ARTIFICIAL INTELLIGENCE 2

Autoencoders

Francesco Spinnato

* francesco.spinnato@unipi.it
University of Pisa

Supervised and Unsupervised Learning

Supervised

In supervised learning, each training example $\mathbf{x}$ is associated with a ground-truth label or target y.

Typical tasks:

- classification
- regression
- forecasting

Unsupervised

In unsupervised learning, no ground-truth labels are provided.

Typical tasks:

- clustering
- dimensionality reduction
- anomaly detection
- data generation

Why Unsupervised Learning in Healthcare?

Labels are often expensive, noisy, or incomplete.

In healthcare and biology, we may have:

- many patient records but few reliable diagnoses
- many images but few expert annotations
- many omics profiles but unclear phenotypes
- many time series but uncertain clinical events

So we want models that can **learn useful structure from the input data itself.**

Representation Learning

A neural network can be used not only to predict a target, but also to learn a useful **representation** of the input.

$$\mathbf{x} \longrightarrow \mathbf{z}$$

where:

- $\mathbf{x}$ is the original input
- $\mathbf{z}$ is a new representation, often called **latent representation**
- $\mathbf{z}$ should preserve the most relevant information in $\mathbf{x}$

The representation $\mathbf{z}$ can then be used for visualization, clustering, anomaly detection, or downstream prediction.

From Data to Latent Representations

Input Space

Original data representation:

- tabular variables
- pixels
- time points
- genes
- medical codes

Often high-dimensional, sparse, noisy, or redundant.

Latent Space

Compressed learned representation:

- lower-dimensional
- continuous
- task-independent
- hopefully more informative

Used as a compact summary of the input.

$$\mathbf{x} \in \mathbb{R}^M \quad \longrightarrow \quad \mathbf{z} \in \mathbb{R}^d \qquad \text{with usually } d < M$$

Autoencoders

Basics

What Is an Autoencoder?

An **autoencoder** is a neural network trained to reconstruct its own input.
It has two main components:

Encoder

Maps the input to a latent representation:

$$\mathbf{z} = f_\theta(\mathbf{x})$$

Decoder

Maps the latent representation back to the input space:

$$\hat{\mathbf{x}} = g_\phi(\mathbf{z})$$

The model is trained so that $\hat{\mathbf{x}}$ is as close as possible to $\mathbf{x}$.

History of Autoencoders

Autoencoders are not a recent idea. They were introduced as neural networks for learning compressed internal representations.

- **1980s:** autoencoder-like networks were studied for dimensionality reduction and representation learning
- **1986:** backpropagation made it possible to train multilayer neural networks effectively
- **2000s:** autoencoders became important for unsupervised pretraining of deep networks
- **2010s:** denoising, sparse, and variational autoencoders became widely used
- **Today:** autoencoders are used for representation learning, anomaly detection, denoising, and generative modeling

Main idea: Learn a useful internal representation by reconstructing the input.

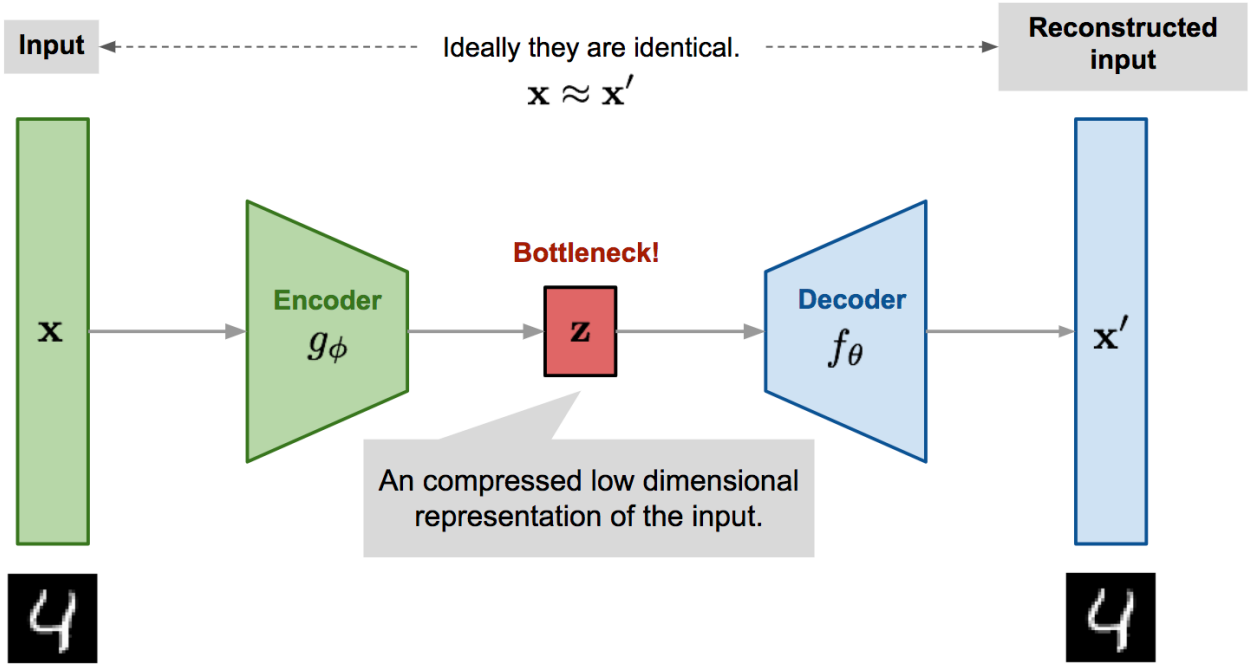
Rumelhart et al. (1986); Hinton and Salakhutdinov (2006); Kingma and Welling (2013).

Autoencoder Architecture

The autoencoder is composed of:

- an **encoder**, which compresses the input
- a **bottleneck**, which stores the latent representation
- a **decoder**, which reconstructs the input

$$\mathbf{x} \xrightarrow{f_\theta} \mathbf{z} \xrightarrow{g_\phi} \hat{\mathbf{x}}$$



Reconstruction Objective

The autoencoder is trained by minimizing the difference between the input $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}} = g_\phi(f_\theta(\mathbf{x}))$.

A common objective for continuous data is the mean squared reconstruction error:

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \left\| \mathbf{x} - \hat{\mathbf{x}} \right\|_2^2$$

Over a dataset:

$$\min_{\theta,\phi} \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{x}_i - g_\phi(f_\theta(\mathbf{x}_i)) \right\|_2^2$$

Why Is This Not Trivial?

At first sight, reconstructing the input may seem useless.

But the model is forced to learn something meaningful if we constrain it, e.g.:

- make the latent representation smaller than the input
- add noise to the input
- force sparse activations
- impose a probabilistic structure on the latent space

Without constraints, the network may simply learn the identity function:

$$\hat{\mathbf{x}} \approx \mathbf{x}$$

The Bottleneck

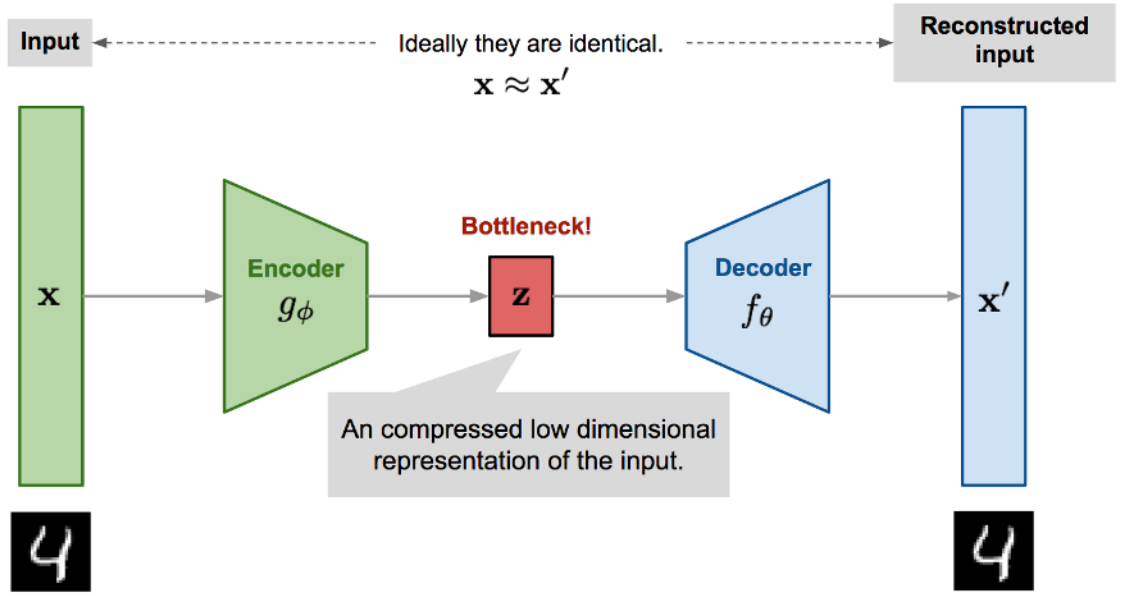
The bottleneck is the central idea.

