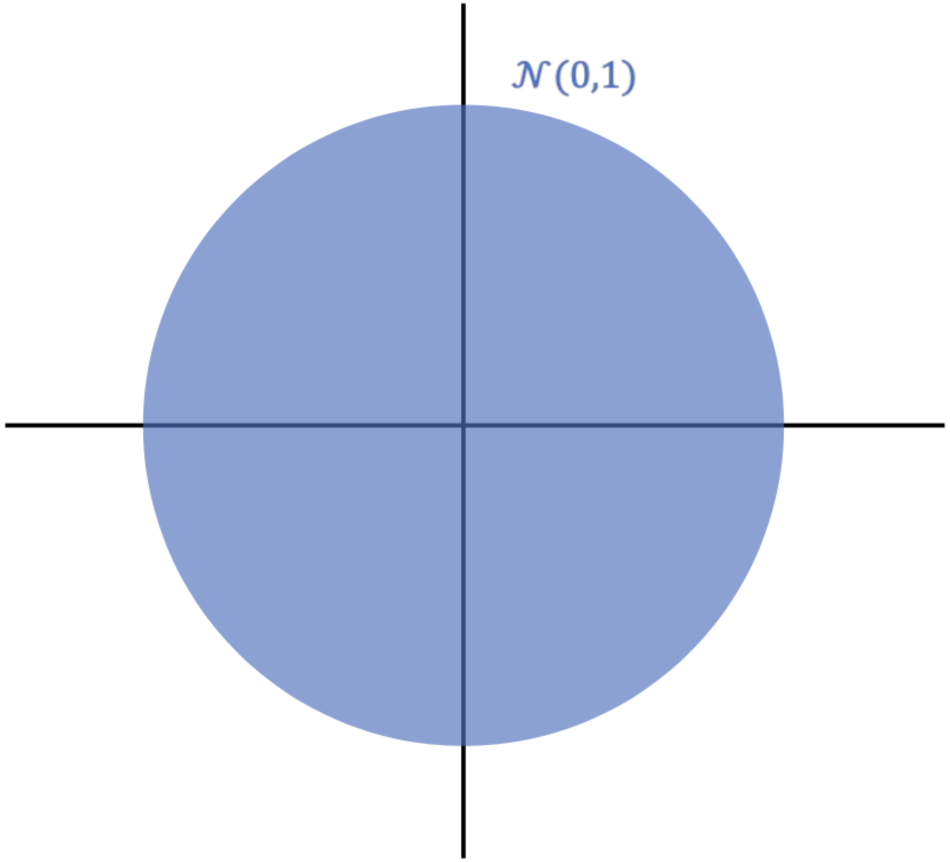
Random Noise $\begin{bmatrix} 0.2 \\ 0.5 \\ -.3 \\ -.2 \\ 0.8 \end{bmatrix} \xrightarrow{\text{VAE}}$

Random Sample

How can we do this?

