Deep Learning and Neural Networks

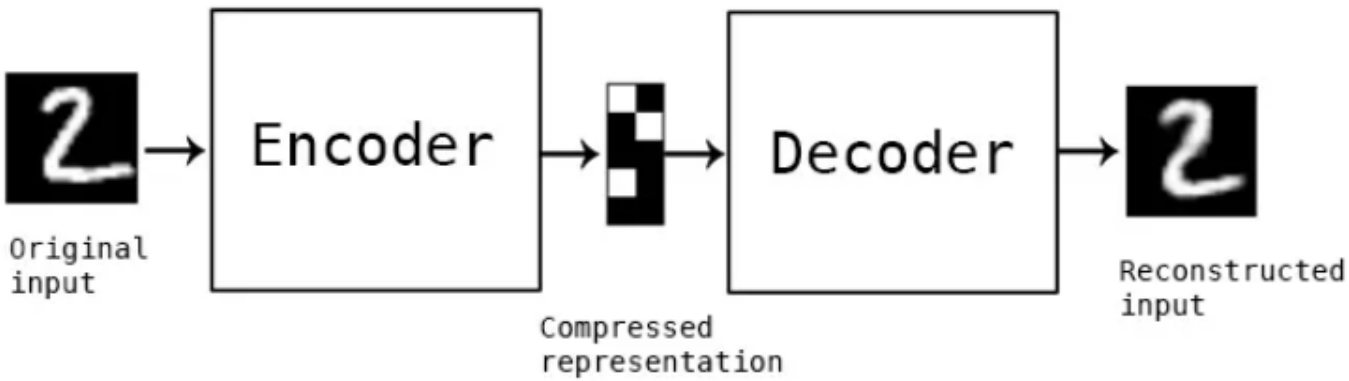
Demetrio Labate

April 3, 2024

Part 5
Autoencoders

Autoencoders

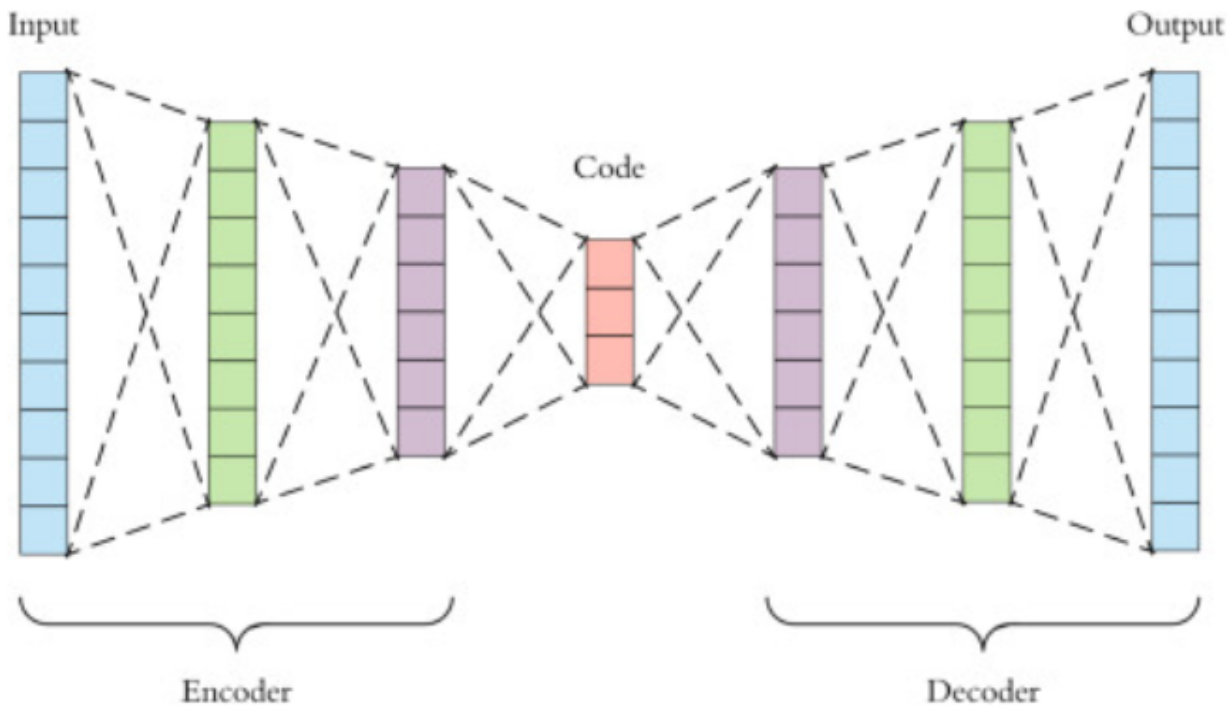
An **autoencoder** is a type of neural network architecture designed to efficiently compress (encode) input data down to its essential features, then reconstruct (decode) the original input from this compressed representation.



Autoencoders

An autoencoder typically consists of three blocks

- **Encoder layer** to compress the input data into a compressed representation.
- **Bottleneck layer** or **code** to represent the compressed input.
- **Decoder layer** to reconstruct the encoded image back to the original dimension.



Autoencoders vs Encoder-decoders

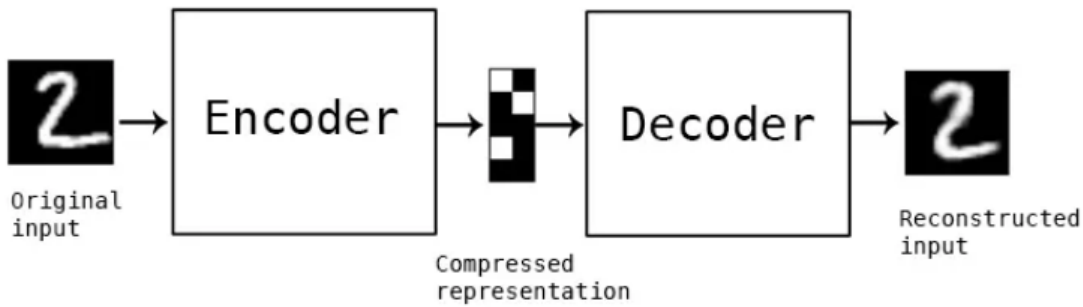
Though all autoencoders include both an encoder and a decoder, **not all encoder-decoder models are autoencoders**.

Encoder-decoder frameworks are used in a variety of deep learning models to extracts/encode features of the input data and take the extracted feature data to the decoder for tasks such as classification or segmentation. In applications of such encoder-decoder models, the output of the neural network is **different** from its input.

For example, in segmentation models like U-Net, the encoder extracts features from the input to determine pixel classification; using the feature map and those pixel-wise classifications, the decoder then constructs segmentation masks for each object in the image. The goal of these encoder-decoder models is to label pixels by their semantic class. they are trained via **supervised learning**, optimizing the model's predictions against ground truth images.

Autoencoders

Autoencoders form a very specific subset of encoder-decoder architectures that are trained via **unsupervised learning** to **reconstruct their own input data**.
Input and output dimensions are the same.



Using **unsupervised machine learning**, autoencoders are trained to discover the **latent variables** of the input data.

The latent variables are not directly observable but they fundamentally inform the way data is distributed. Collectively, the latent variables of a given set of input data are referred to as a **latent space**.

