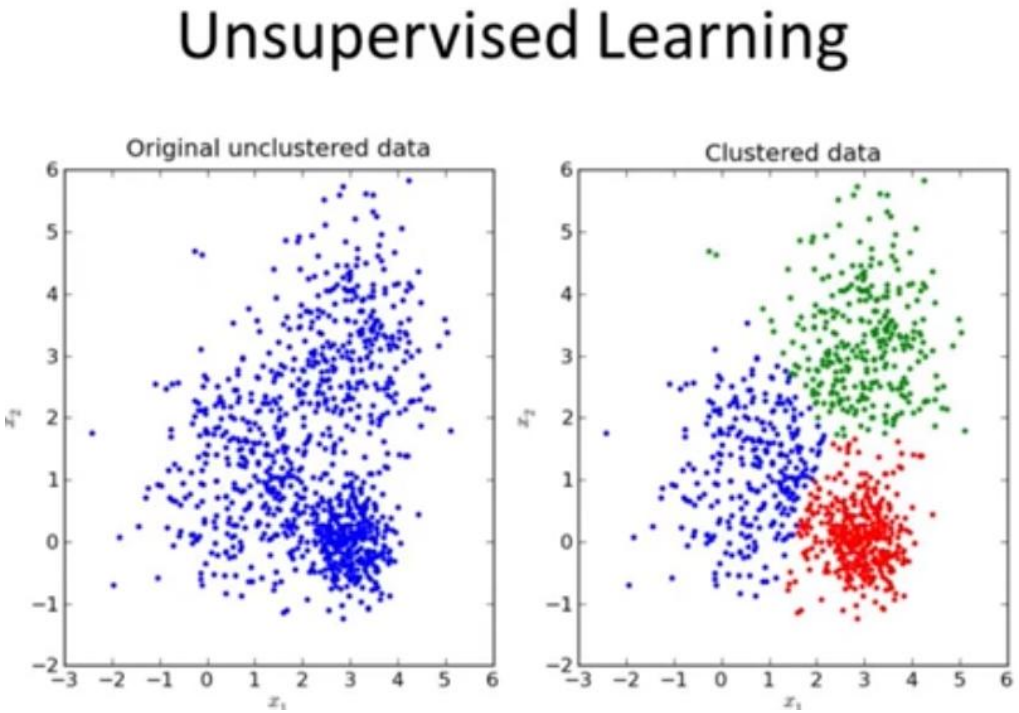
CSE 465
Lecture 13

Generative models (Autoencoders, Variational Autoencoders)

Supervised vs Unsupervised learning

- Supervised Learning
  - Data: (x, y) -- x is data, y is label
  - Goal: Learn function to map x → y
  - Examples: Classification, regression, object detection, semantic segmentation, etc.

- Unsupervised Learning
  - Data: (x) -- x is data, no labels!
  - Goal: Learn some hidden or underlying structure of the data
  - Examples: Clustering, feature or dimensionality reduction, etc



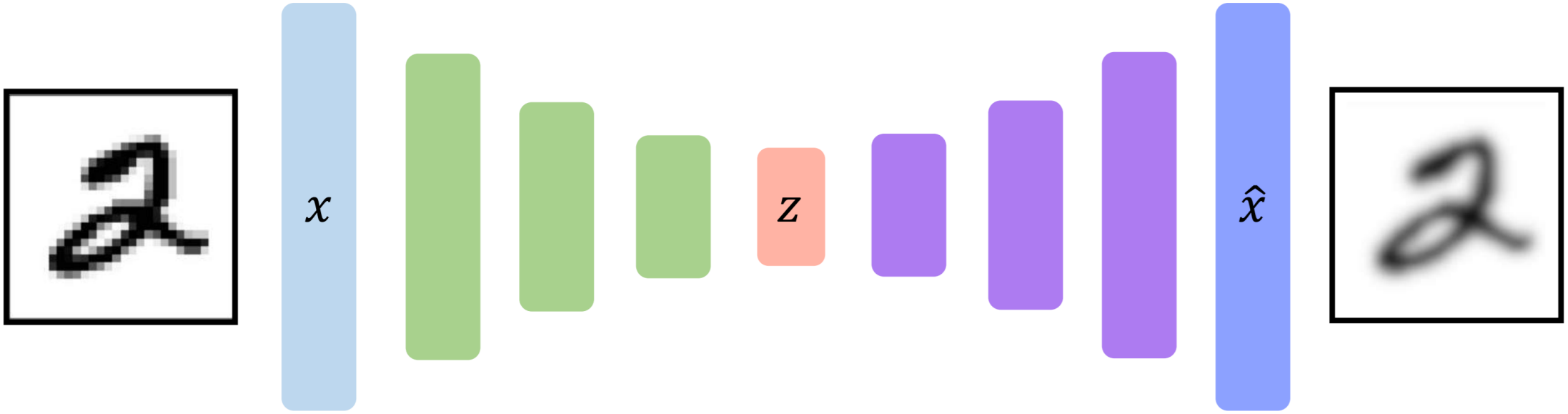
Autoencoders

- Autoencoders are a specific type of feedforward neural networks where the input is the same as the output
- They compress the input into a lower-dimensional *code* and then reconstruct the output from this representation
- The code is a compact "summary" or "compression" of the input, also called the *latent-space representation*

Unsupervised approach for learning a **lower-dimensional** feature representation from unlabeled training data

"Encoder" learns mapping from the data, x, to a low-dimensional latent space, z

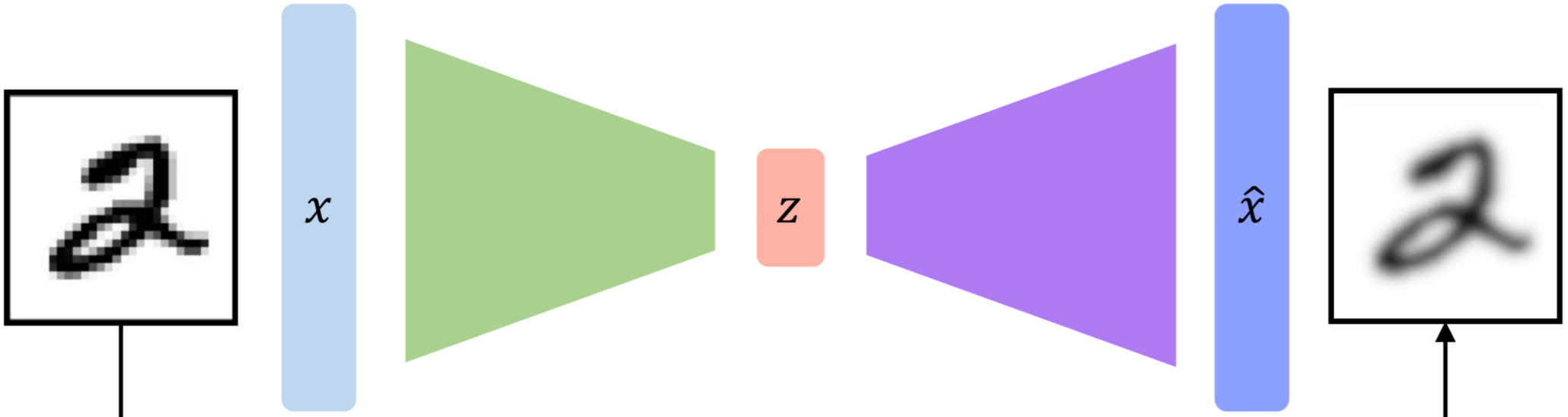
"Decoder" learns mapping back from latent, z, to a reconstructed observation, $\hat{x}$



Learn parameters for autoencoding

How can we learn this latent space?

