AUTOENCODERS

A general principle of generation

- Data is encoded into a different representation
- New data is generated by sampling from the new representation
- GMMs are just one type of encoding-decoding scheme

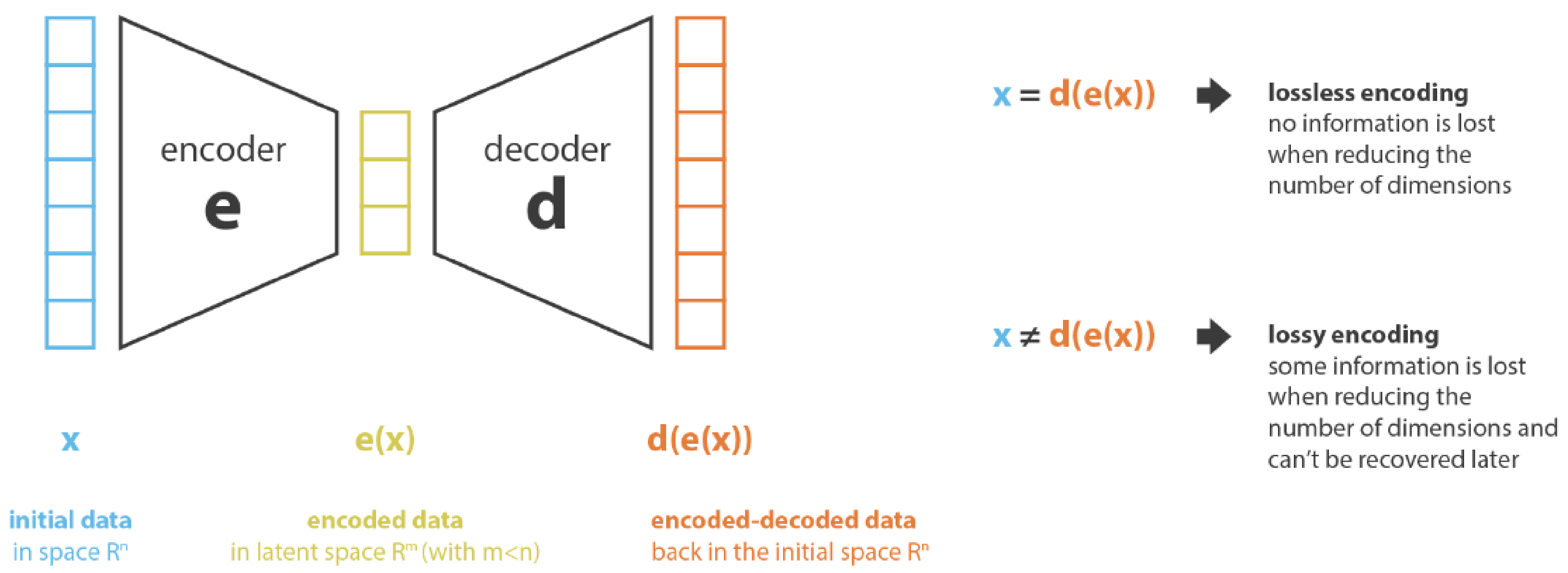
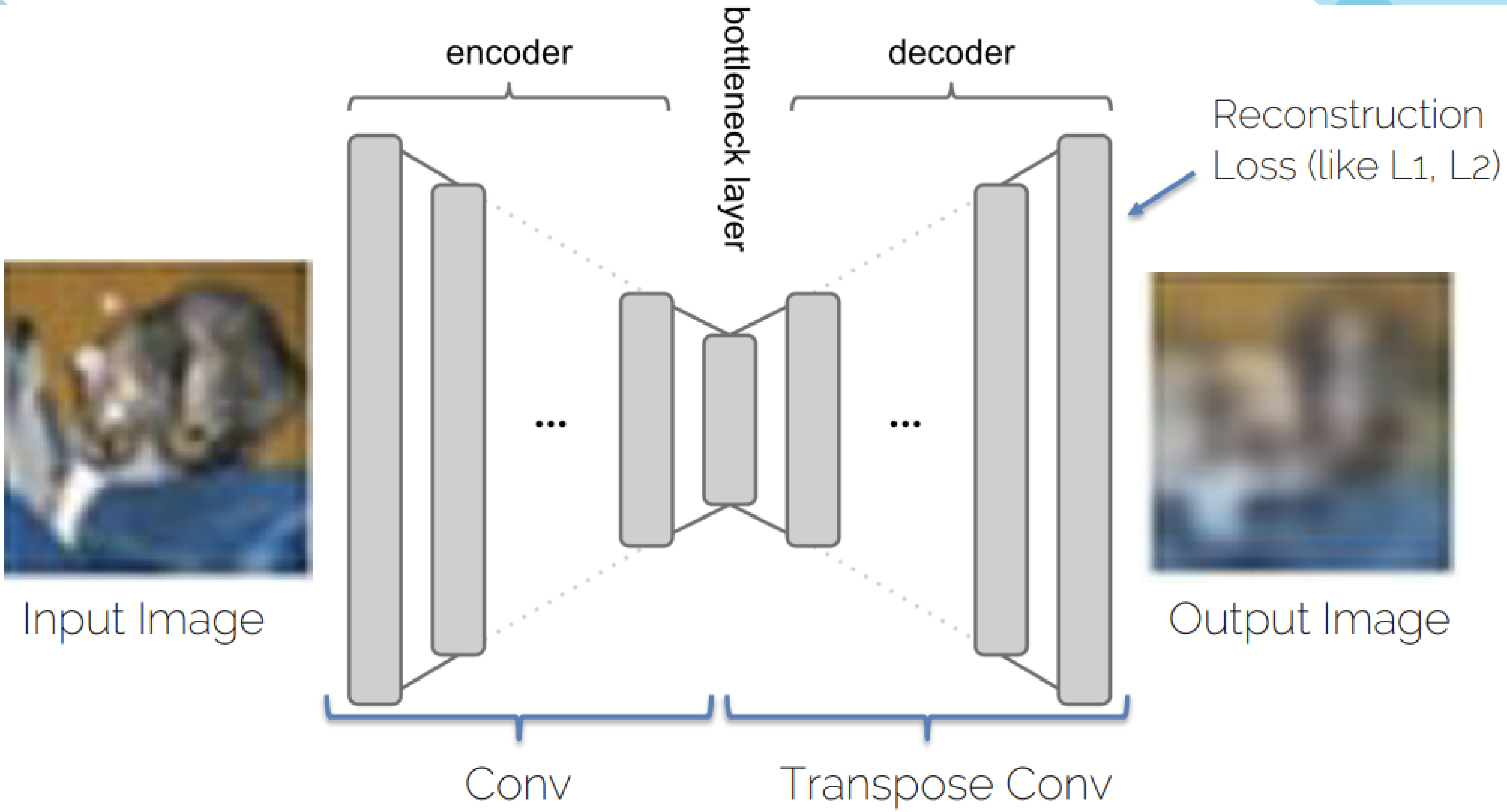