Autoencoders - History

- Early versions of autoencoders were introduced as a method for unsupervised pre-training (Ballard, 1986) and to address the problem of "backpropagation without a teacher" (Rumelhart, Hinton, and Williams, 1986).
- The first formal notion of the autoencoder idea was first proposed by Kramer (1991) as a nonlinear generalization of principal components analysis (PCA).
- Popularized by paper *Autoencoders, Minimum Description Length and Helmholtz Free Energy*, Hinton and Zemel, 1993.
- Most traditional applications: dimensionality reduction, feature learning, learning generative models of data.
- More recently, autoencoders have taken center stage in the deep architecture. Some powerful AIs introduced in the 2010s involve autoencoders, in the form of Restricted Boltzmann Machines, stacked and trained bottom up in unsupervised fashion, followed by a supervised learning phase to train the top layer and fine-tune the entire architecture.

Autoencoders

Autoencoders are flexible neural networks that can be customized for various tasks. They come in different forms, each with unique strengths and limitations.

- **Vanilla Autoencoders:** Basic autoencoders that efficiently encode and decode data.
- **Denoising Autoencoders:** Improved robustness to noise and irrelevant information.
- **Sparse Autoencoders:** Learn more compact and efficient data representations.
- **Contractive Autoencoders:** Generate representations less sensitive to minor data variations.
- **Variational Autoencoders:** Generate new data points that resemble in some form the training data.

The choice of autoencoder depends on the specific task..

Autoencoders

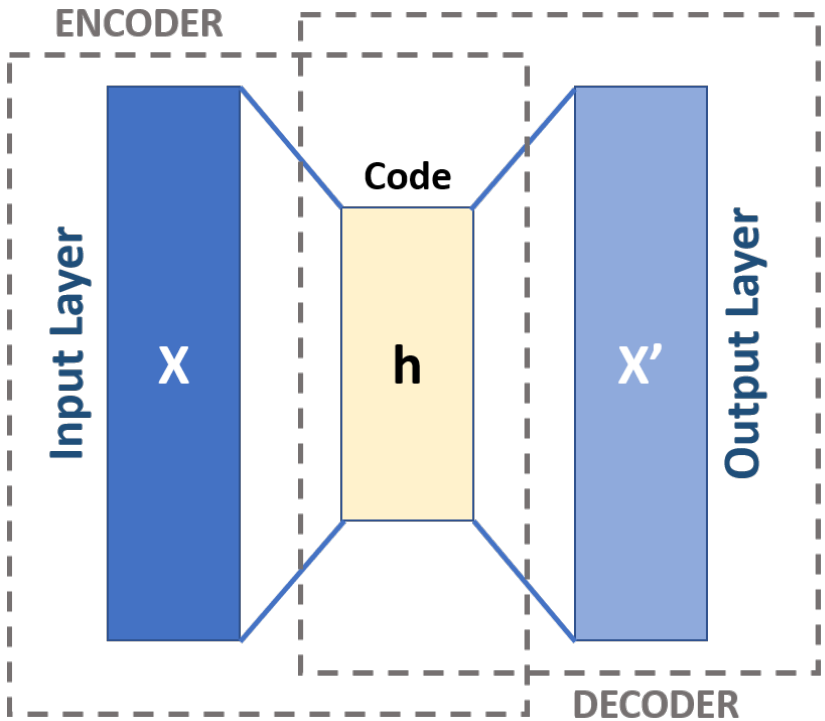
Auto-Encoders are designed to learn a lower-dimensional representation for a higher-dimensional data.

Applications include

1. Dimensionality Reduction
2. Feature Extraction
3. Image Denoising
4. Image Compression
5. Image Search
6. Anomaly Detection
7. Missing Value Imputation

Autoencoders - Architecture

An autoencoder has a structure very similar to a feedforward neural network, the primary difference being that the **number of neurons in the output layer are equal to the number of inputs**.



The autoencoder maps the space of decoded messages $\mathcal{X}$ to the space of encoded messages $\mathcal{Z}$.
In most cases, both $\mathcal{X}$ and $\mathcal{Z}$ are Euclidean spaces, that is, $\mathcal{X} = \mathbb{R}^m$, $\mathcal{Z} = \mathbb{R}^n$ for some m, n.

Autoencoders - Architecture

We can describe the autoencoder algorithm in two parts:

1. a parametrized family of encoder functions f_θ, parametrized by θ, that maps an input $X \in \mathcal{X}$ to a code $h \in \mathcal{Z}$, that is

$$f_\theta(X) = h$$

2. a parametrized family of decoder functions g_ϕ, parametrized by ϕ, that maps $h \in \mathcal{Z}$ to $X' \in \mathcal{X}$, that is

$$g_\phi(h) = X'$$

   The decoder is designed to produces a reconstruction X' of the input X.

Usually, both the encoder and the decoder functions are defined as MLPs. For example, a one-layer-MLP encoder is a function of the form

$$f_{W,b}(X) = \sigma(WX + b)$$

Autoencoders - Loss function

To learn the parameters of the encoder and decoder functions, the autoencoder seeks to minimize some **loss function**, such as mean squared error (MSE).

The loss function penalizes $X' = g_\phi(h) = g_\phi(f_\theta(X))$ for being dissimilar from X.

To define the loss in the continuous setting, let μ be a reference probability distribution on $\mathcal{X}$ and $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a distance function. Then we can define the loss function for the autoencoder as

$$L(\theta, \phi) := \mathbb{E}_{x \sim \mu_{ref}}[d(x, g_\phi(f_\theta(x)))]$$

The **optimal autoencoder** is defined by the optimization problem

$$\arg\min_{\theta,\phi} L(\theta, \phi)$$

Autoencoders - Loss function

In practical situations, the reference distribution is just the empirical distribution given by a dataset $\{X_1, \ldots, X_N\} \subset \mathcal{X}$, so that the reference probability distribution is the Dirac measure

$$\mu_{ref} = \frac{1}{N} \sum_{i=1}^{N} \delta_{x_i}$$

The distance function is typically chosen to be the square L^2 loss:

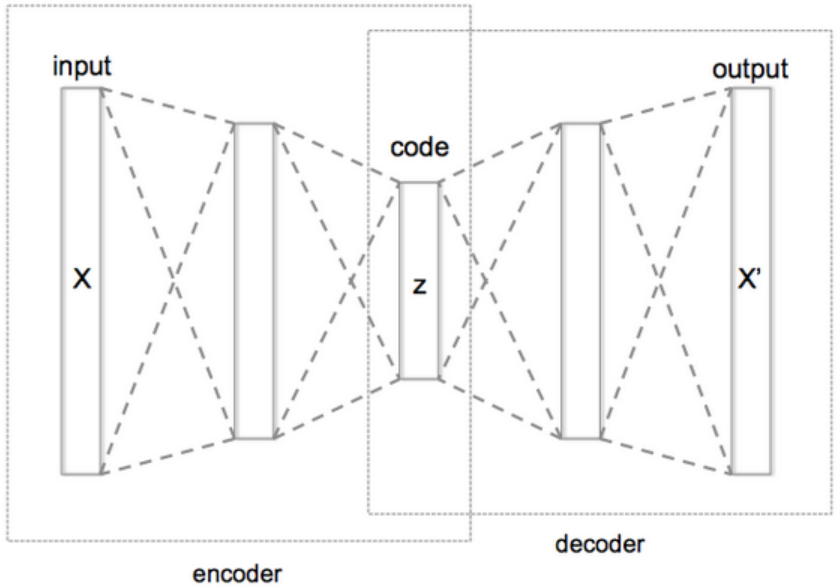
$$d(x, x') = \|x - x'\|_2^2$$

Hence, the problem of searching for the optimal autoencoder is the least-squares optimization problem:

$$\min_{\theta,\phi} L(\theta, \phi) \quad \text{where} \quad L(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} \|X_i - g_\phi(f_\theta(X_i)\|_2^2$$

Autoencoders - Loss function

Let us consider, as an example, an autoencoder where both the encoder and the decoder functions are defined as a one-layer-MLP.