If the latent space is smaller than the input space:

$$\mathbf{x} \in \mathbb{R}^M, \qquad \mathbf{z} \in \mathbb{R}^d, \qquad d < M$$

then the autoencoder must decide what information to preserve.

This encourages the model to learn a lossy compressed representation of the data.



What Does the Latent Space Contain?

The latent representation $\mathbf{z}$ should capture the main factors of variation in the data. Examples:

- In medical images: anatomical structures, texture patterns, lesions or abnormalities
- In omics data: biological pathways, cell types, disease-related expression patterns
- In patient records: clinical phenotype, disease progression, treatment patterns

The model is not explicitly told these factors. It must infer them from reconstruction.

Autoencoders and Principal Component Analysis

A useful intuition:

- PCA learns a **linear** low-dimensional representation
- A basic autoencoder can learn a **nonlinear** low-dimensional representation

With linear activations and mean squared error, a simple undercomplete ($\mathbf{z}$ smaller than $\mathbf{x}$) autoencoder is closely related to PCA.

With nonlinear activations and multiple layers, autoencoders can learn more flexible representations.

PCA: linear compression

Autoencoder: nonlinear compression

Training and Inference

Training

Use the input as its own target:

$$\mathbf{x} \rightarrow \hat{\mathbf{x}}$$

Minimize reconstruction error:

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$$

After Training

Use the encoder to extract the latent representation:

$$\mathbf{z} = f_\theta(\mathbf{x})$$

Use $\mathbf{z}$ for downstream analysis.

Autoencoders Are Not One Architecture

An autoencoder is not a specific neural-network layer, it is a training setup:

$$\text{input} \rightarrow \text{latent representation} \rightarrow \text{reconstruction}$$

The encoder and decoder can be built with different architectures:

Data type	Encoder / Decoder
Tabular data	MLP
Images	CNN
Time series	RNN, CNN, Transformer
Text	RNN, Transformer

The architecture should match the structure of the input data.

MLP Autoencoder

For tabular data, an autoencoder can be built using fully connected layers.

This is useful for:

- dimensionality reduction
- denoising tabular data
- anomaly detection
- omics representations

Example: MLP Autoencoder Forward Pass

Consider a very small undercomplete MLP autoencoder.

$$\mathbf{x} = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \qquad \mathbf{W}_{enc} = \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ 1 & 1 \\ -1 & 0 \end{bmatrix} \qquad \mathbf{W}_{dec} = \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & -1 \end{bmatrix}$$

- The latent representation has dimension 2, while the input has dimension 4.
- Activation function on the latent layer: $f(x) = \mathrm{ReLU}(x)$

Compute:

$$\mathbf{z} = f(\mathbf{x}\mathbf{W}_{enc}) \qquad \hat{\mathbf{x}} = \mathbf{z}\mathbf{W}_{dec}$$

For simplicity, we omit bias terms.

Example: MLP Autoencoder Forward Pass

First compute the encoder output before activation:

$$\mathbf{x}\mathbf{W}_{enc} = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ 1 & 1 \\ -1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \cdot 1 + 2 \cdot 0 + 0 \cdot 1 + 1 \cdot (-1) & 1 \cdot (-1) + 2 \cdot 2 + 0 \cdot 1 + 1 \cdot 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 3 \end{bmatrix}$$

Now apply ReLU:

$$\mathbf{z} = \mathrm{ReLU}\left(\begin{bmatrix} 0 & 3 \end{bmatrix}\right) = \begin{bmatrix} 0 & 3 \end{bmatrix}$$

Example: MLP Autoencoder Forward Pass

Now reconstruct the input using the decoder:

$$\hat{\mathbf{x}} = \mathbf{z}\mathbf{W}_{dec} = \begin{bmatrix} 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \cdot 1 + 3 \cdot 0 & 0 \cdot 0 + 3 \cdot 1 & 0 \cdot (-1) + 3 \cdot 1 & 0 \cdot 2 + 3 \cdot (-1) \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 3 & 3 & -3 \end{bmatrix}$$

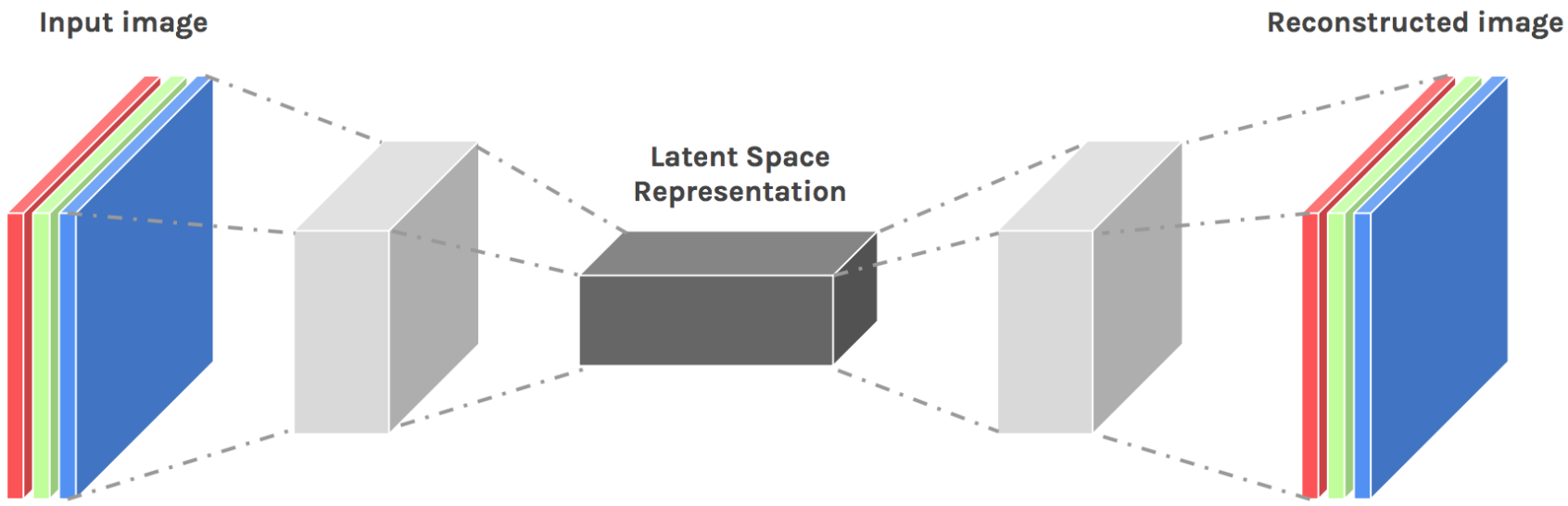
So the autoencoder maps:

$$\mathbf{x} = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \rightarrow \mathbf{z} = \begin{bmatrix} 0 & 3 \end{bmatrix} \rightarrow \hat{\mathbf{x}} = \begin{bmatrix} 0 & 3 & 3 & -3 \end{bmatrix}$$

The reconstruction is not good yet because the weights have not been trained.

CNN Autoencoder

For images, the encoder usually uses convolutional layers.
The decoder reconstructs the image using upsampling or transposed convolutions.



Useful for image denoising, image compression, segmentation pretraining, reconstruction of medical images

How Does the Decoder Increase Image Size?

The decoder commonly uses either:

Upsampling + Convolution

First enlarge the feature map, e.g.:

$$H' \times W' \rightarrow 2H' \times 2W'$$

Then apply a normal convolution to refine the result.

Transposed Convolution

A learnable operation that increases spatial resolution directly.