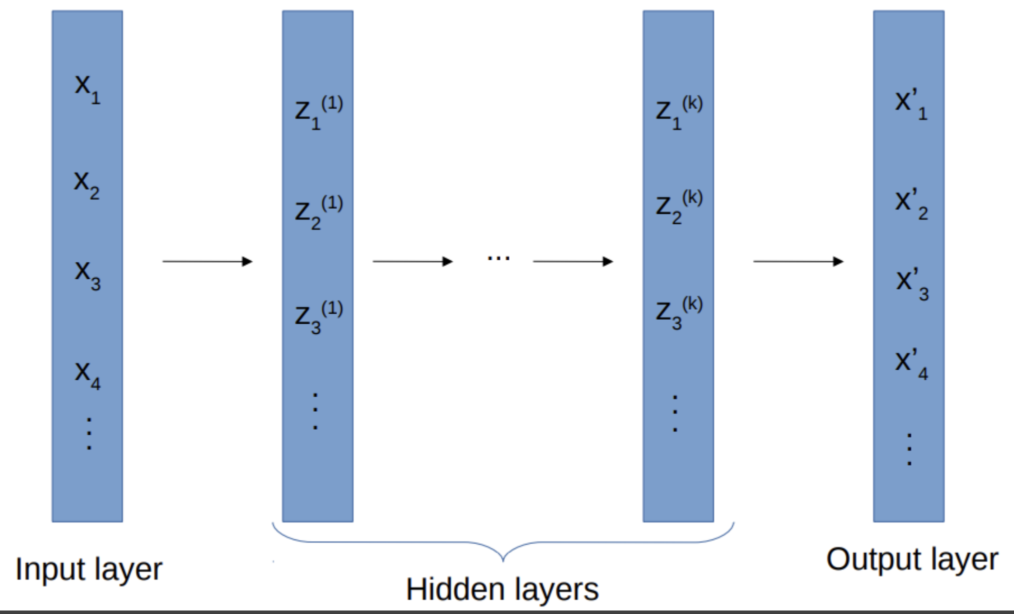
Image credit (link)

Autoencoders consist of two main parts:

The encoder: Compresses the input data into a lower-dimensional latent space.

The decoder: Reconstructs the original data from this compressed representation.


encoder
bottleneck layer
decoder
Reconstruction Loss (like L1, L2)
...
...
Input Image
Output Image
Conv
Transpose Conv