Encoder: $f_\theta = f_{W,b} : \mathcal{X} \mapsto \mathcal{Z}$ given by

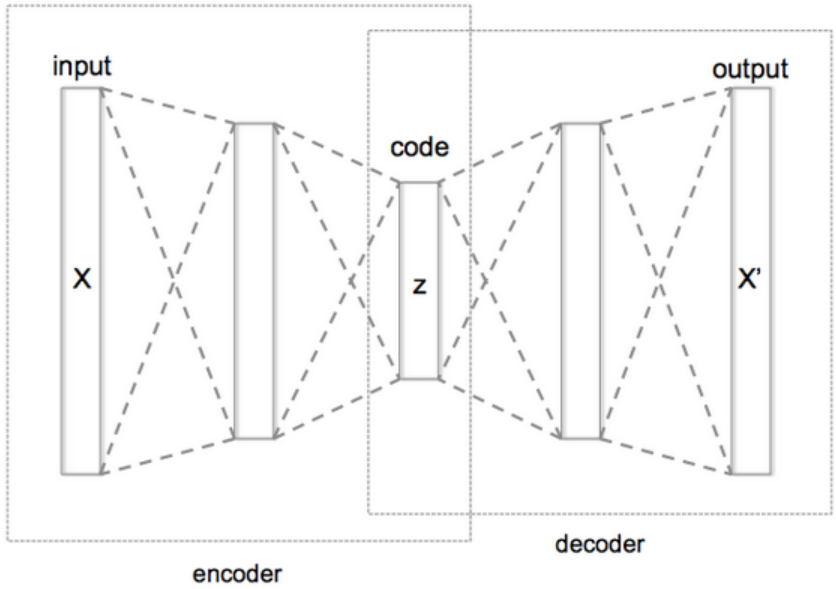
$$f_{W,b}(X) = \sigma(WX + b) = z$$

Decoder: $g_\phi = g_{W',b'} : \mathcal{Z} \mapsto \mathcal{X}$ given by

$$g_{W',b'}(z) = \sigma(W'z + b') = X'$$

Autoencoders - Loss function

Let us consider, as an example, an autoencoder where both the encoder and the decoder functions are defined as a one-layer-MLP.



To minimize the reconstruction error, we minimize the MSE loss over the training samples (X_i)

$$L(W, W', b, b') = \frac{1}{N} \sum_{i=1}^{N} \|X_i - \sigma\left(W'\left(\sigma(WX_i + b)\right) + b'\right)\|_2^2$$

Autoencoders - Learning

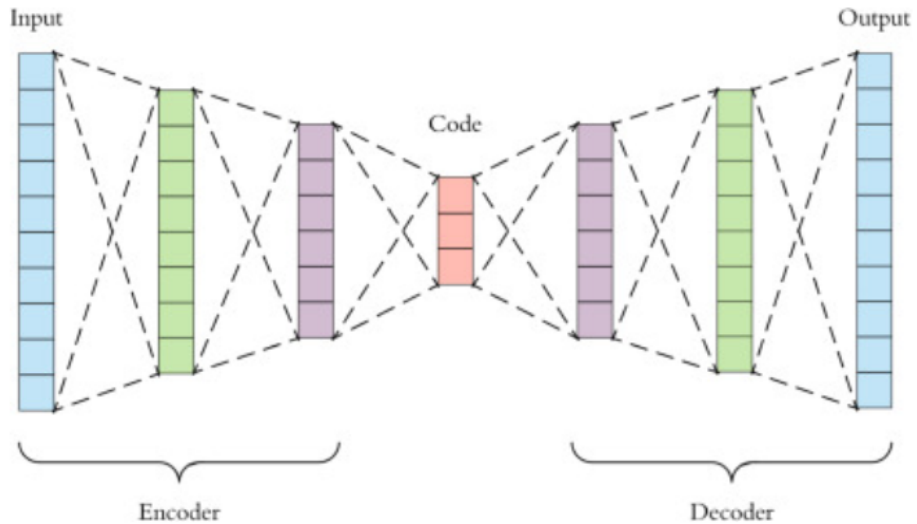
What does an autoencoder learn?

- Learning $g(f(x)) = x$ everywhere is not useful
- Autoencoders are designed to be unable to copy perfectly.
- Autoencoders learn useful properties of the data: can prioritize which aspects of input should be copied.
- Can learn stochastic mappings. they go beyond deterministic functions to mappings $p_{encoder}(x|h)$ and $p_{encoder}(h|x)$.

Autoencoders - Training

There are 4 hyperparameters that we need to set when training an autoencoder:

1. **Code size:** number of nodes in the middle layer. by choosing smaller size than the input dimension results in compression. Smaller code size, higher compression.
2. **Number of layers:** the autoencoder can be as deep as we like. In the figure below we have 2 layers in both the encoder and decoder, without considering the input and output.



Autoencoders - Training

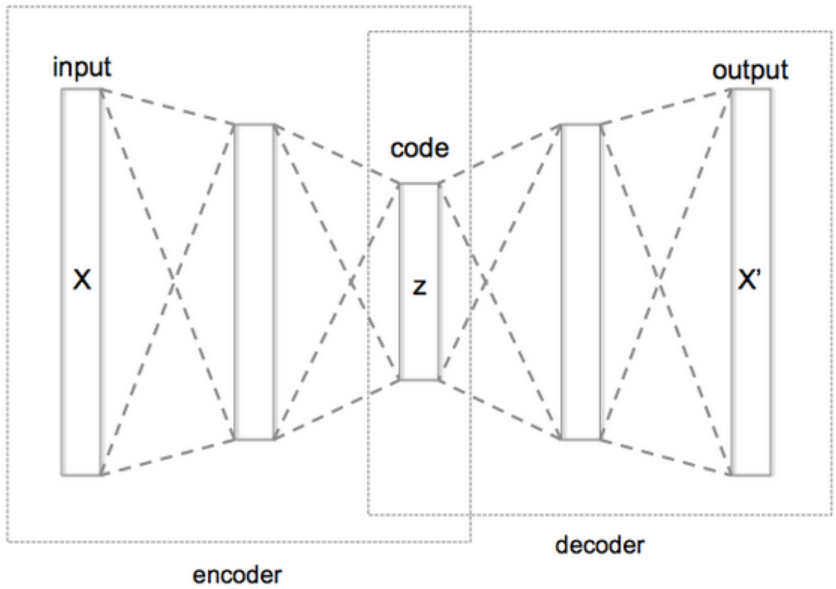
3. **Number of nodes per layer:** the standard autoencoder architecture we have shown in examples is called a **stacked autoencoder** since the layers are stacked one after another. Usually stacked autoencoders look like a "sandwitch", with the number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. Also the decoder is symmetric to the encoder in terms of layer structure. This is not necessary and we have total control over these parameters.
4. **Loss function:** Usual choices are the mean squared error (MSE) or binary crossentropy. If the input values are in the range [0, 1] then we typically use crossentropy, otherwise we use the mean squared error.

