Autoencoders

DEEP LEARNING – MAI

SPRING 2025

CAROLINE KÖNIG

Outline

1. Embeddings
2. Autoencoders
3. Variational autoencoders

Introduction

What are embeddings?

Definition: Embeddings are vector representations of data designed to capture the semantic meaning or relationships between entities.

Significance: They enable machines to process complex data like text, images, and graphs in a structured and meaningful way by creating vectoral representations of the entities.

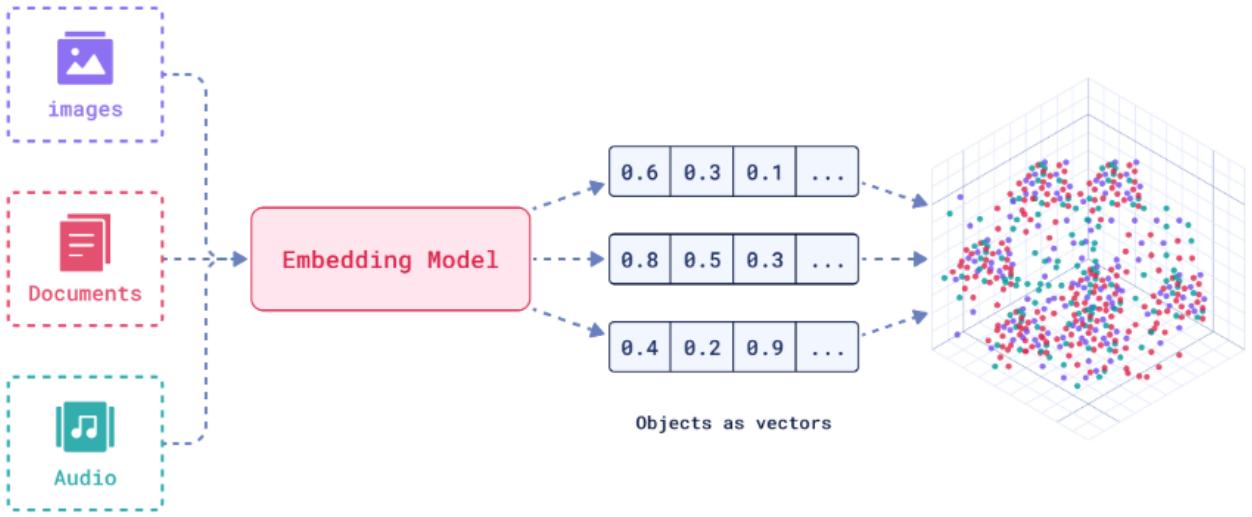
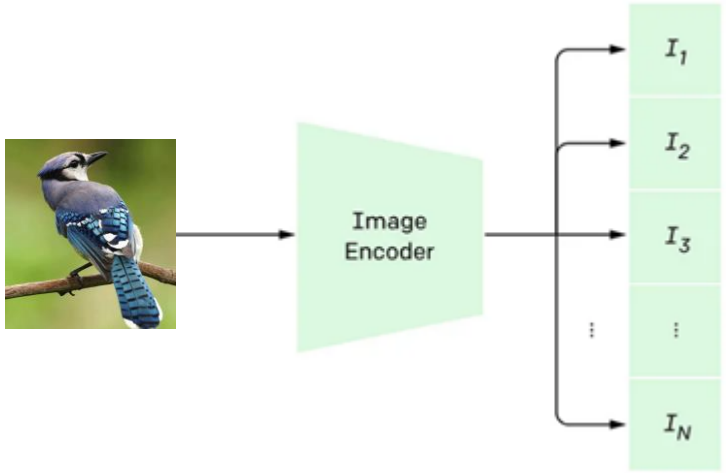
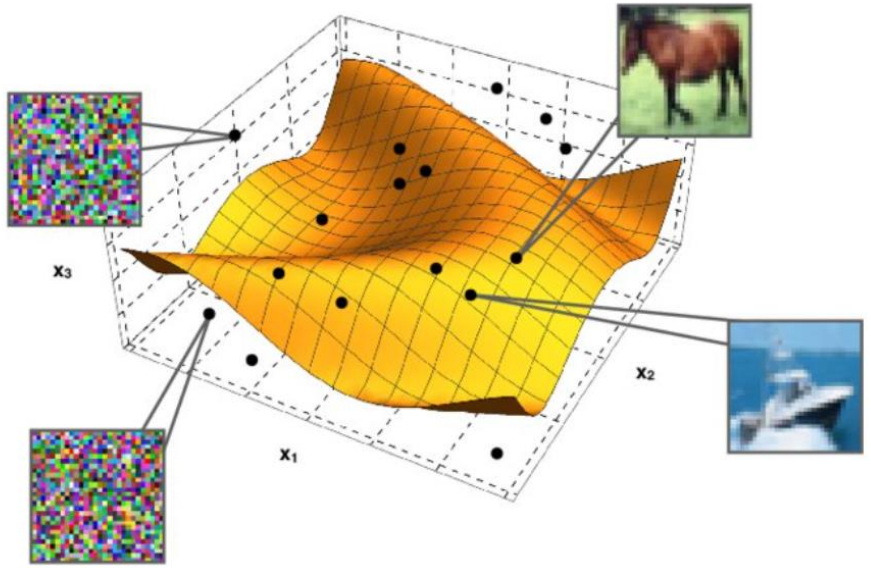


Image embeddings

Neural networks are able to learn vectorial representation of high-dimensional data to create structured and meaningful representations in a lower dimensional space.

Manifolds are essential for tasks like image classification, clustering, and retrieval because they help define regions in embedding space where similar categories reside.





Text embeddings

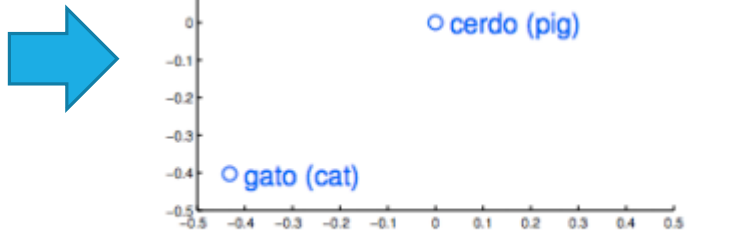
Word2vec is a two-layer neural network that processes text by "vectorizing" words.

Its input is a text corpus, and its output is a set of vectors: feature vectors representing words in that corpus.

While Word2vec is not a deep neural network, it transforms text into a numerical form that deep neural networks can understand.

Example Corpus:

- *La vaca y los caballos se alimentan de hierba.*
- *Pero y gatos son animales domesticos.*
- *Caballo, vaca y cerdo son animales de granja.*
- *El perro vivía en la granja con vacas y cerdos.*



Semantic similarity between words

SIMILARITY METRIC

Cosine similarity to measure the similarity between vectors.

A lack of similarity is expressed as a 90-degree angle.

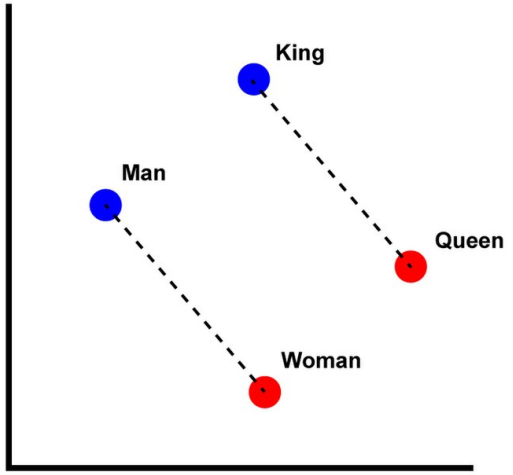
Total similarity (1) corresponds to a 0-degree angle, representing complete overlap.

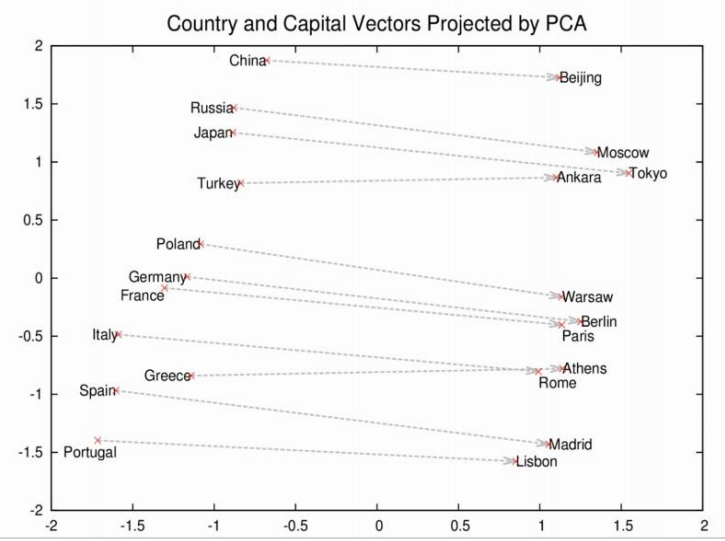
SIMILAR WORDS TO 'SWEDEN'

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

Embedding space is navigable

Word embeddings capture the **semantic relationships** between words, allowing NLP models to better understand context and meaning.



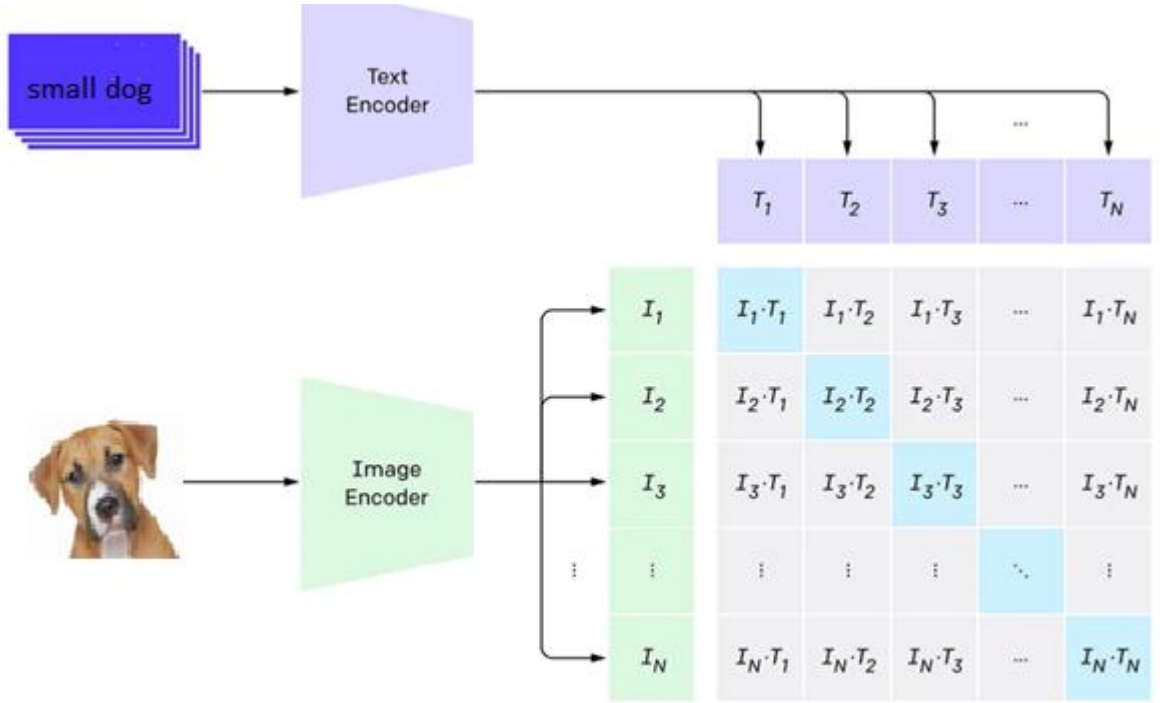


Contrastive Language–Image Pre-training

You can give CLIP an image or a piece of text, and CLIP will output an abstract *representation* of your input.