- Encoder stage : map the input x to z

$$z=\delta(Wx+b)$$

δ : element-wise activation function

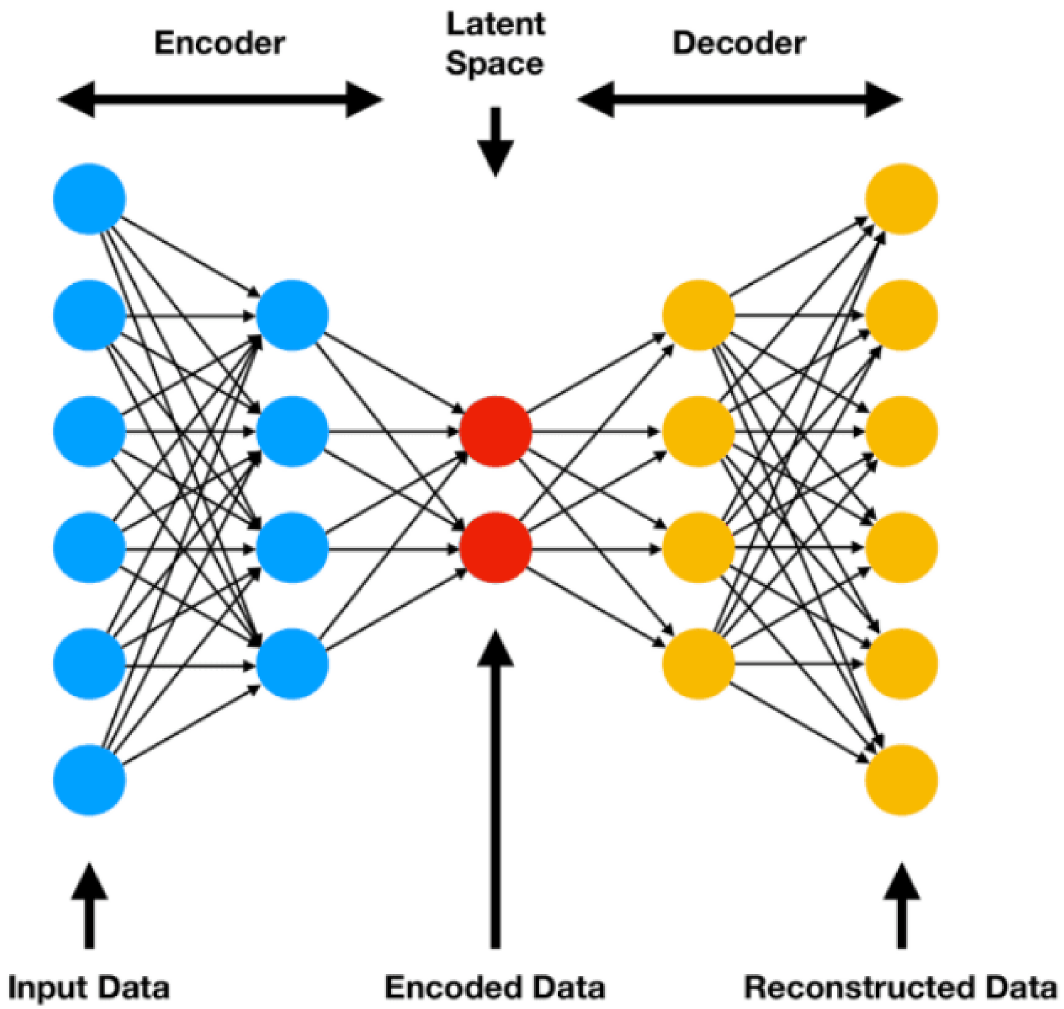
- Decoder stage : map z to reconstruction x'

$$x'=\delta'(W'z+b')$$

- Autoencoders are trained to minimise :

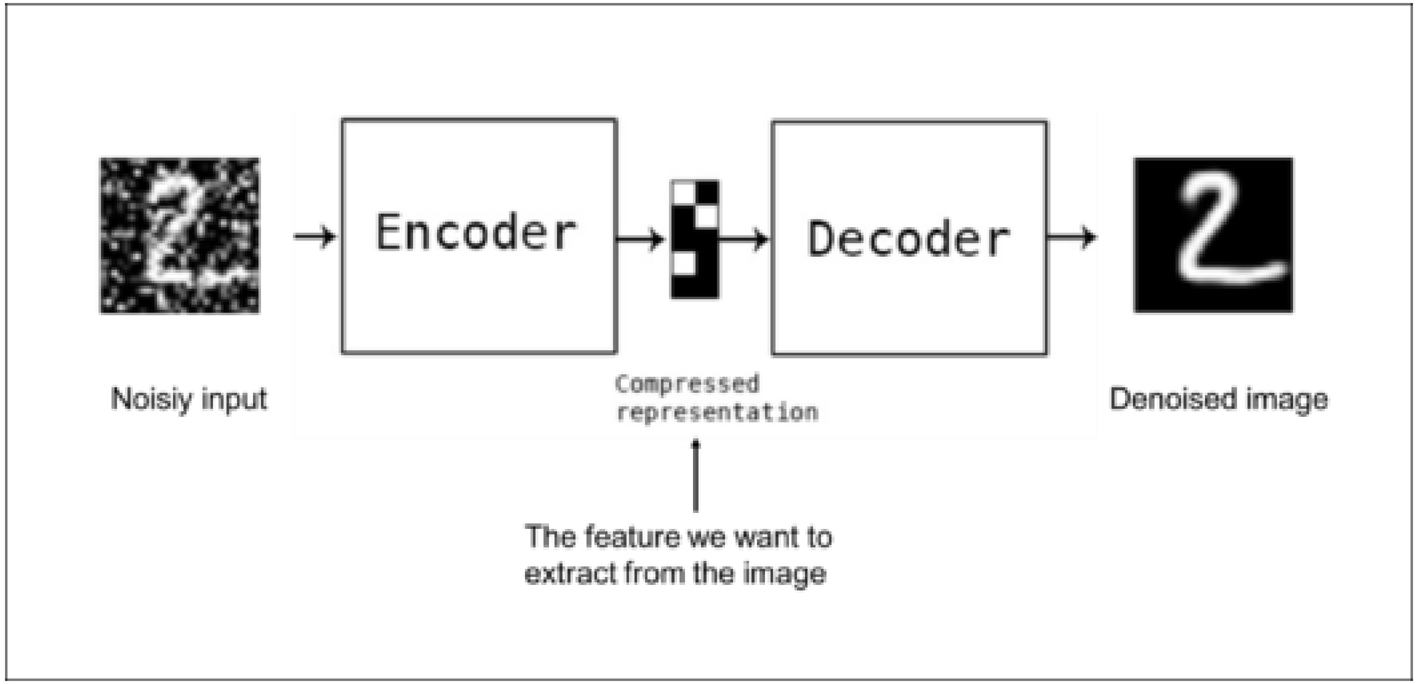
$$L(x,x')=||x-x'||^2$$

Autoencoder



Autoencoders: applications

- Denoising: input clean image + noise and train to reproduce the clean image.