Autoencoders are typically trained via backpropagation combined with minibatch gradient descent.

Autoencoders - Interpretation

An autoencoder is optimized to perform as close to perfect reconstruction as possible.

In many applications, the goal is to create a reduced set of codings that adequately represents the inputs $X \in \mathcal{X}$. Consequently, we constrain the hidden layers so that the number of neurons is less than the number of inputs.



An autoencoder whose internal representation has a smaller dimensionality than the input data is an **undercomplete autoencoder**.

Autoencoders - Interpretation

In an undercomplete autoencoder, the compression of the hidden layers forces the autoencoder to capture the most dominant features of the input data and the representation of these signals are captured in the codings.

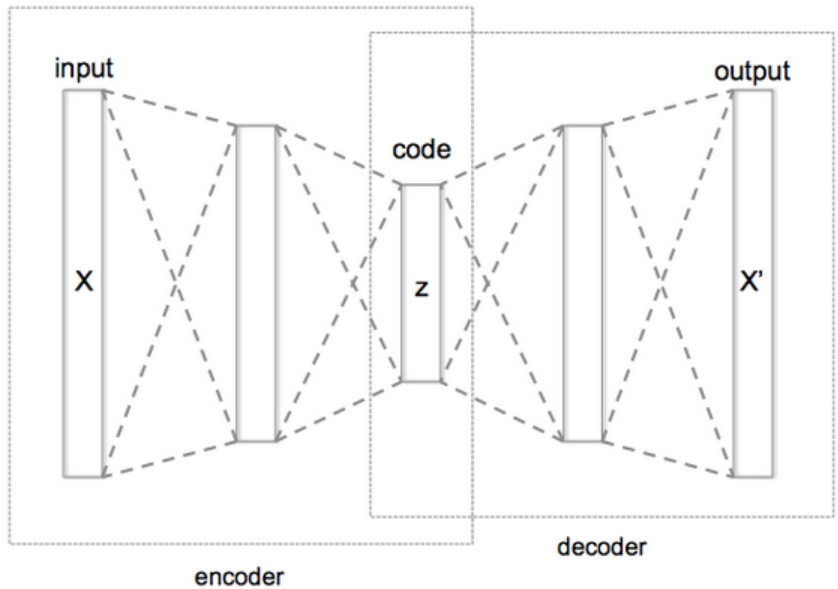
Remark: When the autoencoder uses only linear activation functions and the loss function is MSE, then **the autoencoder learns to span the same subspace as Principal Component Analysis (PCA)**.

When nonlinear activation functions are used, autoencoders provide nonlinear generalizations of PCA

The reduced codings we extract using an undercomplete autoencoder are sometimes referred to as **deep features (DF)** and they are similar in nature to the principal components for PCA.

Autoencoders - Interpretation

Example. The following example demonstrates an implementation of a basic undercomplete autoencoder with three fully connected hidden layers that we apply to find a reduced representation of the MNIST dataset.



We use a single hidden layer with only **two codings**. This is reducing 784 features down to two dimensions; although not very realistic, it allows us to visualize the results and gain some intuition on the algorithm

Autoencoders - Interpretation

We project the MNIST response variable onto the reduced feature space and compare our autoencoder to PCA.

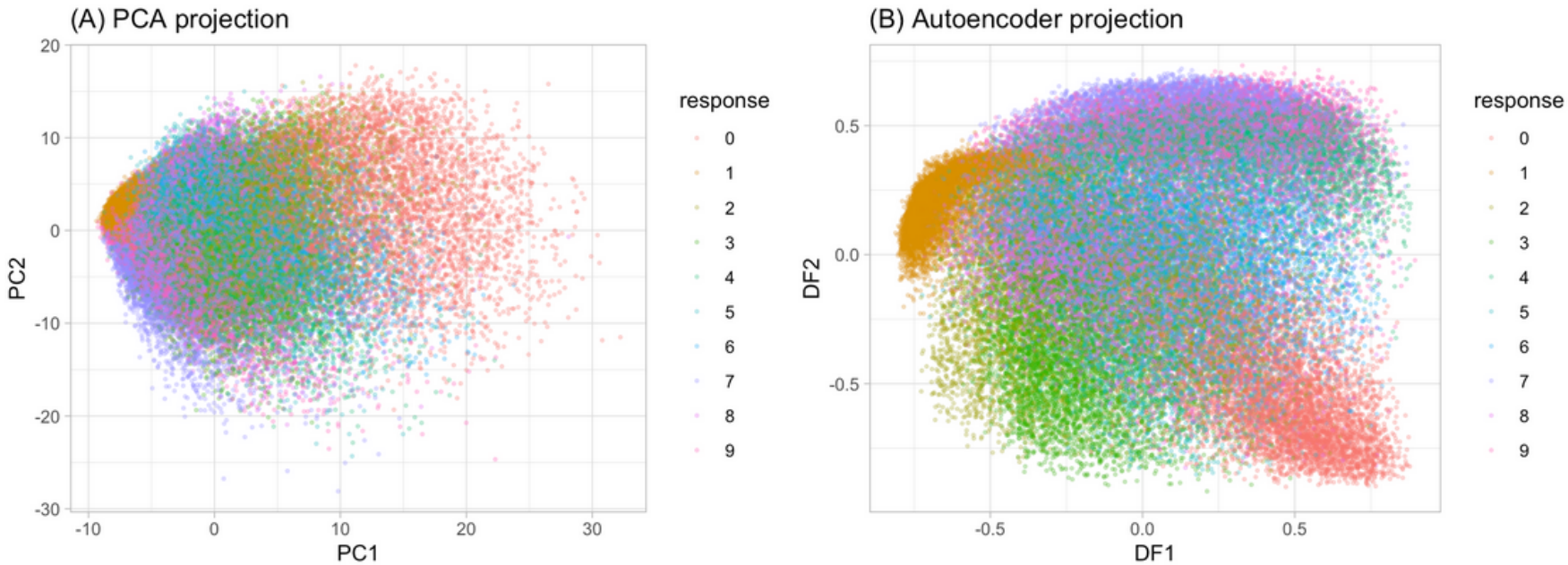


Figure shows that the nonlinear dimensionality reduction of the autoencoder can help to isolate the signals in the features better than PCA.

Undercomplete Autoencoders

The goal of the bottleneck is to prevent the autoencoder from **overfitting** to its training data.

Without sufficiently limiting the capacity of the bottleneck, the network tends toward **learning the identity function** between the input and output: in other words, it may learn to minimize reconstruction loss by simply copying the input directly. By forcing the data to be significantly compressed, the neural network must learn to retain only the features most essential to reconstruction.

If the encoder and decoder have a high enough capacity—that is, if they are processing large or complex data inputs, then the autoencoder (even with a bottleneck) may still learn the identity function anyway, making it useless.
This makes undercomplete autoencoders inflexible and limits their capacity

Autoencoder failings

Autoencoders may fail to learn anything useful in the following cases.

