Laurea Magistrale in Informatica
Dipartimento di Informatica
Università di **To**rino

Autoencoders

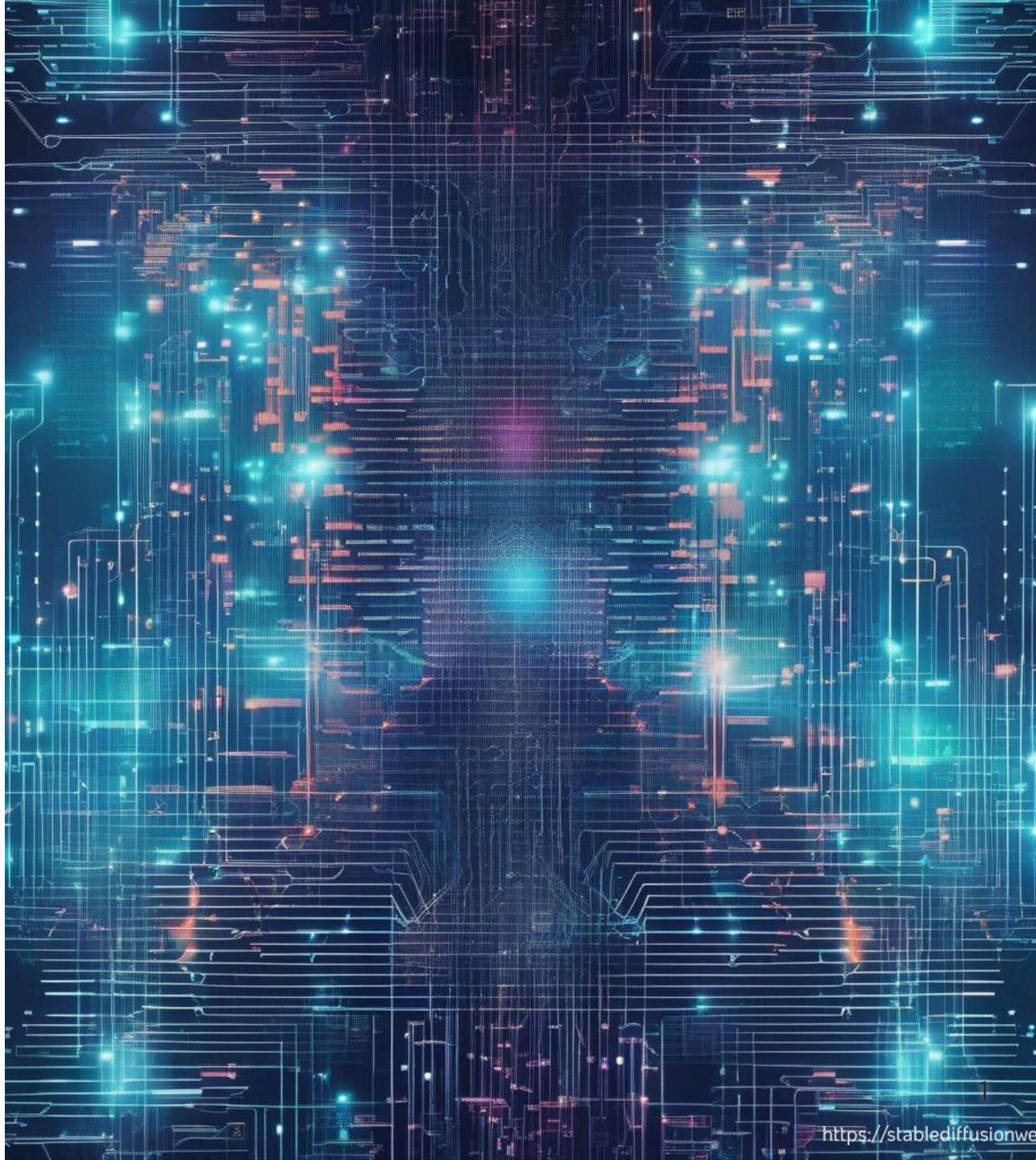
Course

Neural Networks and Deep Learning

Professor

Roberto Esposito
roberto.esposito@unito.it

Image dreamed by stable diffusion
Prompt: "abstract image about neural network autencoders"



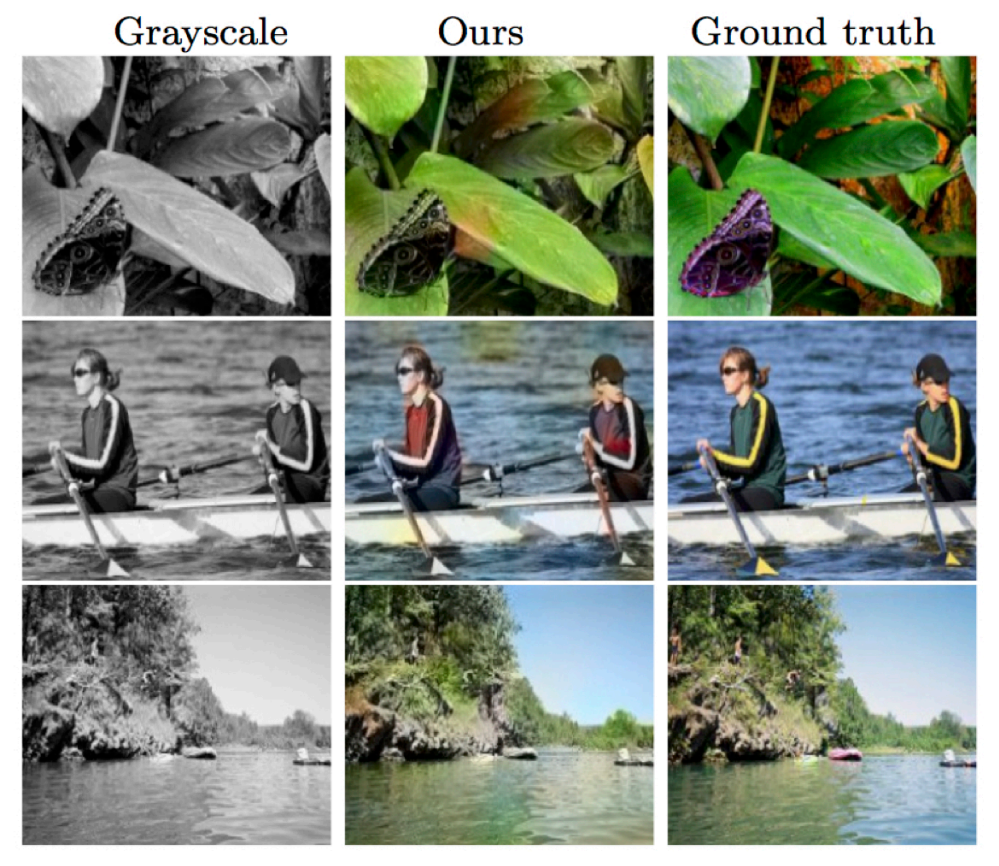
Applications

Autoencoders (AE) are unsupervised learning tools used to **improve supervised networks** (e.g., by using learned AE representation layers to initialize the supervised network), and to **solve many interesting tasks**:

- Image Colorization
- Increase resolution
- Image inpainting
- Machine Translation
- ...

Colorize a b/w image

Image from *Deep Koalarization: Image Colorization using CNNs and Inception-Resnet-v2*. [2]



Superresolution

Images from *Coupled deep autoencoder for single image super-resolution*. [3]

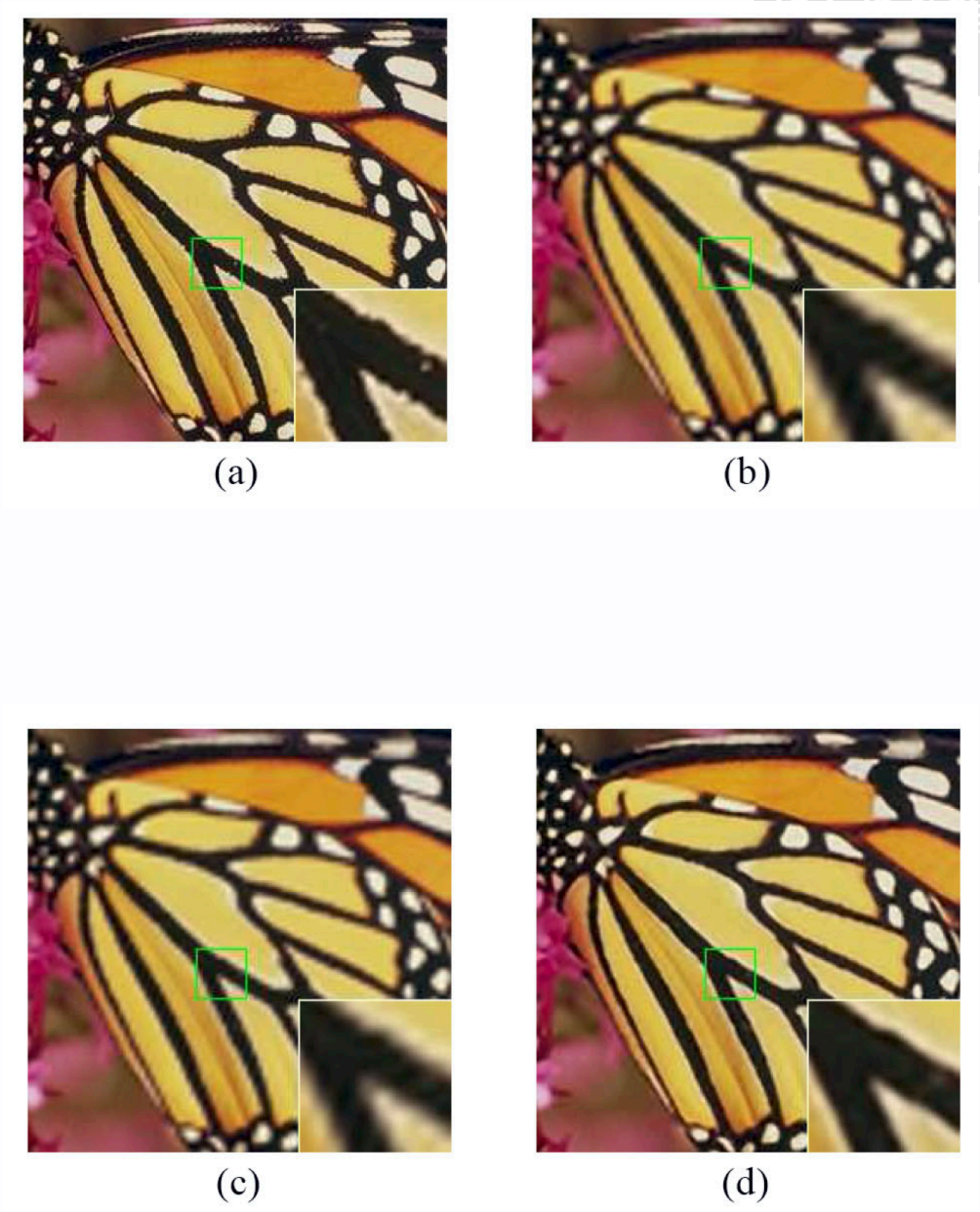


Image Inpainting

Images from *Image restoration using convolutional auto-encoders with symmetric skip connections.* [4]

Definition

An autoencoder is a netural network that is trained to attempt to copy its input to its output via a representation $\mathbf{h}$ built by its hidden layers.

- Encoder: $\mathbf{h} = f(\mathbf{x})$
- Decoder: $\mathbf{r} = g(\mathbf{h})$

An autoencoder is not expected to faithfully copy every input to its output. Instead, they are forced to prioritize which aspects of the input should be preserved.

Traditionally Autoencoders have been used for dimensionality reduction or feature learning.

Today, they are at the forefront of the research on **generative models** due to connections established with **latent variable models**.

> ***Note***
>
> *A latent variable model is a model that has both observed variables* $\mathbf{x}$ *and latent variables* $\mathbf{h}$*, and the goal is to learn the distribution* $p(\mathbf{x}, \mathbf{h})$*. We briefly introduced these ideas at the end of the lectures about representation learning.*

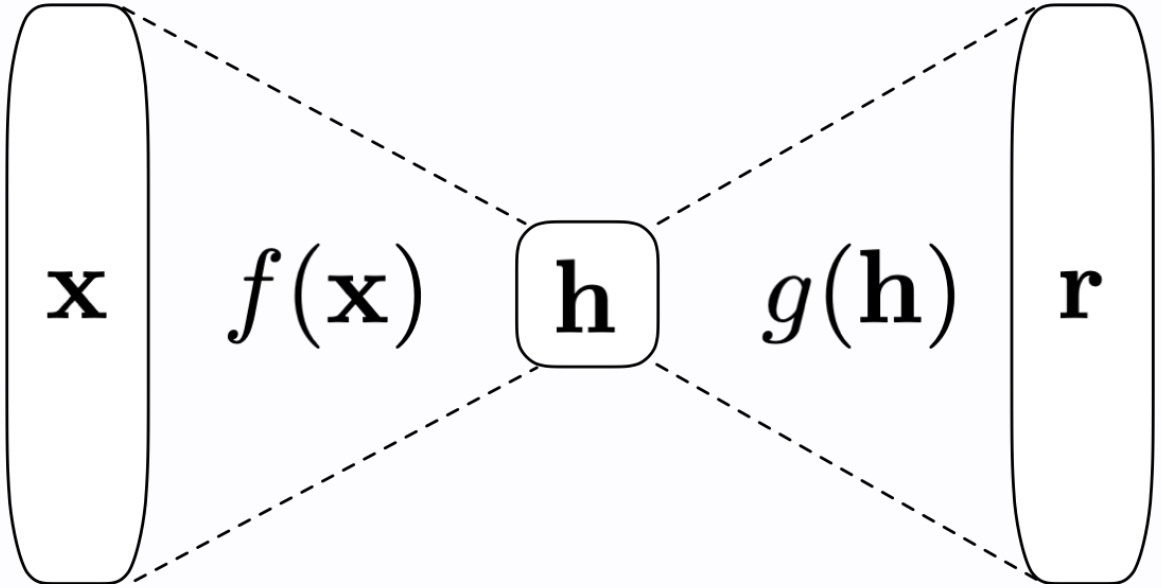
Undercomplete Autoencoders

Goal: only copying the input to the output is not useful, instead we hope that training the autoencoders will result in $\mathbf{h}$ taking on useful properties.