Applications:
- Image classification.
- Image Search.

How-to use for image-search?
- Text-to-image models (DALL-E 2)



Overview embedding learning approaches

Neural network models for learning embeddings

Autoencoders: Compresses data into lower-dimensions through encoder-decoder architectures.

Convolutional Neural Networks: Extracts spatial patterns.

Recurrent Neural Networks: Captures sequential patterns in time-series data.

Transformers: Contextual embeddings powered by self-attention mechanisms.

Graph-neural networks: Node embeddings to represent relationships in graph-structured data.

Word-2-Vec: Creates word embeddings by predicting either a target word from context (CBOW) or context from a target word (Skip-Gram).

Contrastive learning: Learns embeddings by contrasting positive and negative examples.

Conventional approaches: PCA, UMAP, t-SNE

Autoencoders

Introduction

A central goal of deep learning is to discover **representations of data** that are useful for subsequent applications.

Autoencoder or *auto-associative neural networks* are a well-established approach to learning internal representations of data.

Types of autoencoder

- Undercomplete autoencoders
- Sparse autoencoders
- Denoising autoencoders
- Masked autoencoders

Basic idea

Autoencoders learn the internal representation of data.

Autoencoders are designed to be ***unable to learn perfectly***.

Restricted in ways to allow to ***copy only approximately***, and to copy only input that resembles the training data.

The model is forced to prioritize which aspects of the input should be copied, so that it learns useful properties of the data.

Used in unsupervised learning to learn efficient coding of unlabeled data.

How to force approximate learning ?

Definition

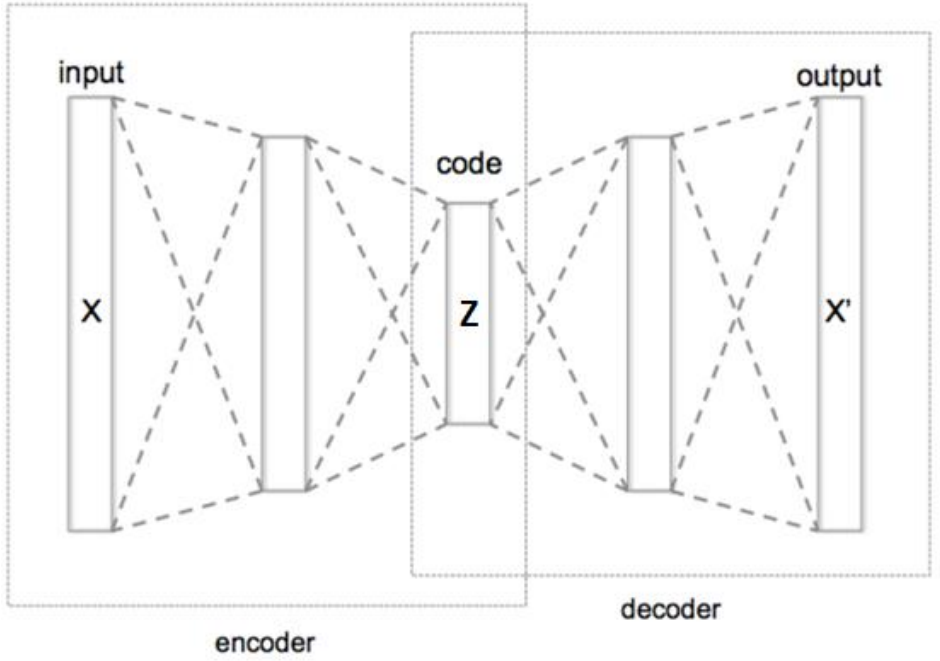
An autoencoder is trained to attempt to **copy its input to its output**.

Trained by backpropagation, setting the target values to be equal to the inputs.

Internally it has a hidden layer *z* that describes a **code** to represent the input.

Encoder function: $z = E(x)$

Decoder function: $x' = D(z)$



Undercomplete Autoencoder

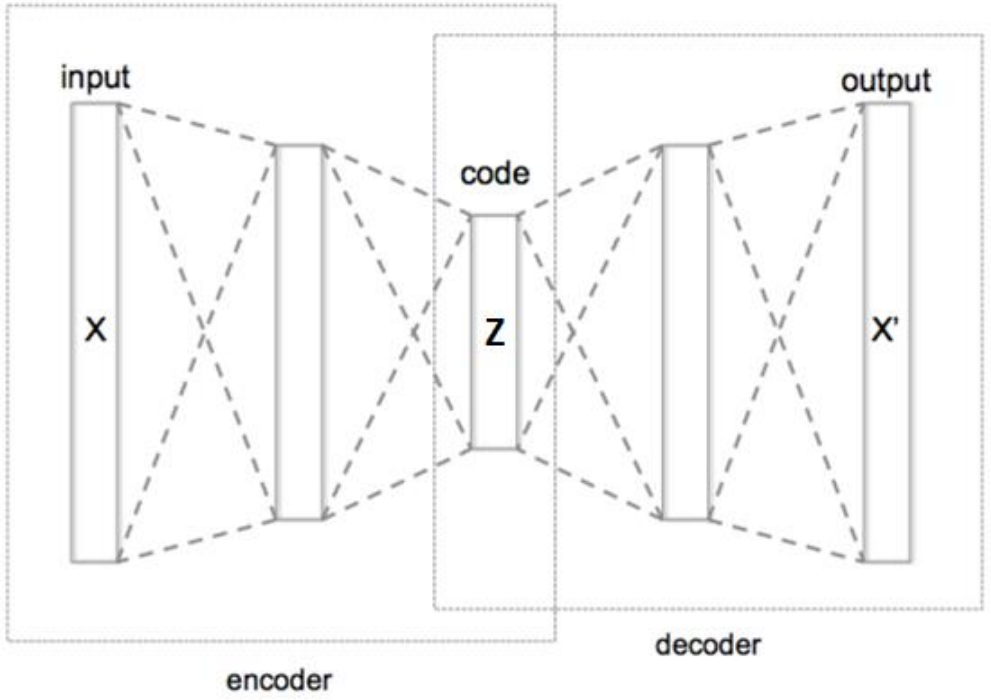
The *hidden layer z* - the bottleneck - has less dimensions than the input dimension.

Trained minimizing the reconstruction error E:

$$E = \|X - X'\|^2$$

The reconstruction error E measures how well the reconstructed data matches the original input.

The code dimension and the capacity of the encoder and decoder depend on the complexity of distribution to be modeled.



Linear autoencoder

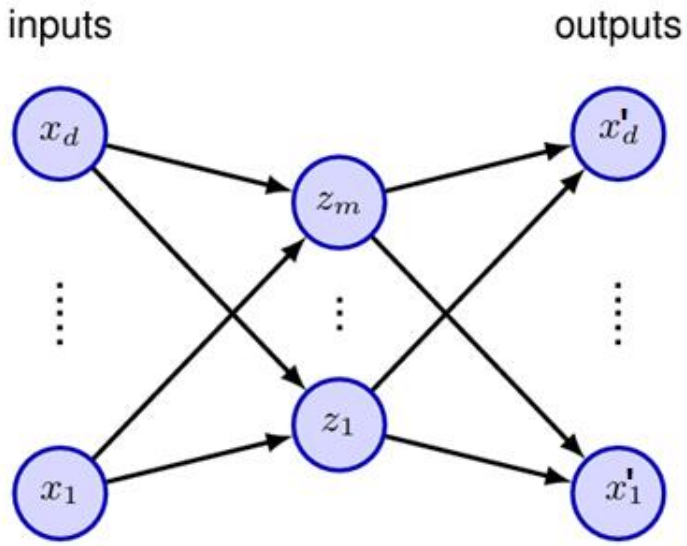
Neural net of D inputs, D output units, and M hidden units, with M < D.

Trained to map input vectors on themselves by minimizing sum-of-square errors.

Learns an *auto-associative mapping*.

$$E = \frac{1}{N}\sum_{i=1}^{N} \|x_i - x'_i\|^2$$

Evaluates the **reconstruction error**.



Equivalent to linear PCA , i.e. projection onto the M-dimensional subspace that is spanned by the first M principal components of the data.

Deep Autoencoder

Adds additional nonlinear layers to the network.

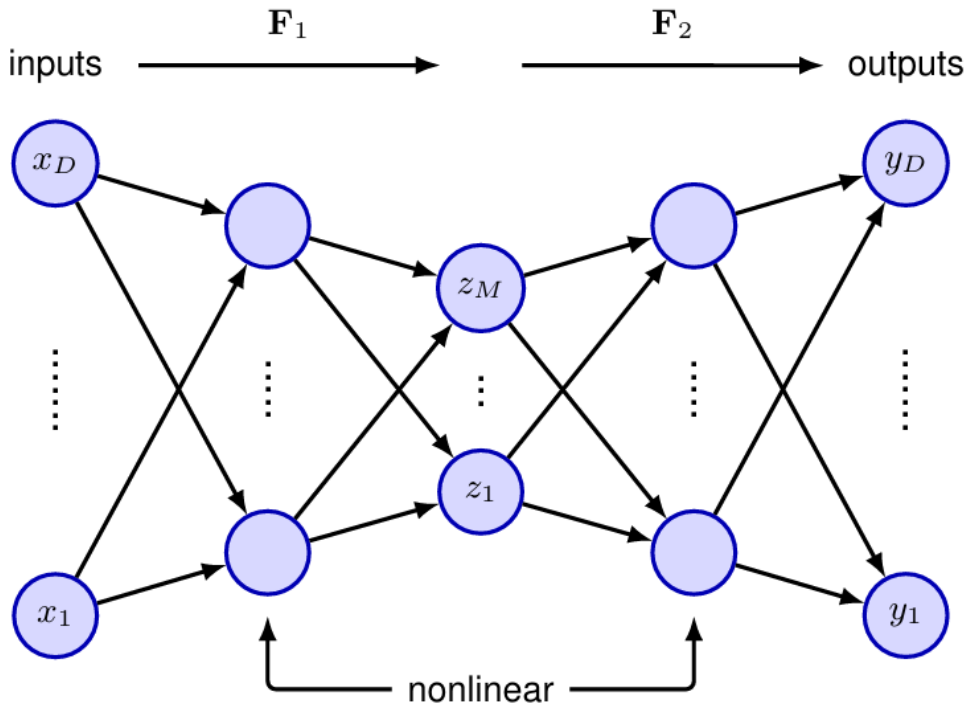
Several fully connected layers with progressively fewer units.