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

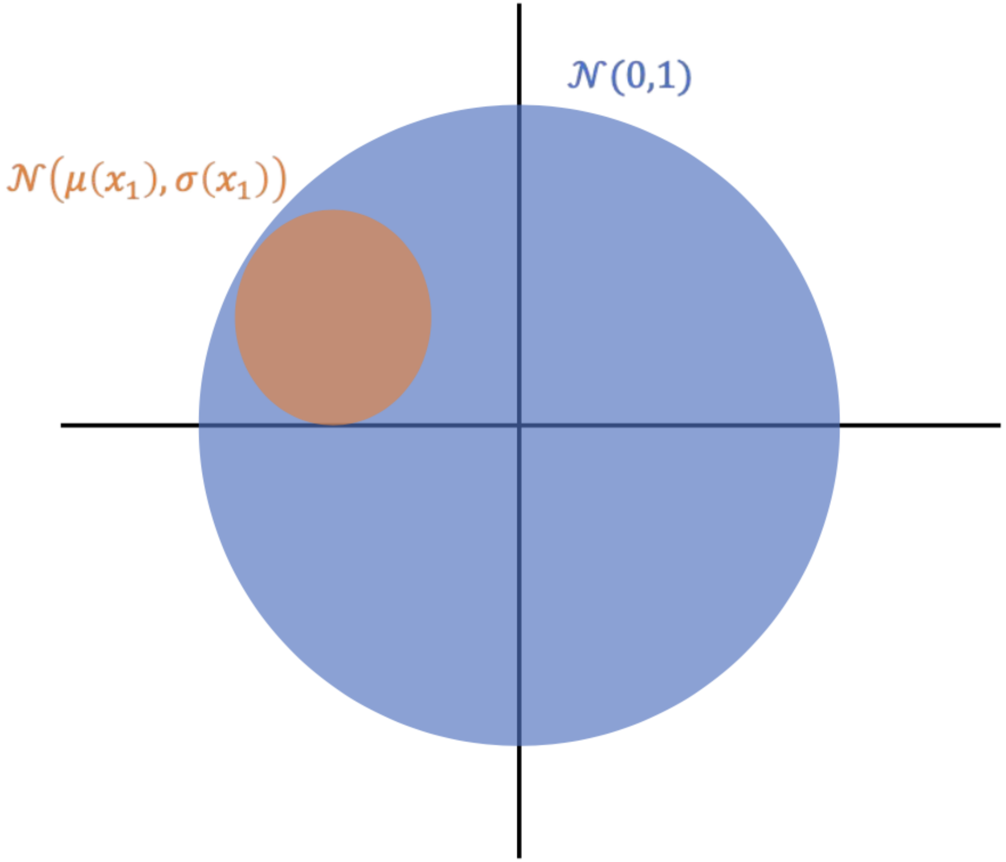
Encoding different points into latent space



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

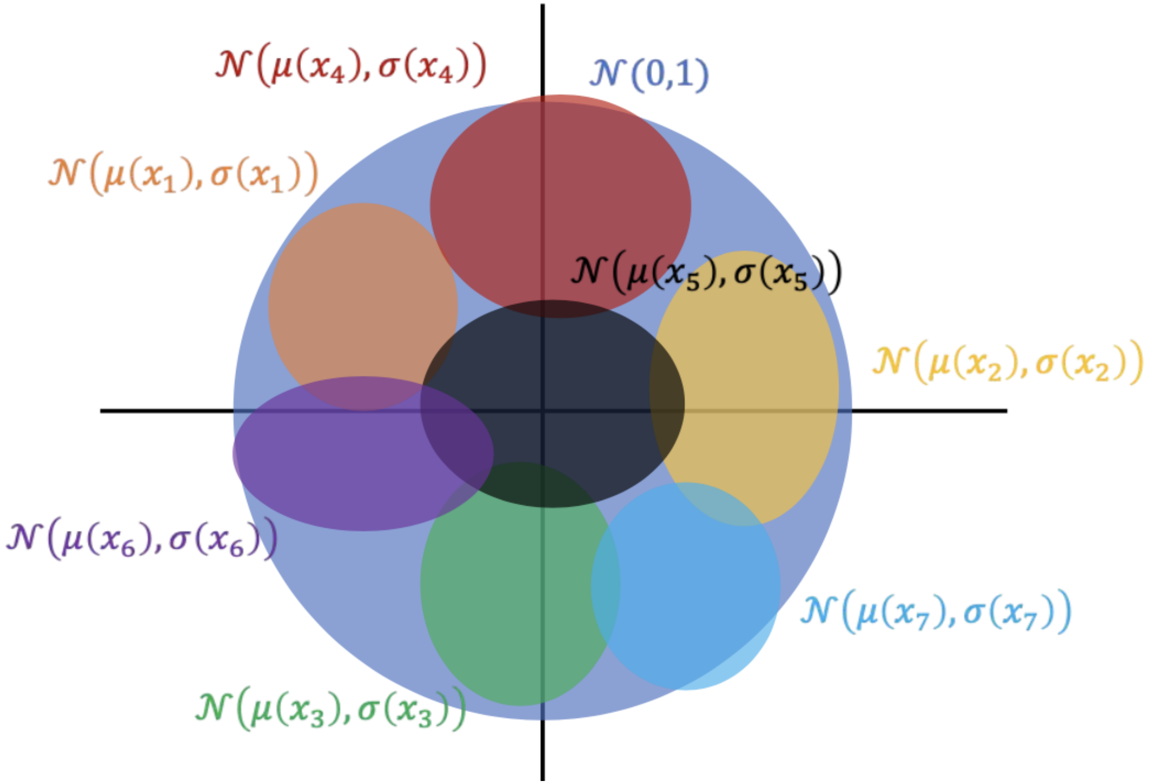
Encoding different points into latent space