One way to do that is by forcing $\mathbf{h}$ to have a smaller dimension than $\mathbf{x}$.

Definition

An autoencoder whose code dimension is less than the input dimension is called ***Undercomplete Autoencoder****.*



Undercomplete Autoencoders

The learning process for an Undercomplete Autoencoders is usually the minimization of a loss function:

$$L(\mathbf{x}, g(f(\mathbf{x})))$$

where L is a loss function penalizing $g(f(\mathbf{x}))$ for being dissimilar from $\mathbf{x}$ (e.g., the mean squared error).

Autoencoders and PCA

When the decoder is linear and L is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA.

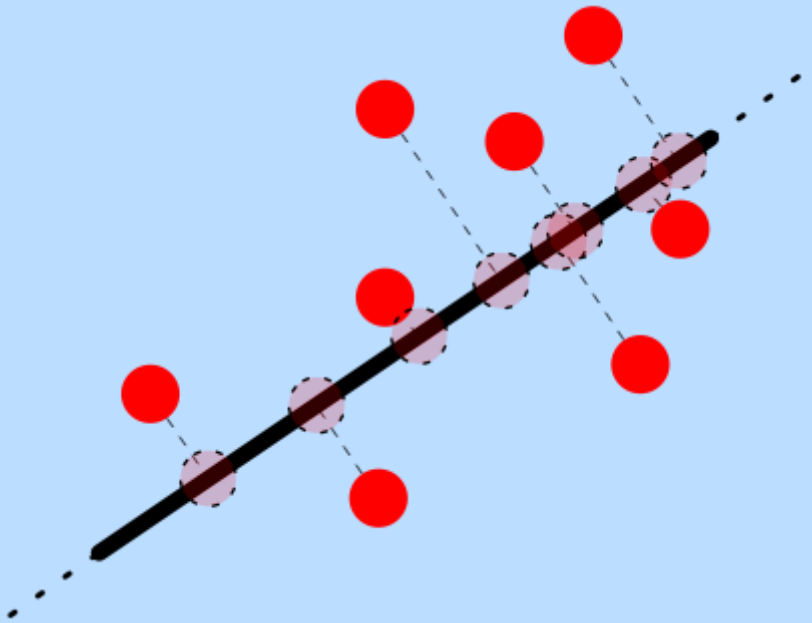
As a side-effect of the learning-to-copy process, the Autoencoder learns the principal subspace of the training data.

Summary of PCA

PCA can be derived as an algorithm to **project each example** x onto a **lower dimensional vector space** with minimal reconstruction error, i.e., for PCA:

$$f(\mathbf{x}) = \arg\min_{\mathbf{h}} \|\mathbf{x} - g(\mathbf{h})\|_2$$

where g is a reconstruction function subject to **few constraints**.



Specifically, g needs to be such that:

- g is linear, i.e., $g(\mathbf{h}) = \mathbf{D}\mathbf{h}$ for some $\mathbf{D}$;
- columns of $\mathbf{D}$ are orthogonal to each other (to simplify the problem);
- columns of $\mathbf{D}$ have unit norm (to make the solution unique).

Note

The proof of this result is beyond the scope of this course. See page 46 of [1] for details.

Under these constraints it can be show that:

- $f(\mathbf{x}) = \arg\min_{\mathbf{h}} \|\mathbf{x} - g(\mathbf{h})\|_2 = \mathbf{D}^T\mathbf{x} = \mathbf{h}_\mathbf{x},$
- $g(\mathbf{h}) = \mathbf{D}\mathbf{h}$

yielding:

- $r(\mathbf{x}) = g(f(\mathbf{x})) = g(\mathbf{h}_\mathbf{x}) = \mathbf{D}\mathbf{D}^T\mathbf{x},$

with: $\mathbf{D}$ **given by the eigenvectors** corresponding to the **largest eigenvalues** of the matrix $\mathbf{X}^T\mathbf{X}$.

Autoencoders and PCA

In summary:

- an undercomplete Autoencoder minimizes a loss $L(\mathbf{x}, g(f(\mathbf{x})))$
- PCA finds the mapping $f(\mathbf{x}) = \arg\min_{\mathbf{h}} \|\mathbf{x} - g(\mathbf{h})\|_2$, imposing that g is a linear model.

It should be apparent that when $L = \|\cdot\|_2$ and the reconstruction layer is modeled by linear units, we have two approaches to solve the same problem.

Question

Assume that the data to be reconstructed lives on a linear manyfold.

Which approach is best and why?

What if the manyfold is highly non-linear?

Regularized Autoencoders

The difference in size between the code $\mathbf{h}$ and the input $\mathbf{x}$ determines how much the autoencoder is forced to learn only the *most relevant* part to the input.

If the size of $\mathbf{h}$ is too big, the autoencoder could simply learn the identity function.

Question

It easy to see that if the size of $\mathbf{h}$ is very large, the autoencoder can just learn the identity function.

What about when the size of $\mathbf{h}$ is 1?

Indeed if the autoencoder has too much capacity (either because the encoder and the decoder are too powerful, or because the code is too large), it will not be able to learn anything useful.

We will be now looking into **regularized autoencoders**: rather than limiting the model capacity by keeping the encoder and decoder shallow and the size of the hidden layers small, **one uses a loss function that encourages the model to have other properties** besides the ability to copy its input to its output.

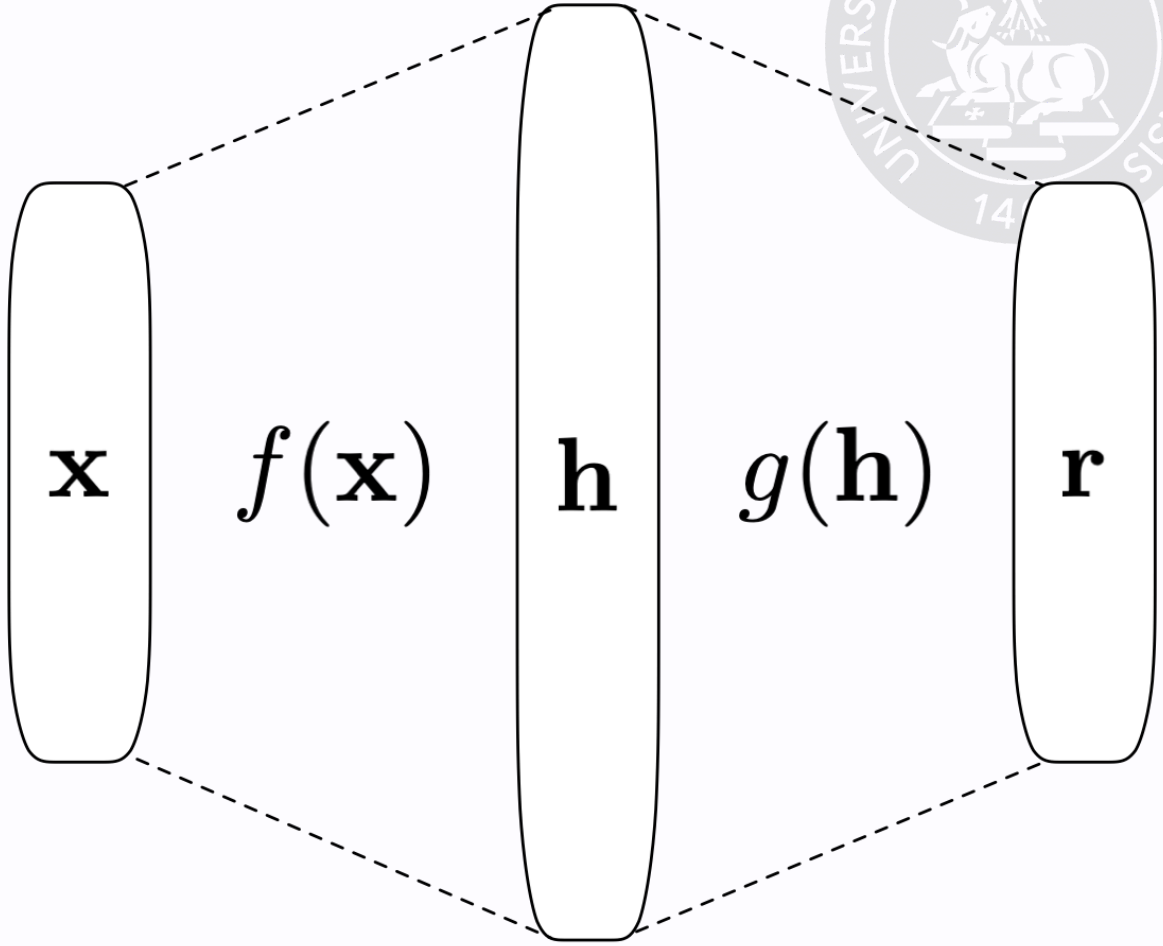
Regularized Autoencoders

Properties exploited for regularization:

- Sparsity of the representation $\Rightarrow$ Sparse autoencoders
- robustness to noise or missing inputs $\Rightarrow$ Denoising autoencoders
- smallness of the derivative $\Rightarrow$ Contractive autoencoders

Definition

*A **sparse autoencoder** is simply an autoencoder whose training criterion involves a sparsity penalty $\Omega(\mathbf{h})$ on the code layer $\mathbf{h}$, in addition to the reconstruction error.*



$$L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h})$$

Generative Models (GM)

A GM is a model that learns the joint distribution $p(\mathbf{x}, \mathbf{y})$ *of the data. After the model is trained, it can generate new data points of a given class by sampling from it. Sometimes* $\mathbf{y}$ *is not an observable variable. In such cases,* $\mathbf{y}$ *is called a* ***latent variable*** *and usually denoted by* $\mathbf{h}$.

GMs differ from a ***discriminative model****, since the latter learn to approximate the conditional probability* $p(\mathbf{y}|\mathbf{x})$ *and cannot therefore generate new data points.*

Sparse Autoencoders as GM

Suppose we have a GM with visible variables $\mathbf{x}$ and latent variables $\mathbf{h}$, with a distribution:

$$p_{\text{model}}(\mathbf{h}, \mathbf{x}) = p_{\text{model}}(\mathbf{h}) p_{\text{model}}(\mathbf{x}|\mathbf{h})$$

where $p_{\text{model}}(\mathbf{h})$ is the model's **a-priori** distribution over the latent variables.

Under this assumptions, the **likelihood of $\mathbf{x}$ given the model** can be computed as the marginalization of the latent variables $\mathbf{h}$:

$$p_{\text{model}}(\mathbf{x}) = \sum_{\mathbf{h}} p_{\text{model}}(\mathbf{h}, \mathbf{x})$$

Important

$p_{\text{model}}(\mathbf{h})$ *is a* ***different prior*** *than the one usually assumed in probabilistic models.*

Usually the term "prior" is the preference of a learning algorithm for a given solution (i.e., for a given model) prior of seeing the data.

Here we are using this term to refer to the ***preference of the model for latent representation*** $\mathbf{h}$ ***prior of seeing*** $\mathbf{x}$.

By taking the logarithm of $p_{\text{model}}(\mathbf{x})$ we obtain the log-likelihood of $\mathbf{x}$ under the model:

$$\log p_{\text{model}}(\mathbf{x}) = \log \sum_{\mathbf{h}} p_{\text{model}}(\mathbf{h}, \mathbf{x}).$$

We can think of the autoencoder as **approximating this sum with a point estimate for just one highly likely value for $\mathbf{h}$**.

Then, given an $\widetilde{\mathbf{h}}$ generated by the encoder:

$$\log p_{\text{model}}(\mathbf{x}) = \log \sum_{\mathbf{h}} p_{\text{model}}(\mathbf{h}, \mathbf{x}) \approx \log p(\widetilde{\mathbf{h}}, \mathbf{x}) = \log p_{\text{model}}(\widetilde{\mathbf{h}}) + \log p_{\text{model}}(\mathbf{x}|\widetilde{\mathbf{h}}).$$

If we now recall that **sparse autoencoders are trained to minimize the loss**:

$$\overbrace{L(\mathbf{x}, g(f(\mathbf{x})))}^{\text{reconstruction loss}} + \overbrace{\Omega(\mathbf{h})}^{\text{sparsity loss}},$$

we can see that, by minimizing **the reconstruction loss,** the autoencoder is learning to approximate the **log-likelihood of the data** under the model.