1. Hidden code h has dimension equal to input X.
2. Even in the case of an undercomplete autoencoder, the capacity of encoder/decoder is too high.
   - Capacity controlled by depth
3. Overcomplete case: hidden code h has dimension greater than input X.
   - Notice that even in the overcomplete case the autoencoder may learn useful features from the data.

Regularized Autoencoders

Regularized autoencoders address the shortcomings of undercomplete autoencoders by introducing **regularization**.

Various techniques exist to prevent autoencoders from learning the identity function, to reduce overfitting and to improve their ability to learn useful features or functions.

- **Denoising Autoencoders**
- **Sparse Autoencoders**
- **Contractive Autoencoders**

Denoising Autoencoders

The **Denoising AutoEncoder (DAE)** is a stochastic version of the autoencoder in which we train the autoencoder to reconstruct the input from a corrupted copy of the inputs.

This forces the codings to learn more robust features of the inputs and prevents them from merely learning the identity function; even if the number of codings is greater than the number of inputs.

We can think of a denoising autoencoder as having two objectives:

1. try to encode the inputs to preserve the essential signals;
2. try to undo the effects of a corruption process stochastically applied to the inputs of the autoencoder.

The latter can only be done by capturing the statistical dependencies between the inputs. Combined, this denoising procedure allows us to implicitly learn useful properties of the inputs

Denoising Autoencoders

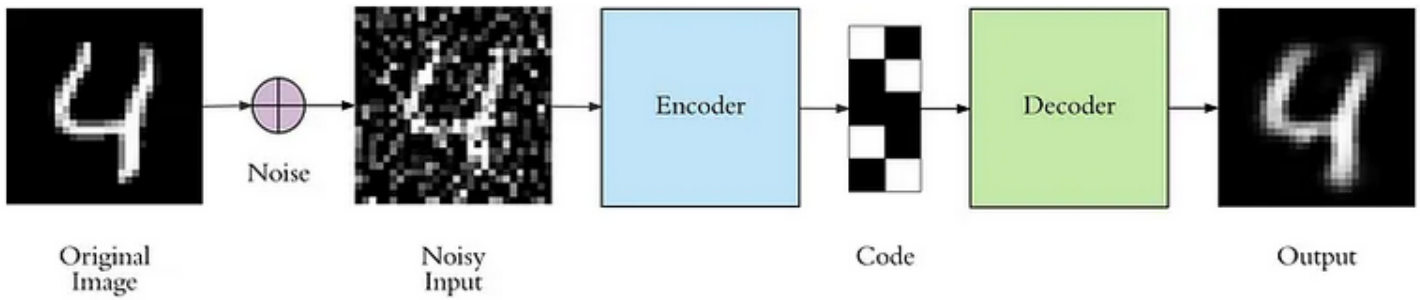
The corruption process typically follows one of the following approaches:

- additive Gaussian noise;
- masking noise: a fraction of the input is randomly chosen and set to 0; this can be done by manually imputing zeros or ones into the inputs or adding a dropout layer between the inputs and first hidden layer;
- salt-and-pepper noise: a fraction of the input is randomly chosen and randomly set to its minimum or maximum value;

Denoising Autoencoders

Training a denoising autoencoder is nearly the same process as training a regular autoencoder.

The only difference is we supply our corrupted inputs as training set and supply the non-corrupted inputs as ground truth.



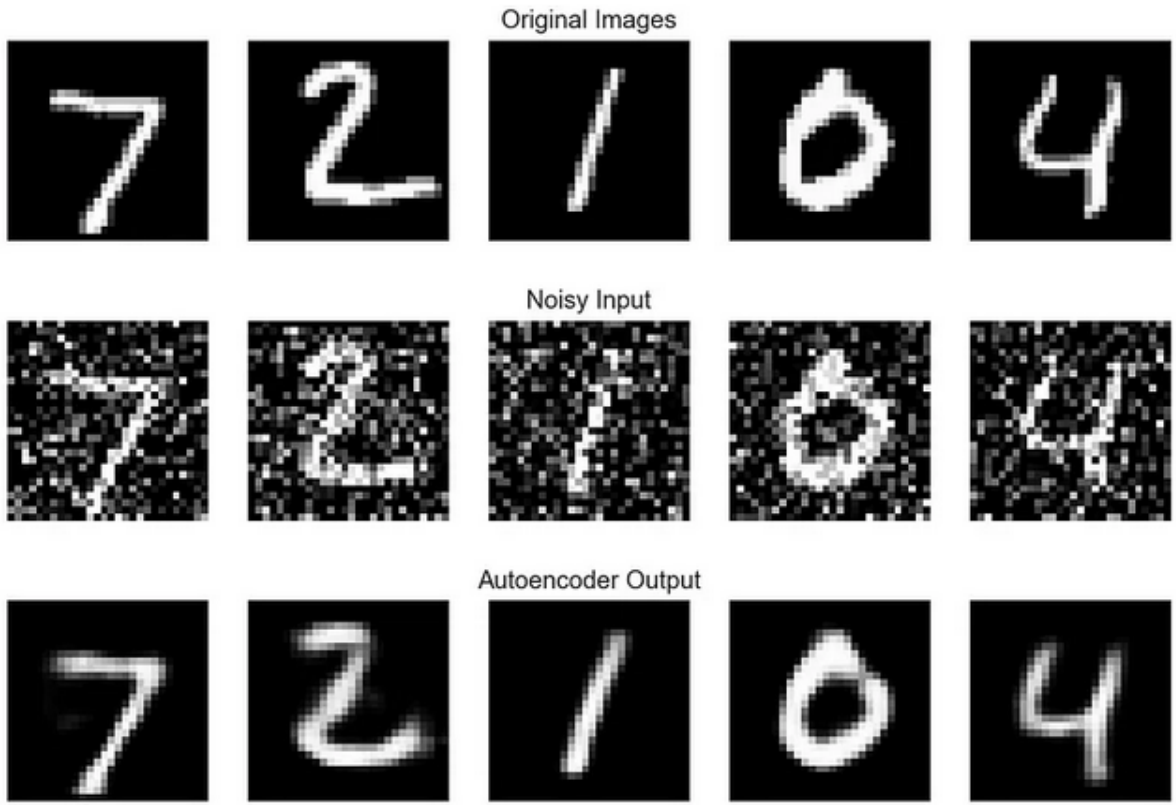
Formally, the DAE is associated to a different loss function as compared to a "vanilla" autoencoder.

Letting the noise process be defined by a probability distribution μ_T over functions $T : \mathcal{X} \to \mathcal{X}$, the problem of training a DAE is the optimization problem

$$\min_{\theta,\phi} L(\theta, \phi) := \mathbb{E}_{x \sim \mu_X, T \sim \mu_T}[d(x, g_\phi(f_\theta(Tx)))]$$

Denoising Autoencoders

The figure below shows an application of a DAE.
The first row shows a sample of the original digits, which are used as the validation data set; the second row shows the Gaussian corrupted inputs used to train the model; the third row shows the reconstructed digits after denoising.



Sparse Autoencoders

Sparse autoencoders are designed to pull out the most influential feature representations of the input data by using a **sparsity constraint** such that only a fraction of the nodes would have nonzero values.