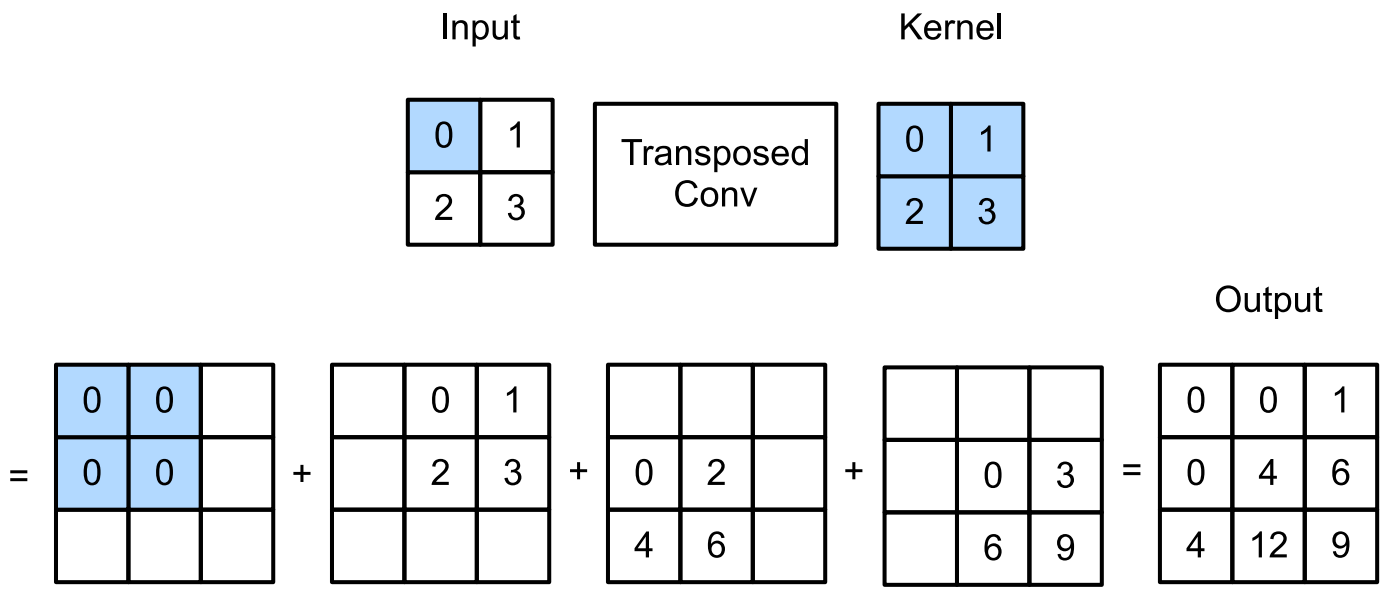
It is often called “deconvolution”, but this name is misleading.

It is not the mathematical inverse of convolution.

Transposed Convolution

Transposed convolution is used in the decoder to make feature maps larger. Instead of sliding a kernel over the input to produce one output value, each input value is used to **place a small weighted patch** into the output.

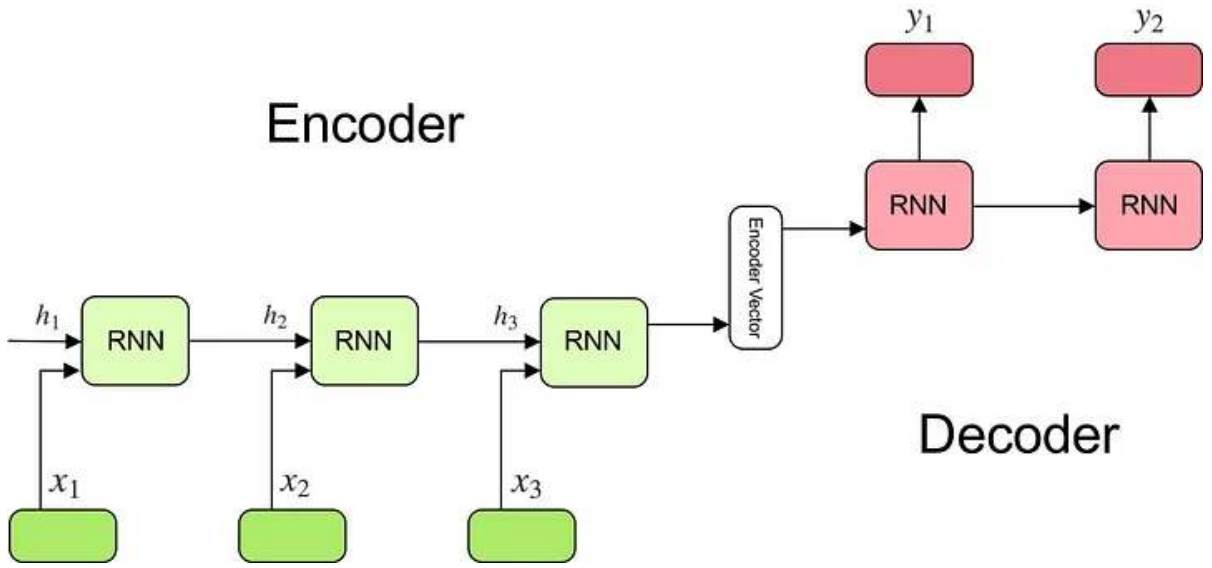
Input values are multiplied by the kernel: $x_{i,j} \cdot \mathbf{K}$. The result is written into the corresponding position of a larger output map. Overlapping regions are summed.



RNN Autoencoder

An RNN autoencoder is an autoencoder for sequential data.

- The encoder reads the input sequence one step at a time
- The decoder reconstructs the sequence one step at a time



Useful for physiological signals, patient trajectories, clinical event sequences, anomaly detection in time series

The Decoder and Context Vector

The encoder reads the input sequence and compresses it into a context vector, $\mathbf{z}$.

The decoder is initialized with this context vector, $\mathbf{h}_0^{dec} = \mathbf{z}$, and reconstructs the sequence step by step:

At each decoder step, the model uses its previous hidden state and, depending on the implementation, either:

- a zero/constant input;
- the context vector $\mathbf{z}$;
- the previous reconstructed value $\hat{\mathbf{x}}_{t-1}$;
- during training, the previous true value $\mathbf{x}_{t-1}$.

Autoencoders

Variants

Denoising Autoencoder

A denoising autoencoder is trained to reconstruct a clean input from a corrupted version of it.

$$\tilde{\mathbf{x}} = \text{corrupt}(\mathbf{x})$$

$$\tilde{\mathbf{x}} \to \mathbf{z} \to \hat{\mathbf{x}}$$

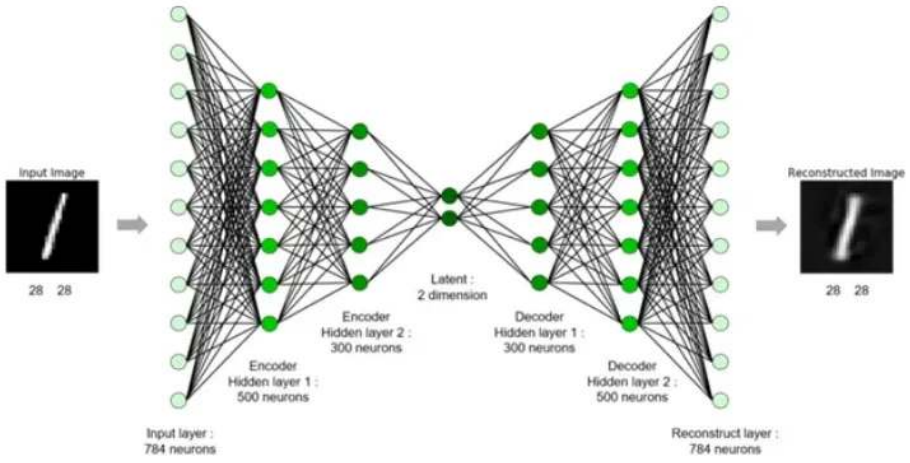
The loss is computed against the original clean input:

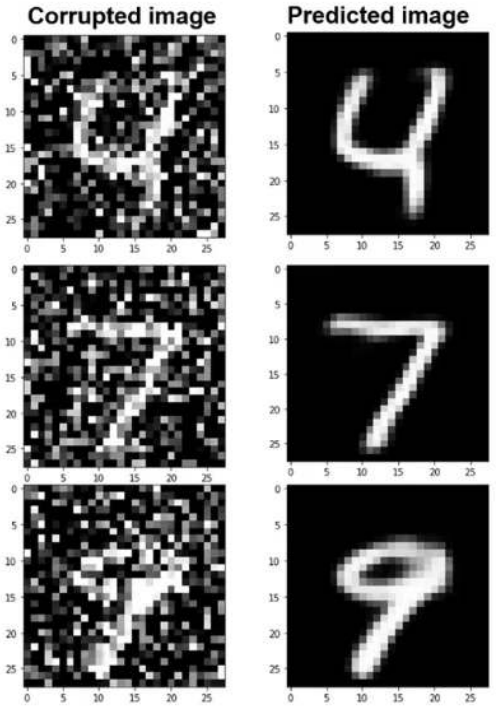
$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \left\| \mathbf{x} - \hat{\mathbf{x}} \right\|_2^2$$

The model learns representations that are robust to noise.

Denoising Autoencoder

The input is intentionally corrupted, e.g.: add Gaussian noise, remove pixels, perturb time points. Then the model learns to recover the original signal.





Why Denoising Helps

Denoising is a way to learn better representations. During training, the model sees a corrupted input $\tilde{\mathbf{x}}$, but must reconstruct the clean input, $\mathbf{x}$.

So it cannot simply copy the input. It must learn stable patterns in the data.

Examples:

- remove noise from medical images
- reconstruct corrupted ECG signals
- impute missing omics values
- learn robust patient representations

Sparse Autoencoder

A sparse autoencoder forces most latent units to be inactive.