Train the model to use these features to **reconstruct the original data**



$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

Loss function doesn't use any labels!!

Autoencoder pytorch implementation

```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True), nn.Linear(64, 12), nn.ReLU(True), nn.Linear(12, 3))
        self.decoder = nn.Sequential(
            nn.Linear(3, 12),
            nn.ReLU(True),
            nn.Linear(12, 64),
            nn.ReLU(True),
            nn.Linear(64, 128),
            nn.ReLU(True), nn.Linear(128, 28 * 28), nn.Tanh())
```

Dimensionality of latent space

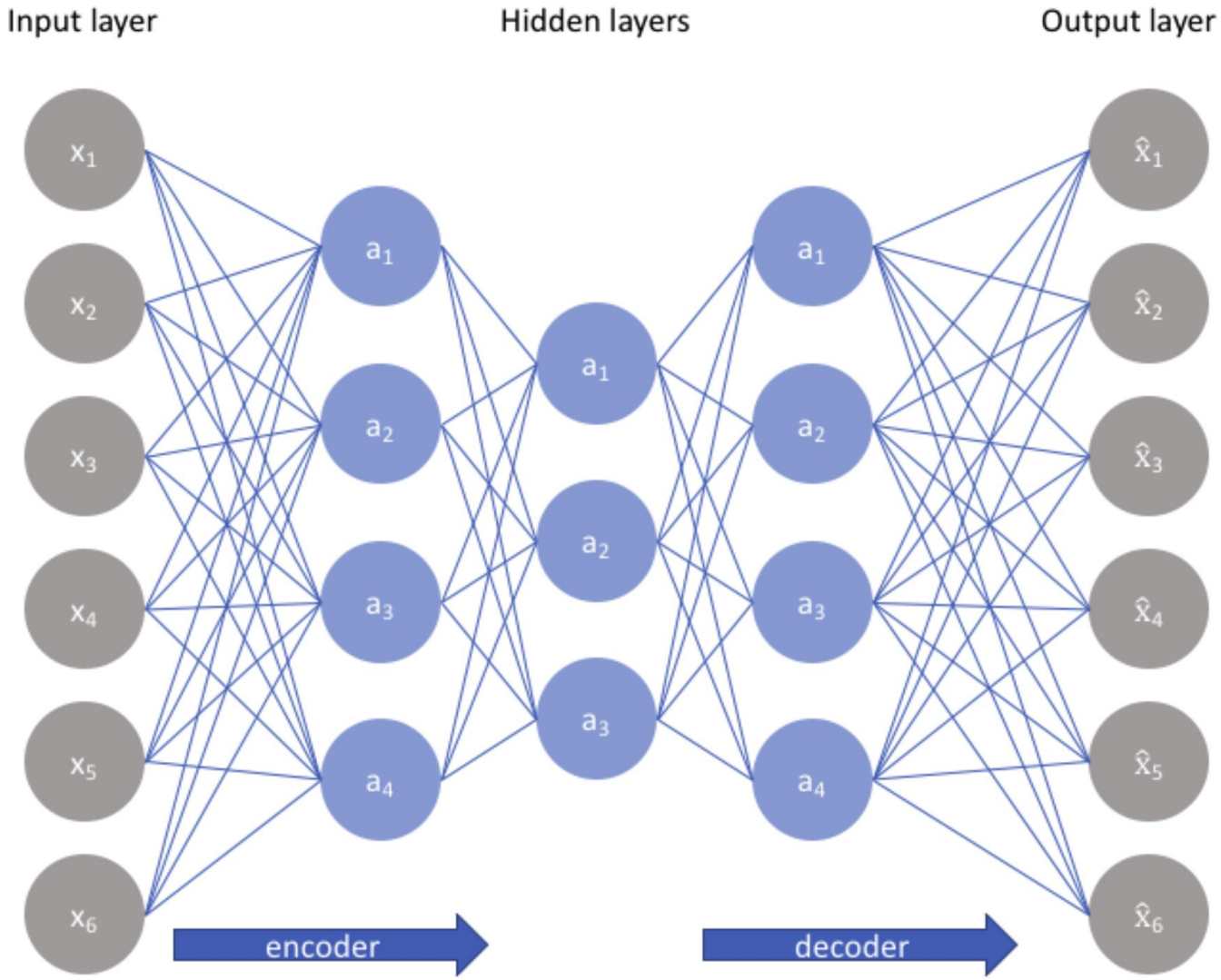
Autoencoding is a form of compression!
Smaller latent space will force a larger training bottleneck