In fact, by minimizing the reconstruction error, the autoencoder is learning to approximate the conditional distribution $p_{\text{model}}(\mathbf{x}|\mathbf{h})$ since it is trying to find the $\mathbf{x}$ that is the best reconstruction given $\mathbf{h}$, i.e., it is actually solving:

$$\arg\max_{\mathbf{x}} p_{\text{model}}(\mathbf{x}|\mathbf{h}) = \arg\min_{\mathbf{x}} -\log p_{\text{model}}(\mathbf{x}|\mathbf{h})$$

If one sets $\Omega(\mathbf{h}) = \lambda \sum_i |h_i|$ (i.e., using the L_1 norm of $\mathbf{h}$, **which is known to encourage sparsity**), minimization of the sparsity term corresponds to maximizing the log-likelihood of the $p(\mathbf{h})$ term assuming a Laplace prior over each component of $\mathbf{h}$ independently. Formally, if

$$p_{\text{model}}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}$$

then

$$-\log p_{\text{model}}(\mathbf{h}) = \sum_i \left(\lambda |h_i| - \log \frac{\lambda}{2} \right) = \Omega(\mathbf{h}) + \text{const}$$

where λ can be treated as an hyper-parameter and the constant term, depending only on λ, is not optimized during learning and can be ignored.

Putting all together, training the network to minimize the reconstruction error and the sparsity penalty is equivalent to training the network to minimize the negative log-likelihood of the data under the model:

$$\min L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h}) \approx \min -\log p_{\text{model}}(\mathbf{x}).$$

From this point of view, the sparsity penalty is not a regularization term: it is just the consequence of modeling the joint distribution taking into account the latent variables and assuming them to have a Laplace prior.

This view provides different motivations for training a sparse autoencoder:

- it is a way of approximately training a generative model;
- learnt features are useful since they describe the latent variables that explain the input.

Designing output units

A general strategy for designing **the output units and the loss function** of a feedforward network is to interpret the network as computing a distribution $p(\mathbf{y}|\mathbf{x})$ and minimize the negative log-likelihood $-\log p(\mathbf{y}|\mathbf{x})$.

In the case of autoencoders, $\mathbf{x}$ is also the target as well as the input. However, we can use the same ideas.

We may see the decoder as providing the conditional distribution $p_{\text{decoder}}(\mathbf{x}|\mathbf{h})$.

Designing output units

In case $\mathbf{x}$ is real valued, **to adopt linear output units and a quadratic loss** corresponds to maximize $p_{\text{decoder}}(\mathbf{x}|\mathbf{h})$ under the assumption that the probability distributes as $\mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \mathbf{I})$, with $\hat{\mathbf{x}}$ being provided by the output linear layer: $\hat{\mathbf{x}} = \mathbf{W}\mathbf{h}$. Let's tart by recalling that the probability of a multivariate normal distribution, with parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, is given by:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

yielding:

$$p_{\text{decoder}}(\mathbf{x}|\mathbf{h}) = c_1 \exp(-(\mathbf{x}-\mathbf{W}\mathbf{h})^T\mathbf{I}(\mathbf{x}-\mathbf{W}\mathbf{h})) = c_1 \exp(-\|\mathbf{x}-\mathbf{W}\mathbf{h}\|^2)$$

implying that minimizing the quadratic loss is equivalent to maximizing the log-likelihood of the data under the model, since:

$$-\log p_{\text{decoder}}(\mathbf{x}|\mathbf{h}) = \|\mathbf{x}-\mathbf{W}\mathbf{h}\|^2 + c$$

Note: since linear units do not saturate, they do not pose any problem for gradient based optimization algorithms.

Designing output units

Similarly, if $\mathbf{x} \in \{0, 1\}^n$, using sigmoid output units and a cross-entropy loss corresponds to use Bernoulli distributions.

The idea is to assume that the individual outputs of the network $\hat{\mathbf{x}}_i = \sigma(z_i)$ give the parameters of the n independent Bernoulli distributions, i.e.:

$$p(\mathbf{x}_i|\mathbf{h}) = \hat{\mathbf{x}}_i^{\mathbf{x}_i}(1 - \hat{\mathbf{x}}_i)^{(1-\mathbf{x}_i)}.$$

By taking the negative $\log$ of the expression above we have:

$$-\log p(\mathbf{x}_i|\mathbf{h}) = -\left[\mathbf{x}_i \log(\hat{\mathbf{x}}_i) + (1 - \mathbf{x}_i)\log(1 - \hat{\mathbf{x}}_i)\right] = -\left[\mathbf{x}_i \log(\sigma(z_i)) + (1 - \mathbf{x}_i)\log(1 - \sigma(z_i))\right]$$

which corresponds to the binary cross entropy loss function.

Stochastic view

More in general, one can view both the encoder and the decoder as modeling some distribution.

In this view:

$$p_{\text{encoder}}(\mathbf{h}|\mathbf{x}) = p_{\text{model}}(\mathbf{h}|\mathbf{x})$$

and

$$p_{\text{decoder}}(\mathbf{x}|\mathbf{h}) = p_{\text{model}}(\mathbf{x}|\mathbf{h})$$

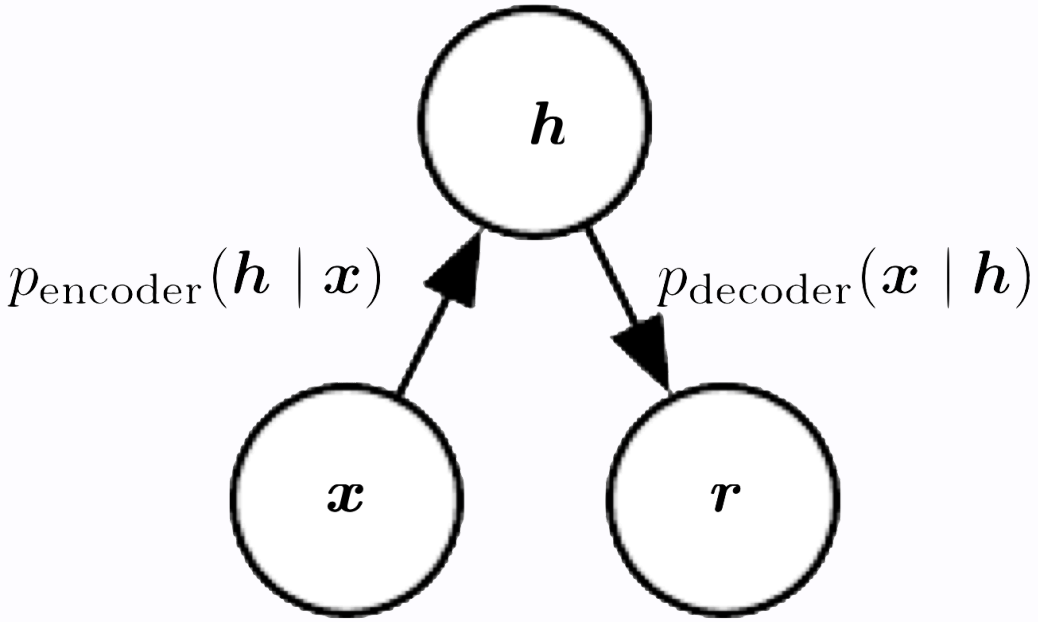


Image from [1] Chap. 14

Stocastic view

In general, **the encoder and decoder distributions are not necessarily conditional distributions compatible with a unique joint distribution** $p_{\text{model}}(\mathbf{x}, \mathbf{h})$.

It can be shown that training the encoder and decoder as a **denoising autoencoder** will tend to make them compatible asymptotically (with enough capacity and examples).

Regularized Autoencoders

Properties exploited for regularization:

- Sparsity of the representation $\Rightarrow$ Sparse autoencoders
- robustness to noise or missing inputs $\Rightarrow$ Denoising autoencoders
- smallness of the derivative $\Rightarrow$ Contractive autoencoders

Denoising Autoencoders

Denoising AutoEncoders (DAE) adopt a different strategy for learning something useful. Rather than adding a penalty Ω to the cost function, they changes the reconstruction error of the cost function. Specifically they minimize:

$$L(\mathbf{x}, g(f(\tilde{\mathbf{x}})))$$

where $\tilde{\mathbf{x}}$ is a copy of $\mathbf{x}$ that has been corrupted by some form of noise.

Denoising autoencoders must therefore undo this corruption rather than simply copying their input. It has been shown that **denoising training forces f and g to implictily learn the structure of $p_{\text{data}}(\mathbf{x})$**.

DAE Stochastic View

A way to undestand denoising autoencoders is to introduce a corruption process $C(\tilde{\mathbf{x}}|\mathbf{x})$ that produces corrupted samples $\tilde{\mathbf{x}}$ given a data sample $\mathbf{x}$.

The autoencoder then learns a **reconstruction distribution** $p_{\text{reconstruct}}(\mathbf{x}|\tilde{\mathbf{x}}) = p_{\text{decoder}}(\mathbf{x}|\mathbf{h}) = g(\mathbf{h})$ and $\mathbf{h} = f(\tilde{\mathbf{x}})$.

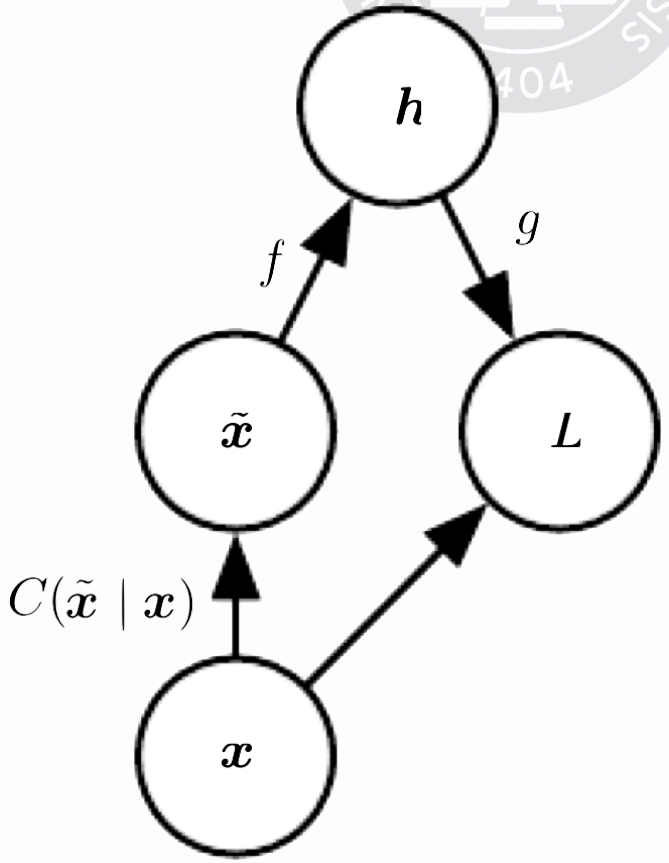


Image from [1] Chap. 14

DAE Stochastic View

Typically we can simply perform gradient based minimization (e.g., minibatch gradient descent) on the negative log likelihood $-\log p_{\text{decoder}}(\mathbf{x}|\mathbf{h})$.