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

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

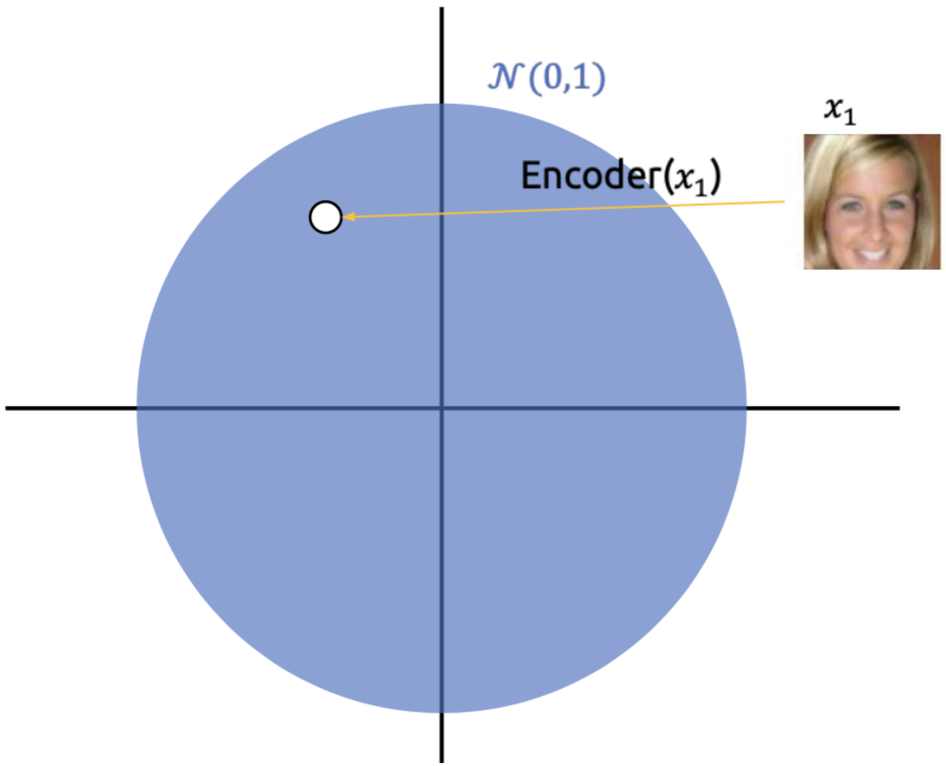


Because of our KL divergence loss, the $\mathcal{N}(\mu, \sigma)$ for any input data point has to be somewhat similar to $\mathcal{N}(0,1)$