2D latent space

5D latent space

Ground Truth

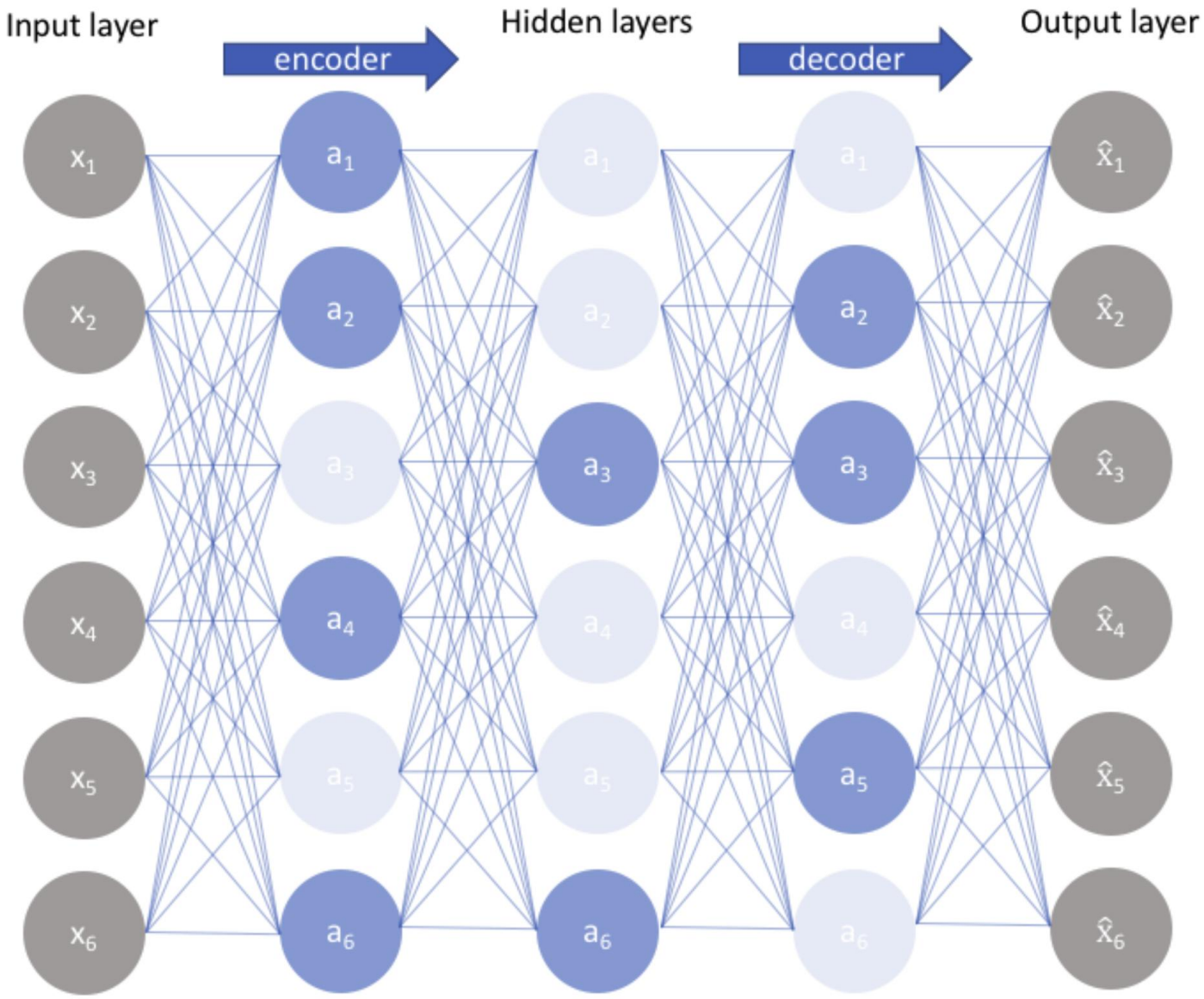
Undercomplete autoencoder



Undercomplete autoencoder

- The simplest architecture for constructing an autoencoder is to constrain the number of nodes present in the hidden layer(s) of the network
- It limits the amount of information that can flow through the network
- By penalizing the network according to the reconstruction error, our model can learn the most important attributes of the input data and how to best reconstruct the original input from an "encoded" state
- Ideally, this encoding will **learn and describe latent attributes of the input data**
- It can remove noise from the data

Sparse autoencoders



Sparse autoencoders

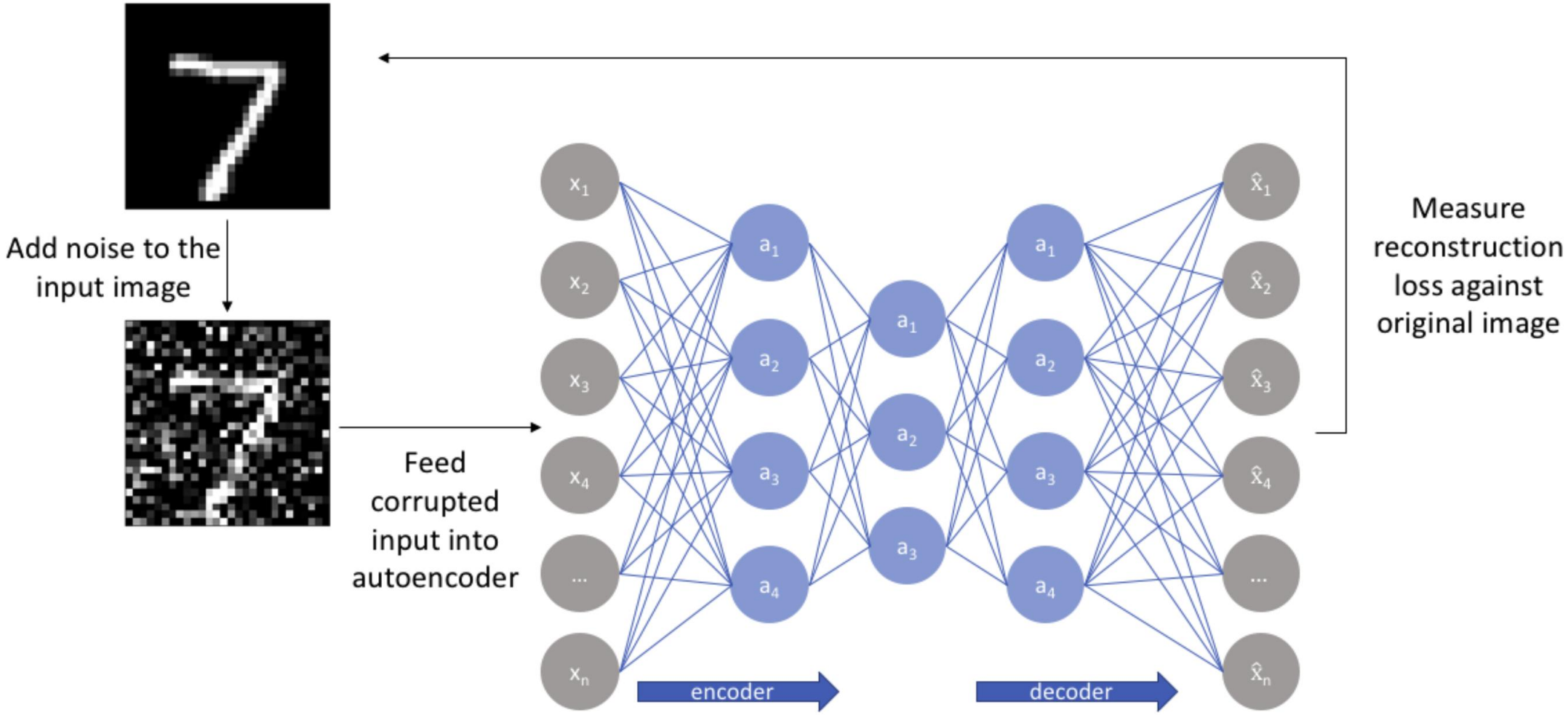
- Sparse autoencoders keep the number of nodes in hidden layers same as the input
- Rather, loss functions are constructed to penalize *activations* within a layer
- Individual nodes of a trained model which activate are *data-dependent*, different inputs will result in activations of different nodes through the network
- An **undercomplete autoencoder** will use the entire network for every observation, a **sparse autoencoder** will be forced to selectively activate regions of the network depending on the input data
- Therefore, sparse autoencoders limits the network's capacity to memorize the input data without limiting the networks capability to extract features from the data

How to keep neurons deactivated?