So long as the encoder is deterministic, the whole autoencoder can be trained end-to-end using stochastic gradient descent and we can therefore view the denoising autoencoder as performing stochastic gradient descent on:

$$-\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\tilde{\mathbf{x}}\sim C(\tilde{\mathbf{x}}|\mathbf{x})}\log p_{\text{decoder}}(\mathbf{x}|\mathbf{h}=f(\tilde{\mathbf{x}}))$$

DAE as approximating a vector field

Autoencoder objective:

- $\min \|g(f(\tilde{\mathbf{x}})) - \mathbf{x}\|^2$

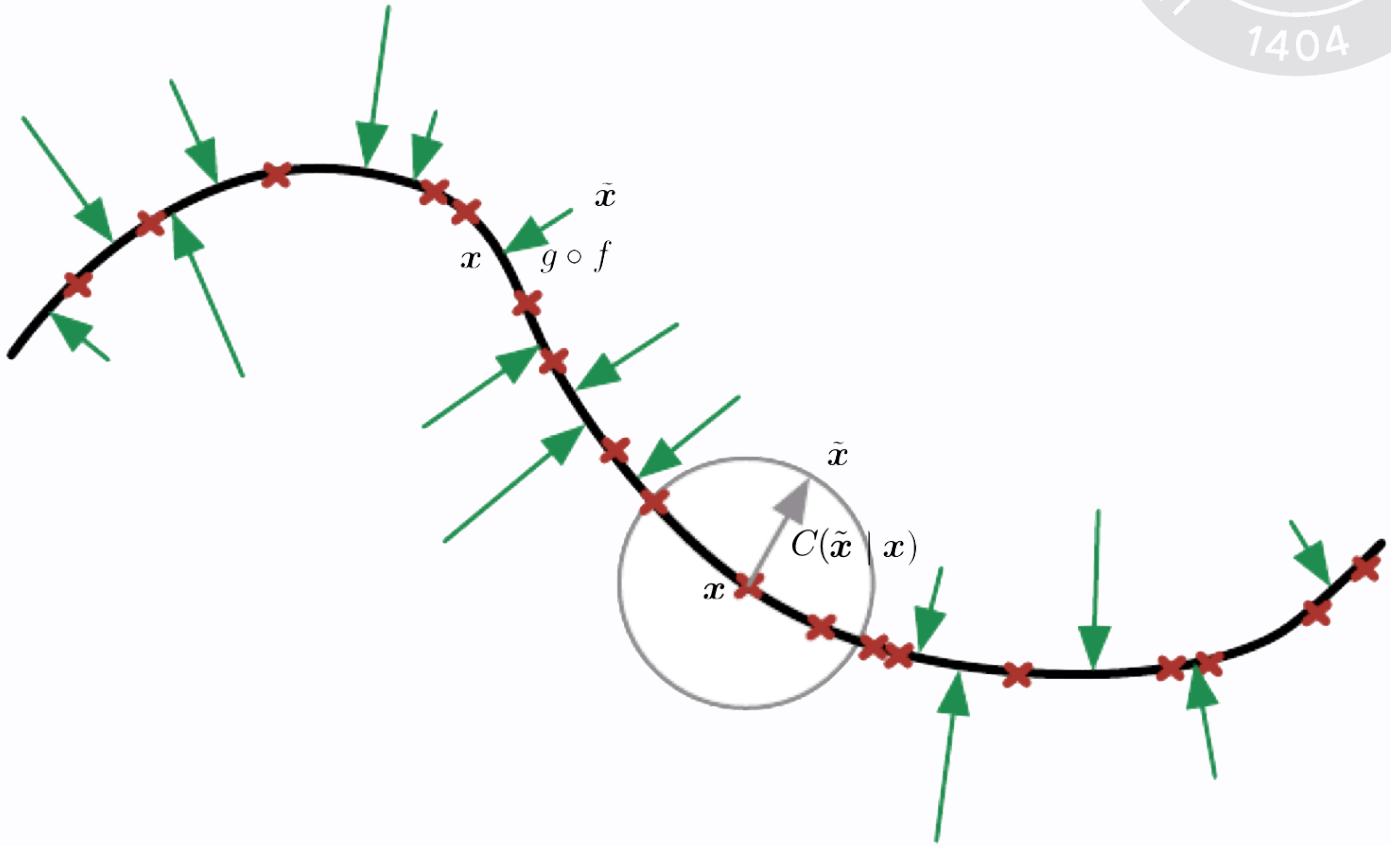


Image from the main text book. Chap. 14

DAE as approximating a vector field

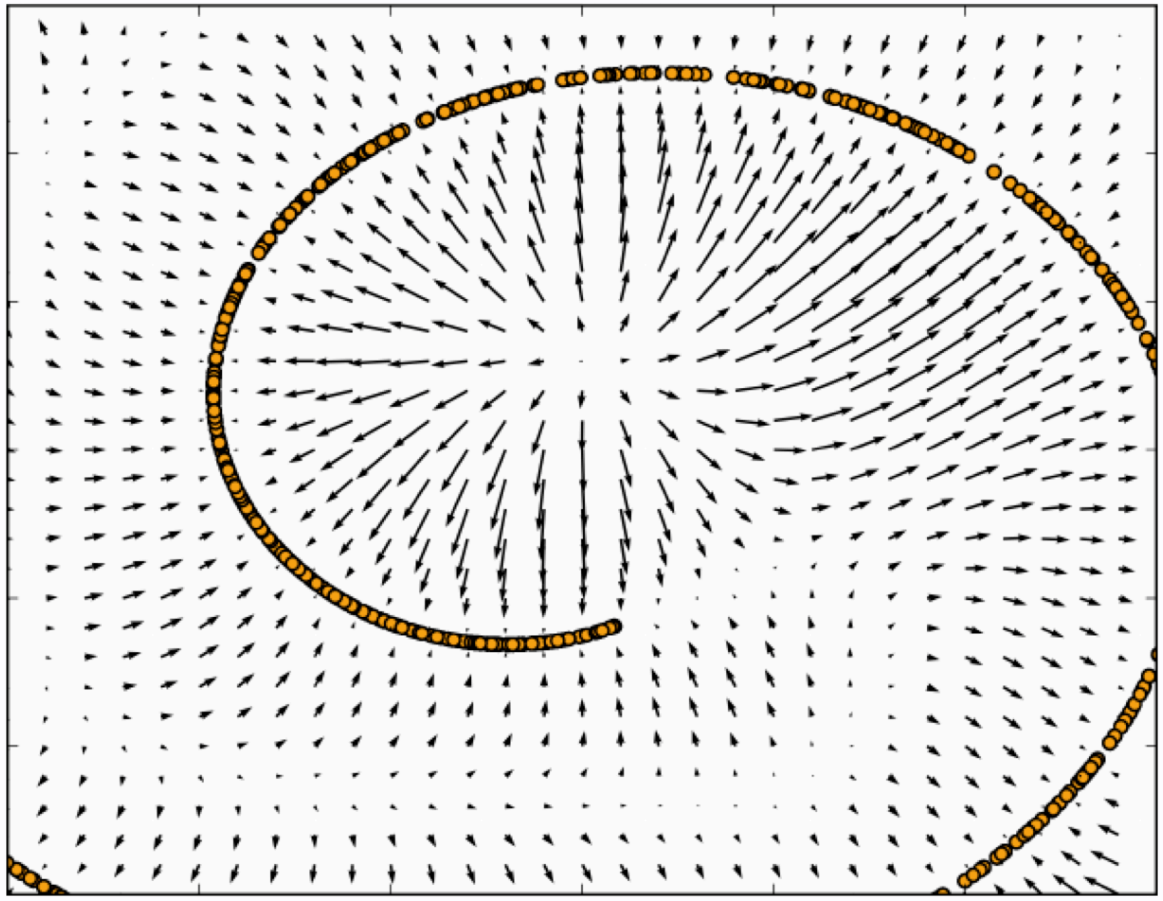
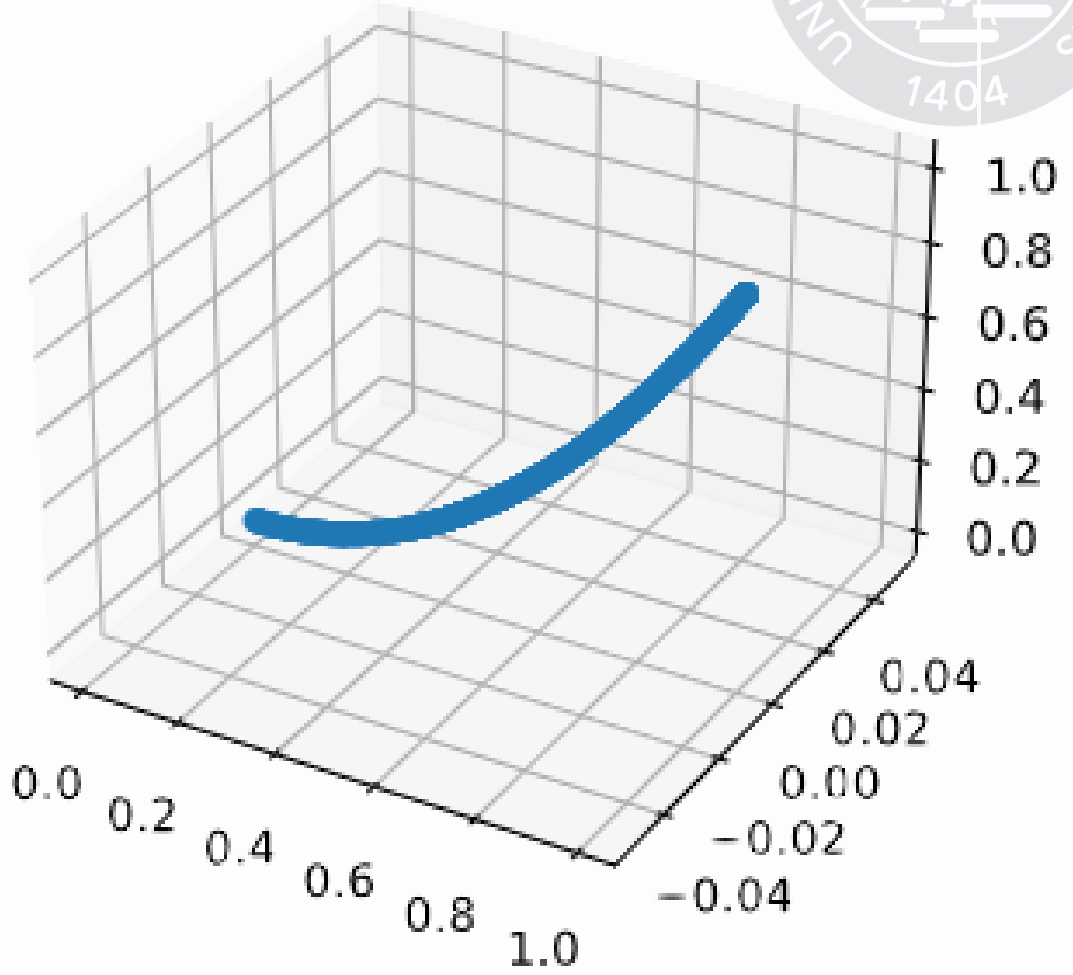


Image from [1] Chap. 14

Learning Manifolds

Many Machine Learning algorithms exploit the idea that data concentrates around low-dimensional manifolds (or a small set of such manifolds).



Learning Manifolds

Autoencoders take this idea further and aim to learn the structure of the manifold.

To understand how autoencoders do that it is necessary to introduce some properties of manifolds.

Manifolds tangent planes

An important characterization of a manifold is the set of its tangent planes.

At a point $\mathbf{x}$ on a d-dimensional manifold, the tangent plane is given by d basis vectors that span the local directions of variation allowed on the manifold.

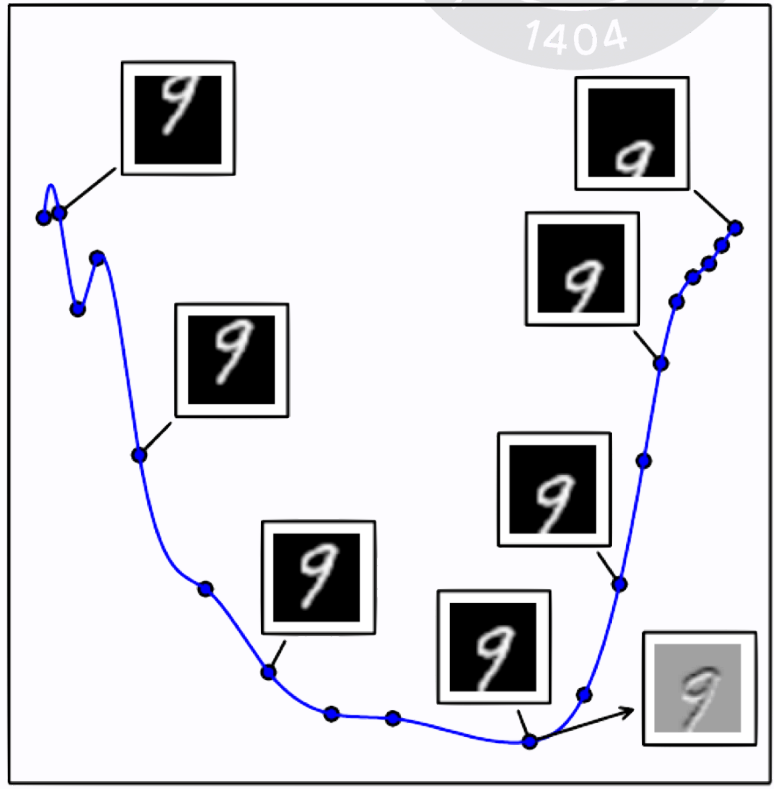
These local directions specify how one can change $\mathbf{x}$ infinitesimally while staying on the manifold.

Manifold tangent planes

The amount of vertical translation defines a coordinate along a one-dimensional manifold that traces out a curved path through image space.

The gray image shows how image changes when moving along the direction of the tangent plane on one of the points on the manifold.

Gray pixels indicate pixels that do not change as we move along the tangent line, white pixels indicate pixels that brighten, and black pixels indicate pixels that darken.



Autoencoders and Manifolds

All autoencoders training procedures involve a compromise between two forces:

1. Learning a representation $\mathbf{h}$ such that the input can be approximately recovered by the decoder.
2. Satisfying some regularity constraint.

The two forces together are useful because the autoencoder can only afford to represent the variations that are needed to reconstruct the example.

Autoencoders and Manifolds

If the data generating distribution concentrates near a low-dimensional manifold, this yields representations that implicitly capture a local coordinate system for this manifold: only the variations tangent to the manifold around $\mathbf{x}$ need to correspond to changes in $\mathbf{h} = f(\mathbf{x})$.