Symmetric architecture.

Using nonlinear activation functions the network effectively performs a nonlinear form of PCA.

Training the network now involves a nonlinear optimization (computational intensive).

Need to control overfitting to avoid learning purely the identity function.



Mappings of Deep Autoencoder

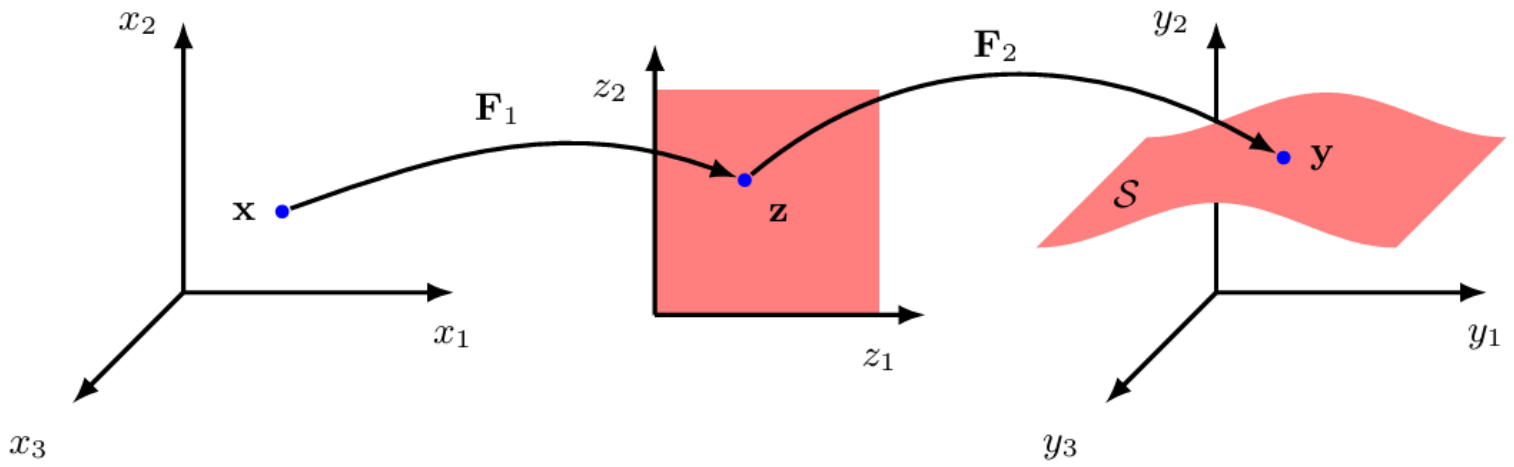


Figure 19.3 Geometrical interpretation of the mappings performed by the network in Figure 19.2 for a model with $D = 3$ inputs and $M = 2$ units in the second layer. The function $\mathbf{F}_2$ from the latent space defines the way in which the manifold $\mathcal{S}$ is embedded within the higher-dimensional data space. Since $\mathbf{F}_2$ can be nonlinear, the embedding of $\mathcal{S}$ can be non-planar, as indicated in the figure. The function $\mathbf{F}_1$ then defines a projection from the original D-dimensional data space into the M-dimensional latent space.

Hyperparameters

Code size: Smaller size results in more compression.

Number of layers: Ability to learn more complex transformations.

Symmetric architecture in terms of the layers of the encoder and decoder.

Loss function:

Mean Squared error for continuous variables (gray-scale or RGB images).

Cross-entropy for binary and categorical data (binary images).

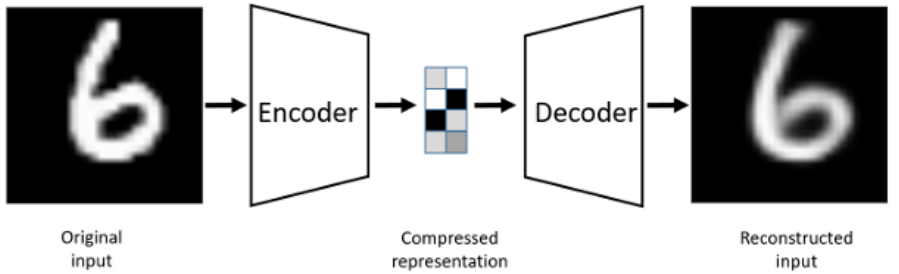
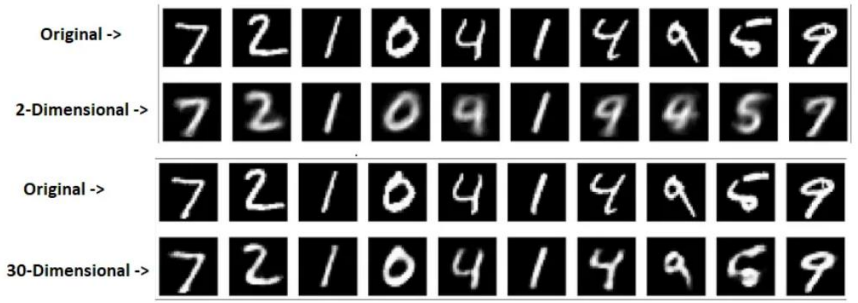


Fig. 1: An autoencoder example. The input image is encoded to a compressed representation and then decoded.



Applications

Properties

Data-specific: able to compress data similar to those they have been trained on.

Applications

Data compression

Image reconstruction

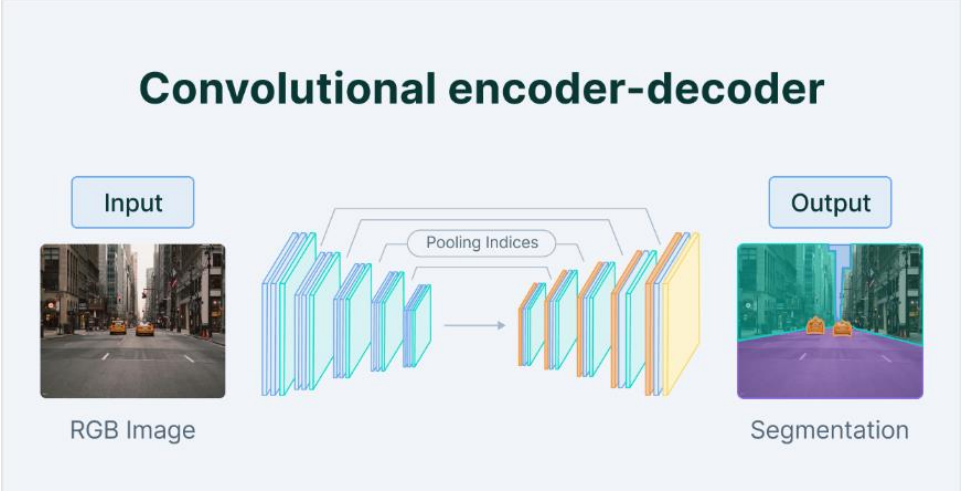
Image colorization

Dimensionality reduction

Watermark removal

Outlier detection

Image segmentation

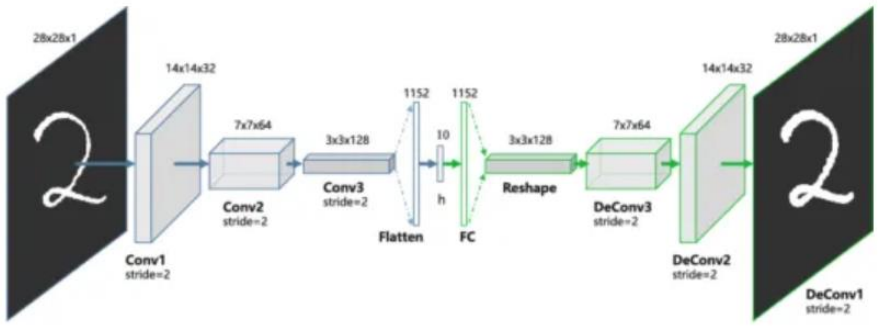


Advanced architectures

IMAGE DATA

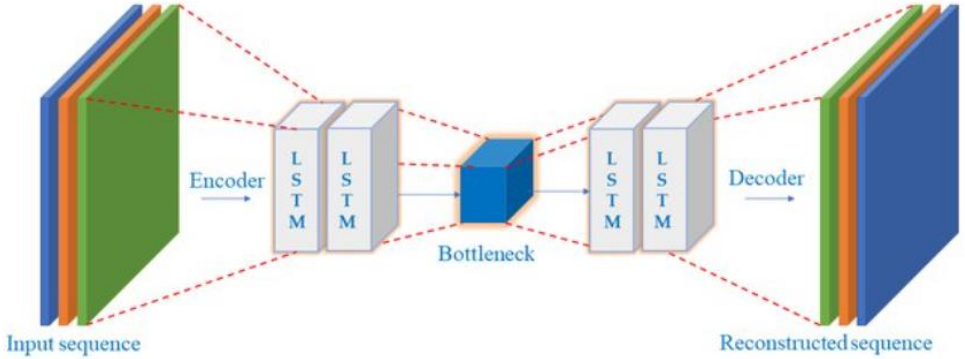
Convolutional layers to capture spatial patterns in images.

Applications: Reconstructs the image modifying the geometry or reflectance in an image.



TIME SERIES DATA

Recurrent layers such as Long Short Term memory (LSTMs) to capture the temporal evolution of the signal.
Applications: Outlier detection in sensor data, signal processing (robust to noise).



Anomaly detection

Train with normal data and evaluate on unseen data.

Data that is significantly different from the normal data have a high reconstruction error.

Used for outlier or novelty detection.

Need to set the anomaly threshold apropriately. For example from the normalized reconstruction error S_a.