- L1 Regularization
  - We can add a term to our loss function that penalizes the absolute value of the vector of activations
  - In layer h for observation i, scaled by a tuning parameter λ

$$\mathcal{L}\left(x, \hat{x}\right) + \lambda \sum_{i} \left| a_i^{(h)} \right|$$

Denoising autoencoders



Denoising autoencoders

- With this approach, **our model isn't able to simply develop a mapping which memorizes the training data because our input and target output are no longer the same**
- Rather, the model learns a vector field for mapping the input data towards a lower-dimensional manifold

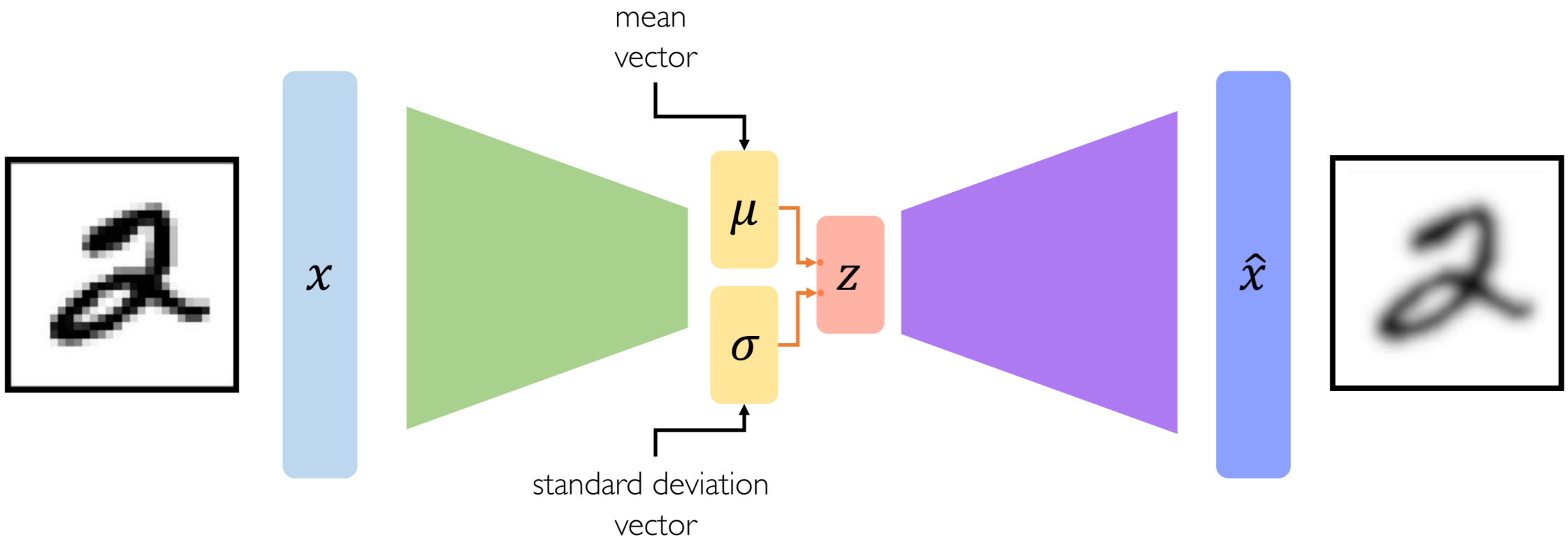
Autoencoders: In summary

- Our input data is converted into an *encoding vector* where each dimension represents some learned attribute about the data
- The most important detail to grasp here is that our encoder network is outputting a *single value* for each encoding dimension
- The decoder network then subsequently takes these values and attempts to recreate the original input

Variational autoencoders

- A variational autoencoder (VAE) provides a *probabilistic* manner for describing an observation in latent space
- Thus, rather than building an encoder which outputs a single value to describe each latent state attribute, we'll formulate our encoder to describe a probability distribution for each latent attribute

Variational autoencoders

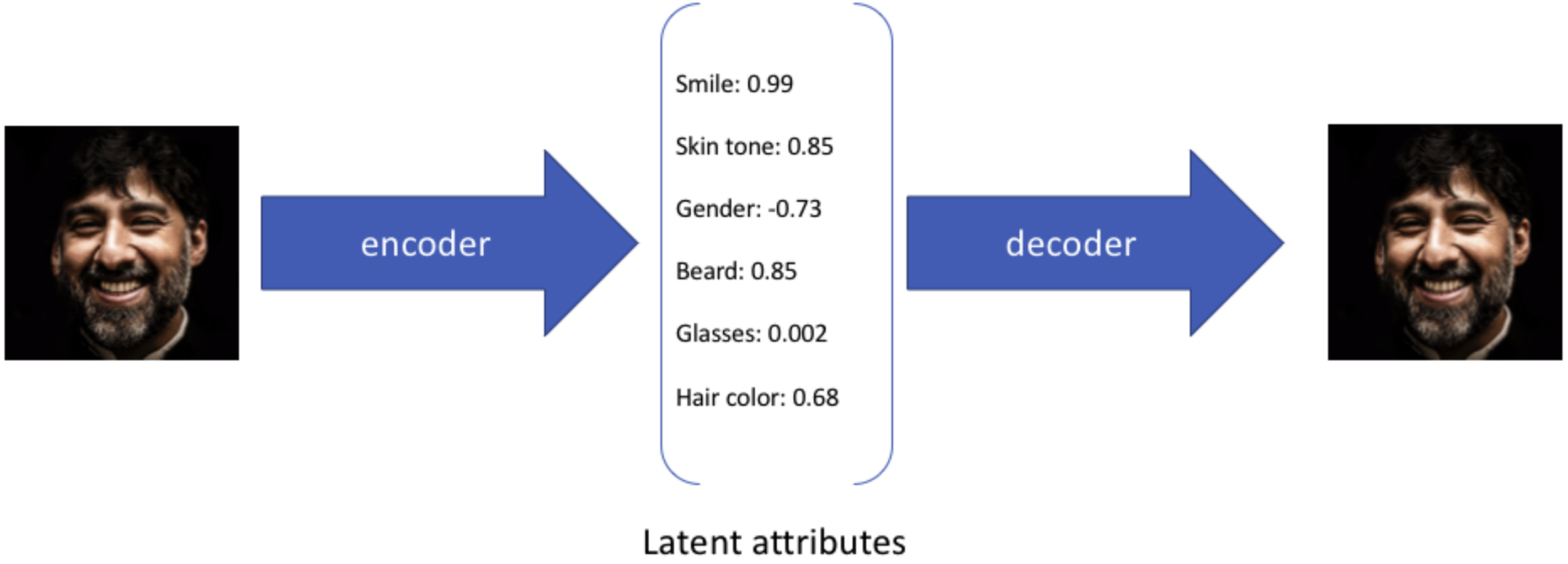


Variational autoencoders are a probabilistic twist on autoencoders!