Hence the encoder learns a mapping from the input space $\mathbf{x}$ to a representation space, a mapping that is only sensitive to changes along the manifold directions, but that is insensitive to changes orthogonal to the manifold.

Regularized Autoencoders

Properties exploited for regularization:

- Sparsity of the representation $\Rightarrow$ Sparse autoencoders
- robustness to noise or missing inputs $\Rightarrow$ Denoising autoencoders
- smallness of the derivative $\Rightarrow$ Contractive autoencoders

Contractive Autoencoders

As in Sparse Autoencoders, Contractive Autoencoders (CAE) optimize a regularized objective $L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h}, \mathbf{x})$, but with a different form of Ω:

$$\Omega(\mathbf{h}, \mathbf{x}) = \lambda \sum_i \|\nabla_{\mathbf{x}} h_i\|^2$$

This forces the model to learn a function that does not change much when $\mathbf{x}$ changes only slightly.

Question

Which are the implications of a very high λ value?

Contractive Autoencoders

The very same regularization term can also be written as:

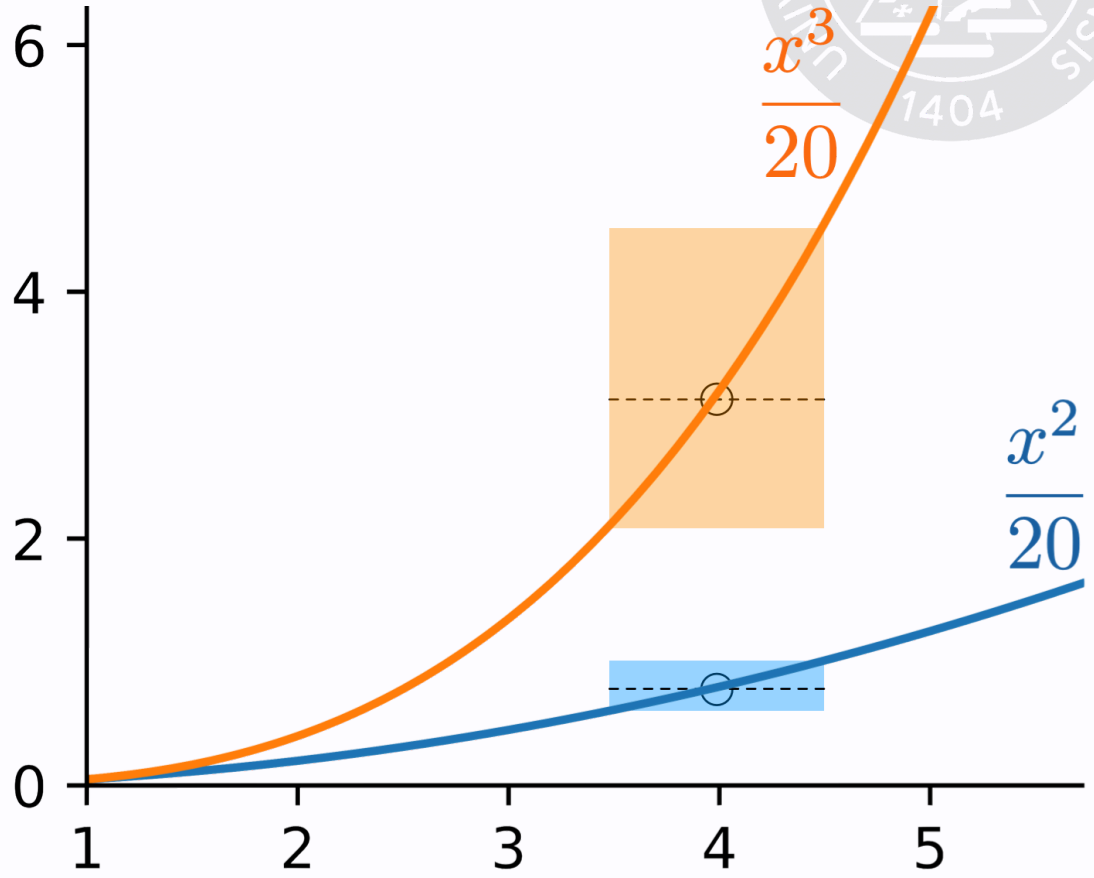
$$\Omega(\mathbf{h}) = \lambda \|\mathbf{J}\|_F^2 = \lambda \left\| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right\|_F^2$$

where $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{J}$ is the Jacobian matrix and:

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} |A_{i,j}|^2}$$

Why the "Contractive" name?

As a side effect of the contractive penalty, the encoder learns to map a neighborhood of inputs onto a smaller neighborhood of outputs.



Why the "Contractive" name?

To best understand this point, notice that without any other competing force (or with a very high λ) the penalty would drive f to be learnt as a constant function (which maps the whole input space onto a single point).

Note: the CAE is contractive only locally, two very different $\mathbf{x}$ and $\mathbf{x}'$ can be mapped to points that are farther apart than the original points.

CAE and DAE

In [5], it is showed that in the limit of small Gaussian input noise, the **denoising reconstruction** error is equivalent to **a contractive penalty** on the **reconstruction function** that maps $\mathbf{x}$ to $\mathbf{r} = g(f(\mathbf{x}))$.

Denoising Autoencoders make the reconstruction function to resist to small errors in the input.

Contractive autoencoders make the feature extraction function to resist small perturbations of the input.

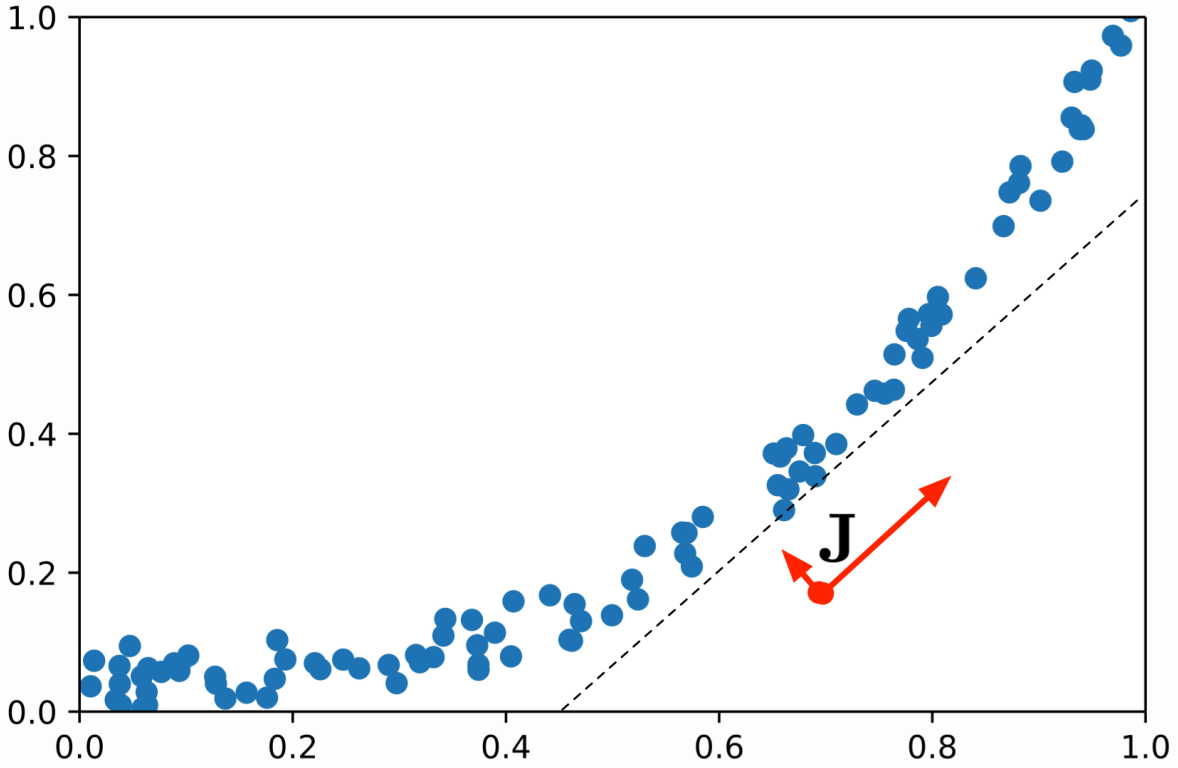
Learning Manifolds

As mentioned, regularized autoencoders learn manifolds by balancing two opposing forces. In the case of the CAE, these two forces are reconstruction error and the contractive penalty $\boldsymbol{\Omega}(\mathbf{h})$.

The compromise between these two forces yields an autoencoder whose derivatives $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$ are mostly tiny. Only a small number of hidden units, corresponding to a small number of directions in the input, may have significant derivatives.

Learning Manifolds

Directions $\mathbf{x}$ with large $\mathbf{Jx}$ rapidly change $\mathbf{h}$. Since such directions are penalized during learning, the ones "surviving" are likely to be directions which approximate the tangent planes of the manifold.



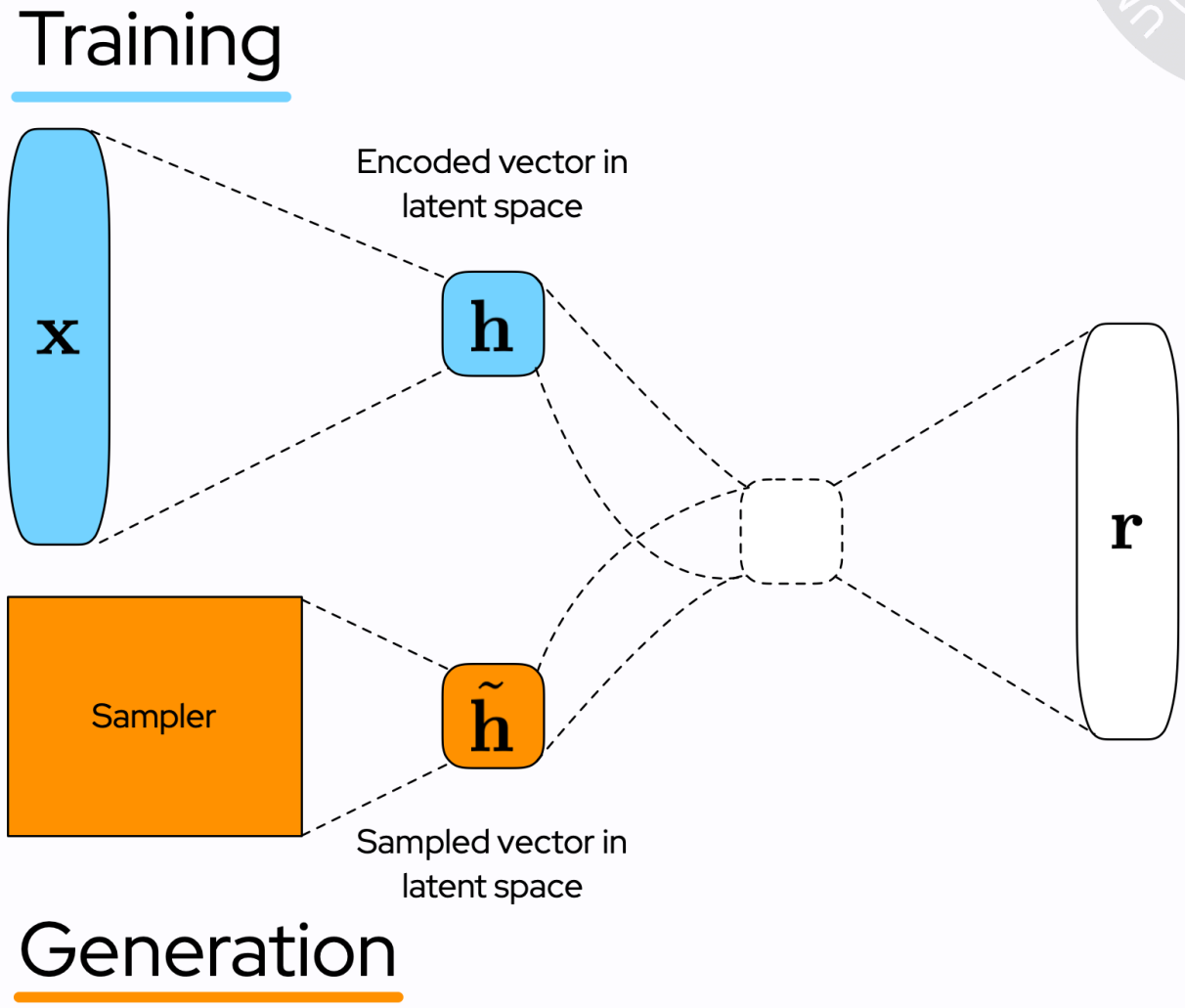
Variational Autoencoders (VAE)

A VAE is an autoencoder whose **encodings distribution is regularised** during the training in order to ensure that its latent space has good properties allowing us to generate some new data.