The latent representation can be larger than the input, but only a few dimensions should be active for each sample.

$$\mathbf{z} = f_\theta(\mathbf{x})$$

with many components close to zero:

$$z_j \approx 0 \quad \text{for most } j$$

The goal is to learn a representation where each sample is described by a small number of active features.

Sparse Autoencoder Objective

A sparse autoencoder adds a sparsity penalty to the reconstruction loss.

$$\mathcal{L}_{total} = \mathcal{L}_{reconstruction} + \lambda \mathcal{L}_{sparsity}$$

A common simple penalty is the L_1 norm of the latent representation:

$$\mathcal{L}_{sparsity} = \|\mathbf{z}\|_1 = \sum_j |z_j|$$

The parameter λ controls how strongly we penalize dense representations.

Why Sparsity Helps

Sparsity encourages the model to use only a few latent dimensions at a time.

This can make representations:

- more compact
- less noisy
- more interpretable
- less likely to simply memorize the input

Undercomplete vs Sparse Autoencoders

Undercomplete Autoencoder

The latent space is smaller than the input:

$$d < M$$

Compression is forced by the bottleneck.

Sparse Autoencoder

The latent space may be large:

$$d \geq M$$

Compression is forced by the sparsity penalty.

Both approaches prevent the model from learning a trivial identity function.

Autoencoders

Downstream Tasks

Using Autoencoders for Downstream Tasks

After training an autoencoder, we often discard the decoder and keep only the encoder: $\mathbf{x} \rightarrow \mathbf{z}$

The latent representation $\mathbf{z}$ can be used as input for another model. For example: $\mathbf{x} \rightarrow \mathbf{z} \rightarrow \hat{y}$

This is useful when the original input is high-dimensional, noisy, or sparse.

Examples:

- classify patients using latent EHR representations
- classify cells using latent omics representations
- cluster patients using latent phenotypes
- visualize high-dimensional biological data

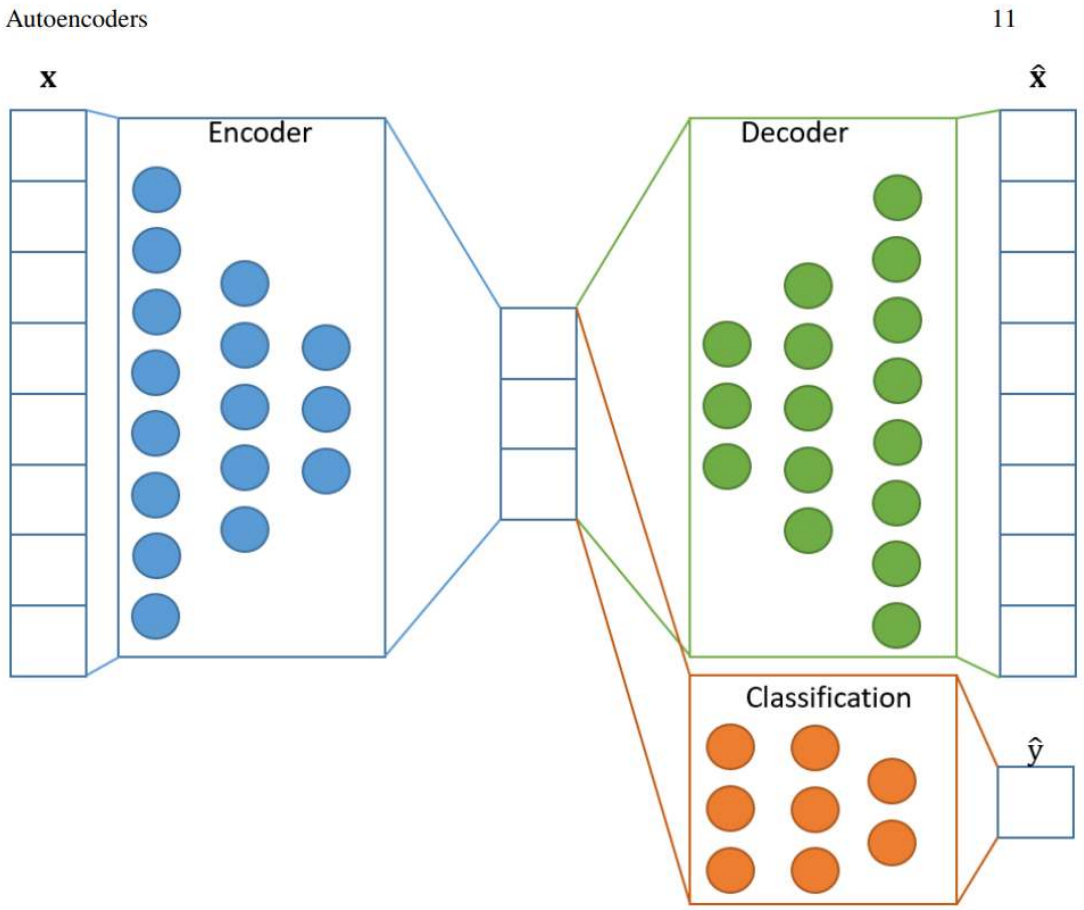
Autoencoder + Classifier

A common strategy is:

1. Train an autoencoder using only $\mathbf{x}$
2. Extract the latent representation $\mathbf{z}$
3. Train a classifier using $\mathbf{z}$ as input

$$\mathbf{x} \xrightarrow{\text{encoder}} \mathbf{z} \xrightarrow{\text{classifier}} \hat{y}$$

- The classifier does not use the original input directly.
- It uses the compressed representation learned by the autoencoder.



Autoencoders for Anomaly Detection

Autoencoders can also be used for anomaly detection.

Train the autoencoder mostly on normal examples:

$$\mathbf{x}_{normal} \rightarrow \mathbf{z} \rightarrow \hat{\mathbf{x}}_{normal}$$

At inference time, compute the reconstruction error:

$$e(\mathbf{x}) = \left\| \mathbf{x} - \hat{\mathbf{x}} \right\|_2^2$$

If the reconstruction error is high, the sample may be anomalous.

$$e(\mathbf{x}) > \tau \quad \Rightarrow \quad \text{anomaly}$$

Why Reconstruction Error Detects Anomalies

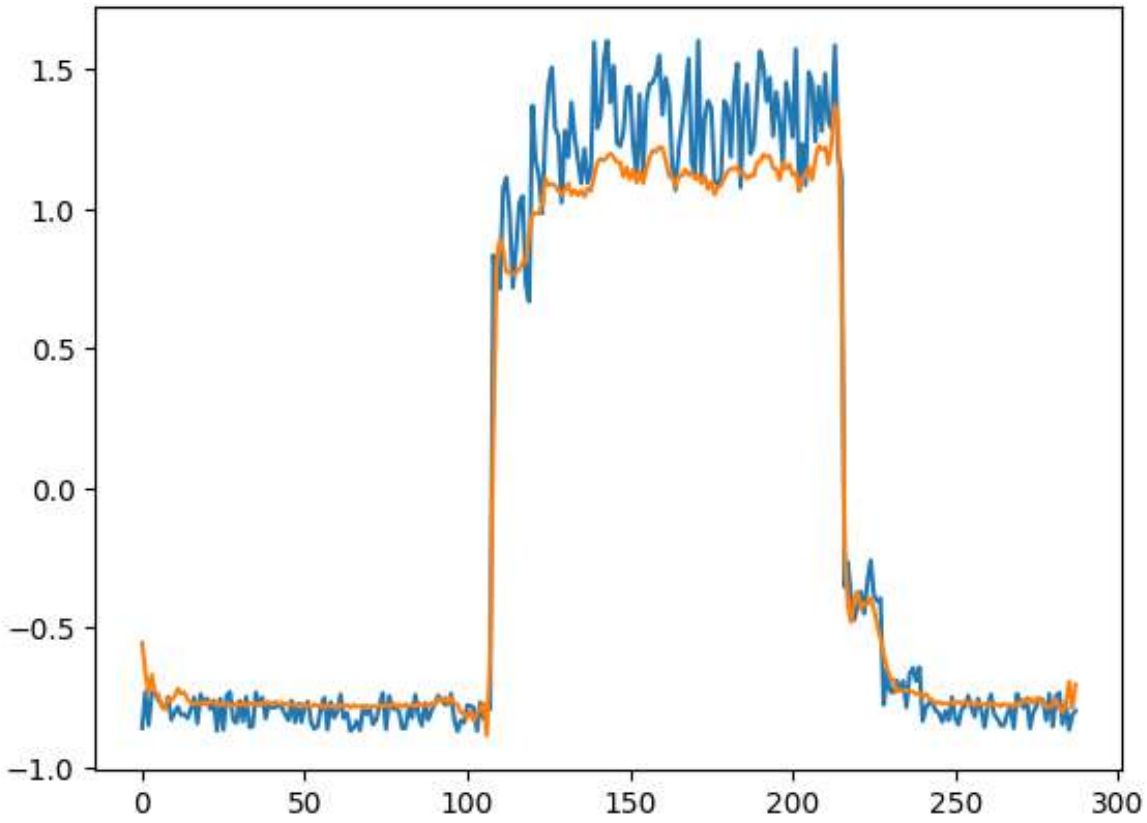
The autoencoder learns to reconstruct patterns seen during training.