Sample from the mean and standard dev. to compute latent sample

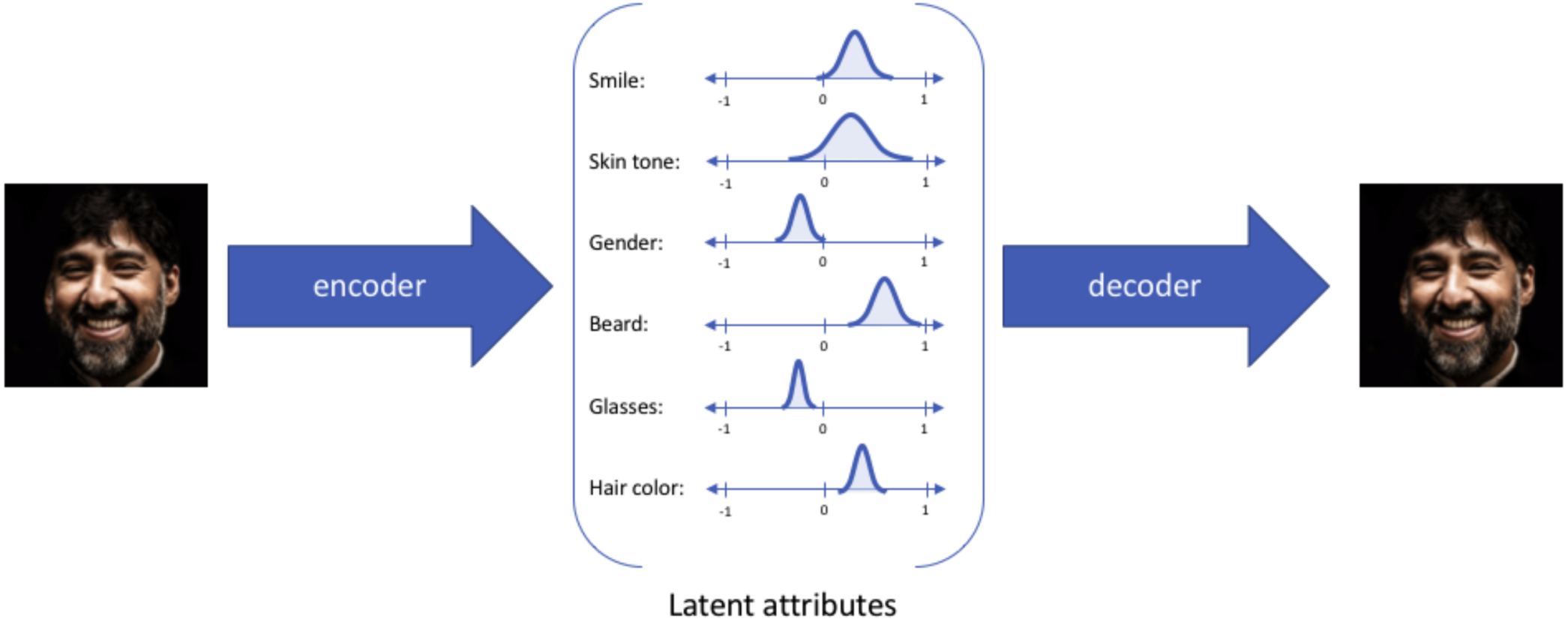
Intuition

- suppose we've trained an autoencoder model on a large dataset of faces with an encoding dimension of 6



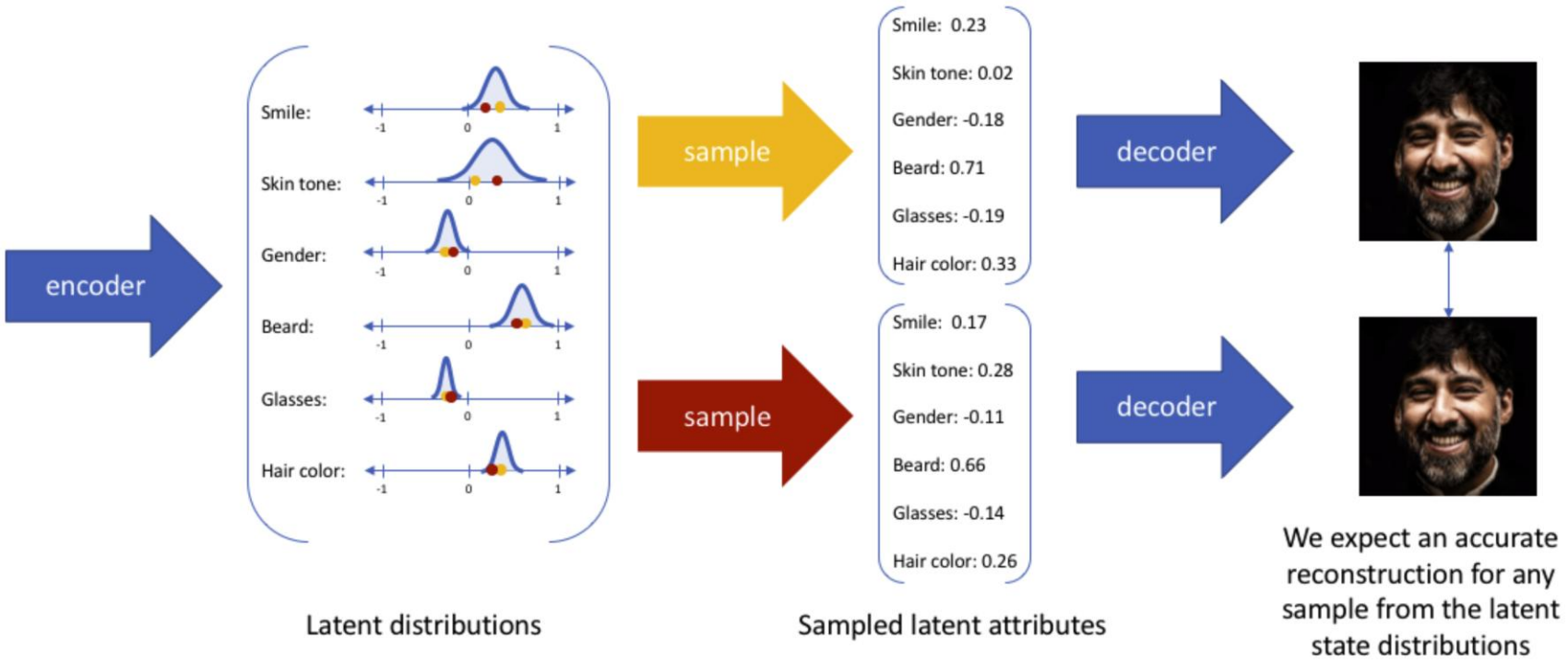
Intuition

- However, we may prefer to represent each latent attribute as a range of possible values (from a distribution)



