

# Deep Learning & Applied AI

Deep generative models

Emanuele Rodolà  
[rodola@di.uniroma1.it](mailto:rodola@di.uniroma1.it)



SAPIENZA  
UNIVERSITÀ DI ROMA

# Generative models

Overall idea:

Learn a **distribution** from some given training samples, and generate new samples from the same distribution.

# Generative models

Overall idea:

Learn a **distribution** from some given training samples, and generate new samples from the same distribution.



<https://this-person-does-not-exist.com/>

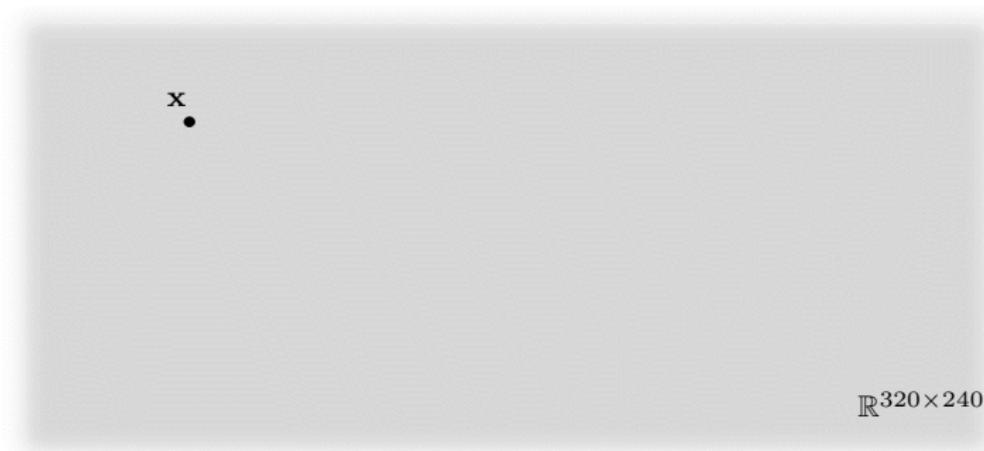
# Generative models

What does it mean to learn a **distribution**?

$$\mathbb{R}^{320 \times 240}$$

# Generative models

What does it mean to learn a **distribution**?



$$\mathbb{R}^{320 \times 240}$$

Questo rettangolo rappresenta lo spazio delle immagini con risoluzione 320x240 e  $x$  è un'immagine nello spazio.

# Generative models

What does it mean to learn a **distribution**?



$\mathbb{R}^{320 \times 240}$

# Generative models

What does it mean to learn a **distribution**?



$\mathbb{R}^{320 \times 240}$

# Generative models

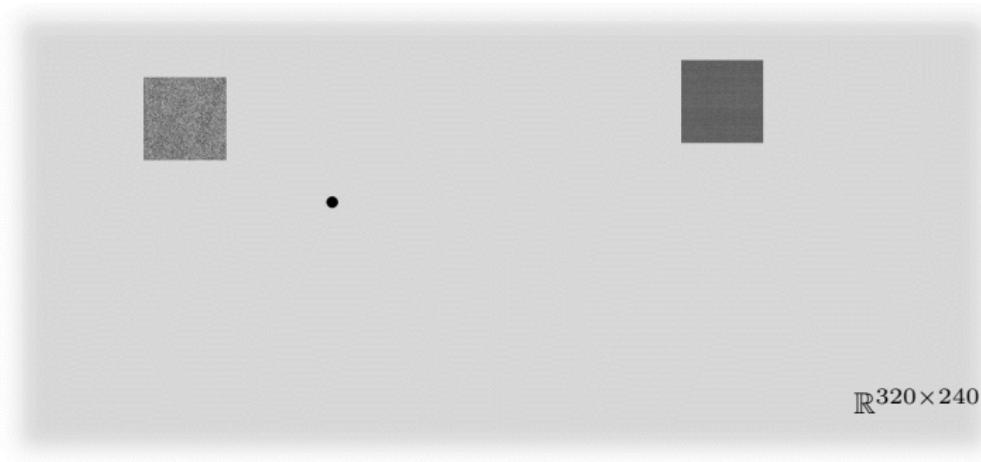
What does it mean to learn a **distribution**?



$\mathbb{R}^{320 \times 240}$

# Generative models

What does it mean to learn a **distribution**?