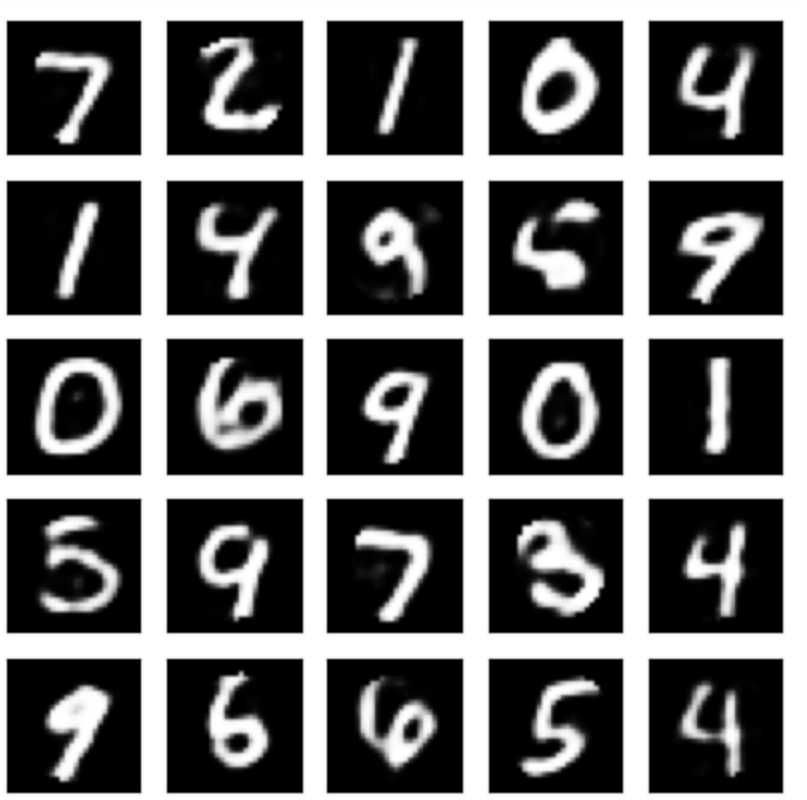
The regularization method is based on *variational inference*, an inference method popular in statistics.

Autoencoders are not good generative models

While this simple schema is intuitive, it does not work well in practice.



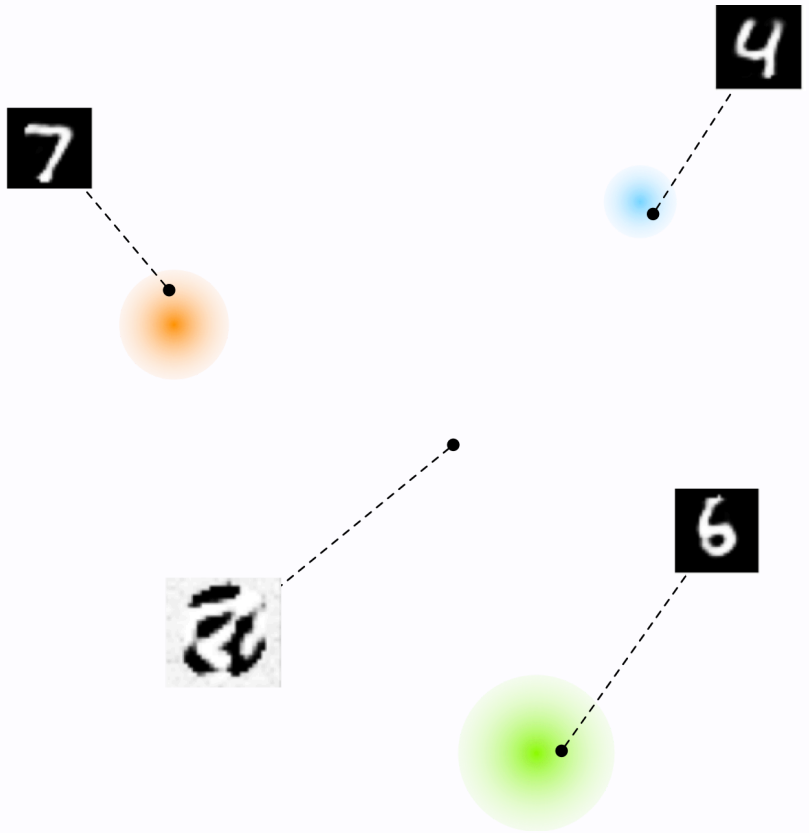
Autoencoders are not good generative models

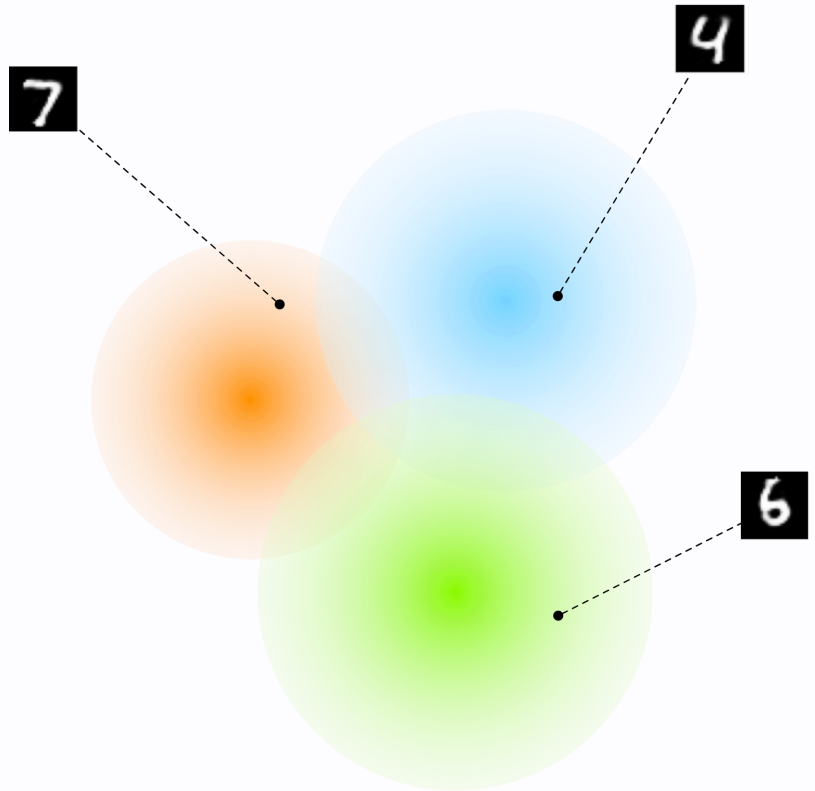


Images from **encoded** vectors in latent space

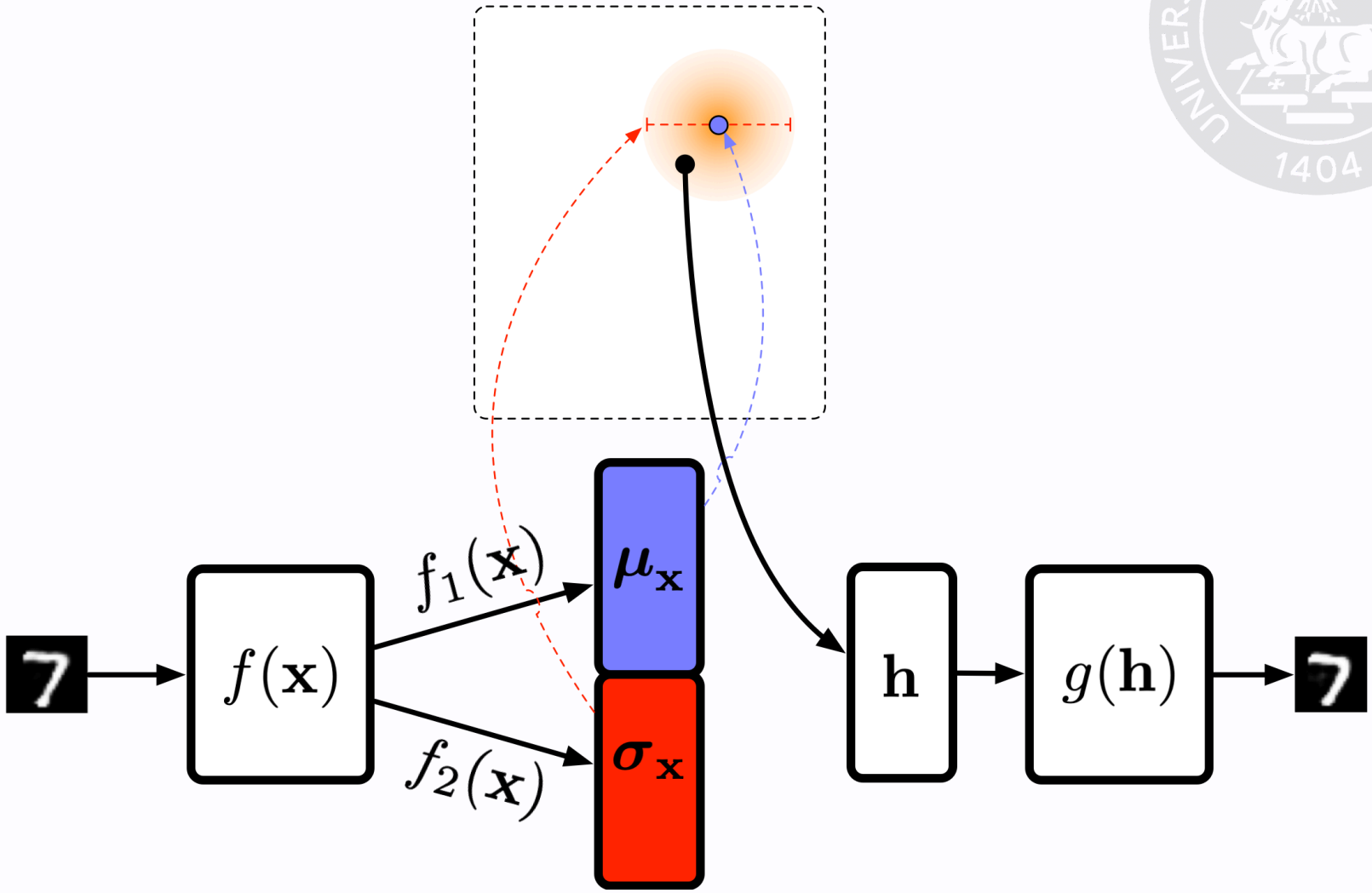
Images from **random** vectors in latent space

Encoding networks needs to be regularized



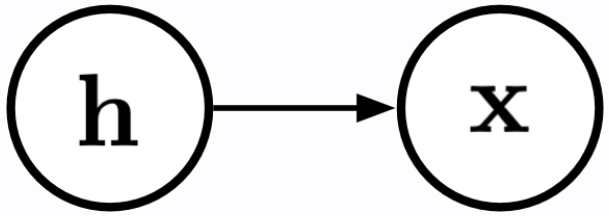


To impose this kind of regularization, images are **encoded into a distribution**. During training, the decoder is trained to decode points sampled from these distributions.



Probabilistic model

We will assume a probabilistic model where a latent variable $\mathbf{h}$ is sampled from a distribution $p(\mathbf{h})$ and then $\mathbf{x}$ is sampled from $p(\mathbf{x}|\mathbf{h})$.



We will also assume:

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

$$p(\mathbf{x}|\mathbf{h}) \sim \mathcal{N}(g(\mathbf{h}), c\mathbf{I}) \propto \exp\left(-(\mathbf{x} - g(\mathbf{h}))^T (cI)^{-1} (\mathbf{x} - g(\mathbf{h}))\right)$$

$$= \exp\left(-\frac{\|\mathbf{x} - g(\mathbf{h})\|^2}{2c}\right)$$

Variational Inference

Even under these assumptions, the inference problem:

$$p(\mathbf{h}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{h})p(\mathbf{h})}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathbf{h})p(\mathbf{h})}{\int p(\mathbf{x}|\mathbf{u})p(\mathbf{u})d\mathbf{u}}$$

is instractable due to the normalization factor in the denominator.

Variational inference is a technique allowing to solve this problem by approximating the hard to compute probability with a simpler one.