Since it is impossible to design a neural network with a flexible number of nodes at its hidden layers, sparse autoencoders work by penalizing the activation of some neurons in hidden layers.
It means that **a penalty directly proportional to the number of neurons activated is applied to the loss function**.

The codes $f_\theta(X)$ for messages tend to be sparse codes, that is, $f_\theta(X)$ is close to zero in most entries. Sparse autoencoders may include more (rather than fewer) hidden units than inputs, but only a small number of the hidden units are allowed to be active at the same time.

Sparse Autoencoders

There are two main ways to enforce sparsity.

The k-**sparse autoencoder** clamps all but the highest-k activations of the latent code to zero.

The k-sparse autoencoder inserts the following "k-sparse function" in the latent layer of a standard autoencoder:

$$f_k(x_1, ..., x_n) = (x_1 b_1, ..., x_n b_n)$$

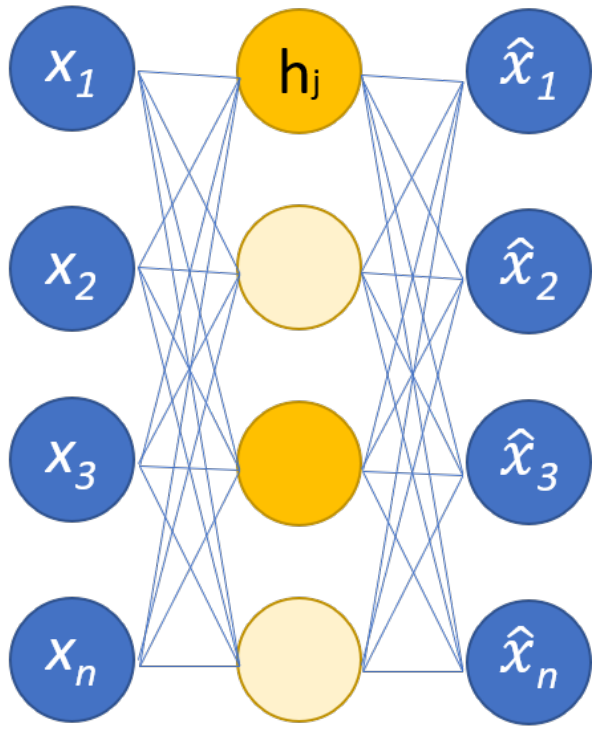
where $b_i = 1$ if $|x_i|$ ranks in the top k, and 0 otherwise.

Backpropagating through f_k is simple: set gradient to 0 for $b_i = 0$ entries, and keep gradient for $b_i = 1$ entries.

This is essentially a generalized ReLU function.

Sparse Autoencoders

Single-layer sparse autoencoder.



The hidden nodes in bright yellow are activated, while the light yellow ones are inactive. The activation depends on the input.

Sparse Autoencoders

Another way to enforce sparsity is, rather than forcing sparsity, to add a **sparsity regularization loss** term.

In this case, we optimize for

$$\min_{\theta,\phi} L(\theta,\phi) + \lambda L_{sparsity}(\theta,\phi)$$

where $\lambda > 0$ measures how much sparsity we want to enforce.

To define a sparsity regularization loss, we need a "desired" sparsity $\hat{\rho}_k$ for each layer k, a weight w_k for how much to enforce each sparsity, and a function $s : [0,1] \times [0,1] \to \mathbb{R}$ to measure how much two sparsities differ. For each input x let the actual sparsity of activation in each layer k be

$$\rho_k(x) = \frac{1}{n}\sum_{i=1}^{n} a_{k,i}(x)$$

where $a_{k,i}(x)$ is the activation in the i-th neuron of the k-th layer upon input x.

Sparse Autoencoders

The sparsity loss upon input x for one layer is $s(\hat{\rho}_k, \rho_k(x))$; hence the sparsity loss for the entire autoencoder is the expected weighted sum of sparsity losses over all K layers:

$$L_{sparsity}(\theta, \phi) = \mathbb{E}_{x \sim \mu_X} \left[\sum_{k=1}^{K} w_k s(\hat{\rho}_k, \rho_k(x)) \right]$$

There are different choices for the function s. Common choices are:

- the Kullback-Leibler divergence,

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

- the L1 loss

$$s(\rho, \hat{\rho}) = |\rho - \hat{\rho}|$$

- the L2 loss

$$s(\rho, \hat{\rho}) = |\rho - \hat{\rho}|^2$$

Contractive Autoencoders

A **Contractive AutoEncoder** (CAE) adds a contractive regularization loss to the standard autoencoder loss that **penalizes the network for changing the output in response to insufficiently large changes in the input**:

$$\min_{\theta,\phi} L(\theta,\phi) + \lambda L_{contractive}(\theta,\phi)$$

where $\lambda > 0$ measures how much contractive-ness we want to enforce.

This penalty term is calculated using the Frobenius norm of the Jacobian matrix of neuron activations in the encoder network with respect to the input.

Recall that the Jacobian matrix contains the first-order derivates of a function and that the Frobenius norm of a matrix is calculated as the square root of the sum of the absolute squares of its elements; it measures the average gain of the matrix along each orthogonal direction in space.

Contractive Autoencoders

Formally, the contractive regularization loss is defined as:

$$L_{contractive}(\theta, \phi) = \mathbb{E}_{x \sim \mu_{ref}} \|\nabla_x f_\theta(x)\|_F^2$$

where f_θ is the encoder function.
To understand what $L_{contractive}$ measures, note that

$$\|f_\theta(x + \delta x) - f_\theta(x)\|_2 \leq \|\nabla_x f_\theta(x)\|_F \|\delta x\|_2$$

for any input $x \in \mathcal{X}$ and small variation δx in it.

Thus, if the Frobenius norm of the gradient $\|\nabla_x f_\theta(x)\|_F^2$ is small, it means that a small neighborhood of the input maps to a small neighborhood of its code.

This is a desired property, as it means small variation in the input leads to small variation in its code.

Autoencoders - Manifold Learning

Lecture by Sargur Srihari

Variational autoencoder

Variational AutoEncoders (VAEs) belong to the families of **variational Bayesian methods**.

Despite the architectural similarities with basic autoencoders, VAEs have different goals and a different mathematical formulation.