$\mathbb{R}^{320 \times 240}$

# Generative models

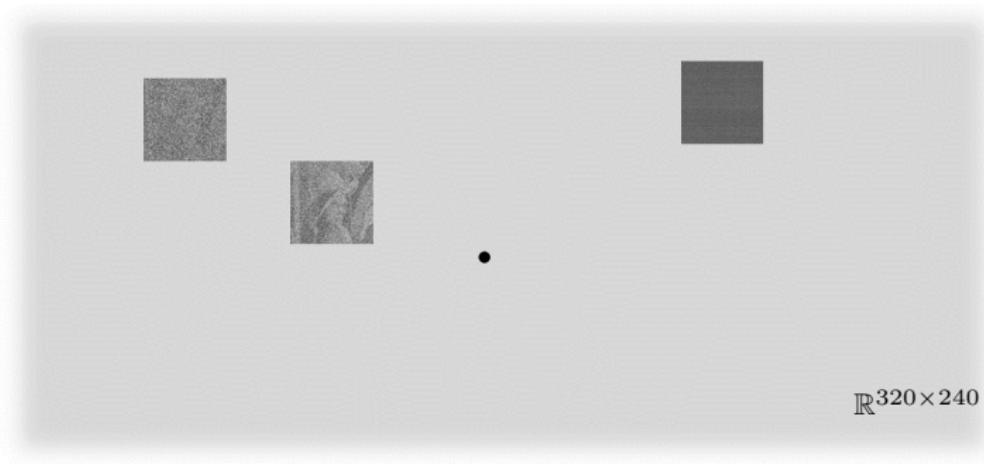
What does it mean to learn a **distribution**?



$\mathbb{R}^{320 \times 240}$

# Generative models

What does it mean to learn a **distribution**?



$\mathbb{R}^{320 \times 240}$

# Generative models

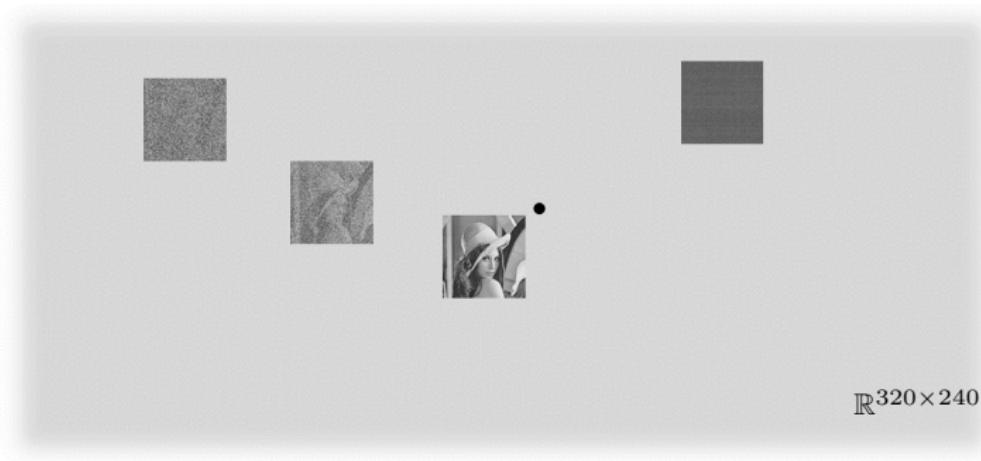
What does it mean to learn a **distribution**?



$\mathbb{R}^{320 \times 240}$

# Generative models

What does it mean to learn a **distribution**?



# Generative models

What does it mean to learn a **distribution**?

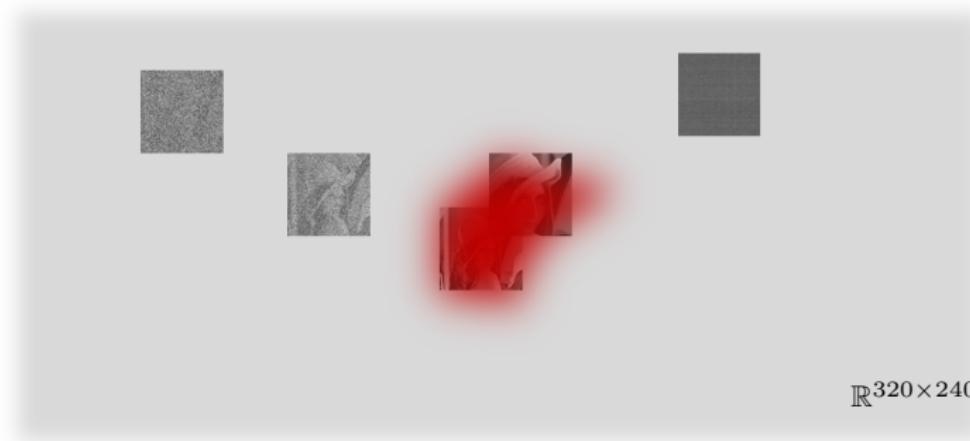


$\mathbb{R}^{320 \times 240}$

Man mano che si esplora lo spazio si trovano immagini con un senso per l'uomo. Possiamo notare che alcune di queste immagini sono "vicine" tra di loro: questo vuol dire che hanno una distribuzione simile...

# Generative models

What does it mean to learn a **distribution**?