If it is trained mostly on normal data:

- normal samples are reconstructed well
- unusual samples are reconstructed poorly

Examples:

- abnormal ECG segments
- rare patient trajectories
- imaging artifacts
- unusual omics profiles
- potential data-quality problems

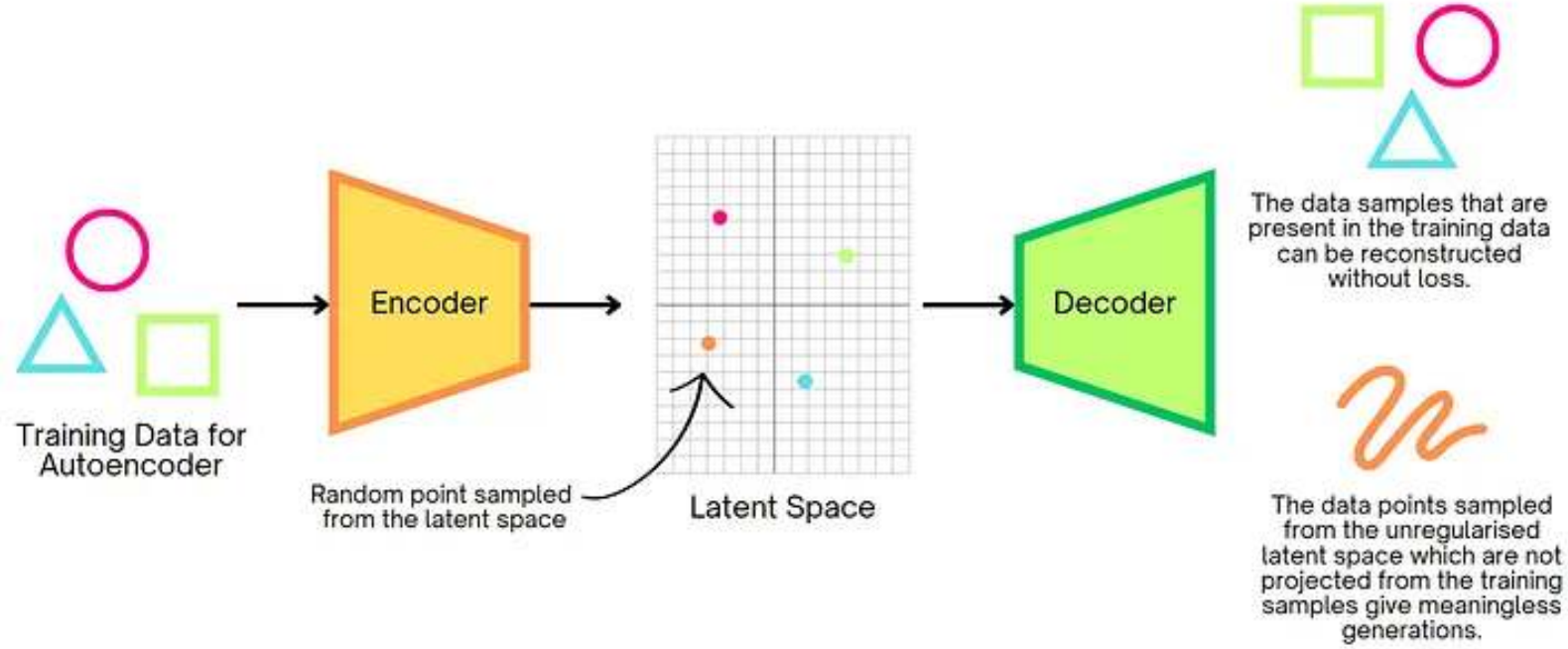


Autoencoders

Variational Autoencoders

From Autoencoders to Generative Models

A basic autoencoder learns: $\mathbf{x} \rightarrow \mathbf{z} \rightarrow \hat{\mathbf{x}}$, but the latent space may be irregular.



This means that sampling a random point $\mathbf{z}$ may not produce a meaningful output: to generate new samples, we need a structured latent space.

Variational Autoencoder

A Variational Autoencoder (VAE) does not encode an input into a single point, instead, it encodes the input into a probability distribution over latent variables.

Usually the encoder outputs $\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\sigma}(\mathbf{x})$ which define a Gaussian distribution:

$$q_\theta(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})$$

VAE: Encoder and Decoder

The encoder produces a distribution:

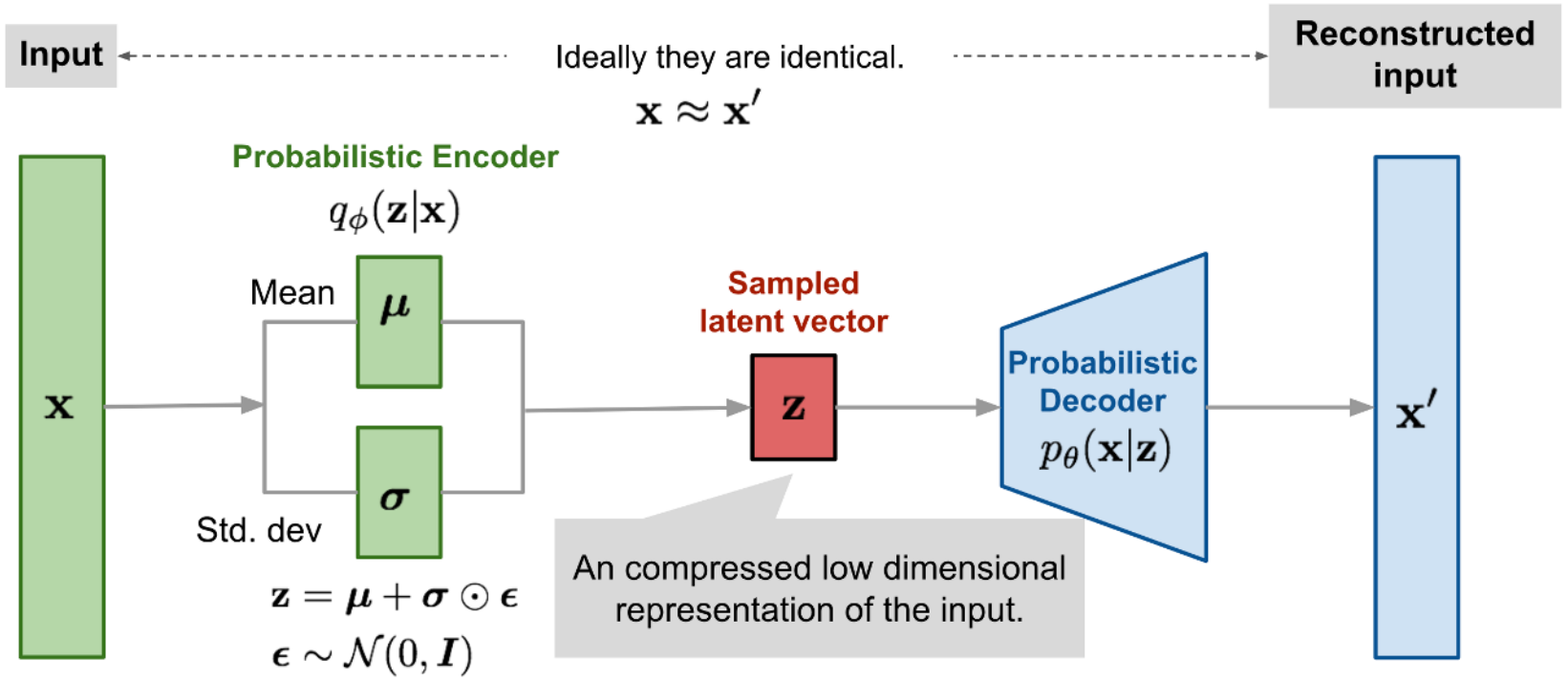
$$\mathbf{x} \rightarrow \boldsymbol{\mu}, \boldsymbol{\sigma}$$