https://www.edureka.co/blog/autoencoders-tutorial/

Autoencoders: Applications

- Image colorization: input black and white and train to produce color images

https://www.edureka.co/blog/autoencoders-tutorial/

Autoencoders: Applications

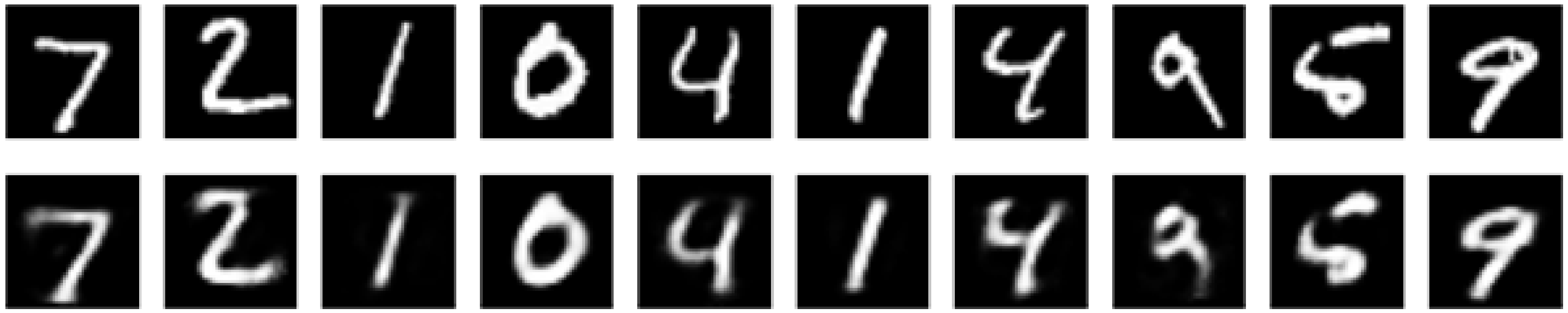
- Watermark removal

https://www.edureka.co/blog/autoencoders-tutorial/

Simple bottleneck layer in Keras

  - input_img = Input(shape=(784,))
  - encoding_dim = 32
  - encoded = Dense(encoding_dim, activation='relu')(input_img)
  - decoded = Dense(784, activation='sigmoid')(encoded)
  - autoencoder = Model(input_img, decoded)

- Maps 28x28 images into a 32 dimensional vector.



https://blog.keras.io/building-autoencoders-in-keras.html

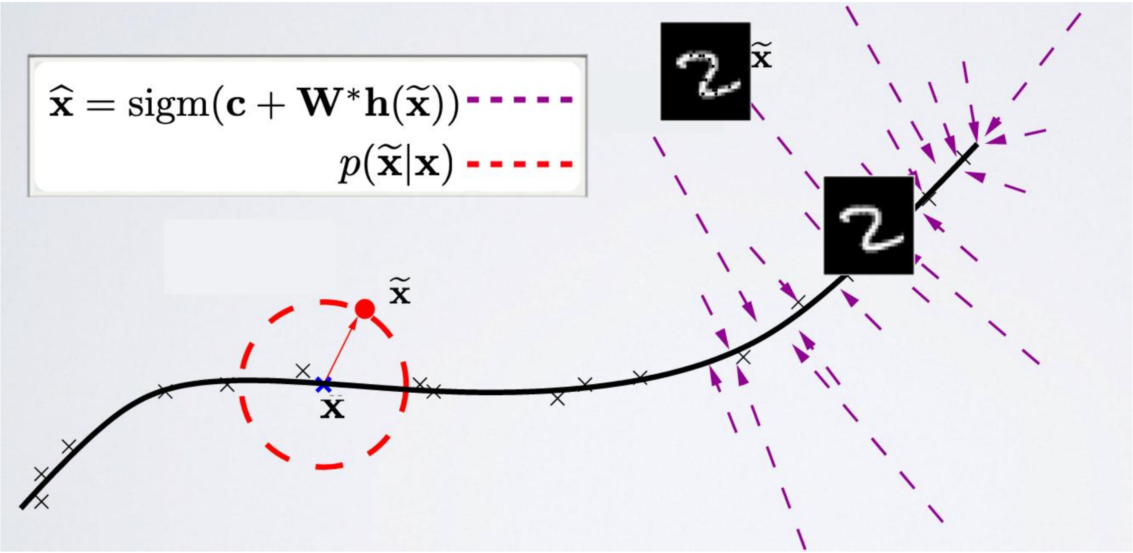
Denoising autoencoders

- Basic autoencoder trains to minimize the loss between x and the reconstruction g(f(x)).
- Denoising autoencoders train to minimize the loss between x and g(f(x+w)), where w is random noise.
- Same possible architectures, different training data.
- Kaggle has a dataset on damaged documents.

https://blog.keras.io/building-autoencoders-in-keras.html

Denoising autoencoders

- Denoising autoencoders can’t simply memorize the input output relationship.
- Intuitively, a denoising autoencoder learns a projection from a neighborhood of our training data back onto the training data.



https://ift6266h17.files.wordpress.com/2017/03/14_autoencoders.pdf

Sparse autoencoders

- Construct a loss function to penalize *activations* within a layer.
- Usually regularize the *weights* of a network, not the activations.

How It Works:

- The autoencoder is trained with an additional **sparsity penalty** (like KL-divergence or L1 regularization) that forces most neurons in the hidden layer to remain inactive for most inputs.
- This encourages the network to learn a **disentangled and meaningful** representation of the input data.

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

Mathematical Formulation of Sparse autoencoders

Given the activation of the hidden unit j as a_j, the sparsity constraint is applied using KL-divergence:

$$KL(\rho||\hat{\rho}) = \sum_j \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

where:

- ρ is the target sparsity (small value, e.g., 0.05),
- $\hat{\rho}_j$ is the average activation of neuron j.

Regularization Used:

•**L1 Regularization (Lasso)**: Adds an absolute value penalty to the activations.