$$S_a = \frac{R(t) - R(t)_{min}}{R(t)_{max}}$$

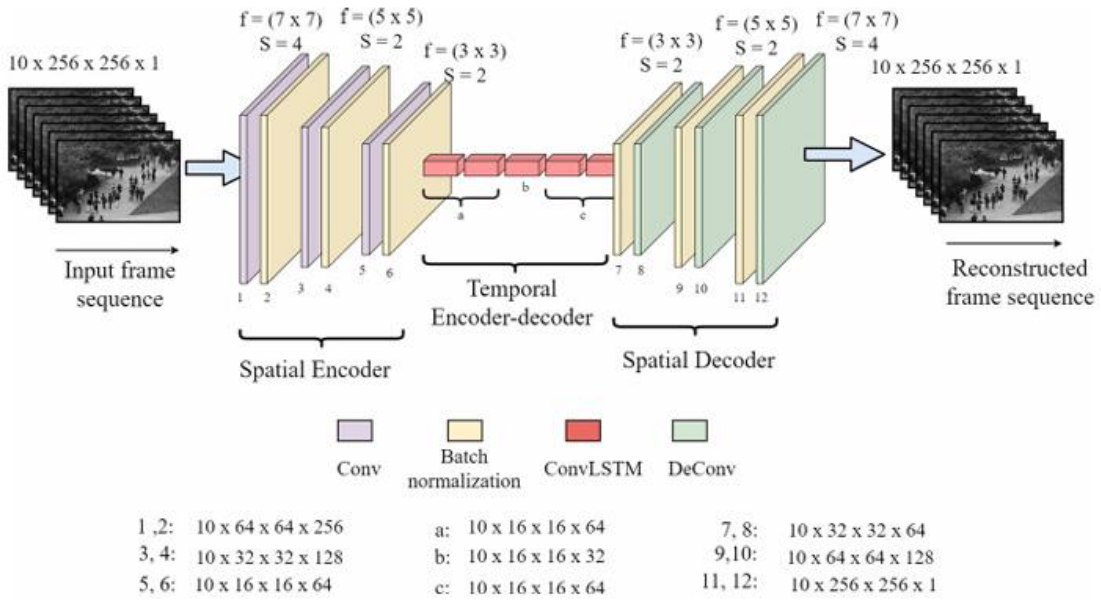


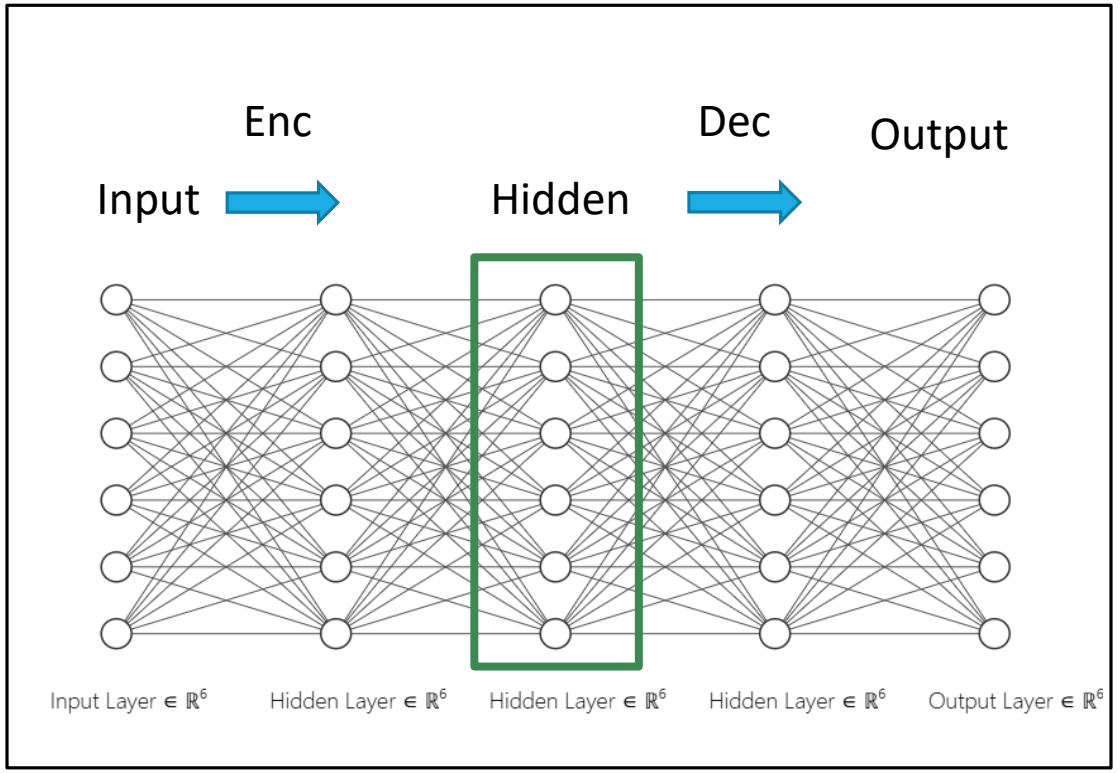
Fig. 8 Proposed architecture of IFA. A bunch of 10 frames are given as input for encoding and the same bunch of 10 frames are reconstructed back by the decoder after various parameters execution. Best viewed in color

Sparse autoencoder

Alternative method for introducing an information bottleneck **without requiring a reduction in the number of nodes** at the hidden layers.

Instead, constructs the loss function such that the activations within a layer are penalized.

Constrains the internal representation by using a ***regularizer*** to encourage a ***sparse representation.***



Sparsity regularization term

Limits the capacity of the autoencoder by adding a term to the cost function penalizing the hidden layer for being larger. The objective function involves a **sparsity penalty:**

$$\mathbf{L} = \|\mathbf{X} - \mathbf{X}'\|^2 + \boldsymbol{\lambda} \cdot \mathbf{Penalty}(\mathbf{s})$$

- λ is the regularization parameter
- Penalty (s) : A function that penalizes deviation from sparsity.

The sparsity constraint can be enforced through various techniques:

L1 – Regularization: Adds a penalization proportional to the absolute values of the weights.

KL-Divergence: Measures the difference between the average activation of the hidden neurons and a target sparsity level.

Sparsity regularization term

Informally, we will think of a neuron as being active if its output value is close to 1, or as being inactive if its output value is close to 0.

The sparse representation constrains the neurons to be inactive most of the time.

Let a_j denote the activation of the hidden unit *j* in the autoencoder and $a_j x_i$ is the activation of the unit given the input x_i.

So the activations of a hidden unit a_j is controlled by setting a threshold *p* on the average activation of a hidden unit j.

For sparsity parameter *p*=0.05 the average activation of each hidden unit is close to 0.05.

Average activation hidden unit a_j

$$\hat{\boldsymbol{p}}_{\boldsymbol{j}} = \frac{\mathbf{1}}{\boldsymbol{N}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} \boldsymbol{a_j x_i}$$

Sparse parameter

$$\hat{\boldsymbol{p}}_{\boldsymbol{j}} = \boldsymbol{p}$$

Controlling the sparsity parameter

Add an extra penalty term to the optimization objective that penalizes significant deviations of $\hat{p}_j$ from the sparsity parameter p .

$$\sum_{j=1}^{m} KL(p||p_j) = \sum_{j=1}^{m} p\ log\frac{p}{p_j} + (1-p)log\frac{1-p}{1-p_j}$$

Kullback-Leibler divergence between a Bernoulli random variable with mean *p* and a Bernoulli random variable with mean $\hat{p}_j$

KL (p $||\hat{p}_j$) measures how different two different distributions are.

Kullback-Leibler Divergence

KL (p $||\hat{p}_j$) = 0 when $\hat{p}_j = \mathrm{p}$

High values as $\hat{p}_j$ approaches 0 or 1.

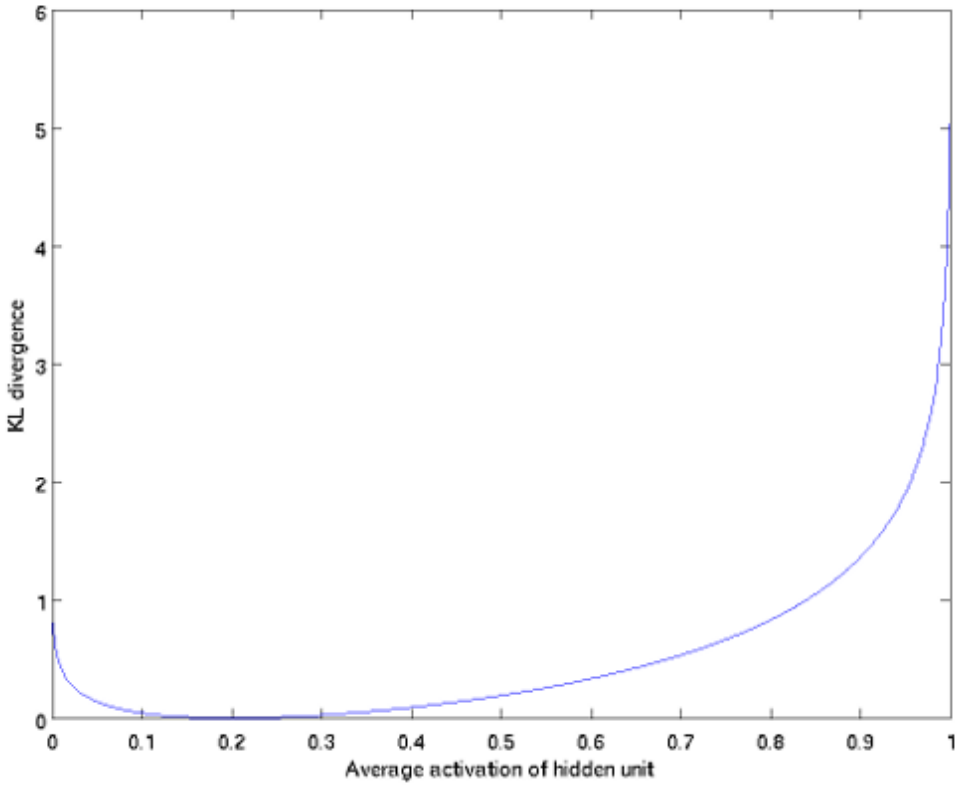
Thus, minimizing the penalty term has the effect of causing $\hat{p}_j$ to be close to p.

The overall cost function is now:

$$\mathbf{L} = \|\mathbf{X} - \mathbf{X}'\|^2 + \boldsymbol{\beta} \cdot \sum_{j=1}^{m} \boldsymbol{KL}(\boldsymbol{p}||\boldsymbol{p}_j)$$

β controls the weight of the sparsity penalty term.

Optimize with back-propagation.



Visualization of activation units

Insights about what the hidden units are learning.

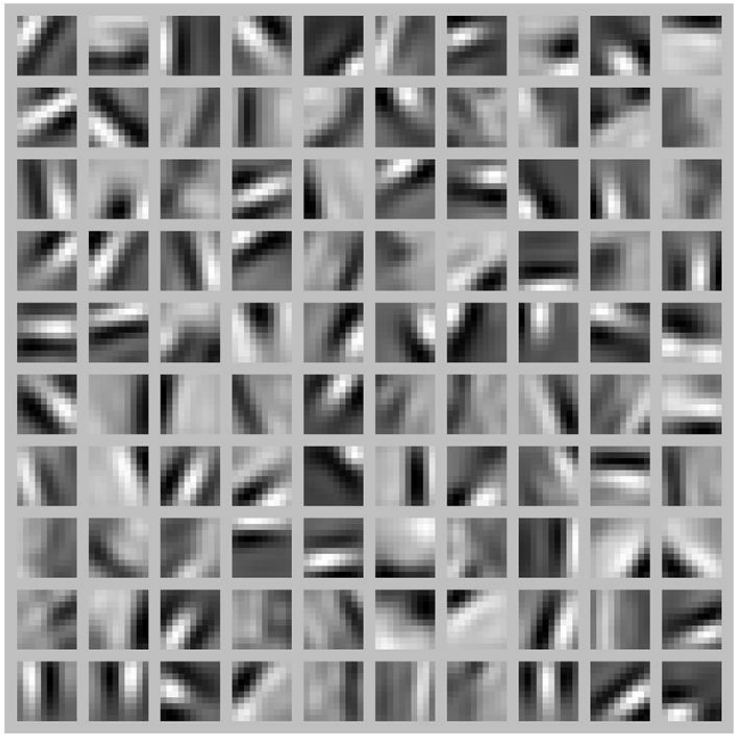
Train an autoencoder on 10x10 images with 100 units at the hidden layer.

$a_i = f(\sum_{i=1}^{100} W_{ij} + b)$

What input image *x* would cause a_i to be maximally activated ?