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

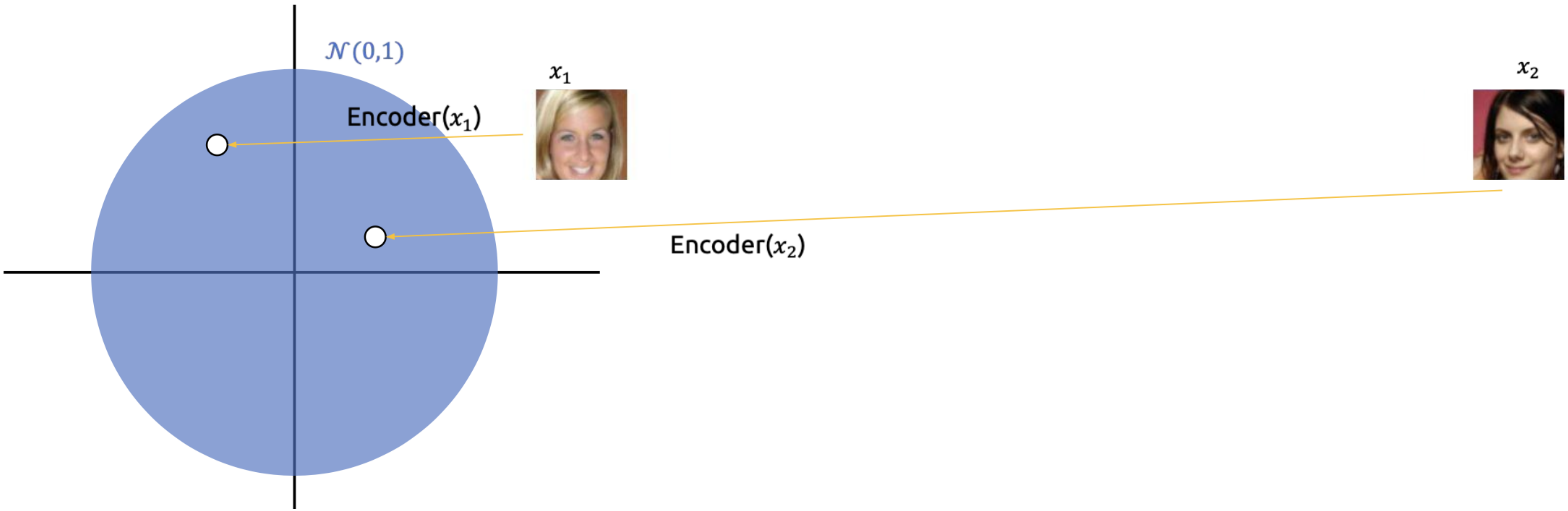
Latent Space Interpolation

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



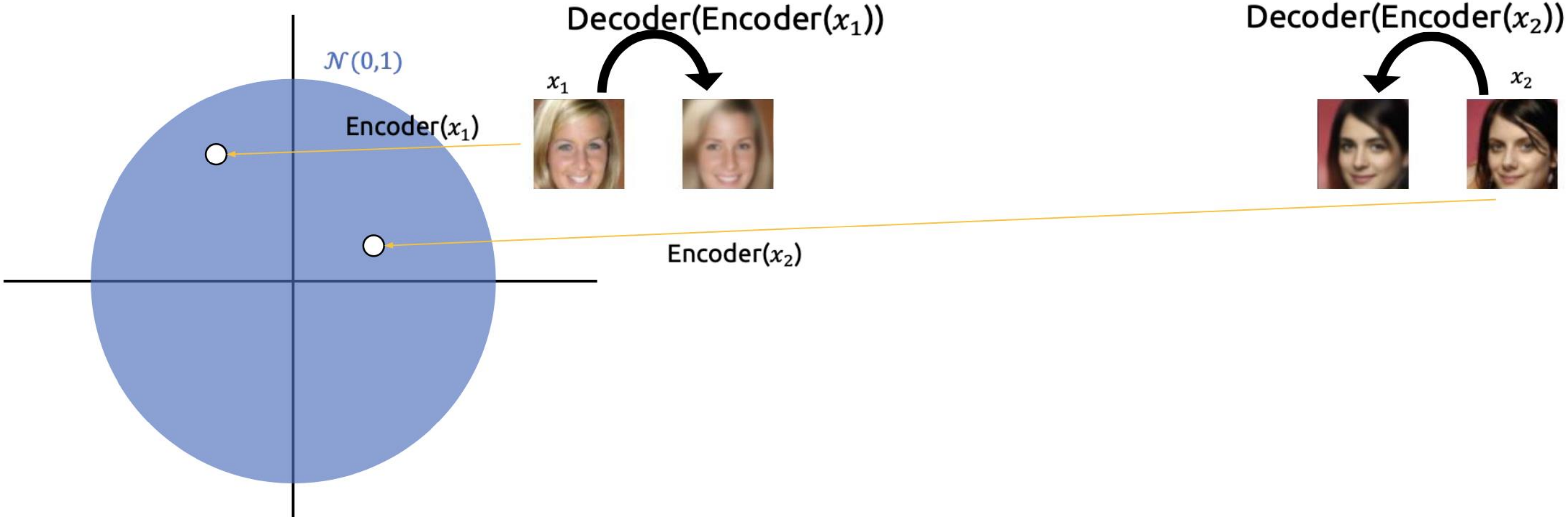