Intuition

By constructing our encoder model to output a range of possible values (a statistical distribution) from which we'll randomly sample to feed into our decoder model, we're essentially enforcing a continuous, smooth latent space representation



Statistical motivation

- Suppose that there exists some hidden variable z which generates an observation x
- We can only see x, but we would like to infer the characteristics of z. In other words, we'd like to compute

$$p(z|x) = \frac{p(x|z)\,p(z)}{p(x)}$$

- Unfortunately, computing p(x) is quite difficult

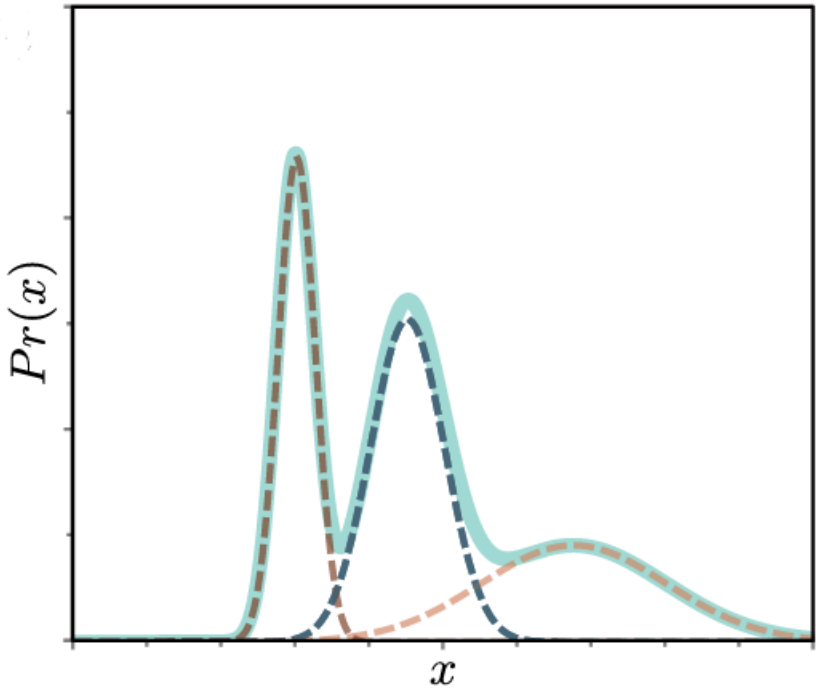
$$p(x) = \int p(x|z)\,p(z)\,dz$$

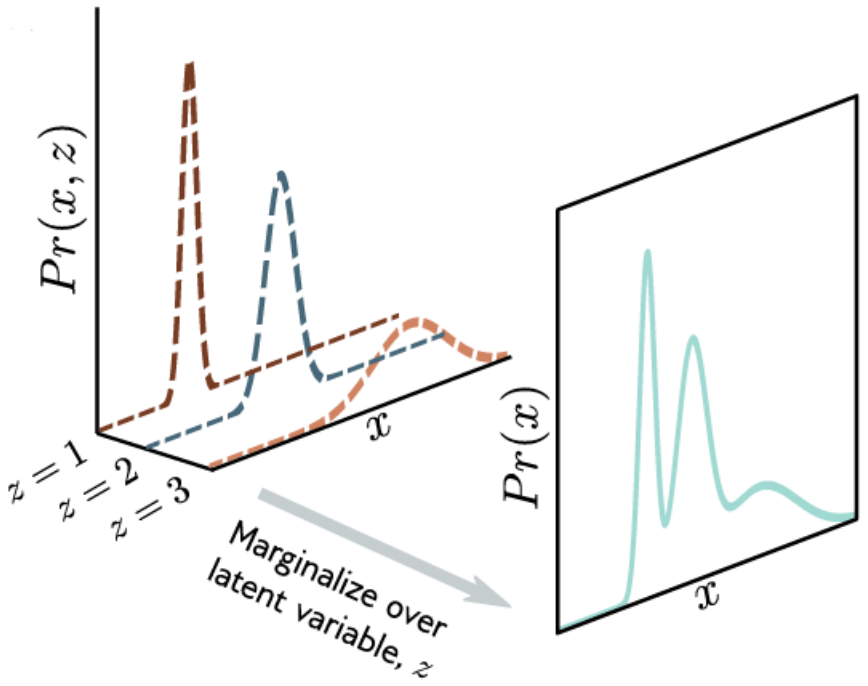
- This usually turns out to be an intractable distribution
- However, we can apply variational inference to estimate this value

Statistical motivation

- It is possible to approximate p(z|x) by another distribution q(z|x) which has a tractable distribution
- If we can define the parameters of q(z|x) such that it is very similar to p(z|x), we can use it to perform approximate inference of the intractable distribution
- KL divergence is a measure of difference between two probability distributions
- Thus, if we wanted to ensure that q(z|x) was similar to p(z|x), we could minimize the KL divergence between the two distributions

$$\min KL\left(q\left(z|x\right)||p\left(z|x\right)\right)$$


$Pr(x)$
x

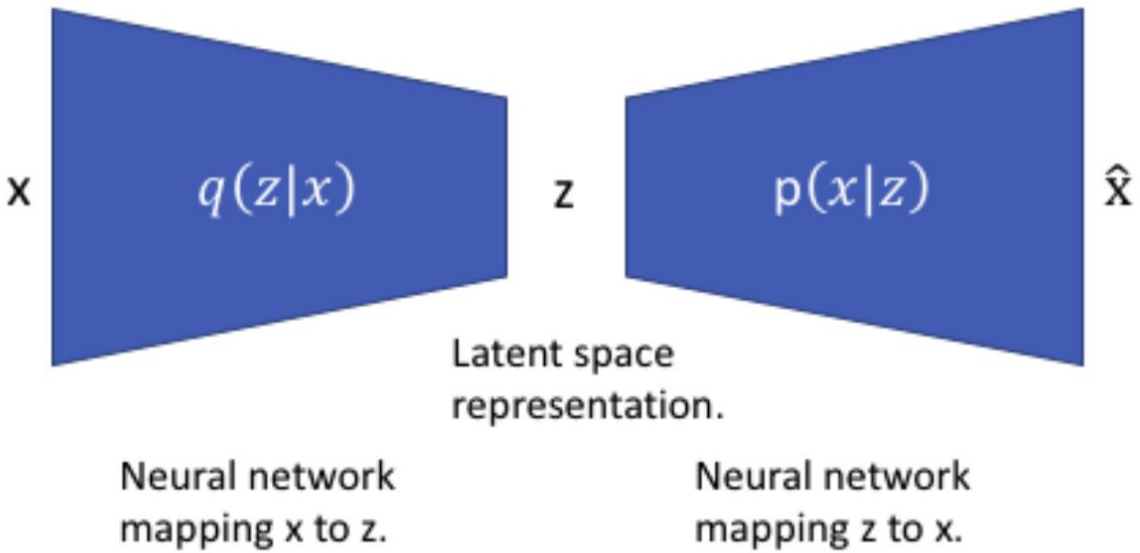

$Pr(x, z)$
x
$z = 1$
$z = 2$
$z = 3$
Marginalize over latent variable, z
$Pr(x)$
x