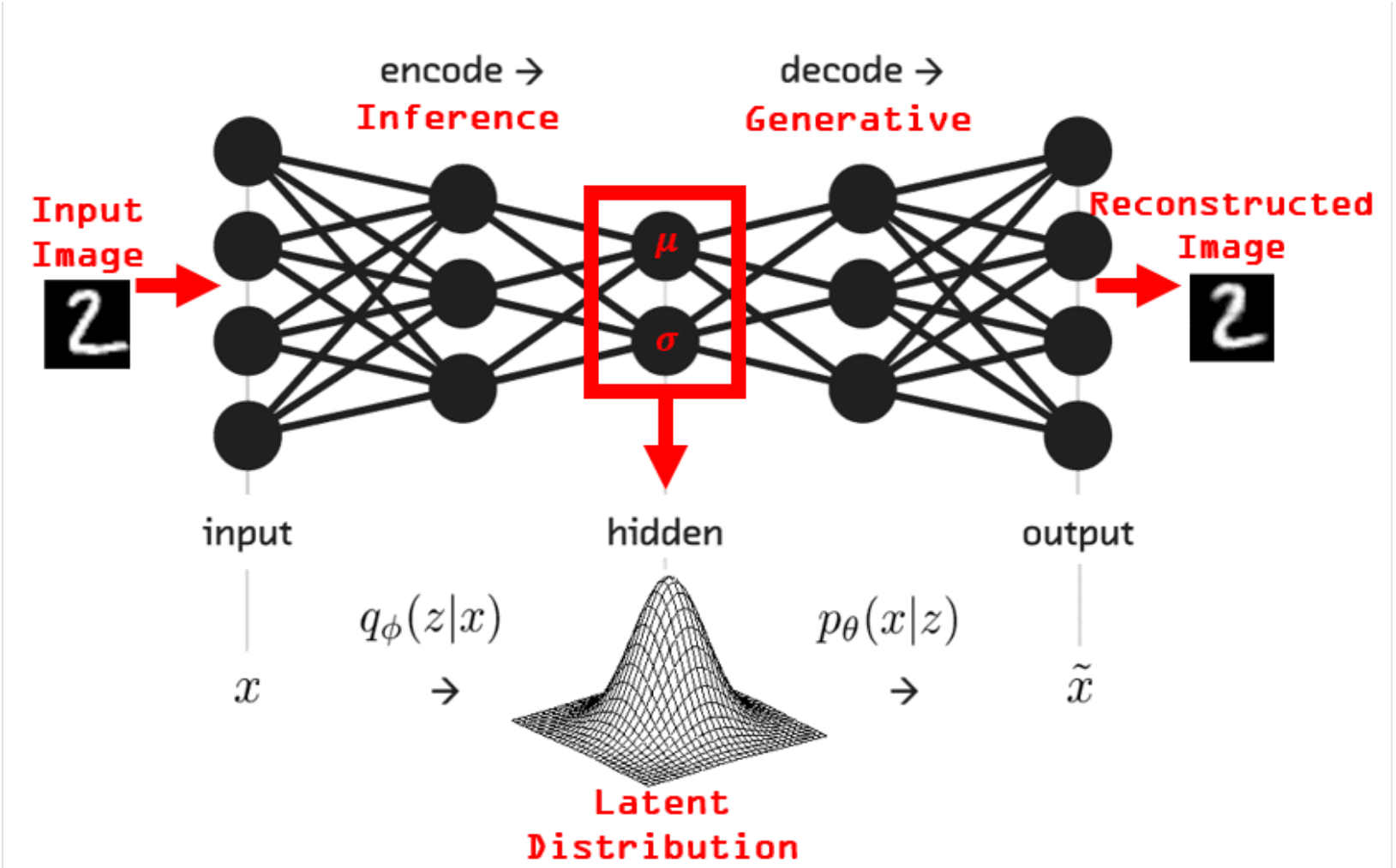
The fundamental difference between VAEs and other types of autoencoders is that while most autoencoders learn discrete latent space models, **VAEs learn continuous latent variable models**. The latent space of a VAE is typically composed by a mixture of distributions instead of a fixed vector.

Given an input dataset x characterized by an unknown probability function $P(x)$ and a multivariate latent encoding vector z, the objective is to model the data as a distribution $p_\theta(x)$, with θ defined as the set of the network parameters so that

$$p_\theta(x) = \int_z p_\theta(x, z) dz$$

Variational autoencoder

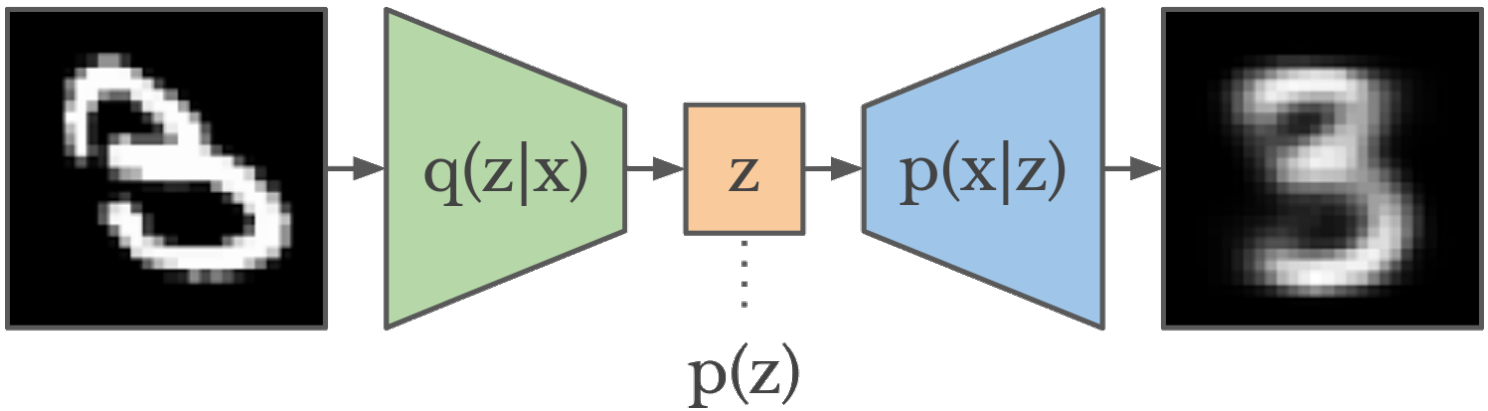
VAEs are trained to **learn the probability distribution** that models the input-data and not the function that maps the input and the output.



Variational autoencoder

Because the latent variables of the VAE capture attributes as a probability distribution — learning the **latent distribution** — the VAE is **generative AI models**.

By learning to encode important features from the inputs in the datasets, VAEs can samples points from the latent distribution and feed them to the decoder to **generate new samples that resemble the original training data**.



Variational autoencoder

Due to these properties, VAEs are useful in applications such as

- Image Generation and Synthesis. VAEs allow for image generation by learning rich latent representations, enabling the creation of high-quality, diverse, and realistic images. Applications range from generating art and photorealistic images to enhancing image quality and data augmentation for computer vision tasks.
- Anomaly Detection. In anomaly detection, VAEs excel at learning the underlying structure of normal data, enabling the identification of anomalies as deviations from known patterns.
- Representation Learning. VAEs facilitate unsupervised learning of meaningful representations from data. They extract essential features and capture latent relationships within complex datasets, aiding downstream tasks like classification, clustering, and recommendation systems.
- Molecular Design. In drug discovery, VAEs assist in generating novel molecular structures by navigating the chemical space. They aid in molecule generation, optimization, and de novo molecular design, accelerating drug development processes.

Variational autoencoder - Loss function

One of the key aspects of VAE is the **loss function**.

Most commonly, it consists of two components:

- The **reconstruction loss** measures how different the reconstructed data are from the original data. As reconstruction loss, mean squared error and cross entropy are often used.
- The **KL-divergence** tries to regularize the process and keep the reconstructed data as diverse as possible. The KL divergence is measured between the probability distribution of training data (the prior distribution) and the distribution of latent variables learned by the VAE (the posterior distribution)

Loss:

$$L_{\theta,\phi}(x) = -\mathbb{E}_{z\sim q_\phi}[\ln p_\theta(x|z)] + D_{KL}(q_\phi(z|x) \parallel p_\theta(z))$$

The first term is the reconstruction loss or expected negative log-likelihood of x

Variational autoencoder - Mathematical formulation

Mathematical formulation

From the point of view of probabilistic modeling, one wants to maximize the likelihood of the data x by their chosen parameterized probability distribution $p_\theta(x)$.