Latent Space Interpolation

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



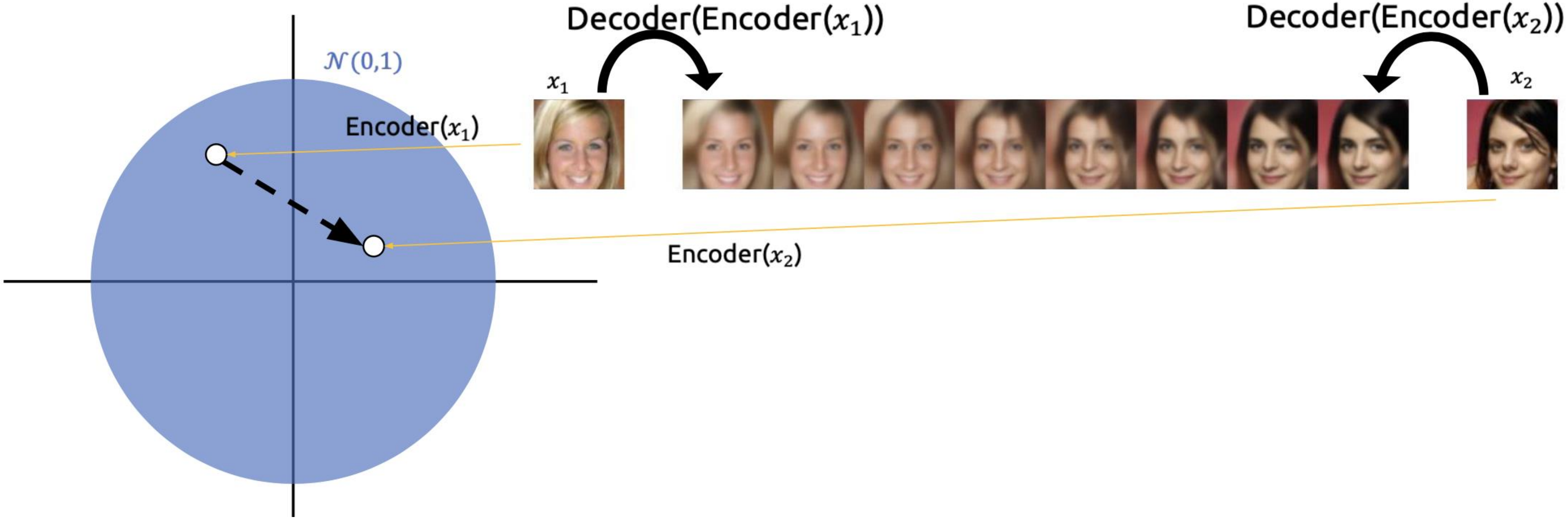
Latent Space Interpolation

- 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