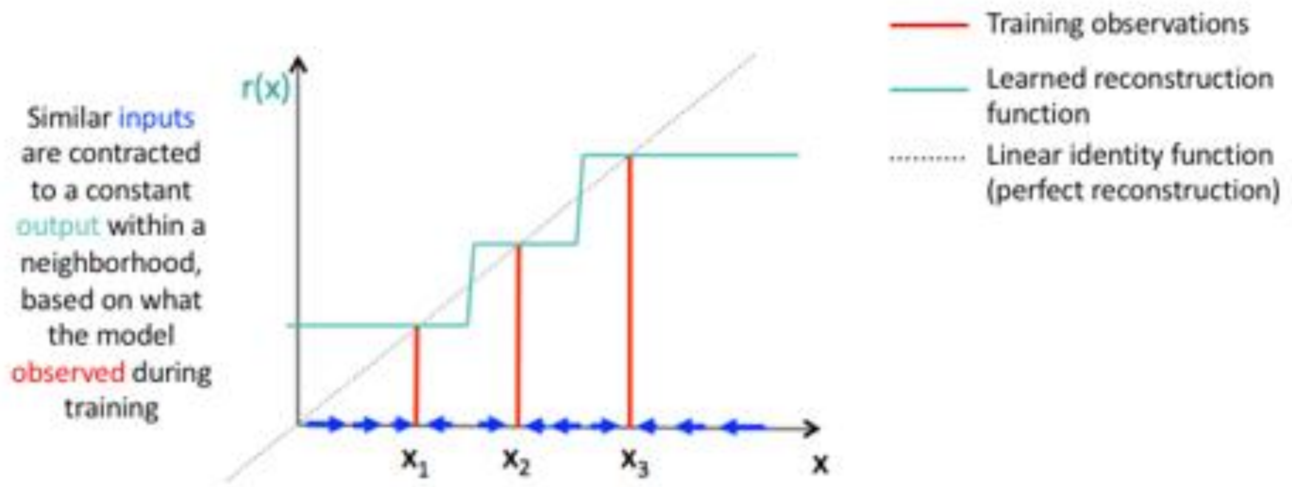
•**KL-Divergence**: Ensures that the average activation of hidden neurons stays close to a low desired value.

Contractive autoencoders

How It Works:

- Instead of enforcing sparsity, a **contractive penalty** is added, which minimizes the **sensitivity of the hidden representation to small input variations**.
- This prevents overfitting and makes the autoencoder robust to noise.

$$\mathcal{L}(x, \hat{x}) + \lambda \sum_i \left\| \nabla_x a_i^{(h)}(x) \right\|^2$$



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

Mathematical Formulation of Contractive Autoencoders

- **Regularization Used:**

Jacobian Regularization: Penalizes large gradients of the encoder output with respect to the input.

The contractive loss adds a penalty term:

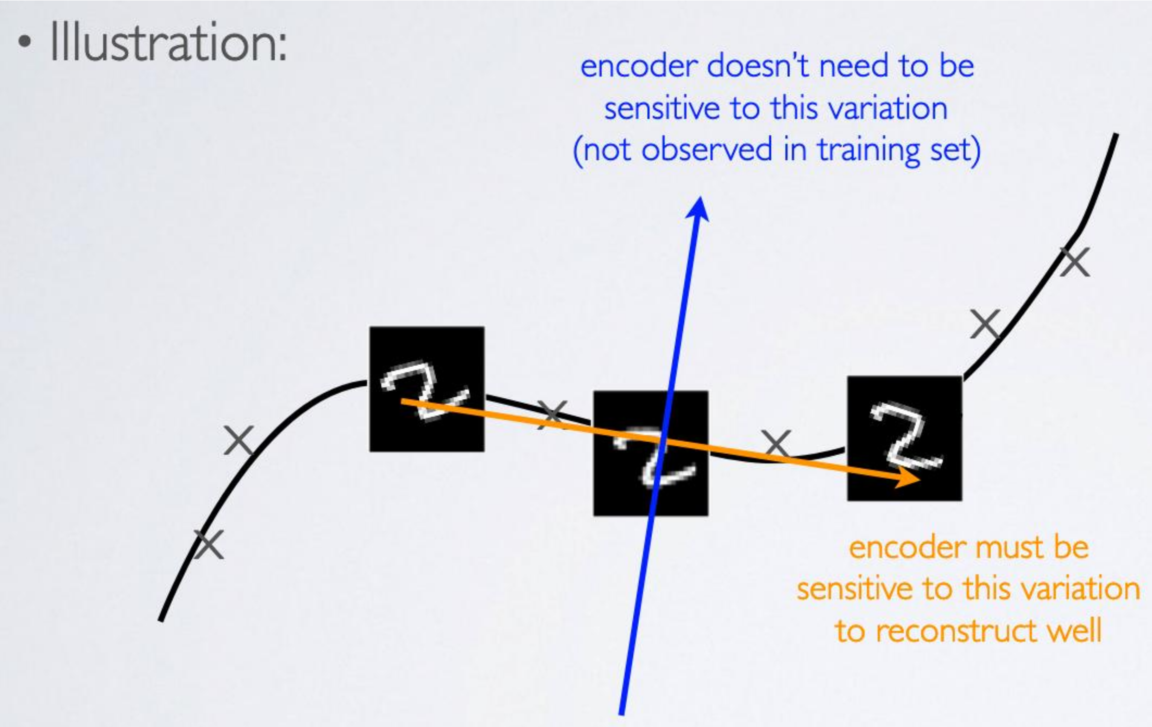
$$\mathcal{L} = \|x - \hat{x}\|^2 + \lambda \sum_j \|\nabla_x h_j\|^2$$

where:

- x is the input,
- $\hat{x}$ is the reconstructed output,
- h_j is the activation of neuron j,
- $\nabla_x h_j$ is the Jacobian matrix of the activations,
- λ controls the weight of the contractive penalty.