This distribution is usually chosen to be a Gaussian $N(x|\mu, \sigma)$ which is parameterized by μ and σ respectively,

We can find $p_\theta(x)$ via marginalizing over z:

$$p_\theta(x) = \int_z p_\theta(x, z)\, dz,$$

where $p_\theta(x, z)$ represents the joint distribution under p_θ of the observable data x and its latent representation or encoding z.

Variational autoencoder - Mathematical formulation

By the chain rule, the last equation can be rewritten as

$$p_\theta(x) = \int_z p_\theta(x|z)\, p_\theta(z)\, dz$$

where

- $p_\theta(z)$ is the **prior**, encoding the representations in the latent space.
- $p_\theta(x|z)$ is the **likelihood**. This is the **probabilistic decoder** describing the distribution of the decoded variable given the encoded one.
- $p_\theta(z|x)$ is the **posterior**. This is the **probabilistic encoder**, describing the distribution of the encoded variable given the decoded one.

Notice that the regularization of the latent space that is missing in simple autoencoders appears here in the definition of the data generation process *as the encoded representations in the latent space are assumed to follow the prior distribution*.

Variational autoencoder - Mathematical formulation

It is commonly assumption that $p_\theta(z)$ is a standard Gaussian distribution

$$p_\theta(z) \sim N(0, I)$$

and that $p_\theta(x|z)$ is a Gaussian distribution whose mean is defined by a deterministic function $f \in F$ of the variable of z and whose covariance matrix has the form of a positive constant c that multiplies the identity matrix I; here F is a fixed function class

$$p_\theta(x|z) \sim N(f(z), cI), \quad f \in F, c > 0$$

In theory, as we know $p_\theta(z)$ and $p_\theta(x|z)$, we can use the Bayes theorem to compute $p_\theta(z|x)$.

However, this Bayesian inference problem is often intractable.

Variational autoencoder - Mathematical formulation

To approximate $p_\theta(z|x)$, we will use **variational inference**.

The idea is to set a parametrized family of distribution - for example the family of Gaussians, whose parameters are the mean and the covariance - and to look for the best approximation of our target distribution among this family.

The best element in the family is one that minimize a given approximation error measurement. This can be found using the Kullback-Leibler divergence between approximation and target.

This is found computationally by gradient descent over the parameters that describe the family.

Variational autoencoder - Mathematical formulation

Thus, we are going to approximate $p_\theta(z|x)$ by a Gaussian distribution $q_\phi(z|x)$ whose mean and covariance are defined by two functions, g and h, of a parameter ϕ.

These two functions are supposed to belong, respectively, to the parametrized families of functions G and H. Thus we can denote

$$q_\phi(z|x) = N(g_\phi(x), h_\phi(x)), \quad g_\phi \in G, h_\phi \in H$$

To ensure that our variational posterior $q_\phi(z|x)$ approximates the true posterior $p_\theta(z|x)$ We use the Kullback-Leibler divergence, which measures the information lost.

Variational autoencoder - Mathematical formulation

The Kullback-Leibler divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$ is expanded as

$$\begin{aligned} D_{KL}(q_\phi(z|x) \parallel p_\theta(z|x)) &= \mathbb{E}_{z\sim q_\phi(\cdot|x)}\left[\ln \frac{q_\phi(z|x)}{p_\theta(z|x)}\right] \\ &= \mathbb{E}_{z\sim q_\phi(\cdot|x)}\left[\ln \frac{q_\phi(z|x)p_\theta(x)}{p_\theta(x,z)}\right] \\ &= \ln p_\theta(x) + \mathbb{E}_{z\sim q_\phi(\cdot|x)}\left[\ln \frac{q_\phi(z|x)}{p_\theta(x,z)}\right] \end{aligned}$$

We parametrize the encoder as E_ϕ, and the decoder as D_θ.

Variational autoencoder - Mathematical formulation

Next we define the **evidence lower bound (ELBO)**:

$$L_{\theta,\phi}(x) := \mathbb{E}_{z\sim q_\phi(\cdot|x)}\left[\ln \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] = \ln p_\theta(x) - D_{KL}(q_\phi(\cdot|x) \parallel p_\theta(\cdot|x))$$

Maximizing the ELBO

$$\theta^*, \phi^* = \operatorname*{argmax}_{\theta,\phi} L_{\theta,\phi}(x)$$

is equivalent to simultaneously maximizing $\ln p_\theta(x)$ and minimizing $D_{KL}(q_\phi(z|x) \parallel p_\theta(z|x))$.

That is, maximizing the log-likelihood of the observed data and minimizing the divergence of the approximate posterior $q_\phi(\cdot|x)$ from the exact posterior $p_\theta(\cdot|x)$.

Variational autoencoder - Mathematical formulation

The form given is not very convenient for maximization, but the following, equivalent form, is:

$$L_{\theta,\phi}(x) = \mathbb{E}_{z\sim q_\phi(\cdot|x)}\left[\ln p_\theta(x|z)\right] - D_{KL}(q_\phi(\cdot|x) \parallel p_\theta(\cdot)).$$

Under the assumption that $x \sim \mathcal{N}(D_\theta(z), I)$, that is, if we model the distribution of x on z to be a Gaussian distribution centered on $D_\theta(z)$ then $\ln p_\theta(x|z)$ is implemented as $-\frac{1}{2}\|x - D_\theta(z)\|_2^2$

The distribution of $q_\phi(z|x)$ and $p_\theta(z)$ are chosen to be Gaussians as $z|x \sim \mathcal{N}(E_\phi(x), \sigma_\phi(x)^2 I)$ and $z \sim \mathcal{N}(0, I)$. Hence we can aply the formula for KL divergence of Gaussians to conclude

$$\begin{aligned} L_{\theta,\phi}(x) &= -\frac{1}{2}\mathbb{E}_{z\sim q_\phi(\cdot|x)}\left[\|x - D_\theta(z)\|_2^2\right] \\ &- \frac{1}{2}\left(N\sigma_\phi(x)^2 + \|E_\phi(x)\|_2^2 - 2N\ln\sigma_\phi(x)\right) + Const, \end{aligned}$$

where N is the dimension of z.

Variational autoencoder

The regularized loss function enables VAEs to generate new samples that resemble the data it was trained on.

To generate a new sample, the VAE samples a random latent vector from within the unit Gaussian — in other words, selects a random starting point from within the normal distribution — shifts it by the mean of the latent distribution and scales it by the variance of the latent distribution.

This process, called the **reparameterization trick**, avoids direct sampling of the variational distribution: because the process is random, it has no derivative, hence eliminates the need for backpropagation.

Variational Autoencoders

Example. We train a VAE to be used as a generative model for generating digits. This means that we use the decoder to generate data similar to the data used determine the latent space.

We design it using fully-connected encoders and decoders and we train it using some images from the MNIST dataset.

The input dimension is 784 which is the flattened dimension of MNIST images (28×28).

In the encoder, the mean μ and variance σ^2 vectors are our variational representation vectors.

The final encoder dimension has dimension 2 which are the μ and σ^2 vectors. These continuous vectors define our latent space distribution that allows us to sample images in VAE.

Variational Autoencoder (VAE) — PyTorch Tutorial

Variational Autoencoders - Generative models