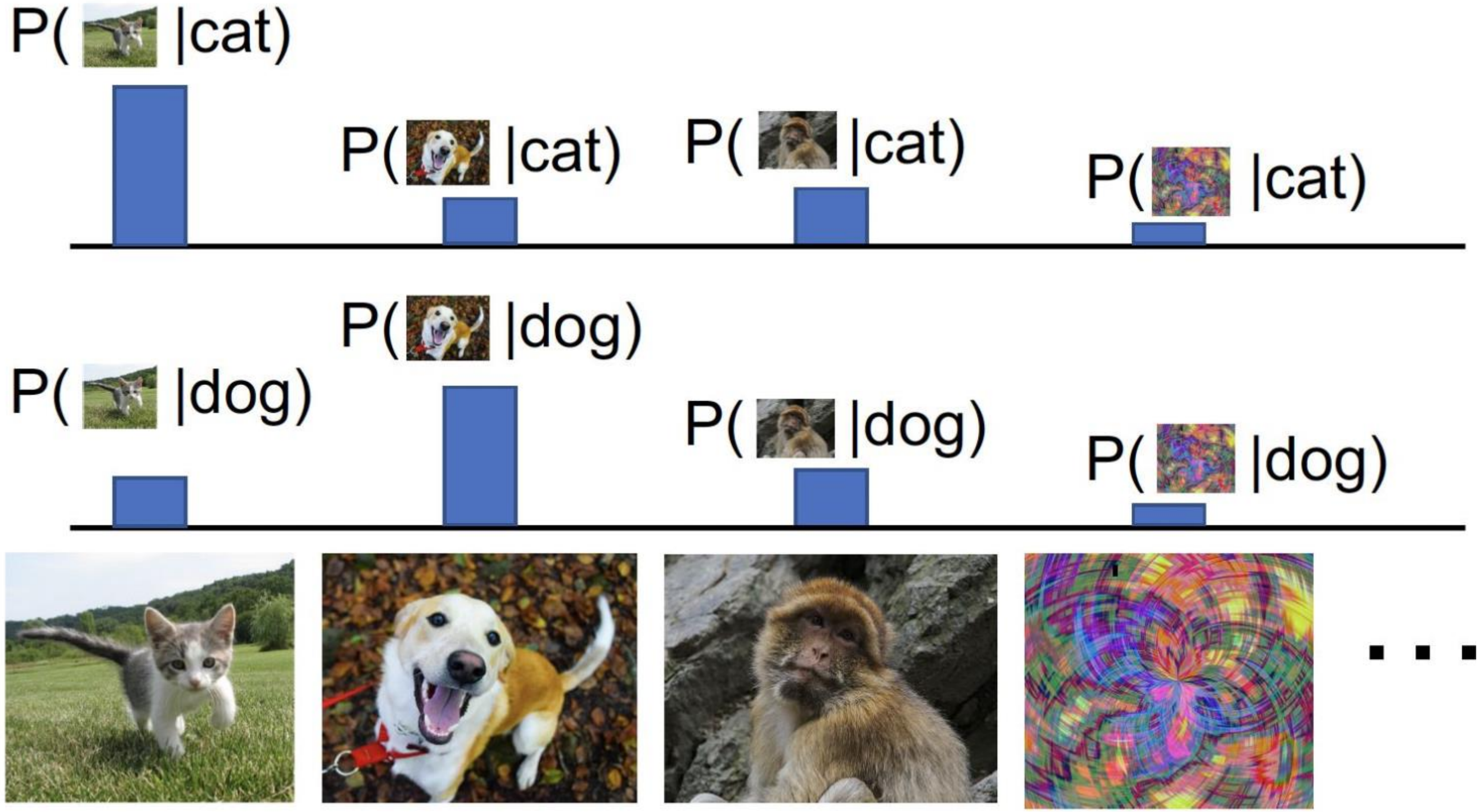


Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

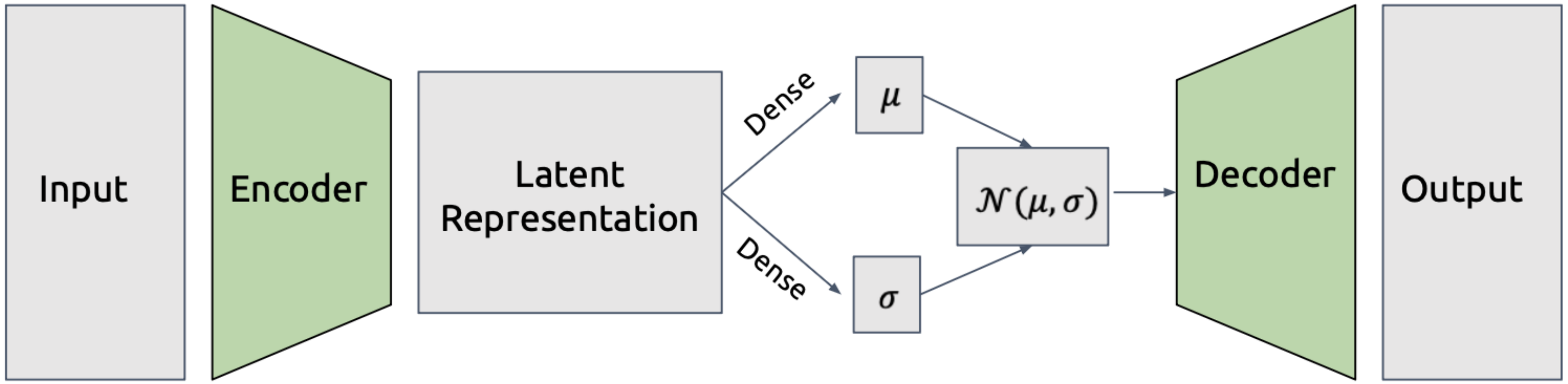


Conditional Generative Model: Each possible label induces a competition among all images