The input which maximally activates hidden unit i is given by setting the pixel x_j to:

$$x_j = \frac{W_{ij}}{\sqrt{\sum_1^{100}(W_{ij})^2}}$$



Each square shows the input image *x* that maximally actives one of 100 hidden units.

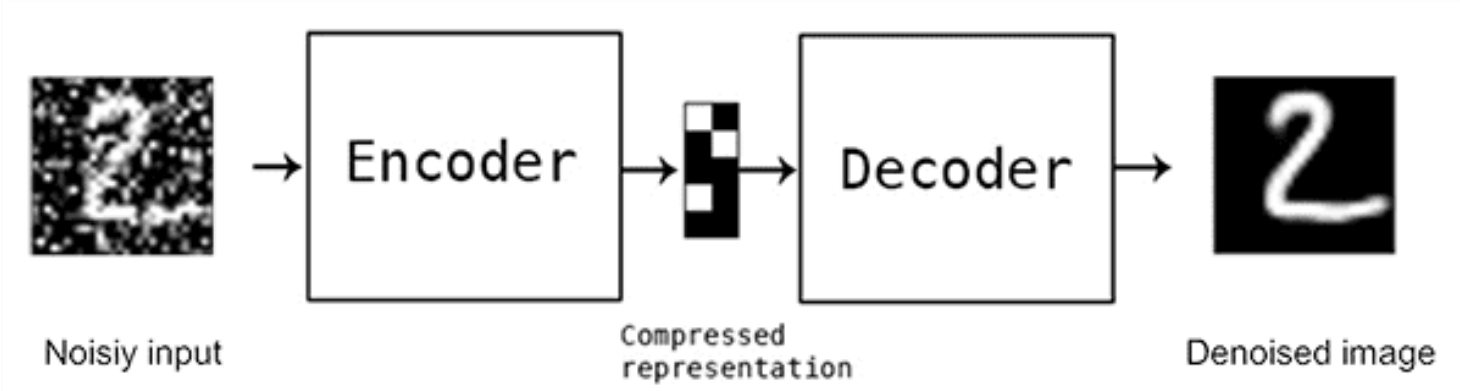
Denoising autoencoder (DAE)

Distorts the input data partially so that the model learns a generalization of the features.

Each input vector x_i is corrupted with noise to a given modified vector $\tilde{x_i}$, which is then the input for an autoencoder.

The network is trained to construct the original noise-free input vector by minimizing the MSE:

$$E = \frac{1}{N}\sum_{i=1}^{N}\|\tilde{x_i} - x_i\|^2$$



Estimating the scores

DAEs trained via SGD approximate the underlying score function of the data (for some cases of noise).

It provides a consistent estimator of probability distributions based on encouraging the model to have the same score as the data distribution at every training point *x*. In this context, the score is the gradient of the log-density ∇_xP(x) of the data distribution P(x).

Learns to reverse the distortion vector $\widetilde{x_i} - x_i$ and therefore learns a vector for each point in data space that points towards the manifold and therefore towards the region of high data density.

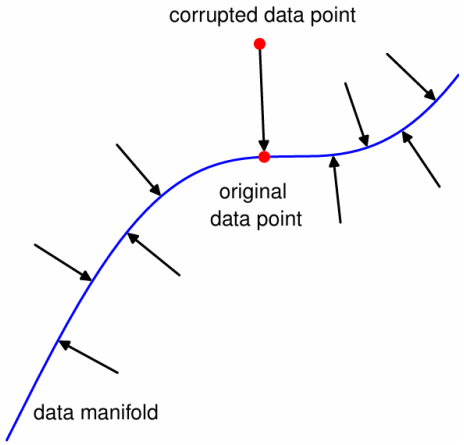


Figure 19.4 In a denoising autoencoder, data points, which are assumed to live on a lower-dimensional manifold in data space, are corrupted with additive noise. The autoencoder learns to map corrupted data points back to their original values and therefore learns a vector for each point in data space that points towards the manifold.

Denoising autoencoder

How to add noise to the input data ?

Setting a random choosen subset of input variables to zero (masking values).

Add independent zero-mean Gaussian noise to every input variable.

Apply salt-and-pepper noise (flipping randomly some input pixels or values).

Benefits

The autoencoder is able to handle missing data when trained on masked data.

The autoencoder tend to generalize well to unseen, real-world data with different level of noise and learns robust features.

Masked autoencoder

Similar to DAEs. Corruption is masking, or dropping out, part of the input image.

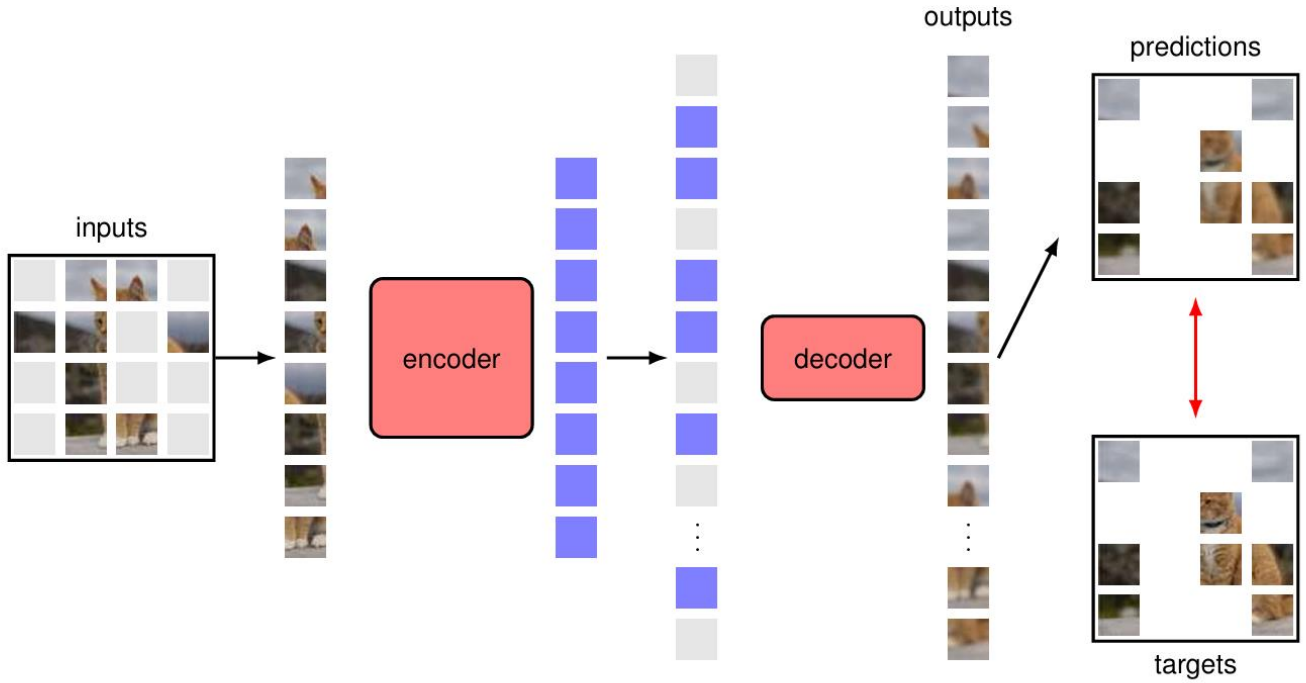


Figure 19.5 Architecture of a masked autoencoder during the training phase. Note that the target is the complement of the input as the loss is only applied on masked patches. After training, the decoder is discarded and the encoder is used to map images to an internal representation for use in downstream tasks.

Example masked autoencoder

Figure 19.6 Four examples of images reconstructed using a trained masked autoencoder, in which 80% of the input patches are masked. In each case the masked image is on the left, the reconstructed image is in the centre, and the original image is on the right. [From He *et al.* (2021) with permission.]

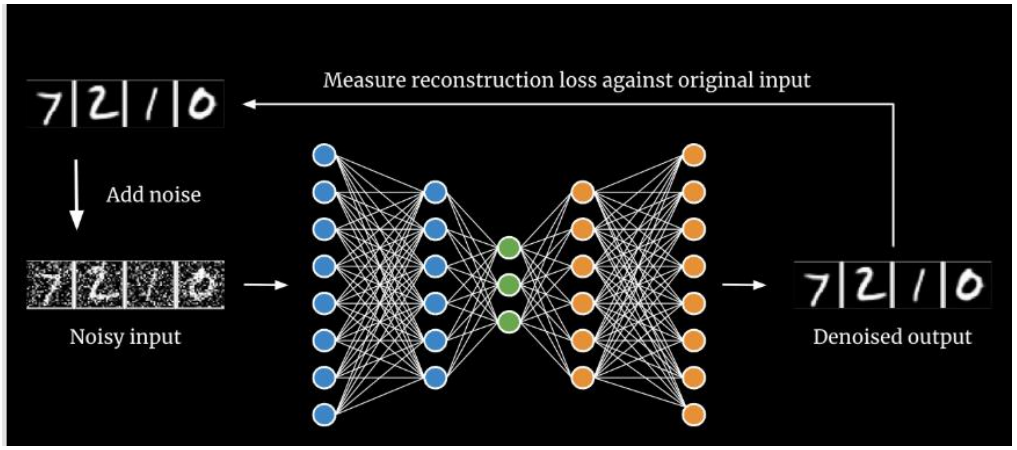
DAE applications

Denoising: Cleaning and enhancing images or audio signals, making them valuable in speech-processing tasks.

Sensor Data Processing: DAEs are valuable in processing sensor data, removing noise, and extracting relevant information from sensor readings.

Data Compression: Autoencoders, including DAEs, can be utilized for data compression by learning compact representations of input data.

Feature Learning: Masked DAEs are especially effective in unsupervised feature learning, capturing relevant features in the data without explicit labels.



Controlling overfitting

Undercomplete autoencoders, with code dimension less than the input dimension.

Choose the **code dimension** and the **capacity of the encoder and decoder** based on the complexity of distribution to be modeled.

Regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output.

These other properties include **sparsity** of the representation, **smallness of the derivative** of the representation, and **robustness to noise** or to missing inputs.

A regularized autoencoder can be nonlinear and overcomplete but still learn something useful about the data distribution, even if the model capacity is great enough to learn a trivial identity function.

Variational autoencoders

Motivation – Generative Models

One major division in machine learning is generative versus discriminative modeling.

While discriminative modeling aims to learn a predictor given the observations, generative modeling aims to solve the more general problem of learning a joint distribution over all the variables.

In discriminative methods we directly learn a map in the same direction as we intend to make future predictions. This is in the opposite direction than the generative model.

Discriminative classifier

Learns the conditional probability $P(y|x)$, i.e. the probability of the class label given the characteristics x.

Generative classifier

Builds a model to describe how the data looks like for a class, i.e. model the distribution of inputs characteristics for a label.

Model the probability $P(x, y)$. Given $p(y)$ we can compute $P(x|y)$ and apply Bayes rule for $P(y|x)$.