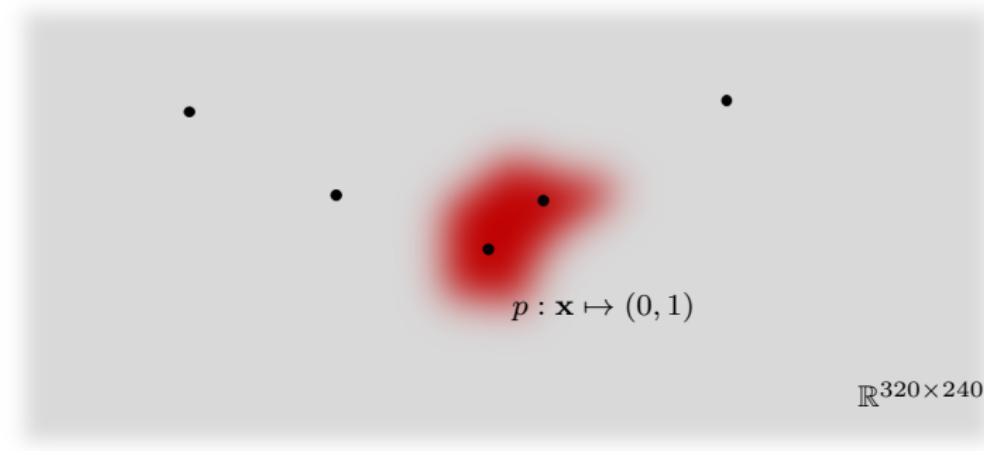
$\mathbb{R}^{320 \times 240}$

... come queste due.

# Generative models

What does it mean to learn a distribution?

Possiamo associare una probabilità a queste immagini che ci indica quanto un sample è probabile che faccia parte della distribuzione che vogliamo imparare (vicino 0 = poco probabile, vicino 1 = molto probabile) (notare che 0 e 1 non sono compresi nell'intervallo)

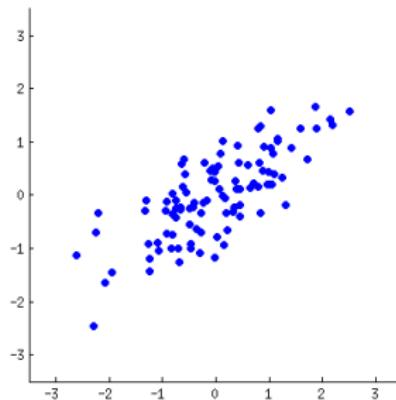


The probability distribution  $p$  is defined on the entire data space.

Tuttavia, già in questo esempio possiamo osservare come lo spazio di rappresentazione in esame sia troppo dispersivo. Occorre quindi trovare una semplificazione...

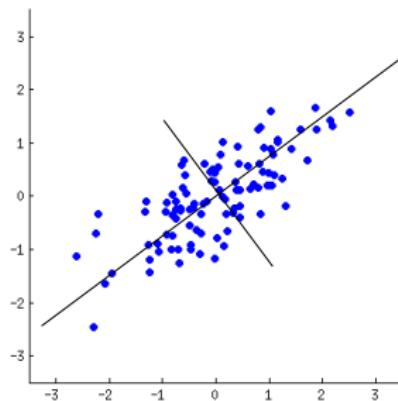
# Principal component analysis (PCA)

Regard our data as  $n$  points in  $\mathbb{R}^d$ :



# Principal component analysis (PCA)

Regard our data as  $n$  points in  $\mathbb{R}^d$ :

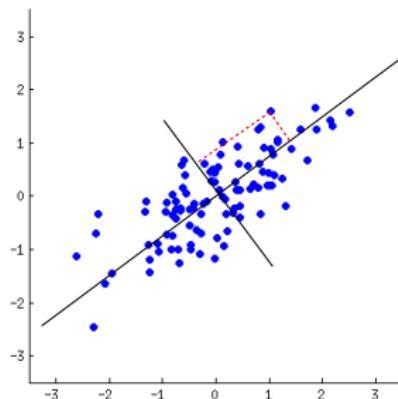


Overall idea:

- Find  $k \leq d$  **orthogonal directions** with the most variance.

# Principal component analysis (PCA)

Regard our data as  $n$  points in  $\mathbb{R}^d$ :



Overall idea:

- Find  $k \leq d$  **orthogonal directions** with the most variance.
- **Project** all the data points onto these directions.

# Principal component analysis (PCA)

In matrix notation:

$$\underbrace{\begin{pmatrix} \text{---} & \mathbf{x}_1^\top & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{x}_n^\top & \text{---} \end{pmatrix}}_{n \times d}$$

# Principal component analysis (PCA)

In matrix notation:

$$\underbrace{\begin{pmatrix} \text{---} & \mathbf{x}_1^\top & \text{---} \\ \vdots & & \vdots \\ \text{---} & \mathbf{x}_n^\top & \text{---} \end{pmatrix}}_{n \times d} \underbrace{\begin{pmatrix} | & & | \\ \mathbf{w}_1 & \cdots & \mathbf{w}_k \\ | & & | \end{pmatrix}}_{d \times k}$$

# Principal component analysis (PCA)

In matrix notation:

$$\underbrace{\begin{pmatrix} \_ & \mathbf{x}_1^\top & \_ \\ \_ & \vdots & \_ \\ \_ & \mathbf{x}_n^\top & \_ \end{pmatrix}}_{n \times d} \underbrace{\begin{pmatrix} | & & | \\ \mathbf{w}_1 & \cdots & \mathbf{w}_k \\ | & & | \end{pmatrix}}_{d \times k} = \underbrace{\begin{pmatrix} \_ & \mathbf{z}_1^\top & \_ \\ \_ & \vdots & \_ \\ \_ & \mathbf{z}_n^\top & \_ \end{pmatrix}}_{n \times k}$$