Contractive autoencoders

- Contractive autoencoders make the *feature extraction function* (ie. encoder) resist infinitesimal perturbations of the input.



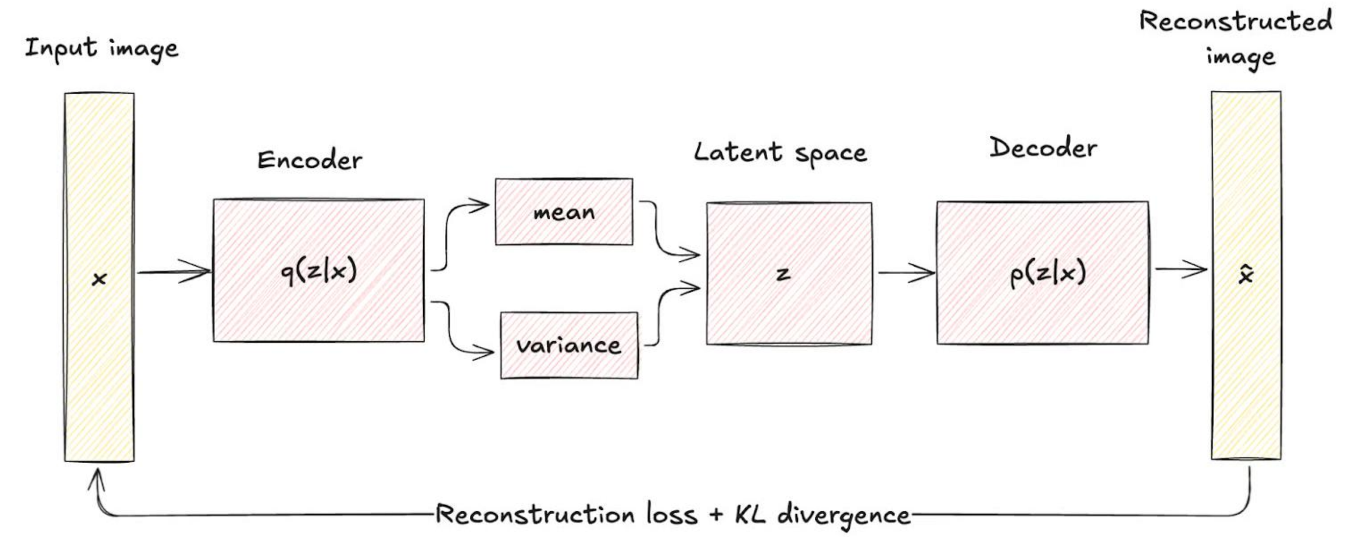
https://ift6266h17.files.wordpress.com/2017/03/14_autoencoders.pdf

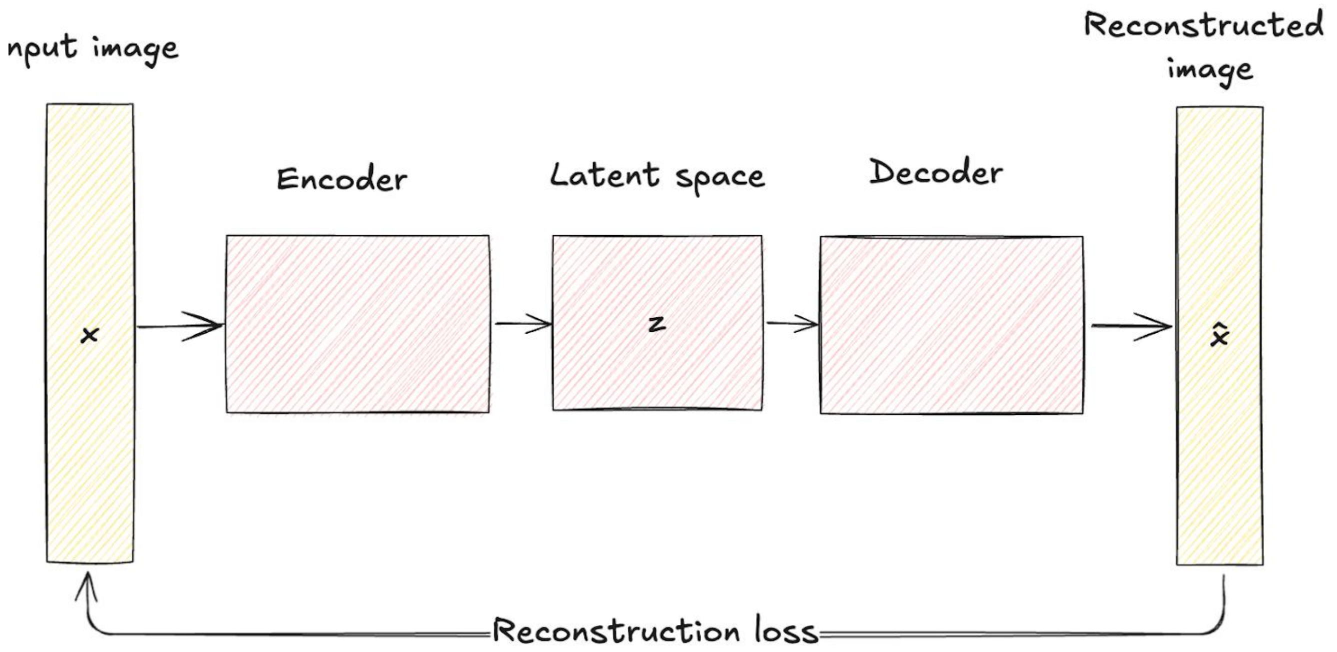
Variational Autoencoder

- **Key innovation of VAEs lies in their ability to generate new, high-quality data by learning a structured, continuous latent space.**

- Traditional autoencoders produce a fixed point in the latent space (the network that maps the input data *x* to a fixed, lower-dimensional latent space representation *z*. This process is deterministic, meaning each input is encoded into a specific point in the latent space). The decoder network then reconstructs the original data from this fixed latent representation, aiming to minimize the difference between the input and its reconstruction.