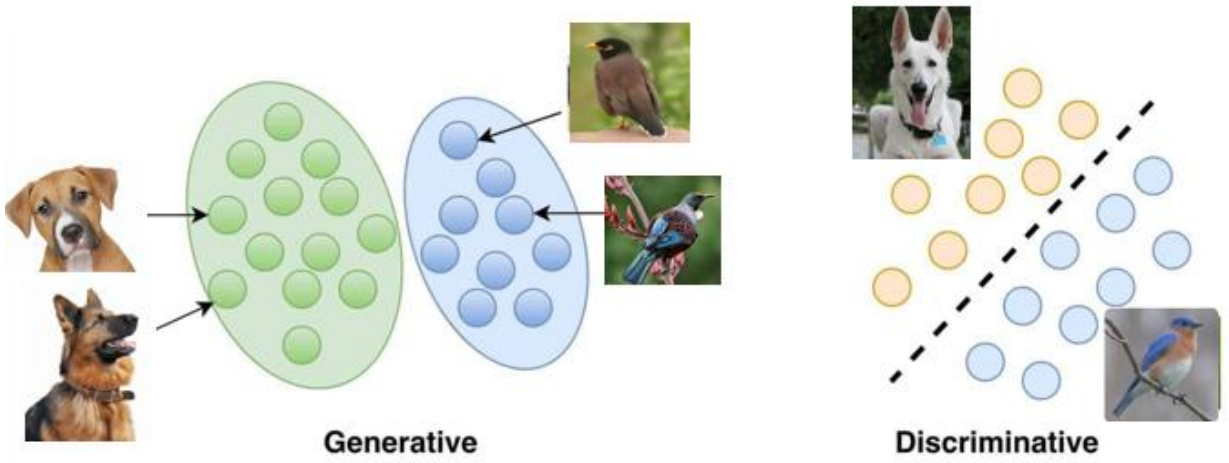
Generative vs Discriminative Models

DETECT THE LANGUAGE IN A TEXT

Generative model: Learn each language and determine to which language the text most probably belongs.

Discriminative approach: Identify the linguistic differences without learning any language.

OBJECT DETECTION



Stochastic models

Stochastic autoencoder are generative models that attempt to **describe data generation through a probabilistic distribution**.

The **variational autoencoder** (VAE) is a stochastic autoencoder for unsupervised representation learning designed to capture semantically meaningful, statistically independent and causal factors of variation in the data.

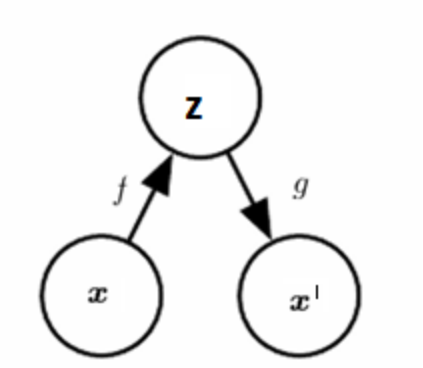
Use probabilistic techniques to model the data-generating process and incorporates randomness into both the encoding and decoding processes.

Deterministic autoencoder

Deterministic autoencoder map an input *x* to an output x' (called reconstruction) through an internal representation *z*.

The autoencoder has two components:

- Encoder mapping *z=f(x)*
- Decoder mapping *x'=g(z)*.

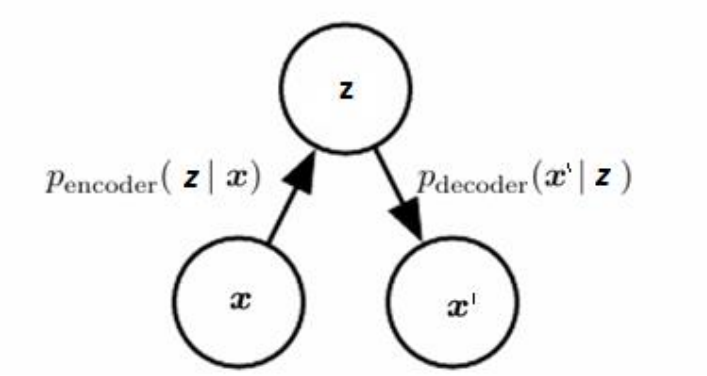


Stochastic autoencoder

Stochastic autoencoder are a generalization from the encoding function f(x) to an encoding distribution $p_{encoder}(z|x)$ and from the decoding function g(z) to a decoding distribution $p_{decoder}(x|z)$.

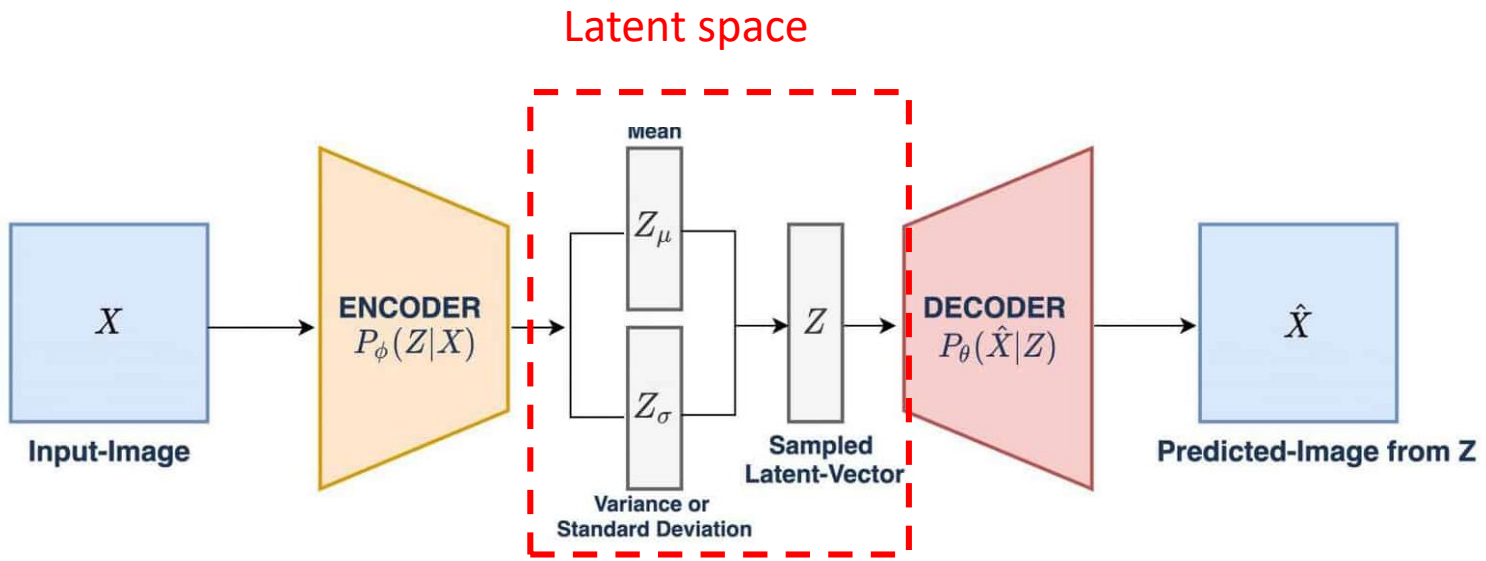
Any latent variable model $P_{model}(z, x)$ is defined by a:

- Stochastic Encoder: $P(z|x) = p_{model}(z|x)$
- Stochastic Decoder: $P(x|z) = p_{model}(x|z)$



Variational Autoencoders

The VAE is a parameterized model comprising an encoder and a decoder. The encoder learns the function $p_{encoder}\ (z|x)$ obtaining an estimate for its parameters, this is also called the inference model or recognition network. The decoder learns the function $p_{decoder}\ (x|z)$ that corresponds to a generative model. The latent space is structured to follow a Gaussian distribution $p_{model}(z)$ of mean zero and variance 1.



Latent space

The latent space z in a Variational Autoencoder (VAE) represents a **multitude of individual Gaussian distributions**, one for each data point.

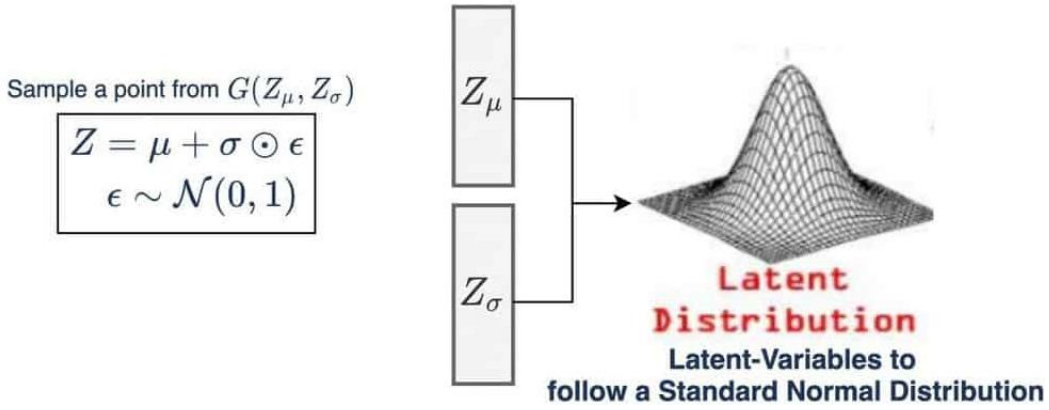
For each input data point, the encoder generates a distinct Gaussian distribution in the latent space. This is parameterized by:

Mean (μ): A unique mean vector for the latent variables of the input.

Variance (σ^2): A unique variance vector representing uncertainty.

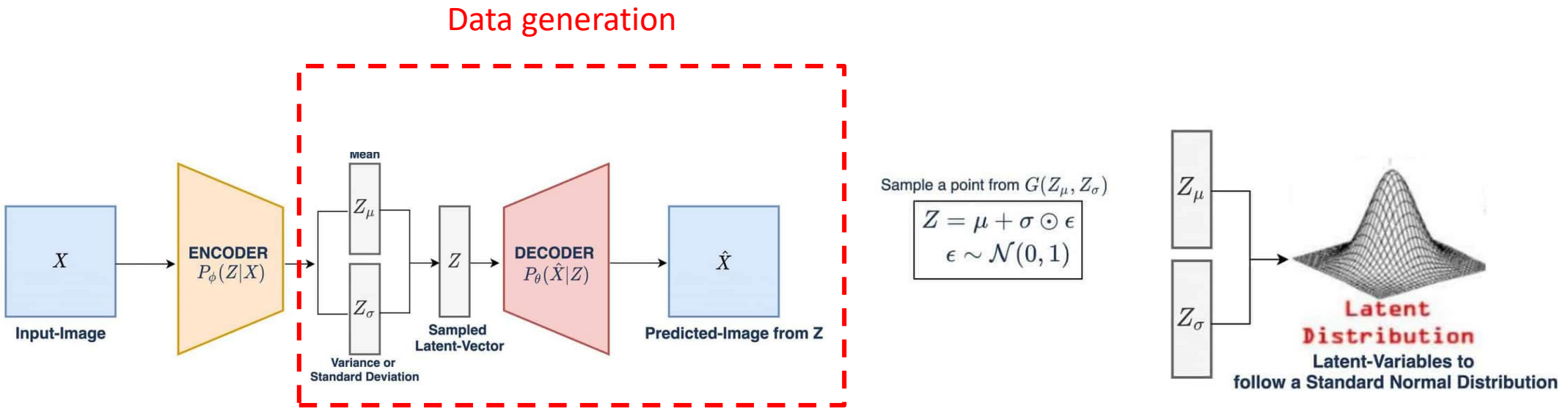
These individual distributions are learned by the model to describe the variability in the data.