# Principal component analysis (PCA)

In matrix notation:

ogni riga è un sample e le colonne sono le dimensioni del sample

$$\underbrace{\begin{pmatrix} \cdots & \mathbf{x}_1^\top & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_n^\top & \cdots \end{pmatrix}}_{n \times d} \underbrace{\begin{pmatrix} | & & | \\ \mathbf{w}_1 & \cdots & \mathbf{w}_k \\ | & & | \end{pmatrix}}_{d \times k} = \underbrace{\begin{pmatrix} \cdots & \mathbf{z}_1^\top & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{z}_n^\top & \cdots \end{pmatrix}}_{n \times k}$$

Assuming  $\mathbf{W}^\top \mathbf{W} = \mathbf{I}$ , for  $k = d$  we get:

$$\mathbf{X}^\top \mathbf{W} = \mathbf{Z}^\top$$

$$\mathbf{X} = \mathbf{WZ}$$

quindi in pratica posso ricostruire i samples originali partendo dalla nuova rappresentazione

tuttavia noi siamo interessati al caso  $k < d$ , che fa diventare l'equazione...

# Principal component analysis (PCA)

In matrix notation:

$$\underbrace{\begin{pmatrix} \_ & \mathbf{x}_1^\top & \_ \\ \_ & \vdots & \_ \\ \_ & \mathbf{x}_n^\top & \_ \end{pmatrix}}_{n \times d} \underbrace{\begin{pmatrix} | & & | \\ \mathbf{w}_1 & \cdots & \mathbf{w}_k \\ | & & | \end{pmatrix}}_{d \times k} = \underbrace{\begin{pmatrix} \_ & \mathbf{z}_1^\top & \_ \\ \_ & \vdots & \_ \\ \_ & \mathbf{z}_n^\top & \_ \end{pmatrix}}_{n \times k}$$

Assuming  $\mathbf{W}^\top \mathbf{W} = \mathbf{I}$ , for  $k < d$  we get:

$$\mathbf{X}^\top \mathbf{W} = \mathbf{Z}^\top$$

$$\mathbf{X} \approx \mathbf{WZ} \quad \dots \text{meno precisa}$$

# Principal component analysis (PCA)

In matrix notation:

$$\underbrace{\begin{pmatrix} \_ & \mathbf{x}_1^\top & \_ \\ \_ & \vdots & \_ \\ \_ & \mathbf{x}_n^\top & \_ \end{pmatrix}}_{n \times d} \underbrace{\begin{pmatrix} | & & | \\ \mathbf{w}_1 & \cdots & \mathbf{w}_k \\ | & & | \end{pmatrix}}_{d \times k} = \underbrace{\begin{pmatrix} \_ & \mathbf{z}_1^\top & \_ \\ \_ & \vdots & \_ \\ \_ & \mathbf{z}_n^\top & \_ \end{pmatrix}}_{n \times k}$$

Assuming  $\mathbf{W}^\top \mathbf{W} = \mathbf{I}$ , for  $k < d$  we get:

$$\mathbf{X}^\top \mathbf{W} = \mathbf{Z}^\top \quad \text{projection}$$

$$\mathbf{X} \approx \mathbf{WZ} \quad \text{reconstruction}$$

# Principal component analysis (PCA)

In matrix notation:

$$\underbrace{\begin{pmatrix} \_ & \mathbf{x}_1^\top & \_ \\ \_ & \vdots & \_ \\ \_ & \mathbf{x}_n^\top & \_ \end{pmatrix}}_{n \times d} \underbrace{\begin{pmatrix} | & & | \\ \mathbf{w}_1 & \cdots & \mathbf{w}_k \\ | & & | \end{pmatrix}}_{d \times k} = \underbrace{\begin{pmatrix} \_ & \mathbf{z}_1^\top & \_ \\ \_ & \vdots & \_ \\ \_ & \mathbf{z}_n^\top & \_ \end{pmatrix}}_{n \times k}$$

Assuming  $\mathbf{W}^\top \mathbf{W} = \mathbf{I}$ , for  $k < d$  we get:

$$\mathbf{X}^\top \mathbf{W} = \mathbf{Z}^\top \quad \text{projection}$$

$$\mathbf{X} \approx \mathbf{WZ} \quad \text{reconstruction}$$

We call the columns of  $\mathbf{W}$  **principal components**.

They are unknown and must be computed.

# Principal component analysis (PCA)

We seek the **direction  $w$**  (a column of  $\mathbf{W}$ ) that:

- Minimizes the **projection/reconstruction error**.
- Maximizes the **variance** of the projected data.

# Principal component analysis (PCA)

We seek the **direction  $w$**  (a column of  $W$ ) that:

- Minimizes the **projection/reconstruction error**. anche detto **residual error**
- Maximizes the **variance of the projected data**. Il grafico d'esempio mostra una riduzione da 2D a 1D

ATTENZIONE: scegliendo solo un eigenvector la riduzione dimensionale sarà da  $d$  a 1!!!  
Per operare una riduzione da  $d$  a  $k$  (con  $k < d$ ) dovrò scegliere  $k$  eigenvectors dalla matrice  $W$ .  
Una riduzione da 3D a 2D ad esempio si può fare scegliendo i primi 2 eigenvectors di  $W$

# Principal component analysis (PCA)

Assume the data points  $\mathbf{X}$  are **centered** at zero.