Then we sample a latent vector:

$$\mathbf{z} \sim q_\theta(\mathbf{z}|\mathbf{x})$$

The decoder reconstructs the input from the sampled latent vector:

$$\hat{\mathbf{x}} = g_\phi(\mathbf{z})$$



Reparameterization Trick

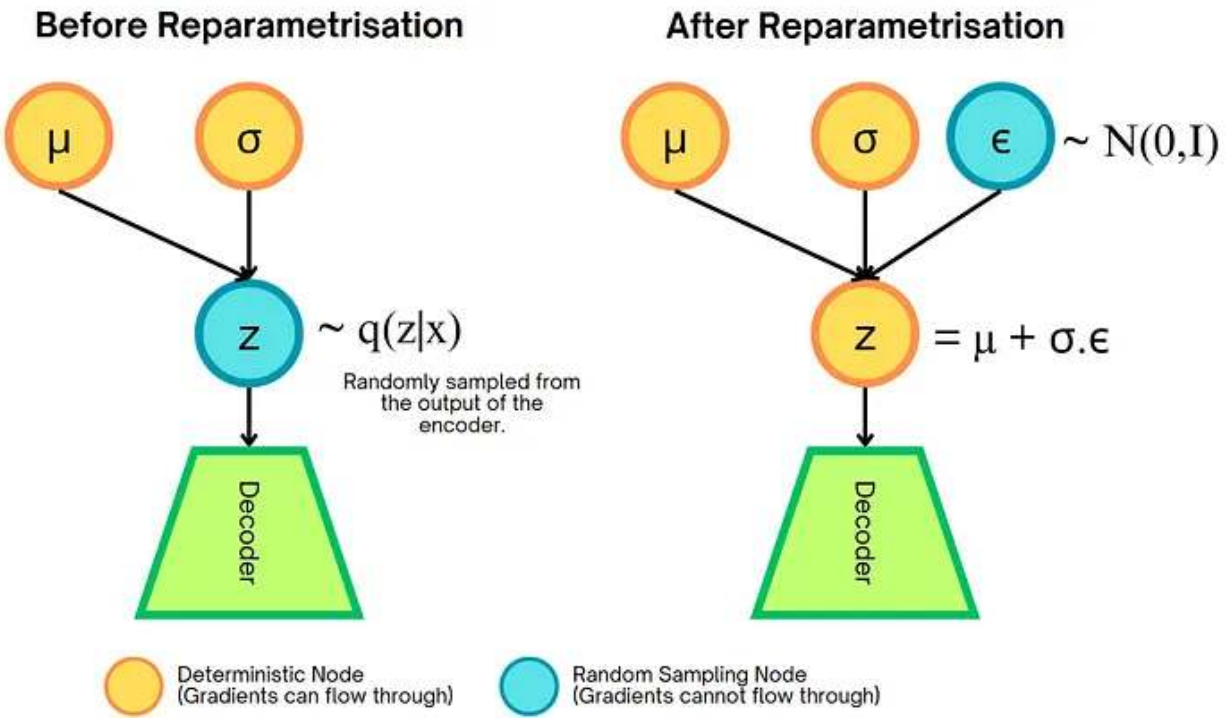
Sampling $\mathbf{z} \sim q_\theta(\mathbf{z}|\mathbf{x})$ is a stochastic process and therefore we cannot backpropagate the gradient.

The reparameterization trick writes the same sampling operation as:

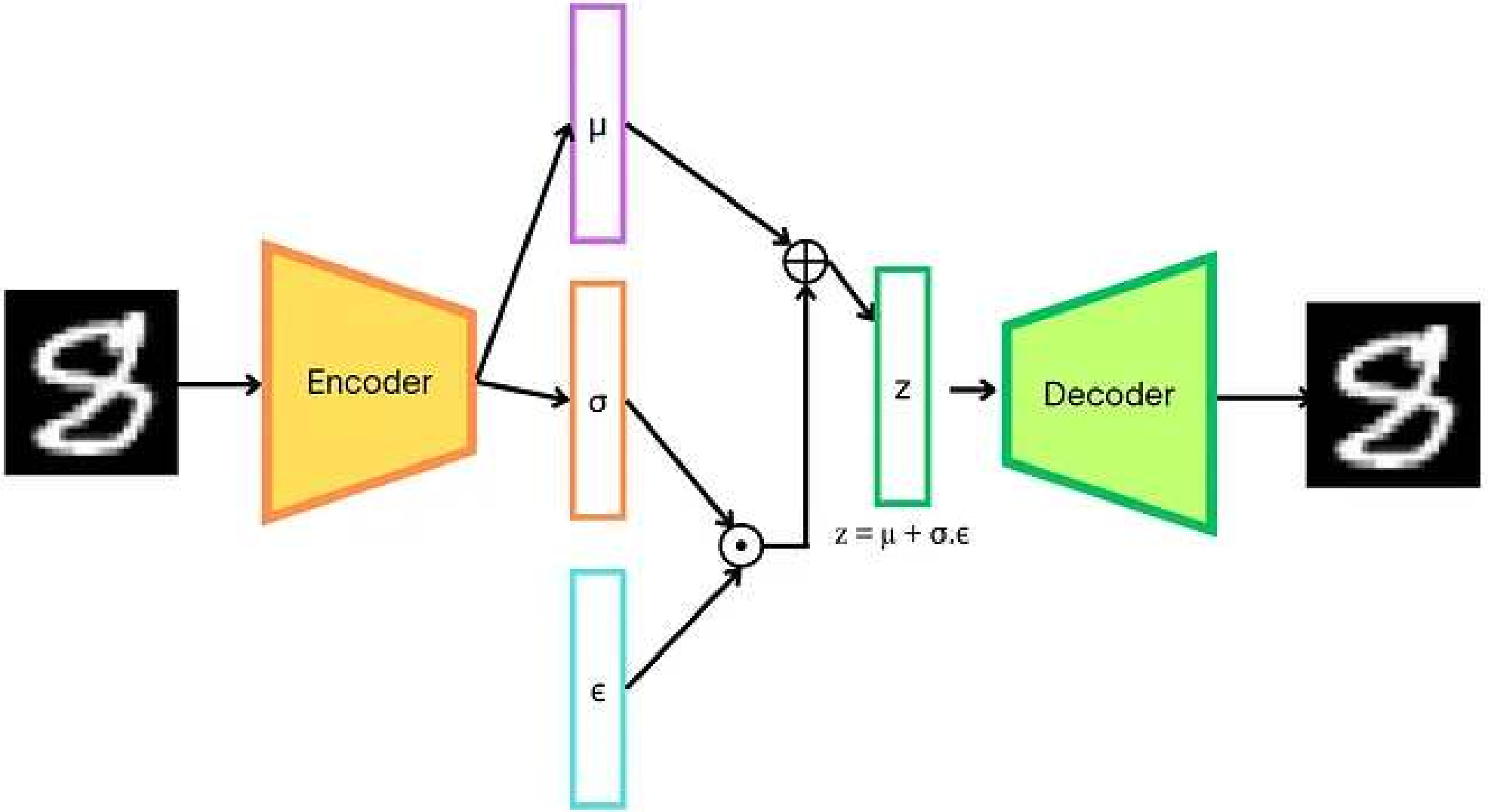
$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Now the random part is only $\boldsymbol{\epsilon}$, while $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ remain differentiable outputs of the encoder.

This allows backpropagation through the sampling step.



Reparametrization Trick



VAE Objective

$$\mathcal{L}_{VAE} = \mathcal{L}_{rec} + \mathcal{L}_{KL}$$

- Reconstruction term $\mathcal{L}_{rec}$ that makes the output similar to the input.
- Regularization term $\mathcal{L}_{KL}$ that makes the latent distribution close to a simple prior, usually $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

If the encoder distribution is $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})$ then the KL term is:

$$\mathcal{L}_{KL} = \frac{1}{2}\sum_{j=1}^{d}\left(\mu_j^2 + \sigma_j^2 - \log(\sigma_j^2) - 1\right)$$

- The term μ_j^2 penalizes latent means far from 0.
- The term $\sigma_j^2 - \log(\sigma_j^2) - 1$ penalizes variances far from 1.

Why the KL Term Matters

The KL term forces latent codes to occupy a regular, continuous space.

Without it:

- latent points may be scattered irregularly
- empty regions may decode to meaningless outputs
- random sampling may fail

With it:

- nearby latent points decode to similar outputs
- random samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ can generate new data
- interpolation in latent space becomes meaningful

Sampling from a VAE

After training, we can generate new samples without an input.