After training the decoder is kept to obtain new data points processing samples from a prior distribution $p_{model}(z)$.

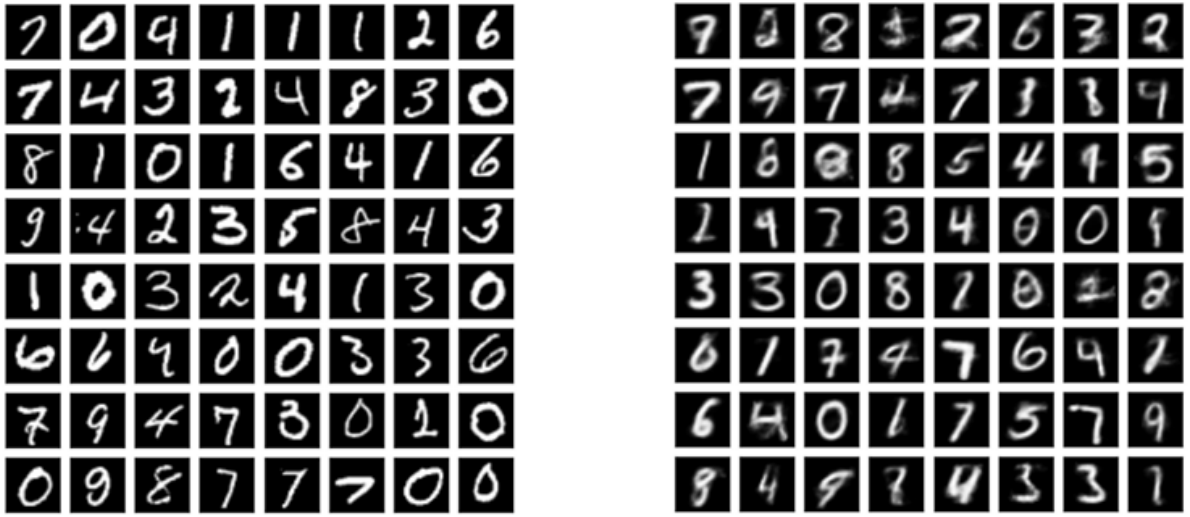


Generating new samples

To generate a sample from the model, the VAE first draws a sample *z* from the code distribution $p_{model}(z)$. The sample is then run through a differentiable generator network $p_{decoder}\,(x|z)$ to generate a new data point x.



Generation of images



(a) Sample from the original MNIST dataset.

(b) VAE generated MNIST images.

Fig. 4: Generated images of from a variational autoencoder, trained on the MNIST dataset with a prior $p_\theta(\mathbf{z}) = \mathcal{N}(0, \mathcal{I})$. Left: original images from the dataset. Right: generated images.

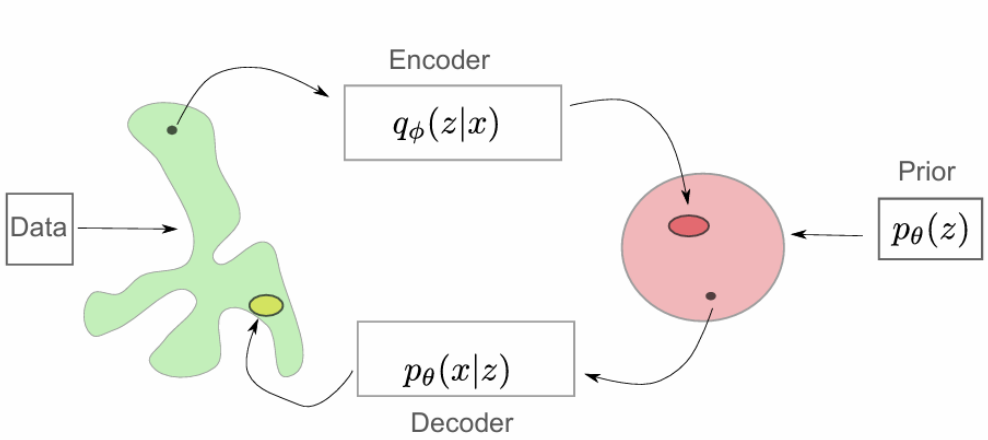
Variational autoencoder

VAEs describe data generation with probabilistic distributions.

Given an observed dataset $X = \{x_i\}_{i=1}^{N}$ assume a generative model for each datum x_i conditioned on a un unobserved random latent variable z_i, where θ are the parameters governing the generative distribution.

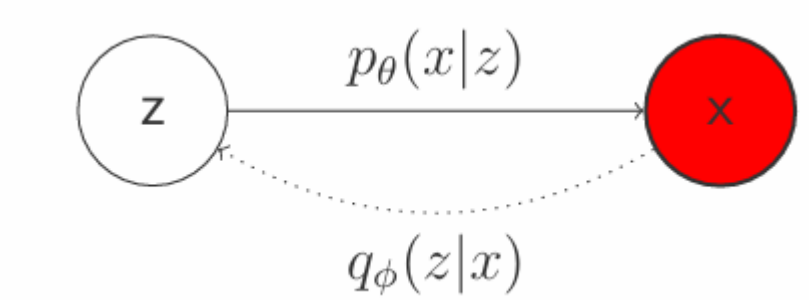
Simetrically, assume an approximate posterior distribution of the latent variable z_i given the datum x_i , learned by an encoder and governed by the parameters ϕ.

Prior distribution of the latent variable z_i denoted by $p_\Theta(z)$.



Implementing VAEs

The goal is to have a probability distribution that is able to map from the original data space to the latent space and back:



Three key ideas in the VAE:

- Use of the **evidence lower bound (ELBO)** to approximate the likelihood function $p(z|x)$, leading to a close relationship to the Expectation Maximization (EM) algorithm.
- **Amortized inference** in which a second model, the encoder network, is used to approximate the posterior distributions over latent variables in the Encoder step, rather than evaluating the posterior distribution for each data point exactly.
- Making the training of the encoder model tractable using the **reparameterization trick**.

Latent variable model

An approach to approximate $p(z|x)$ is variational inference.

Variational inference substitutes an intractable probability distribution *p* by another well-behaved distribution *q*.

The goal is to transform the distribution *q* so it is as close possible to *p* tuning its parameter ϕ.

Basically we need a distribution q_ϕ that is close to p_θ .

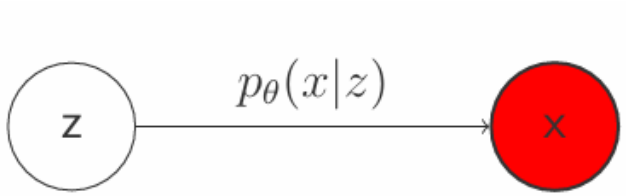
The closeness between two probability distributions can be measured using the Kullback-Leibler divergence.

$$KL(q||p) = -\sum_x q(x) \log \frac{p(x)}{q(x)} = \sum_x q(x) \log \frac{q(x)}{p(x)}$$

Latent variable model (LVM)

LVMs define the relationship among latent and observed variables as probabilistic models.

Latent variable models assume that there are a set of hidden variables that generate the observed variable.



Calculate $p(z|x) = \frac{p(x|z)\,p(z)}{p(X)}$ implies computing $P(x) = \int P(x|z)P(z)dz$

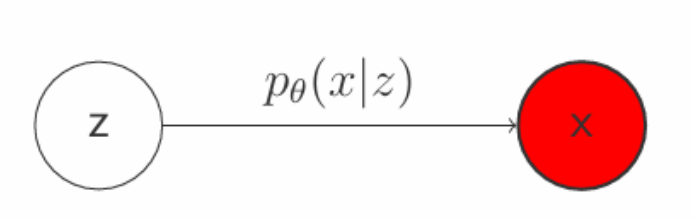
Calculate p(x) is usually intractable, because it requires integrating over all z, so that calculating analytically the posterior $p(z|x)$ is hard.

An approach to approximate $p(z|x)$ is **Variational Inference**.

Variational inference substitutes an intractable probability distribution p by another well-behaved distribution q.

While z implies x in the generative model, we need to infer z given x during training and data processing to learn the latent structure.

Latent variable model



P(z|x): The posterior (how likely z is given the data x).

P(x|z): The likelihood (how likely the data x is for a given z).

P(z): The prior on the latent variable z.

P(x): The marginal likelihood or evidence, defined as: *P(x)=* $\int P(x|z)P(z)dz$

The problem is that P(x) involves a high-dimensional integral over all possible values of z

Evidence Lower Bound (ELBO)

Choose a distribution $q_\phi(z|x)$ as a substitute to $p_\theta(z|x)$.

In order to choose the parameters q_ϕ we need to minimize the KL Divergence.

$$D_{KL}(q_\phi(z|x)|\ p_\theta(z|x)\ ) = \sum_{z\sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(x,z)} + \log\ p_\theta(x)$$

$\log\ p_\theta(x)$ is the likelihood of the distribution of samples, rewritten as:

$$\log\ p_\theta(x) = D_{KL}(q_\phi(z|x)|\ p_\theta(z|x)\ ) - \sum_{\sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(x,z)}$$

The second term is called the Variational Lower Bound or **Evidence Lower Bound** (ELBO)

The KL Divergence is always larger or equal to 0, so that the ELBO is a lower bound of the probability distribution p(x).

Evidence Lower Bound (ELBO)

The ELBO can be expressed as the sum of the **reconstruction term** (log likelihood) and the negative of the **regularization term** (KL divergence).

$$
\begin{aligned}
\sum_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} &= \sum_z q_\phi(z|x) \log \frac{p_\theta(x|z)p_\theta(z)}{q_\phi(z|x)} \\
&= \sum_z q_\phi(z|x) \log p_\theta(x|z) \frac{p_\theta(z)}{q_\phi(z|x)} \\
&= \sum_z q_\phi(z|x) \left[\log p_\theta(x|z) + \log \frac{p_\theta(z)}{q_\phi(z|x)} \right] \\
&= \sum_z q_\phi(z|x) \log p_\theta(x|z) + \sum_z q_\phi(z|x) \log \frac{p_\theta(z)}{q_\phi(z|x)} \\
&= \sum_z q_\phi(z|x) \log p_\theta(x|z) - KL(q_\phi(z|x)||p_\theta(z))
\end{aligned}
$$

Maximizing ELBO

Maximizing the Evidence Lower Bound (ELBO) allows to learn the parameters of a latent variable model by optimizing two critical aspects: the likelihood of the observed data and the structure of the latent space.

$$\text{ELBO} = \mathbb{E}q\phi(z|x)[\log p_\theta(x|z)] - KL(q_\phi(z|x)\|p_\theta(z))$$

Reconstruction term: This is the log-likelihood of the data, averaged over the posterior $q_\phi(z|x)$, which is the encoder's approximation of the true latent distribution. It measures how well the model reconstructs the observed data *x* from the latent representation *z*.

Amortized inference refers to the use of a neural network (encoder) with learnable parameters ϕ to approximate the posterior $q_\phi(z|x)$.