For a given  $\mathbf{w}$ , the projection of all  $n$  points onto  $\mathbf{w}$  is  $\mathbf{X}^\top \mathbf{w}$ .

# Principal component analysis (PCA)

Assume the data points  $\mathbf{X}$  are **centered** at zero.

For a given  $\mathbf{w}$ , the projection of all  $n$  points onto  $\mathbf{w}$  is  $\mathbf{X}^\top \mathbf{w}$ .

The **variance** to maximize is  $\|\mathbf{X}^\top \mathbf{w}\|_2^2$ :

$$(\mathbf{X}^\top \mathbf{w})^\top (\mathbf{X}^\top \mathbf{w})$$

# Principal component analysis (PCA)

Assume the data points  $\mathbf{X}$  are **centered** at zero.

For a given  $\mathbf{w}$ , the projection of all  $n$  points onto  $\mathbf{w}$  is  $\mathbf{X}^\top \mathbf{w}$ .

The **variance** to maximize is  $\|\mathbf{X}^\top \mathbf{w}\|_2^2$ :

$$(\mathbf{X}^\top \mathbf{w})^\top (\mathbf{X}^\top \mathbf{w}) = \mathbf{w}^\top (\mathbf{X} \mathbf{X}^\top) \mathbf{w}$$

# Principal component analysis (PCA)

Assume the data points  $\mathbf{X}$  are **centered** at zero.

For a given  $\mathbf{w}$ , the projection of all  $n$  points onto  $\mathbf{w}$  is  $\mathbf{X}^\top \mathbf{w}$ .

The **variance** to maximize is  $\|\mathbf{X}^\top \mathbf{w}\|_2^2$ :

$$(\mathbf{X}^\top \mathbf{w})^\top (\mathbf{X}^\top \mathbf{w}) = \mathbf{w}^\top \underbrace{(\mathbf{X} \mathbf{X}^\top)}_{\mathbf{C}} \mathbf{w}$$

where  $\mathbf{C} \in \mathbb{R}^{d \times d}$  is the symmetric **covariance matrix**.

# Principal component analysis (PCA)

Assume the data points  $\mathbf{X}$  are **centered** at zero.

For a given  $\mathbf{w}$ , the projection of all  $n$  points onto  $\mathbf{w}$  is  $\mathbf{X}^\top \mathbf{w}$ .

The **variance** to maximize is  $\|\mathbf{X}^\top \mathbf{w}\|_2^2$ :

$$(\mathbf{X}^\top \mathbf{w})^\top (\mathbf{X}^\top \mathbf{w}) = \mathbf{w}^\top \underbrace{(\mathbf{X} \mathbf{X}^\top)}_{\mathbf{C}} \mathbf{w}$$

where  $\mathbf{C} \in \mathbb{R}^{d \times d}$  is the symmetric **covariance matrix**.

We want to solve the problem:

$$\max_{\mathbf{w}} \mathbf{w}^\top \mathbf{C} \mathbf{w} \quad \text{s.t. } \|\mathbf{w}\|_2 = 1$$

# Principal component analysis (PCA)

Assume the data points  $\mathbf{X}$  are **centered** at zero.

For a given  $\mathbf{w}$ , the projection of all  $n$  points onto  $\mathbf{w}$  is  $\mathbf{X}^\top \mathbf{w}$ .

The **variance** to maximize is  $\|\mathbf{X}^\top \mathbf{w}\|_2^2$  

ha riscritto la  $L_2$  norma  $(\mathbf{X}^\top \mathbf{w})^\top (\mathbf{X}^\top \mathbf{w}) = \mathbf{w}^\top \underbrace{(\mathbf{X} \mathbf{X}^\top)}_{\mathbf{C}} \mathbf{w}$

where  $\mathbf{C} \in \mathbb{R}^{d \times d}$  is the symmetric **covariance matrix**.  $\mathbf{C}$  ha dimensione  $d \times d$ , quindi è piccola

We want to solve the problem:

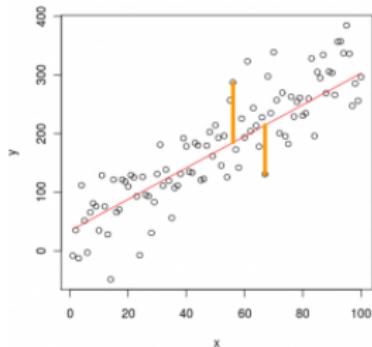
$$\max_{\mathbf{w}} \mathbf{w}^\top \mathbf{C} \mathbf{w} \quad \text{s.t. } \|\mathbf{w}\|_2 = 1$$

siccome  $\mathbf{w}$  indica la direzione,  
impone che la lunghezza sia 1  
(non ci interessa)

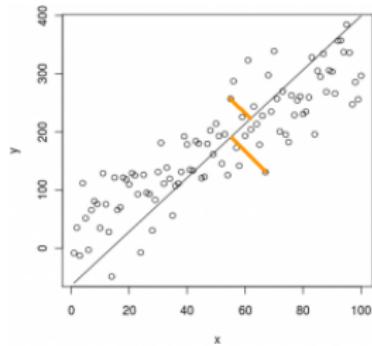
The solution is  $\mathbf{w} =$  principal **eigenvector** of  $\mathbf{C}$  (**Courant minmax principle**), and the value  $\mathbf{w}^\top \mathbf{C} \mathbf{w}$  is the corresponding **eigenvalue**.