Statistical motivation

$Q(z|X)$ that project our data X into latent variable space z, the latent variable

$P(X|z)$ that generate data given latent variable

Statistical motivation



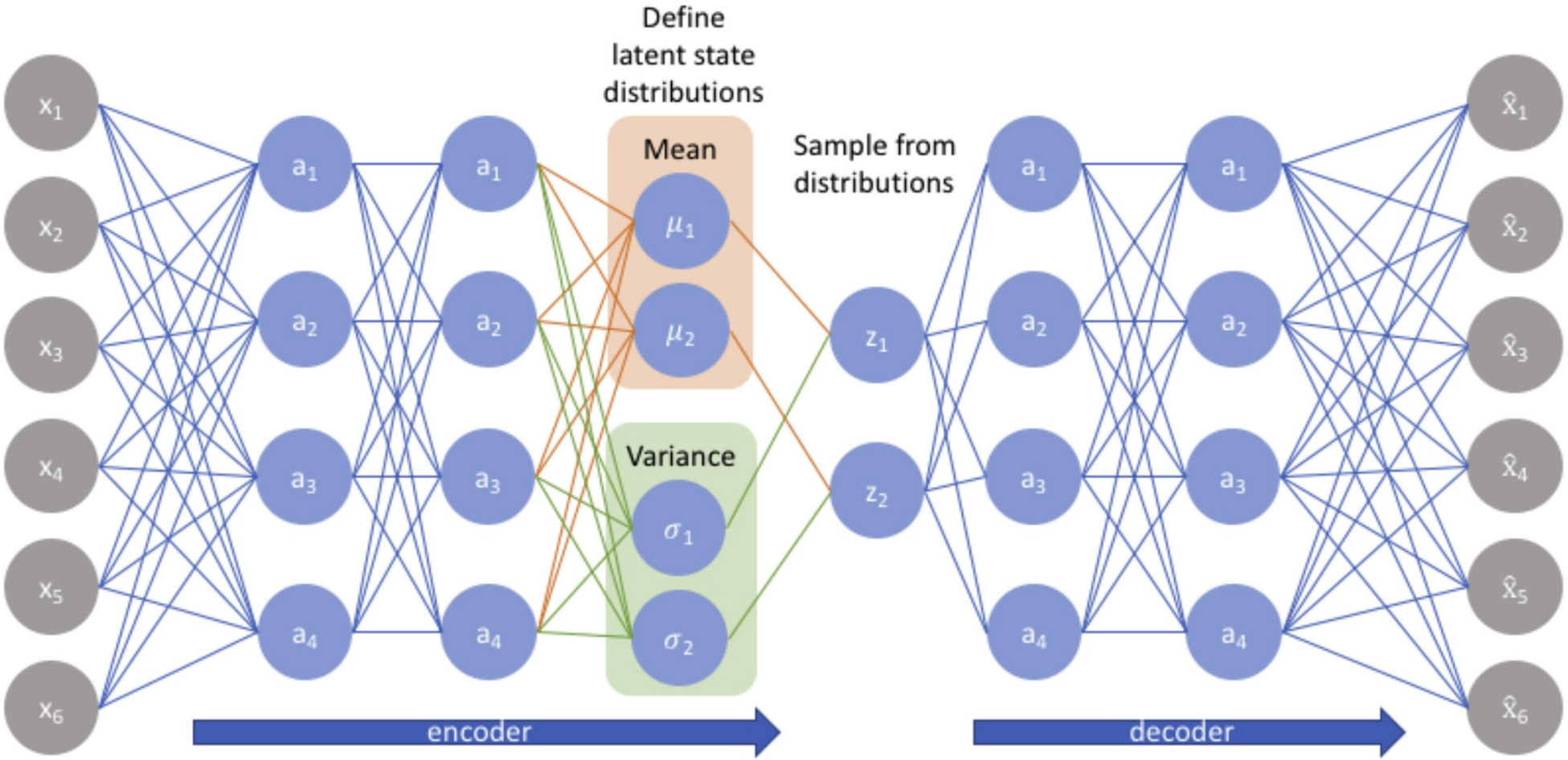
$$\mathcal{L}\left(x, \hat{x}\right) + \sum_{j} KL\left(q_j\left(z|x\right)||p\left(z\right)\right)$$

- Loss function for this network consists of two terms
  - Penalizes reconstruction error
  - A second term which encourages our learned distribution q(z|x) to be similar to the true prior distribution p(z)
- We most often assume that the prior follows a unit Gaussian distribution, for each dimension j of the latent space

Implementation

- Rather than directly outputting values for the latent state as we would in a standard autoencoder, the encoder model of a VAE will output parameters describing a distribution for each dimension in the latent space
- Since we're assuming that our prior follows a normal distribution, we'll output *two* vectors describing the mean and variance of the latent state distributions
- If we were to build a true multivariate Gaussian model, we'd need to define a covariance matrix describing how each of the dimensions are correlated