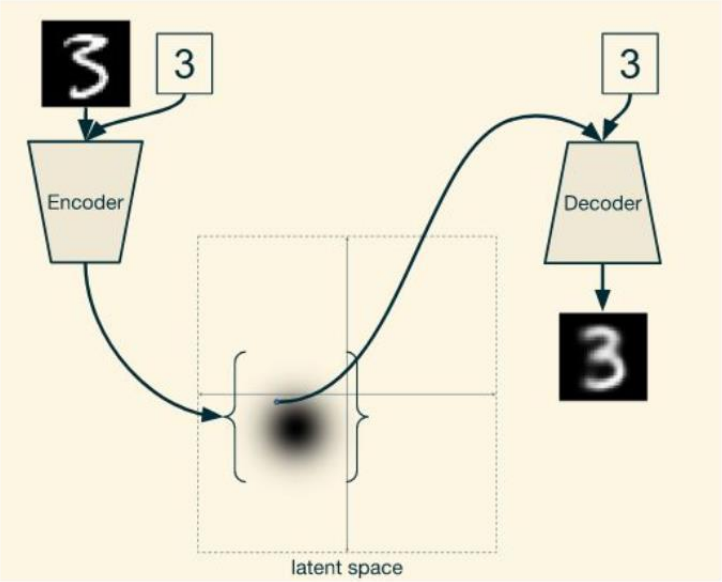
Credit: UMich EECS498

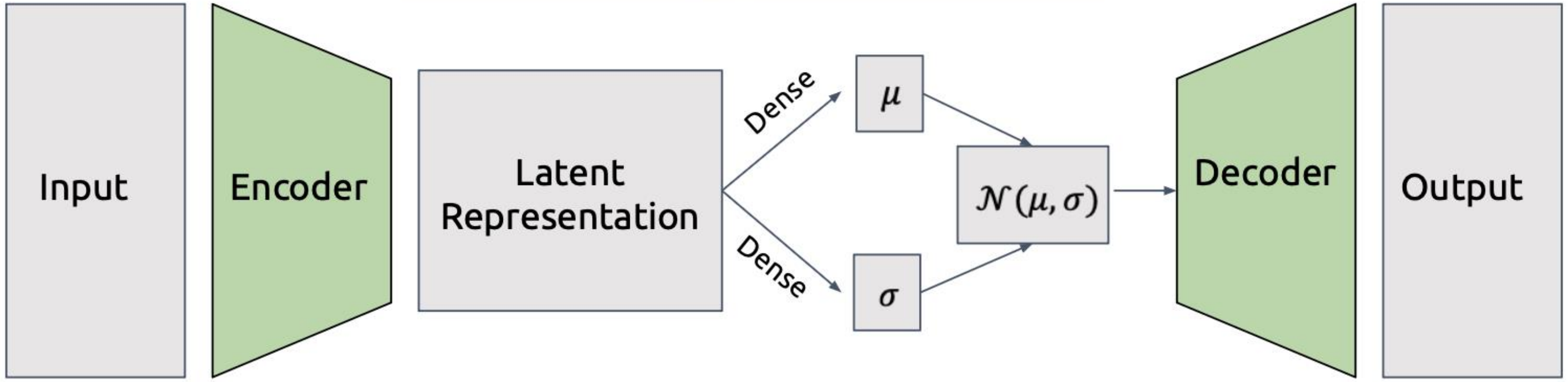
Conditional VAE

Any ideas?