# PCA is not linear regression

With linear regression we measure the error along the  $y$  coordinate:



With PCA we measure the error orthogonal to the principal direction:



## PCA as a generative model

Given the  $\mathbf{W}$  satisfying, for the observations  $\mathbf{X}$ :

$$\mathbf{X}^\top \mathbf{W} = \mathbf{Z}^\top \quad \text{projection}$$

$$\mathbf{X} \approx \mathbf{WZ} \quad \text{reconstruction}$$

We can generate new data just by sampling  $\mathbf{z}_{\text{new}} \in \mathbb{R}^k$  and computing:

$$\mathbf{x}_{\text{new}} = \mathbf{Wz}_{\text{new}}$$

# PCA as a generative model

Given the  $\mathbf{W}$  satisfying, for the observations  $\mathbf{X}$ :

$$\mathbf{X}^\top \mathbf{W} = \mathbf{Z}^\top \quad \text{projection}$$

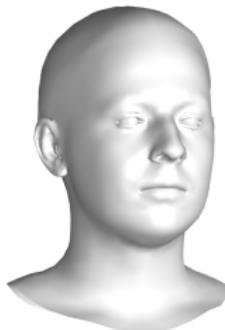
$$\mathbf{X} \approx \mathbf{WZ} \quad \text{reconstruction}$$

We can generate new data just by sampling   $\mathbf{z}_v \in \mathbb{R}^k$  and computing:

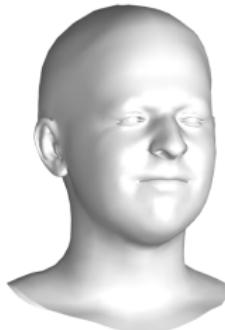
$$\mathbf{x}_{\text{new}} = \mathbf{Wz}_{\text{new}}$$

la matrice  $\mathbf{W}$  che abbiamo trovato  
non garantisce in alcun modo che i  
nuovi samples avranno senso

Example:

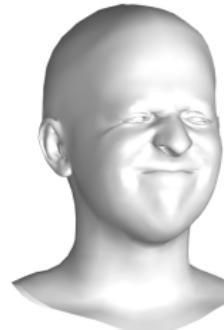


Data point  $\mathbf{x}_1$



Generated

$$\mathbf{x}_{\text{new}} = \frac{1}{2}\mathbf{W}(\mathbf{z}_1 + \mathbf{z}_2)$$



Data point  $\mathbf{x}_2$

## Codes

Consider again the relations:

$$\mathbf{W}^\top \mathbf{x} = \mathbf{z} \quad \text{projection}$$

$$\mathbf{x} \approx \mathbf{W}\mathbf{z} \quad \text{reconstruction}$$

From a different perspective, PCA gives us a [parametric model](#).

## Codes

Consider again the relations:

$$\mathbf{W}^\top \mathbf{x} = \mathbf{z} \quad \text{encoding}$$
$$\mathbf{x} \approx \mathbf{W}\mathbf{z} \quad \text{decoding}$$

From a different perspective, PCA gives us a [parametric model](#).

Each data point  $\mathbf{x}$  is transformed into a low-dimensional code  $\mathbf{z} \in \mathbb{R}^k$ , where the dimension  $k < d$  is fixed.

The [encoding](#) and [decoding](#) procedures are linear.

# Codes

Consider again the relations:

$$\mathbf{W}^\top \mathbf{x} = \mathbf{z} \quad \text{encoding}$$
$$\mathbf{x} \approx \mathbf{W}\mathbf{z} \quad \text{decoding}$$

From a different perspective, PCA gives us a [parametric model](#).

Each data point  $\mathbf{x}$  is transformed into a low-dimensional code  $\mathbf{z} \in \mathbb{R}^k$ , where the dimension  $k < d$  is fixed.

The [encoding](#) and [decoding](#) procedures are linear.

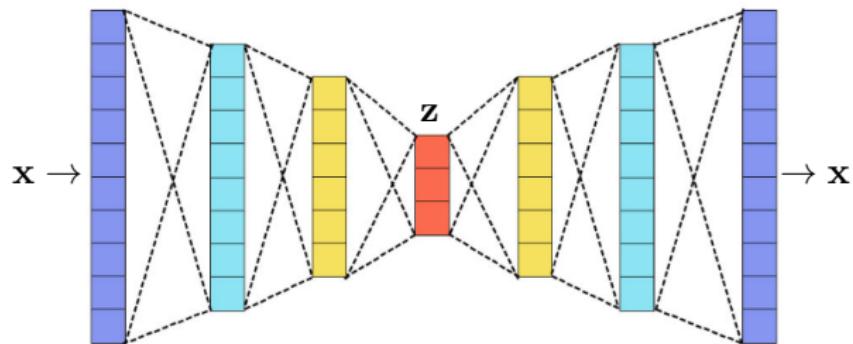
per PCA!