Latent space structure

Similar digits in the MNIST dataset are clustered together in a 2D latent space representation.

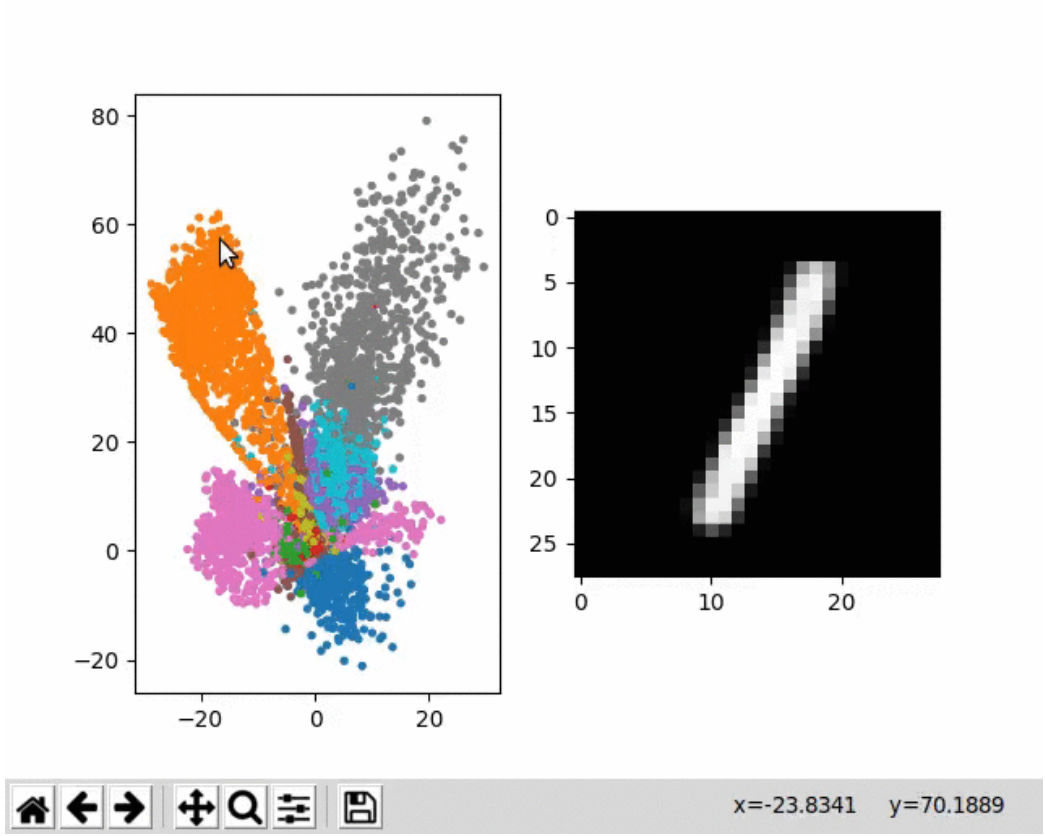
Groups inputs with similar features close together in the latent space.

Applications of clustering in latent space:

Data retrieval: Items similar to a query can be found by exploring its neighborhood in latent space.

Feature extraction: Understand the latent representation helps identify the most significant characteristics of data.

Anomaly Detection: Outliers in latent space can signal unusual inputs.



Disentangled autoencoders

Beta – VAEs: Framework for automated discovery of interpretable factorized latent representations.

Add a multiplicative factor β to the KL-Divergence term of the ELBO.

$$\text{ELBO} = \mathbb{E}q\phi(z|x)[\log p_\theta(x|z)] - \beta KL(q_\phi(z|x)\|p_\theta(z))$$

Effect of β:

When β = 1, the model behaves like a standard VAE.

When β > 1, the model emphasizes disentangling the latent space at the cost of reconstruction accuracy.

Purpose: By encouraging disentanglement, Beta-VAE learns latent representations where each dimension corresponds to an independent generative factor (e.g., object shape, size, or color in an image dataset).

Latent factors

With $\beta > 1$ the model learns a more efficient latent representation of the data, which is disentangled if the data contains at least some underlying factors of variation that are independent.

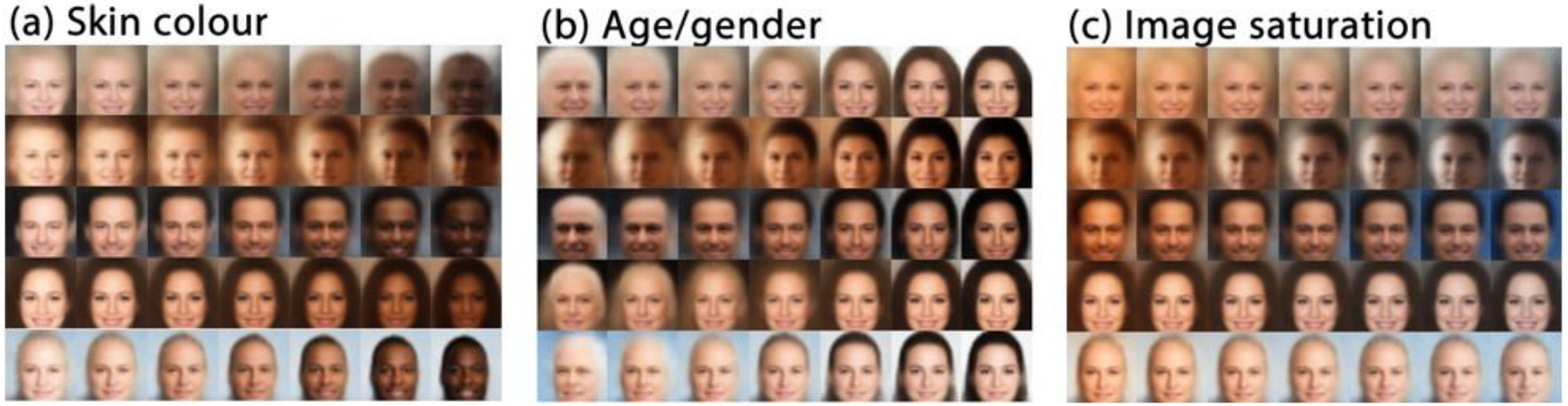


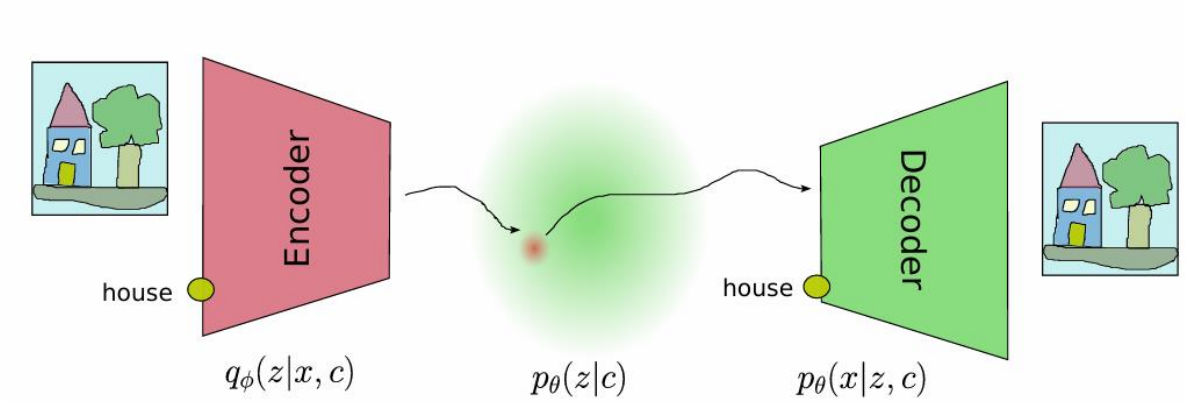
Figure 4: **Latent factors learnt by β-VAE on celebA:** traversal of individual latents demonstrates that β-VAE discovered in an unsupervised manner factors that encode skin colour, transition from an elderly male to younger female, and image saturation.

Conditioned VAEs

Conditioned Variational Autoencoders (CVAEs) are an extension of Variational Autoencoders (VAEs) designed to model conditional distributions.

In CVAEs, the encoder and decoder are conditioned on auxiliary information, such as a class label, attribute, or another form of metadata related to the data.

$$L_{\theta,\phi}(x,c) = \mathbb{E}z \sim q\phi(z|x,c)[\log p_\theta(x|z,c)] - \mathrm{KL}(q_\phi(z|x,c)\|p_\theta(z|c))$$



Tradeoff Reconstruction - KL term

The tradeoff between the **reconstruction term** and the **KL divergence** in a Variational Autoencoder (VAE) can be interpreted as balancing **data reconstruction fidelity** and **latent space regularization**:

Reconstruction Term:

Ensures that the model reconstructs the input *x* accurately.

A strong focus on this term prioritizes detailed reconstructions, potentially sacrificing a well-structured latent space.

KL Divergence:

Emphasizing KL divergence encourages disentanglement and smoothness in the latent space but may reduce reconstruction quality if over-optimized.

Posterior collapse

Posterior collapse is a common issue in Variational Autoencoders (VAEs), where the approximate posterior $q_\phi(z|x)$ learned by the encoder becomes overly simplistic and collapses to the prior $p_\theta(z)$, regardless of the input data x.

Causes of Posterior Collapse

KL Divergence Dominance:

- Excessive emphasis on minimizing KL divergence in the loss function forces $q_\phi(z|x)$ to resemble the prior $p_\theta(z)$, regardless of *x*.

Overpowering Decoder:

- A strong decoder can learn to reconstruct *x* directly, making the latent representation z unnecessary.

Latent Space Dimension Mismatch:

- If the latent space is too small or not expressive enough, fails to encode enough information about *x*.

Summary

Embeddings are widely used in ML applications to capture the identity and relationship between objects of different types.

Autoencoders are effective for feature learning and dimensionality reduction from unlabeled data.

Different types of autoencoders: Undercomplete, regularized and denoising variants.

Variational autoencoders learn a probability distribution of the data and can be used for data generation.

Training of VAE is balancing the data reconstruction fidelity and latent space regularization.

Variants of VAE such as beta-VAEs and conditional VAEs allow to understand and control data generation (at least to some extent).

Thank you for your attention!

Contact:

Caroline.leonore.konig@upc.edu

References

[1] Do, J. S., Kareem, A. B., & Hur, J. W. (2023). Lstm-autoencoder for vibration anomaly detection in vertical carousel storage and retrieval system (vcsrs). *Sensors*, *23*(2), 1009.

[2] Aslam, N., & Kolekar, M. H. (2022). Unsupervised anomalous event detection in videos using spatio-temporal inter-fused autoencoder. *Multimedia Tools and Applications*, *81*(29), 42457-42482.

[3] Bishop, C. M., & Bishop, H. (2023). *Deep learning: Foundations and concepts*. Springer Nature.

[4] Bengio, Y., Goodfellow, I., & Courville, A. (2017). *Deep learning* (Vol. 1, pp. 23-24). Cambridge, MA, USA: MIT press.

[5] Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), 307-392.

[6] Dubois, Y., Kastanos, A., Lines, D., & Melman, B. Understanding Disentangling in VAE.

[7] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., ... & Lerchner, A. (2017, February). beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*.