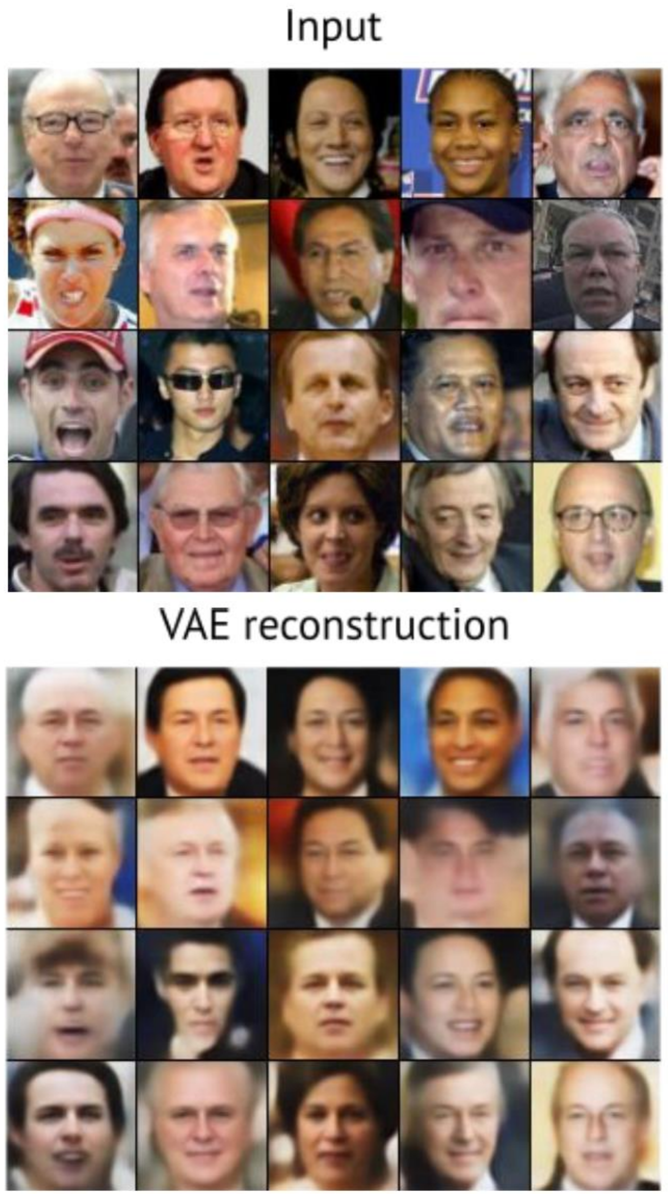
Conditional VAE





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

VAE output



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

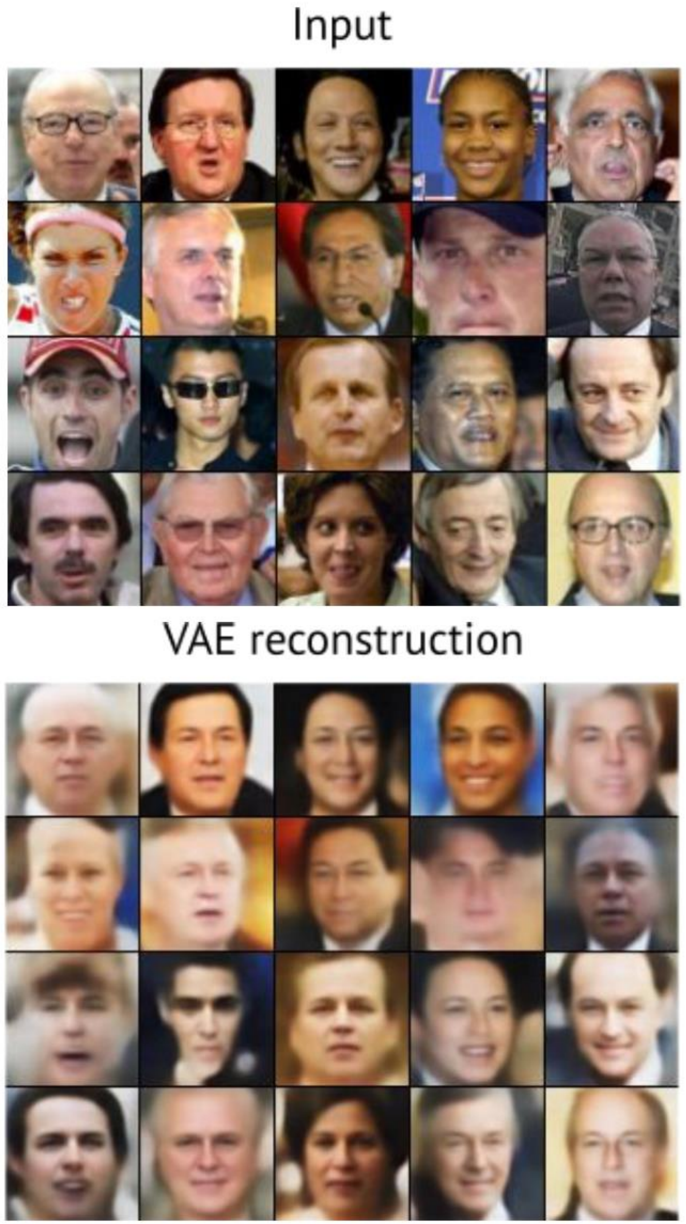
What's the issue here?

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!



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-17b86023588a