How to generalize this idea?

[sostituendo alla matrice  \$W\$  una rete neurale \(così passiamo da operazioni lineari a operazioni non lineari\)](#)

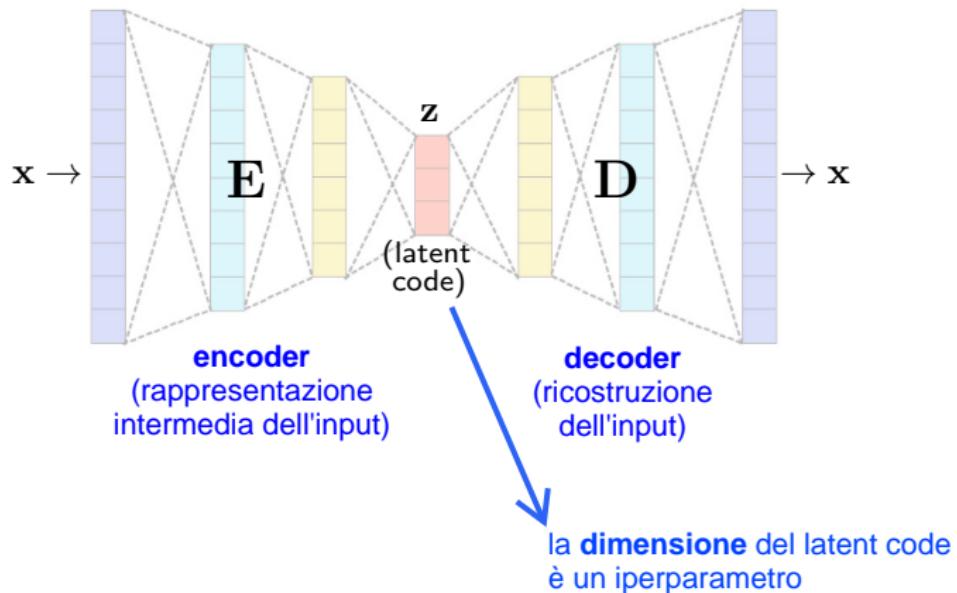
# Autoencoders (AE)

We can construct powerful parametric models using deep nets.



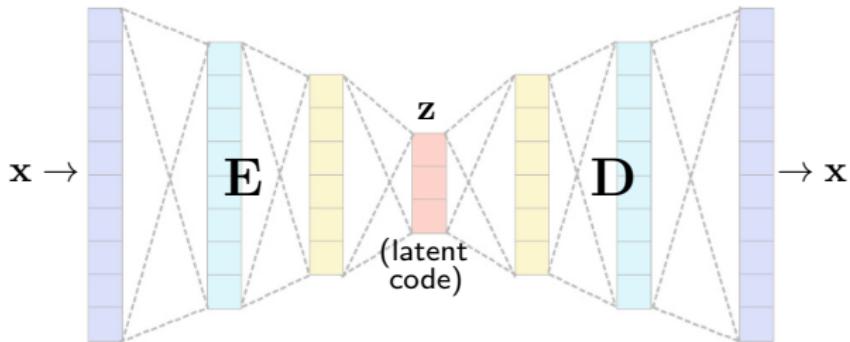
# Autoencoders (AE)

We can construct powerful parametric models using deep nets.



# Autoencoders (AE)

We can construct powerful parametric models using deep nets.

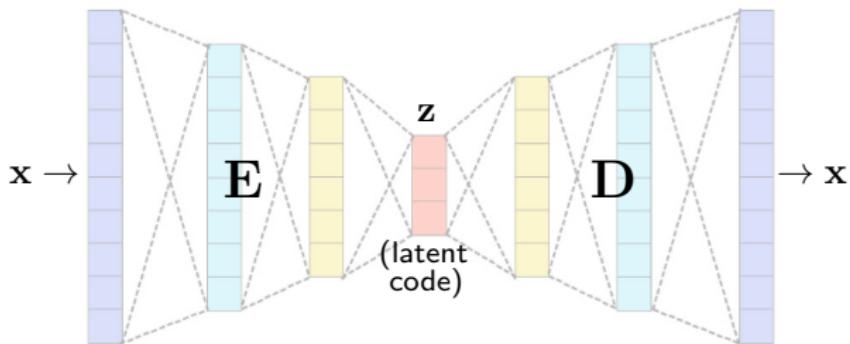


For a given dataset  $\{\mathbf{x}_i\}$ , we require the encoder  $E$  and decoder (or generator)  $D$  to minimize the **reconstruction loss**:

$$\ell_\Theta = \sum_i \|\mathbf{x}_i - D_\Theta(E_\Theta(\mathbf{x}_i))\|$$

# Autoencoders (AE)

We can construct powerful parametric models using deep nets.