- Encoder in a VAE outputs parameters of a probability distribution—typically the mean and variance of a Gaussian distribution (encoder in a VAE maps the input data to a probability distribution over the latent variables, typically modeled as a Gaussian distribution with mean μ and variance σ^2). During training, we sample a point from this distribution to feed into the decoder.



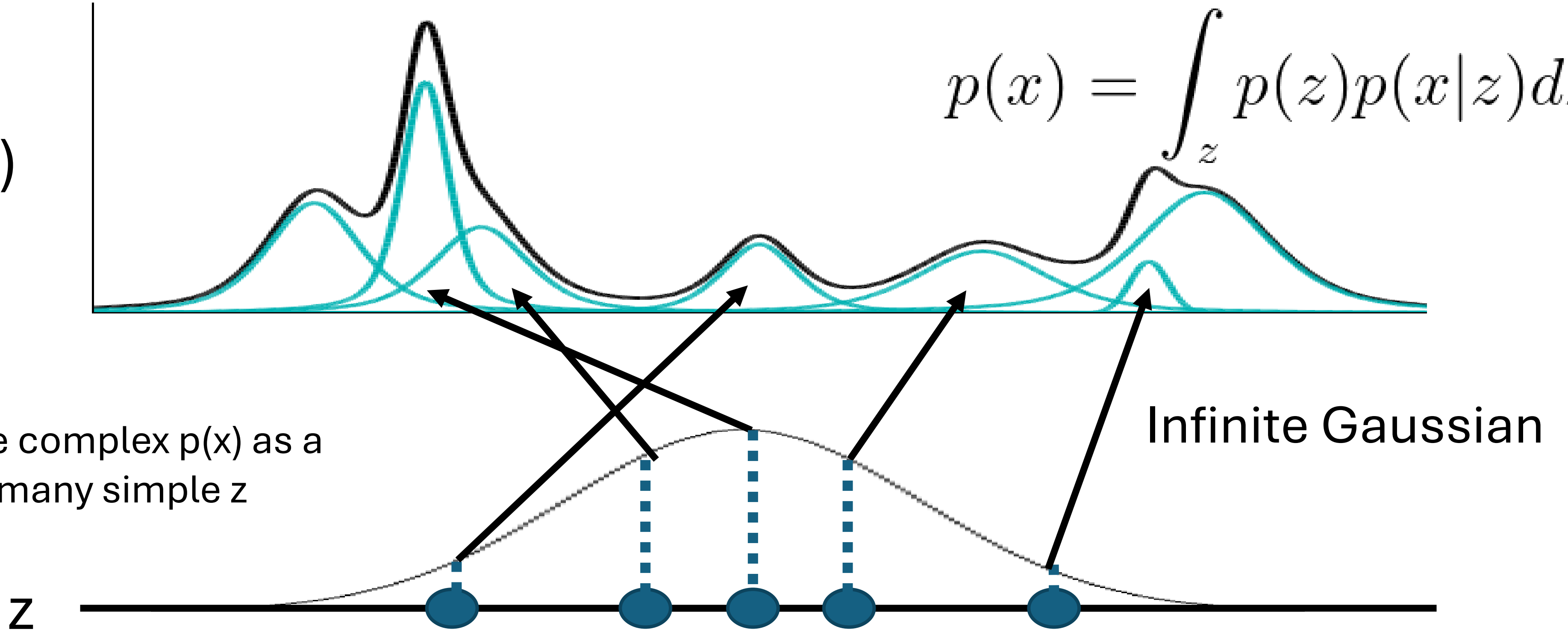


VAEs concept

$$z \sim N(0,1)$$
$$x|z \sim N(\mu(z), \sigma(z))$$

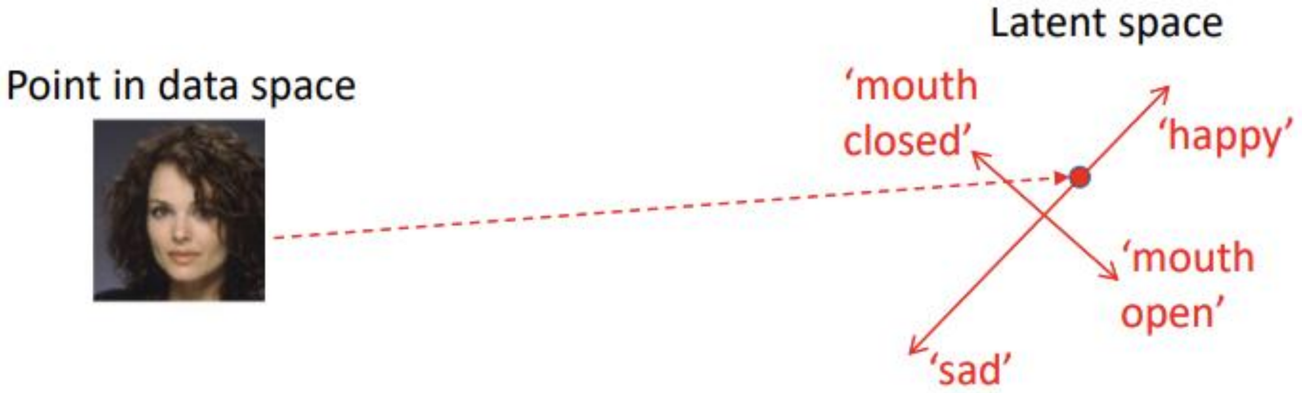
Each dimension of z represents a data attribute

$$p(x) = \int_z p(z)p(x|z)dz$$



We approximate complex p(x) as a composition of many simple z

This approach encodes each input into a distribution rather than a single point, adding a layer of variability and uncertainty.