Specifically, we will seek a distribution $q_{\mathbf{x}}(\mathbf{h}) \sim \mathcal{N}(f_1(\mathbf{x}), f_2(\mathbf{x})) \approx p(\mathbf{h}|\mathbf{x})$

Our goal is now to find f_1 and f_2 allowing the best possible approximation of $p(\mathbf{h}|\mathbf{x})$. In other words, we want to solve

$$\arg\min_{f_1,f_2} KL(q_{\mathbf{x}}(\mathbf{h})||p(\mathbf{h}|\mathbf{x}))$$

$$= \arg\min_{f_1,f_2} E_{\mathbf{h}\sim q_{\mathbf{x}}}\left[\log \frac{q_{\mathbf{x}}(\mathbf{h})}{p(\mathbf{h}|\mathbf{x})}\right]$$

$$= \arg\min_{f_1,f_2} E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log q_{\mathbf{x}}(\mathbf{h})] - E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log p(\mathbf{h}|\mathbf{x})]$$

$$= \arg\min_{f_1,f_2} E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log q_{\mathbf{x}}(\mathbf{h})] - E_{\mathbf{h}\sim q_{\mathbf{x}}}\left[\log \frac{p(\mathbf{x}|\mathbf{h})p(\mathbf{h})}{p(\mathbf{x})}\right]$$

$$= \arg\min_{f_1,f_2} E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log q_{\mathbf{x}}(\mathbf{h})] - E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log p(\mathbf{x}|\mathbf{h})] - E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log p(\mathbf{h})] + E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log p(\mathbf{x})]$$

$$= \arg\min_{f_1,f_2} -(E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log p(\mathbf{x}|\mathbf{h})] - (E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log q_{\mathbf{x}}(\mathbf{h})] - E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log p(\mathbf{h})]))$$

$$= \arg\max_{f_1,f_2} E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log p(\mathbf{x}|\mathbf{h})] - (E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log q_{\mathbf{x}}(\mathbf{h})] - E_{\mathbf{h}\sim q_{\mathbf{x}}}[\log p(\mathbf{h})])$$

$$= \arg\max_{f_1,f_2} E_{\mathbf{h}\sim q_{\mathbf{x}}}\left[-\frac{\|\mathbf{x}-g(\mathbf{h})\|^2}{2c}\right] - KL(q_{\mathbf{x}}(\mathbf{h})||p(\mathbf{h}))$$

This derivation allows us to determine f_1 and f_2. What about the decoder function g?

The goal is to find the function g that maximizes the likelihood of $p(\mathbf{x}|\mathbf{h})$ under the assumption that $\mathbf{h} \sim q_{\mathbf{x}}(\mathbf{h})$, i.e., we want:

$$g = \arg\max_g E_{\mathbf{h}\sim q_{\mathbf{x}}(\mathbf{h})}[\log p(\mathbf{x}|\mathbf{h})] = \arg\max_g E_{\mathbf{h}\sim q_{\mathbf{x}}(\mathbf{h})}\left[-\frac{\|\mathbf{x}-g(\mathbf{h})\|^2}{2c}\right]$$

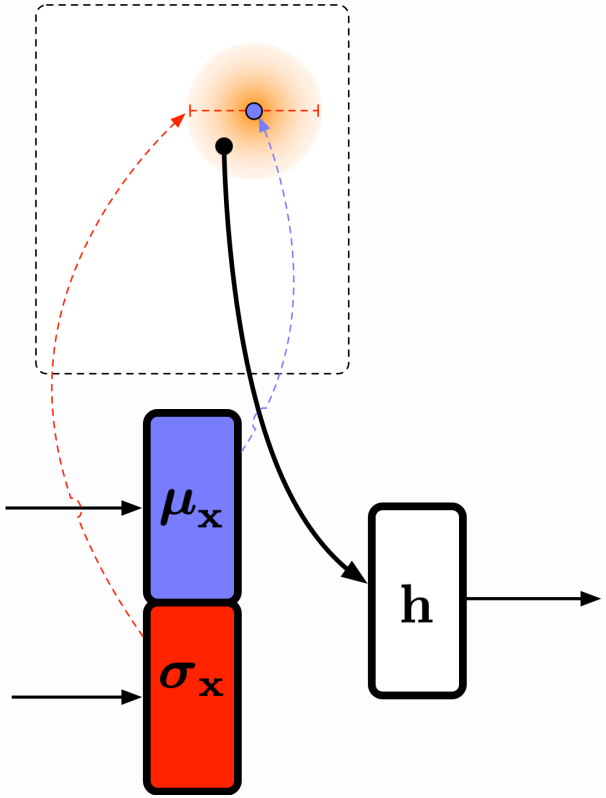
Putting everything together, we want to solve for:

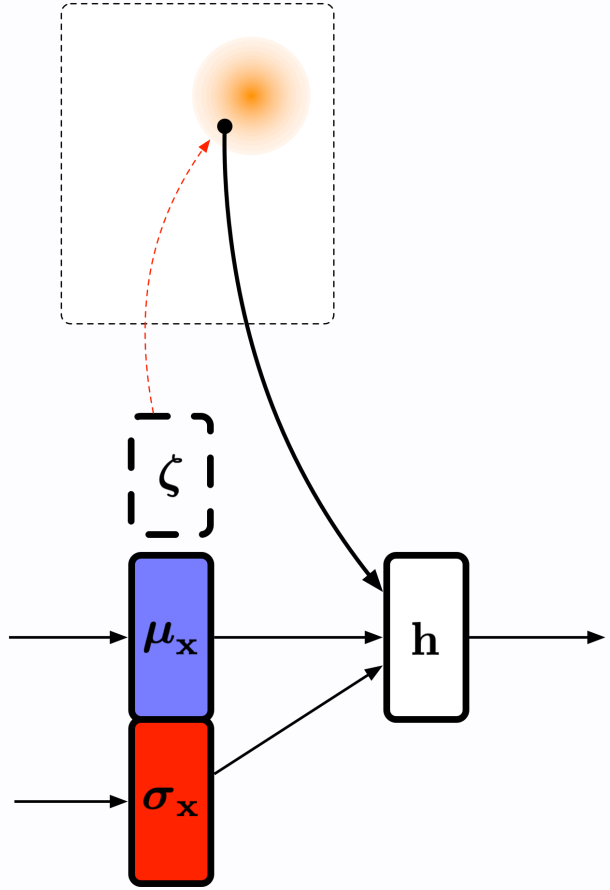
$$\arg\max_{f_1,f_2,g} E_{\mathbf{h}\sim q_{\mathbf{x}}}\left[-\frac{\|\mathbf{x}-g(\mathbf{h})\|^2}{2c}\right] - KL(q_{\mathbf{x}}(\mathbf{h})\|p(\mathbf{h}))$$

Or, equivalently:

$$\arg\min_{f_1,f_2,g} E_{\mathbf{h}\sim q_{\mathbf{x}}}\left[\frac{\|\mathbf{x}-g(\mathbf{h})\|^2}{2c}\right] + KL(q_{\mathbf{x}}(\mathbf{h})\|p(\mathbf{h}))$$

Reparametrization trick





Sampling from $\mathcal{N}(\boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\sigma}_\mathbf{x})$ is equivalent to sampling $\boldsymbol{\zeta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then setting $\mathbf{h} = \boldsymbol{\sigma}_\mathbf{x}\boldsymbol{\zeta} + \boldsymbol{\mu}_\mathbf{x}$.

VAE loss function

Putting all together, the loss function used in VAEs is:

$$C\|\mathbf{x} - g(\mathbf{h})\|^2 + KL(\mathcal{N}(\boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\sigma}_\mathbf{x})\|\mathcal{N}(\mathbf{0}, \mathbf{I}))$$

and it can be (*not so easily*) shown that:

$$KL(\mathcal{N}(\boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\sigma}_\mathbf{x})\|\mathcal{N}(\mathbf{0}, \mathbf{I})) = \frac{1}{2}\left[-1 - \log(\boldsymbol{\sigma}_\mathbf{x}^2) + \boldsymbol{\sigma}_\mathbf{x}^2 + \boldsymbol{\mu}_\mathbf{x}^2\right]$$

where:

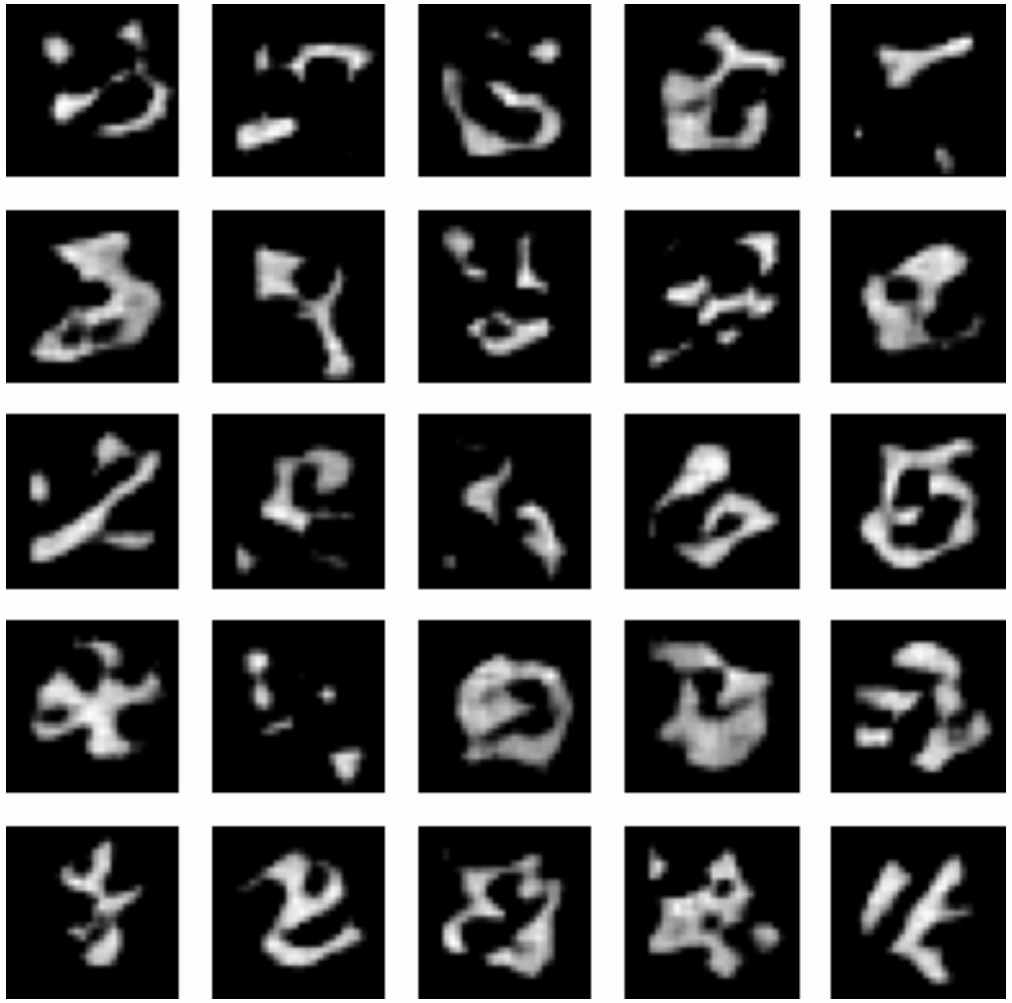
- $\boldsymbol{\mu}_\mathbf{x} = f_1(\mathbf{x})$
- $\boldsymbol{\sigma}_\mathbf{x} = f_2(\mathbf{x})$

Implementation hints

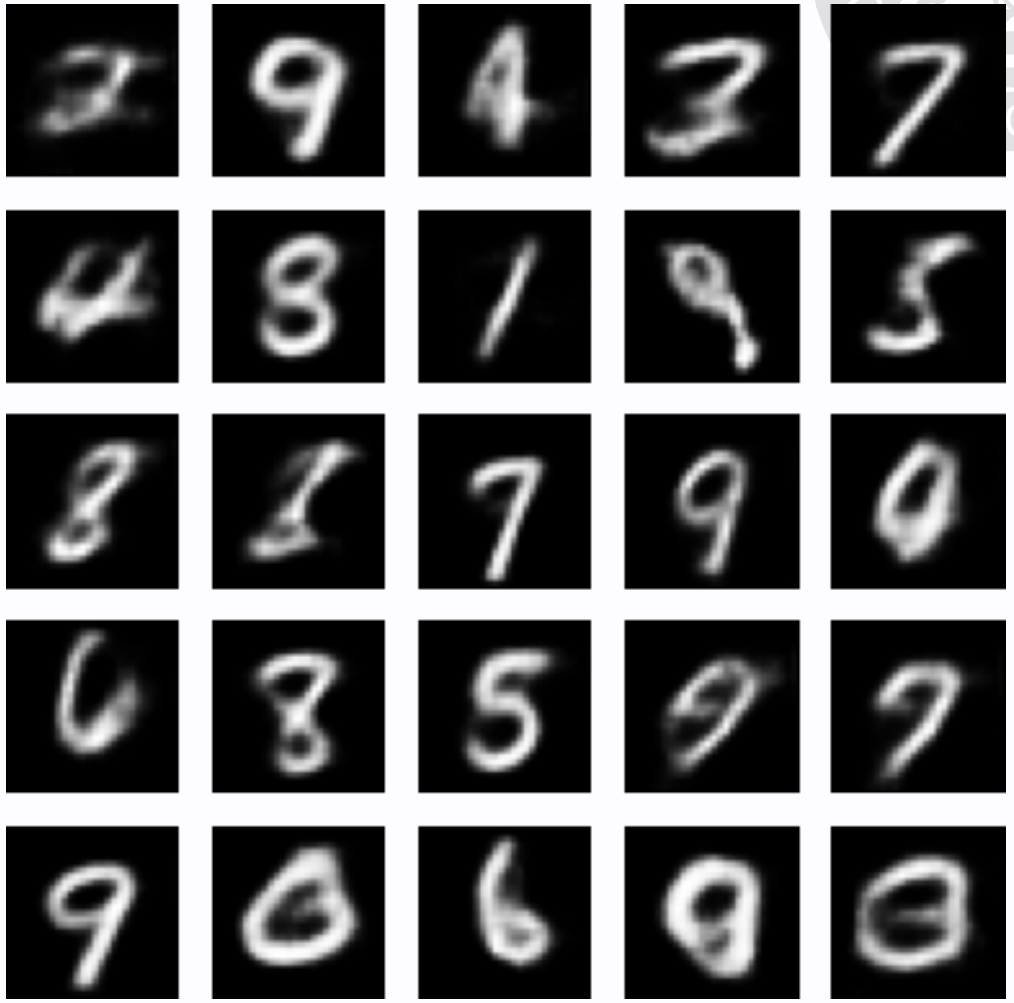

```python
class VAEModel(torch.nn.Module):
    ...

    def encode(self, x):
        x = ...

        sigma = self.sigma(x)
        mu = self.mu(x)

        z = self.reparameterize(mu, sigma)
        return z, mu, sigma

    def forward(self, x):
        z, mu, sigma = encode(x)
        x = decode(z)

        return x, mu, sigma

    def reparameterize(self, mu, sigma):
        epsilon = torch.randn_like(mu)
        return mu + epsilon * sigma
```

```python
def loss_fn(x, y):
    x_, mu, sigma = x

    # Reconstruction loss
    recon_loss =
        F.mse_loss(x_, y, reduction="sum")

    # KL loss
    kl_loss = 0.5 * torch.sum(
        -1 - torch.log(sigma**2)
        + sigma**2 + mu**2
    )

    return recon_loss + kl_loss,
           recon_loss,
           kl_loss
```