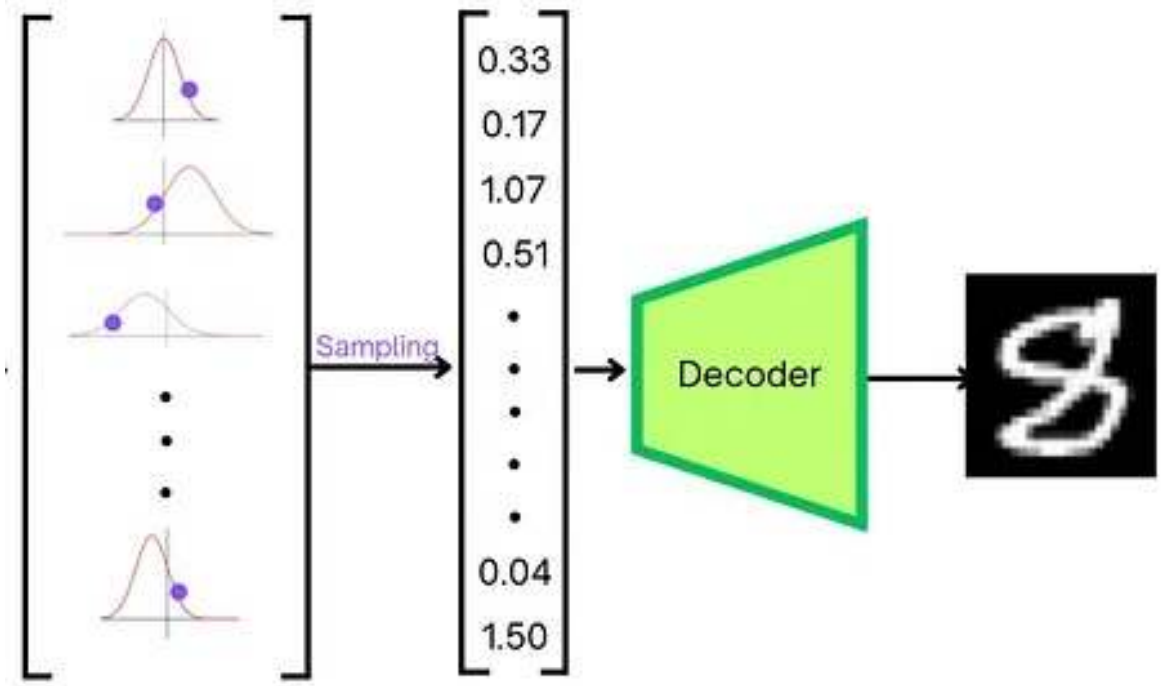
Sample a latent vector:

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

Decode it:

$$\tilde{\mathbf{x}} = g_\phi(\mathbf{z})$$

So the VAE is a generative model.



AEs vs VAEs

Basic Autoencoder

Main goal:

- reconstruction
- compression
- representation learning

Usually not a proper generative model.

Variational Autoencoder

Main goal:

- structured latent space
- reconstruction
- generation by sampling

A proper generative model.

Limitations

Autoencoders are useful, but they have limitations.

- Good reconstruction does not guarantee useful clinical representations
- Latent dimensions may be hard to interpret
- The model can learn shortcuts or irrelevant patterns
- Synthetic biomedical data must be validated carefully
- Reconstruction error may miss clinically important abnormalities

In healthcare, reconstruction quality is not the same as clinical validity.

Autoencoders

Are not the only generative models

Other Generative Models

VAEs are one family of generative models.

Other important families:

- **GANs**: train a generator against a discriminator
- **Diffusion models**: generate data by learning to denoise step by step
- **Autoregressive models**: generate one element at a time

Generative Adversarial Networks

A GAN is composed of two neural networks trained against each other.

Generator

Takes random noise and generates synthetic data:

$$\mathbf{z} \to G(\mathbf{z}) = \tilde{\mathbf{x}}$$

Discriminator

Tries to distinguish real data from generated data:

$$D(\mathbf{x}) \to \text{real}/\text{fake}$$

The generator tries to fool the discriminator.
The discriminator tries not to be fooled.

GAN Training

GAN training is an adversarial game.

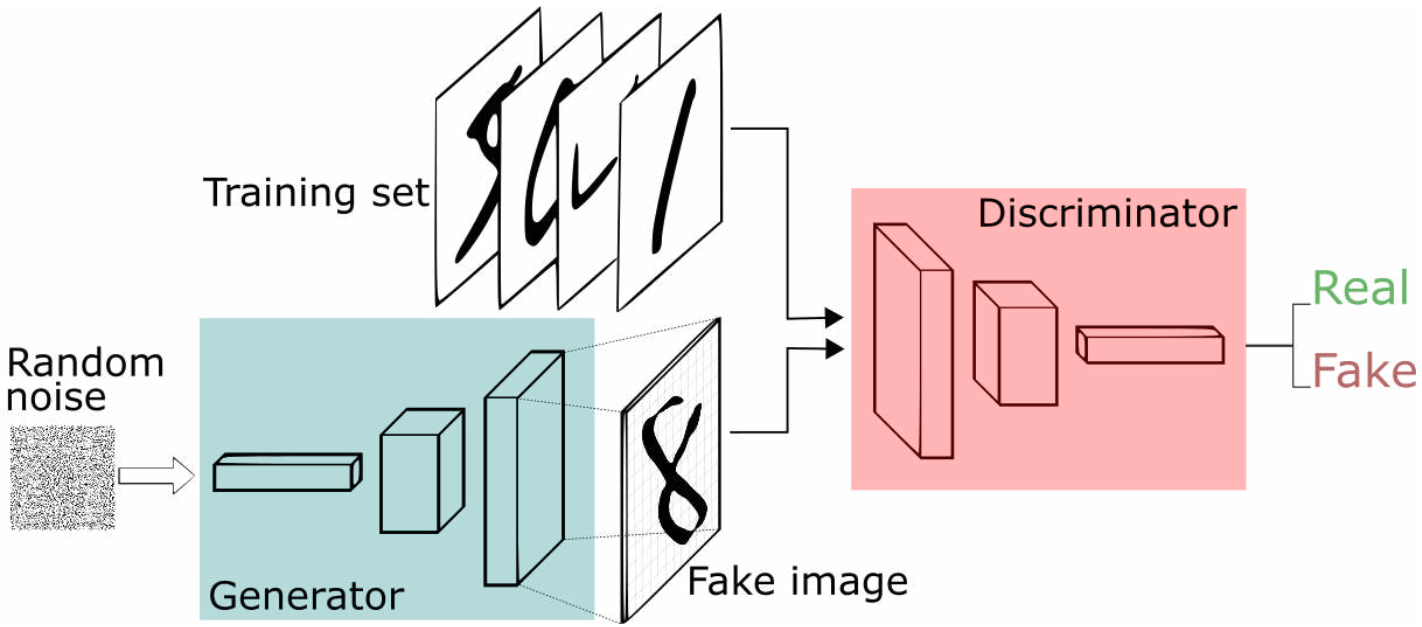
$$\mathbf{z} \sim p(\mathbf{z}) \rightarrow G(\mathbf{z}) \rightarrow D(G(\mathbf{z}))$$

The discriminator learns to classify:

$$\mathbf{x} \rightarrow \text{real}$$

$$G(\mathbf{z}) \rightarrow \text{fake}$$

The generator learns to produce samples that the discriminator classifies as real.



GANs can generate sharp samples, but training can be unstable.

GANs in Biomedical Data

GANs have been used for:

- medical image synthesis
- image-to-image translation
- data augmentation
- synthetic patient data

However:

- generated samples may be unrealistic
- training can suffer from mode collapse
- evaluation is difficult
- clinical validity must be checked carefully

In healthcare, a realistic-looking image is not automatically clinically valid.

Diffusion Models

Diffusion models generate data by learning to denoise.

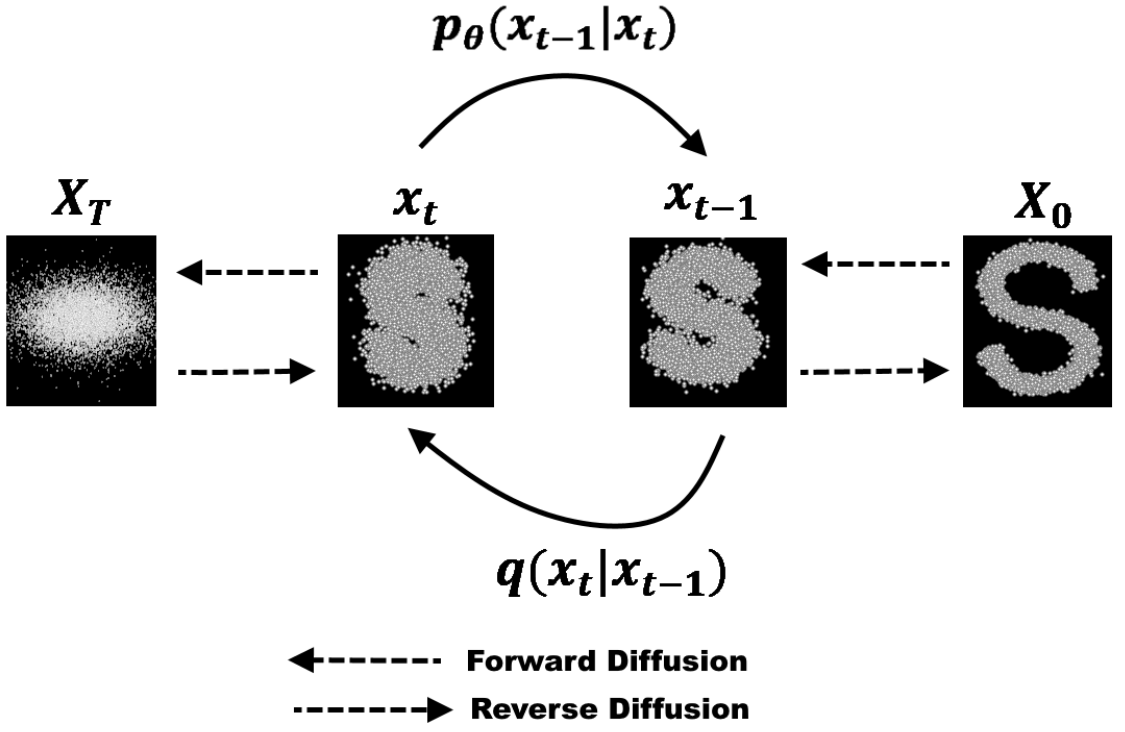
Training idea:

1. Start with a real sample $\mathbf{x}_0$
2. Gradually add noise
3. Train a neural network to reverse the process

$$\mathbf{x}_0 \rightarrow \mathbf{x}_1 \rightarrow \cdots \rightarrow \mathbf{x}_T$$

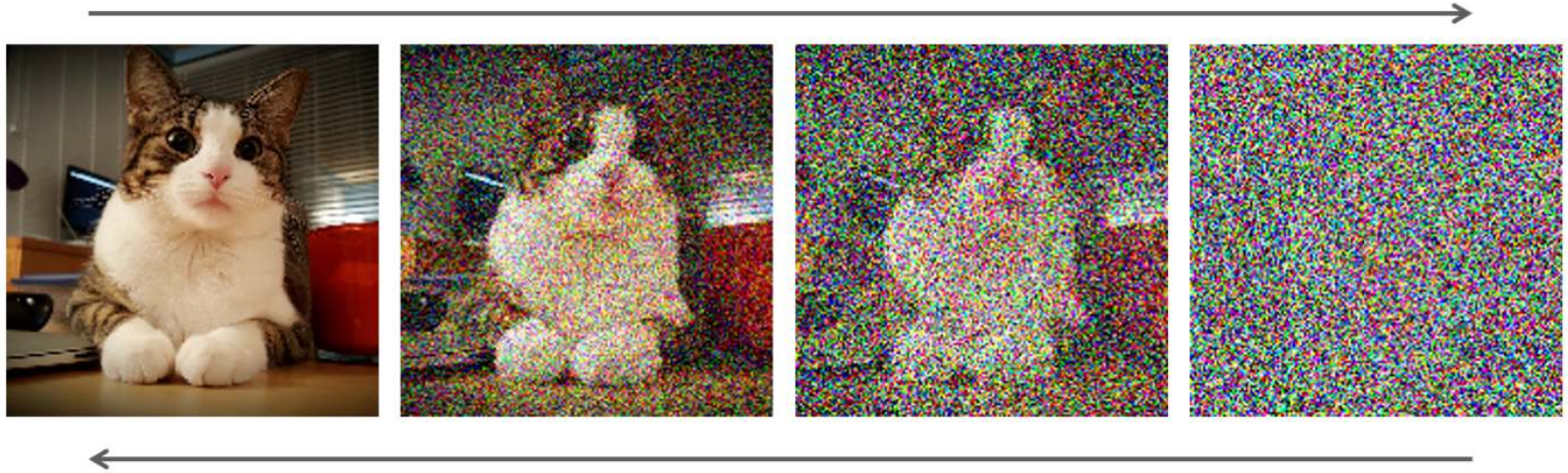
At generation time, we start from noise and denoise step by step:

$$\mathbf{x}_T \rightarrow \mathbf{x}_{T-1} \rightarrow \cdots \rightarrow \mathbf{x}_0$$



Diffusion Models: Intuition

The model learns a sequence of small denoising steps.



Diffusion models are the basis of many modern image generation systems.

Generative Models: Comparison

Model	Main idea	Strength	Limitation
VAE	Learn structured latent space	Easy sampling, interpretable latent space	Samples may be blurry
GAN	Generator vs discriminator	Sharp samples	Unstable training
Diffusion	Learn to denoise noise	High-quality samples	Slow generation

Autoencoders

Biomedical Applications

Biomedical Applications

Autoencoders and VAEs can be used for:

- dimensionality reduction of omics data
- denoising medical images
- reconstruction of corrupted signals
- anomaly detection in ECG or imaging
- patient phenotyping from EHR data
- synthetic data generation
- latent-space visualization

The important point is not only reconstruction.

The useful object is often the learned representation $\mathbf{z}$.

Molecule Generation with VAE

A VAE can be used to map molecules from a discrete representation to a continuous latent space. The goal is to generate molecules that are both realistic and novel.

$$\text{SMILES string} \rightarrow \mathbf{z} \rightarrow \text{generated SMILES string}$$

Data

- QM9: small molecules with up to 9 heavy atoms
- ZINC: drug-like commercially available molecules

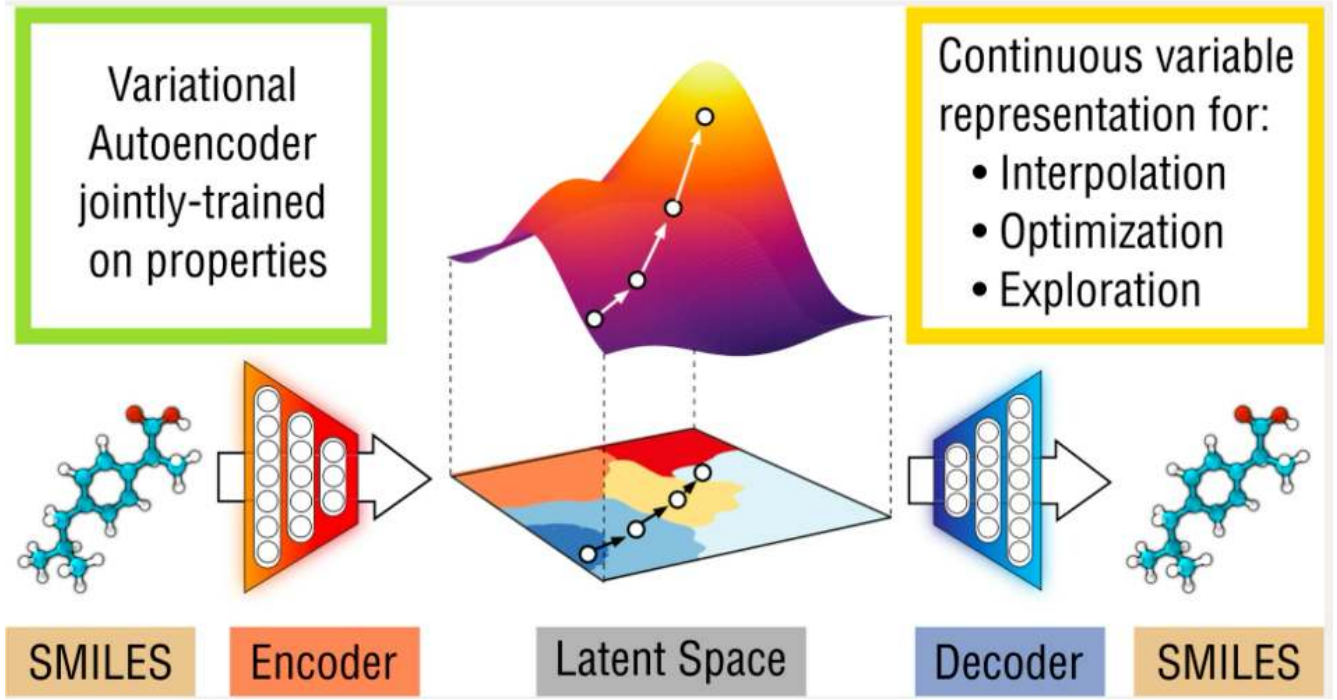
Model

- Input: SMILES strings
- Encoder: 1D CNN
- Latent space: continuous vector $\mathbf{z}$
- Decoder: GRU layers
- Extra network: predicts chemical properties

Simplified Molecular Input Line Entry System (SMILES) strings are a text format for representing molecules.

Molecule Generation with VAE

A VAE can be used to map molecules from a discrete representation to a continuous latent space. The goal is to generate molecules that are both realistic and novel.



Simplified Molecular Input Line Entry System (SMILES) strings are a text format for representing molecules.

References

- Introduction to Deep Learning for Healthcare, Chapter 8
- Introduction to Deep Learning for Healthcare, Chapter 12
- https://lilianweng.github.io/posts/2018-08-12-vae/
- https://medium.com/@raajanwankhade/variational-autoencoders-vaes-explained-and-implemented-11e2931d05f3