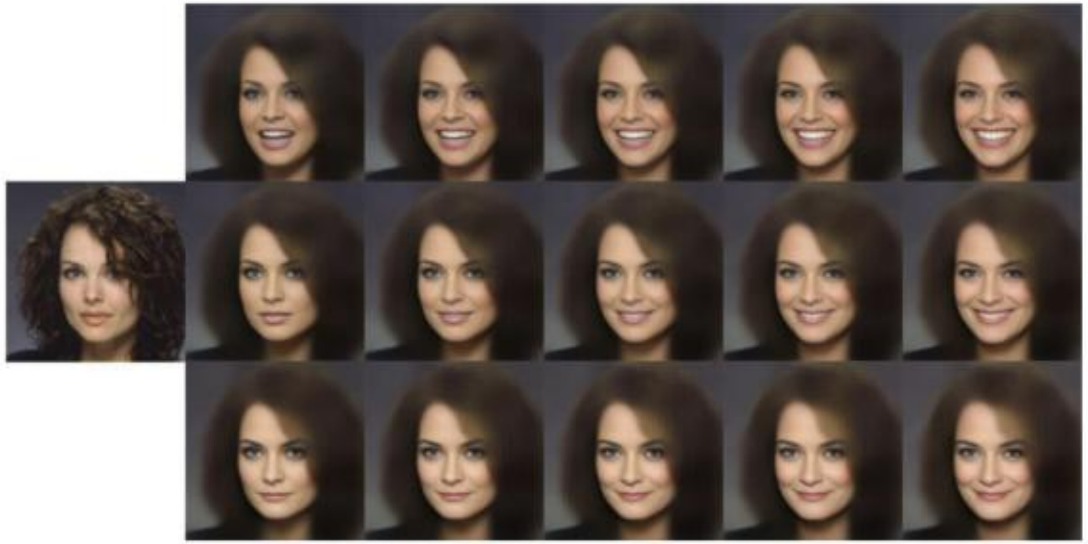
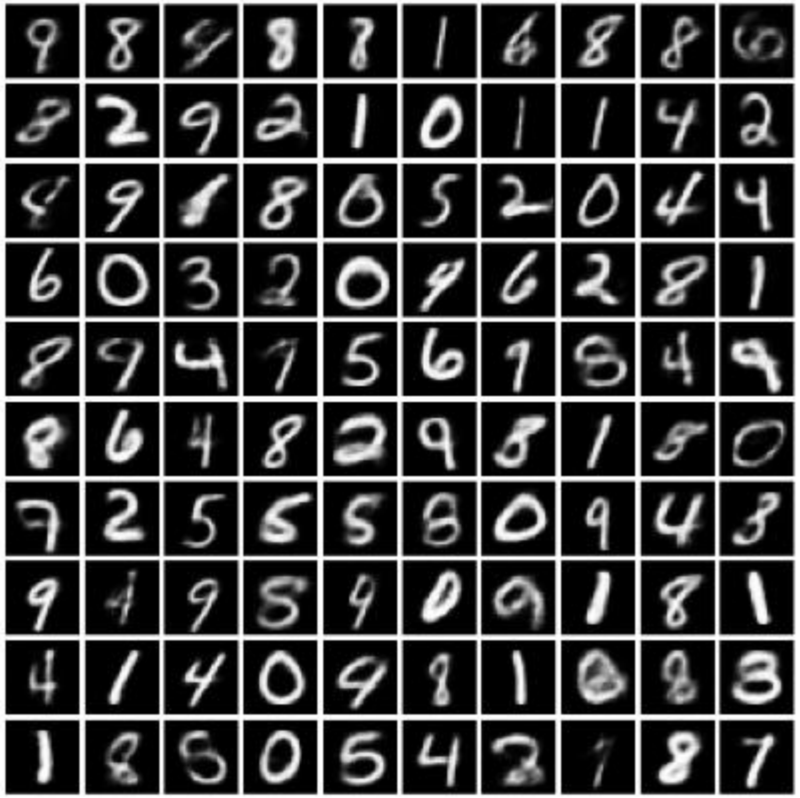


Figure 7: Decoupling attribute vectors for smiling (x-axis) and mouth open (y-axis) allows for more flexible latent space transformations. Input shown at left with reconstruction adjacent. (model: VAE from Lamb 16 on CelebA)

Figure 4.4: VAEs can be used for image resynthesis. In this example by White, 2016, an original image (left) is modified in a latent space in the direction of a *smile vector*, producing a range of versions of the original, from smiling to sadness.

https://arxiv.org/vc/arxiv/papers/1609/1609.04468v2.pdf

VAE outputs



Samples from a VAE trained on MNIST

Samples from a VAE trained on a faces dataset

VAE limitations

- People have mostly moved on from VAEs to use GANs for generating synthetic high-dimensional data
- VAEs are theoretically complex
- Don’t generalize very well
- Are pragmatically under-constrained
  - Reconstruction error need not be exactly correlated with realism

Realistic

Fake

Who is real ?

[Play](#)

https://www.whichfaceisreal.com/index.php