Generation ($\boldsymbol{h} \in \mathbb{R}^{50}$)

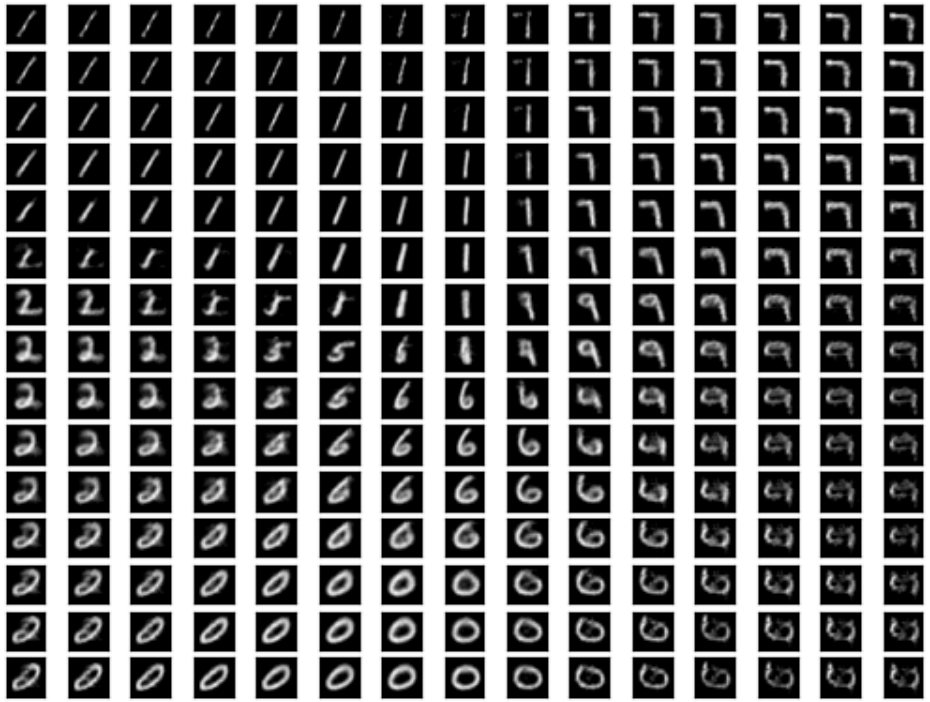


Autoencoders

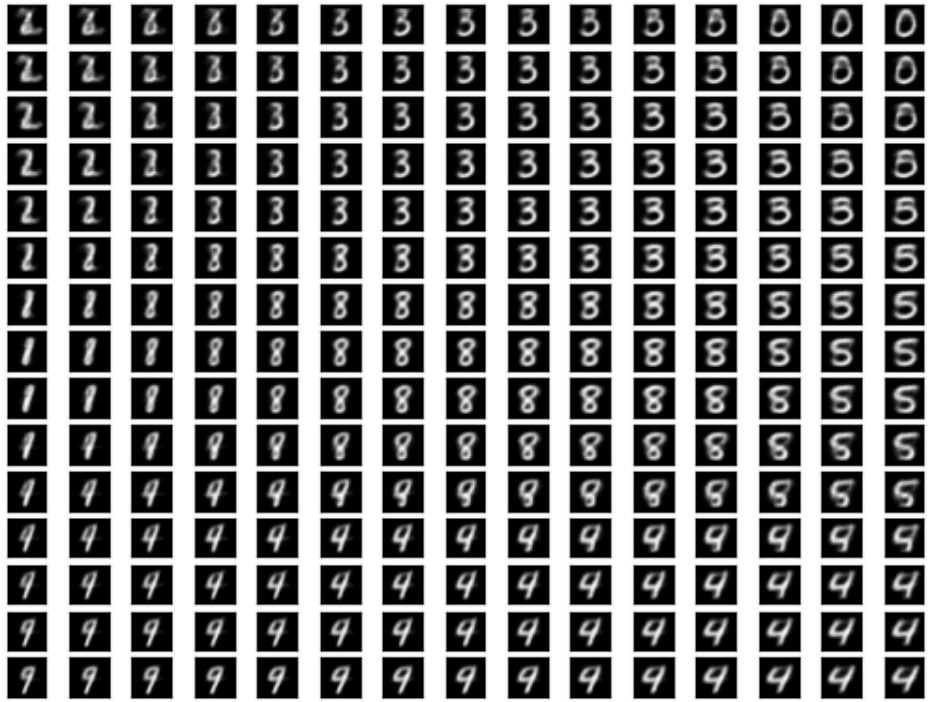


Variational Autoencoders

Interpolation btwn representations ($\boldsymbol{h} \in \mathbb{R}^3$)

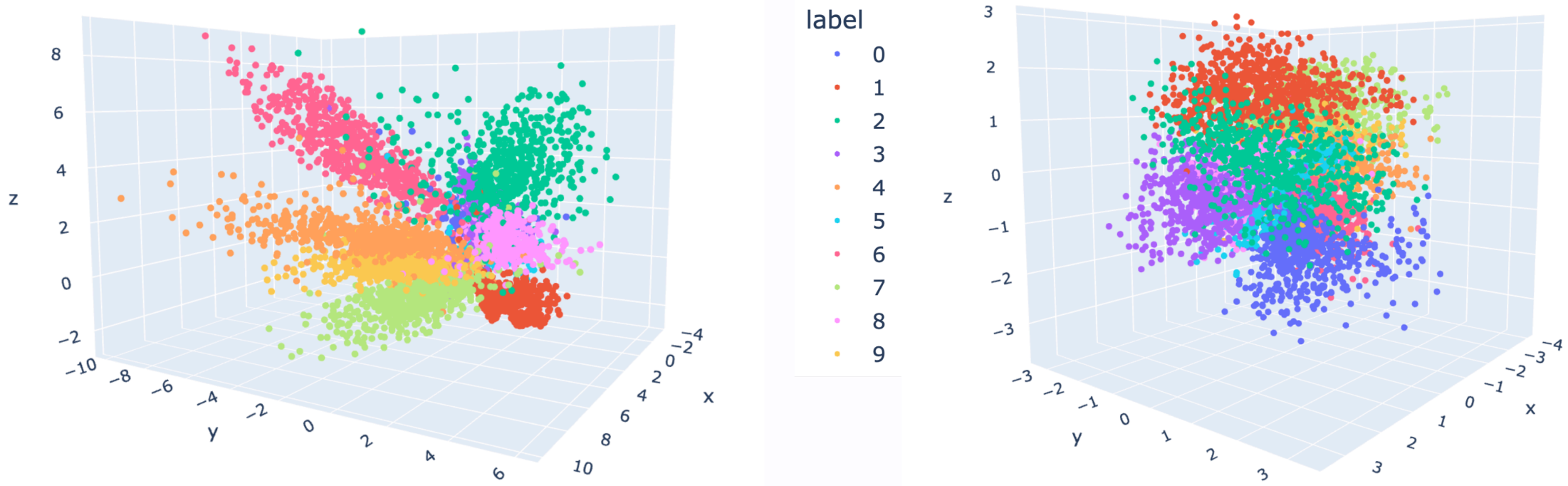


Autoencoders



Variational Autoencoders

Latent space organization ($\boldsymbol{h} \in \mathbb{R}^3$)



[Autoencoders](#)

[Variational Autoencoders](#)

Notable Applications

Dimensionality reduction

Dimensionality reduction has been the first motivation to study autoencoders. By learning (deep) autoencoders that map input to lower dimensional outputs we get several benefits:

- many tasks can be more easily solved;
- models of smaller spaces consume less memory and runtime;
- often dimensionality reduction places semantically related examples nearby. This helps generalization.

Notable Applications

Semantic Hashing

In information retrieval there is great benefit provided by dimensionality reduction.

if we train the dimensionality reduction algorithm to produce a code that is low- dimensional and binary, then we can store all database entries in a hash table mapping binary code vectors to entries.

Semantic Hashing

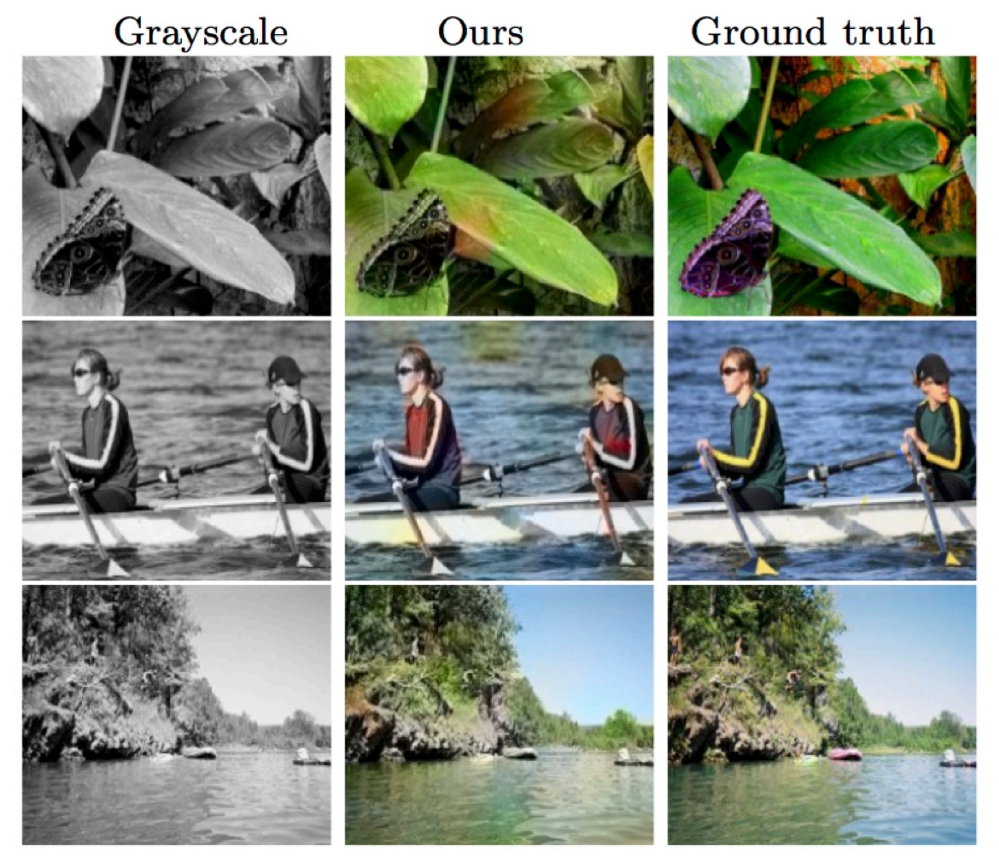
The hash table allows to retrieve all similar elements at once since they will share the same binary code.

Search for slightly less similar elements can be made by swapping single bits in the code.

To produce these binary codes, one typically use sigmoid units in the final layer and train them to saturation (by injecting additive noise just before the sigmoid).

Question

How would you force an autoencoder to build the image colorization system?



Question

How would you force an autoencoder to build the image inpainting system?

References

- [1] Goodfellow, Ian, et al. Deep learning. Cambridge: MIT press, 2016.
- [2] Baldassarre et al., *Deep Koalarization: Image Colorization using CNNs and Inception-Resnet-v2*. ArXiv:1712.03400, Dec. 2017
- [3] Zeng, Kun, et al. *Coupled deep autoencoder for single image super-resolution*. IEEE transactions on cybernetics, 2015, 47.1: 27-37.
- [4] Mao, Xiao-Jiao et al. *Image restoration using convolutional auto-encoders with symmetric skip connections*. arXiv preprint arXiv:1606.08921, 2016.
- [5] Alain, Guillaume, Yoshua Bengio, and Salah Rifai. "Regularized auto-encoders estimate local statistics." ICLR. 2013.