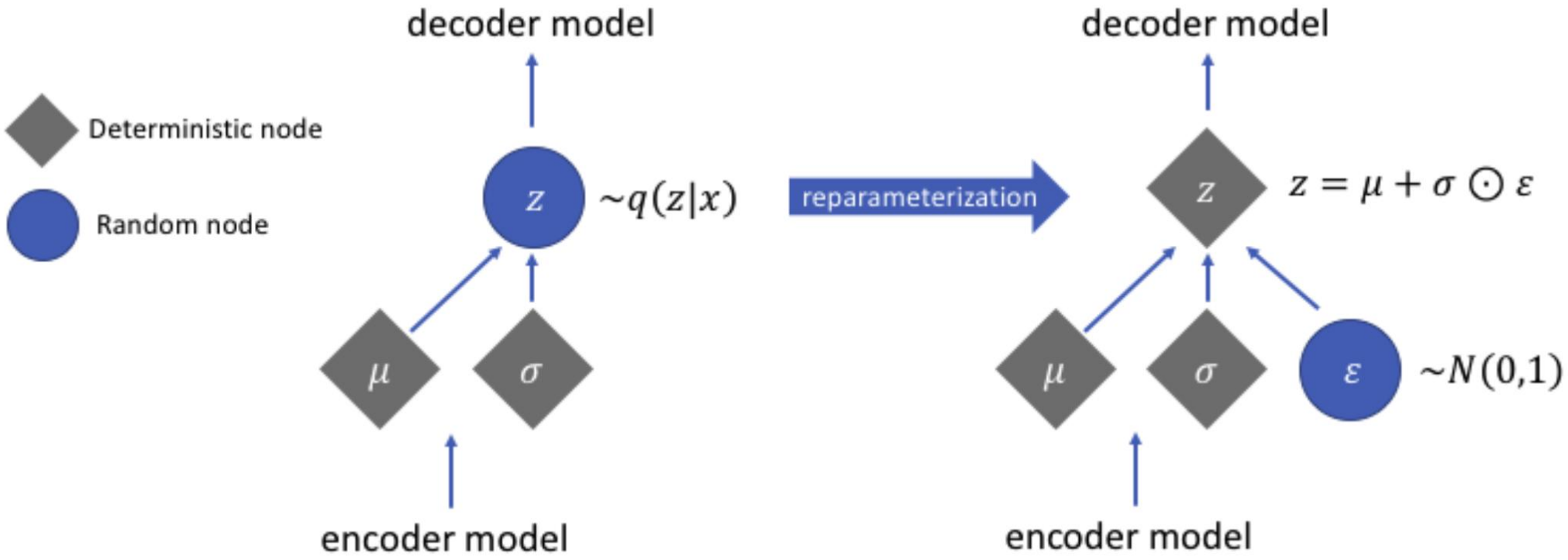
Implementation



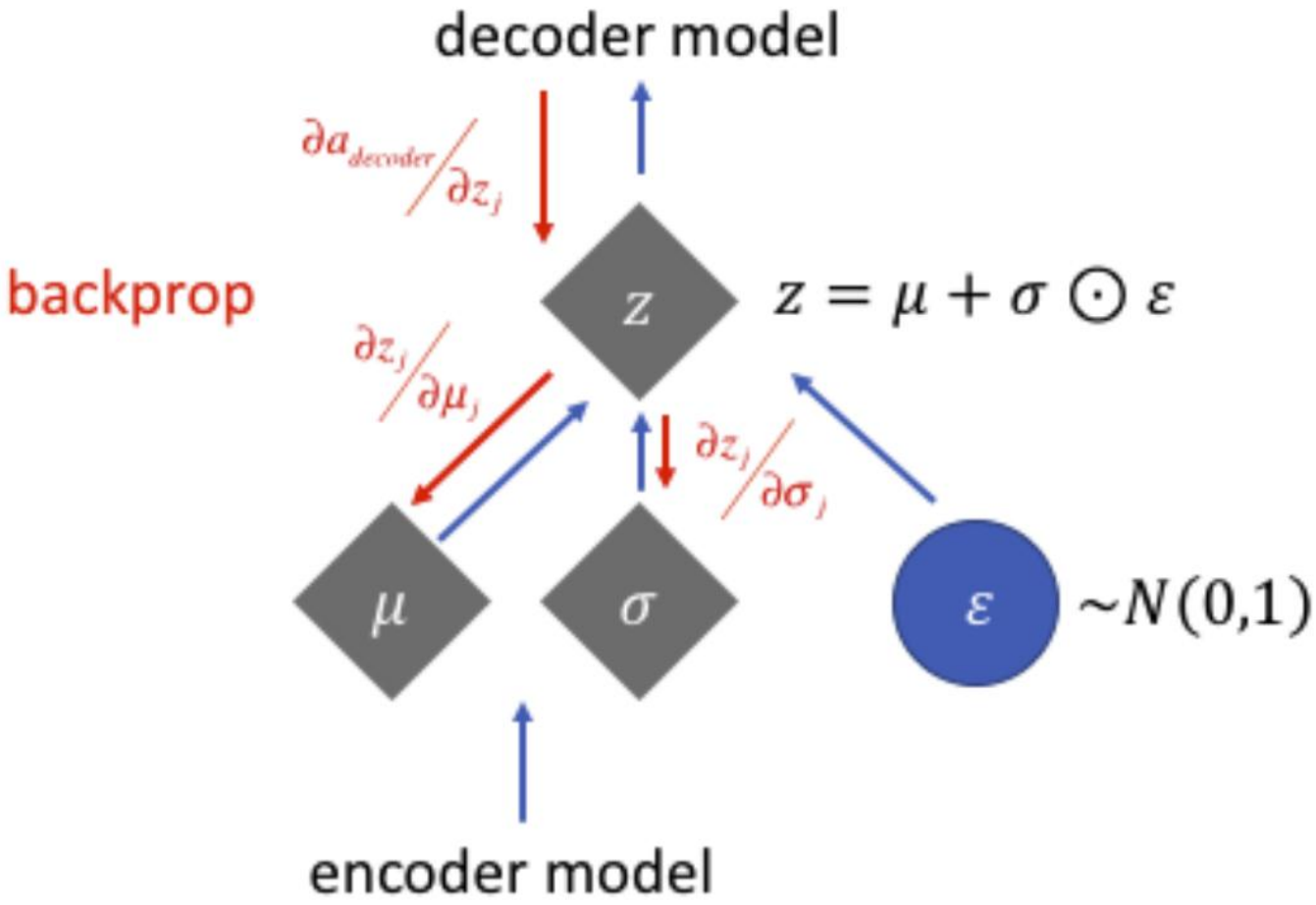
Implementation

- However, the sampling process (while decoding) requires some extra attention
- When training the model, we need to be able to calculate the relationship of each parameter in the network with respect to the final output loss using backpropagation
- We cannot do this for a random sampling process
- However we can do the "reparameterization trick“
  - We randomly sample ε from a unit Gaussian, and then shift the randomly sampled ε by the latent distribution's mean μ and scale it by the latent distribution's variance σ

Reparameterization trick



After the reparameterization trick



Statistical motivation – Self Study!!!

$$D_{KL}[Q(z|X)\|P(z|X)] = E[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log P(X)$$

$$D_{KL}[Q(z|X)\|P(z|X)] - \log P(X) = E[\log Q(z|X) - \log P(X|z) - \log P(z)]$$

$$D_{KL}[Q(z|X)\|P(z|X)] - \log P(X) = E[\log Q(z|X) - \log P(X|z) - \log P(z)]$$

$$\log P(X) - D_{KL}[Q(z|X)\|P(z|X)] = E[\log P(X|z) - (\log Q(z|X) - \log P(z))]$$

$$= E[\log P(X|z)] - E[\log Q(z|X) - \log P(z)]$$

$$= E[\log P(X|z)] - D_{KL}[Q(z|X)\|P(z)]$$