For a given dataset  $\{\mathbf{x}_i\}$ , we require the encoder  $E$  and decoder (or generator)  $D$  to minimize the **reconstruction loss**:

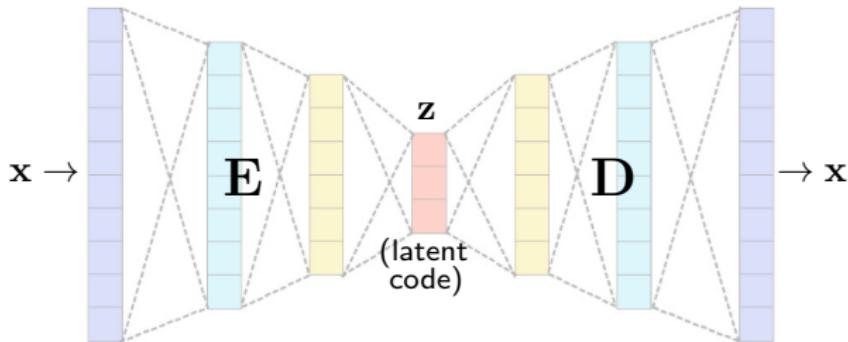
$$\ell_\Theta = \sum_i \|\mathbf{x}_i - D_\Theta(E_\Theta(\mathbf{x}_i))\|$$

The choice of the metric depends on the data.

# Autoencoders (AE)



We can construct powerful parametric models using deep nets.



For a given dataset  $\{\mathbf{x}_i\}$ , we require the encoder  $E$  and decoder (or generator)  $D$  to minimize the **reconstruction loss**:

$$\ell_\Theta = \sum_i \|\mathbf{x}_i - D_\Theta(E_\Theta(\mathbf{x}_i))\|$$

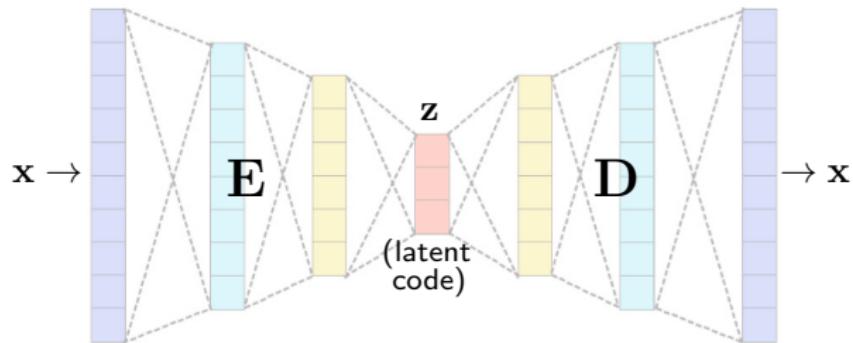
The choice of the metric depends on the data.

Quindi vogliamo che l'errore tra l'input originale e la sua ricostruzione sia quanto più vicino allo 0 possibile, in base alla norma scelta (come una energy function, dipende dall'input)

If the layers are **linear**, the codes  $\mathbf{z}_i$  span exactly the same space as PCA.

# Autoencoders (AE)

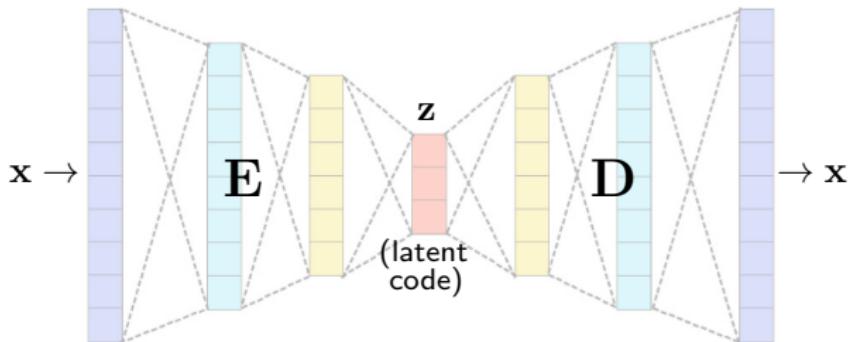
We can construct powerful parametric models using deep nets.



- The **bottleneck** prevents trivial solutions.

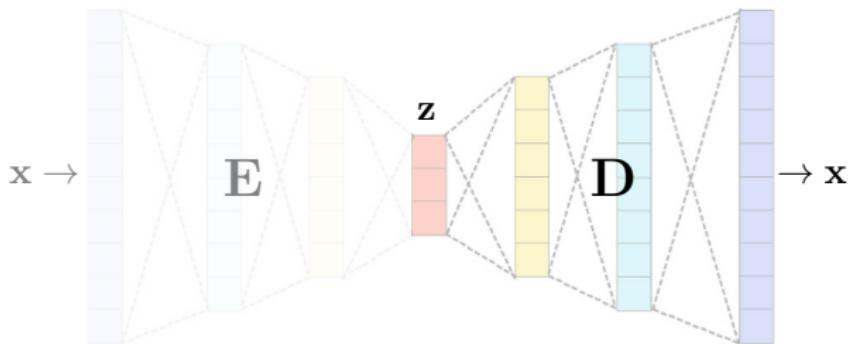
# Autoencoders (AE)

We can construct powerful parametric models using deep nets.



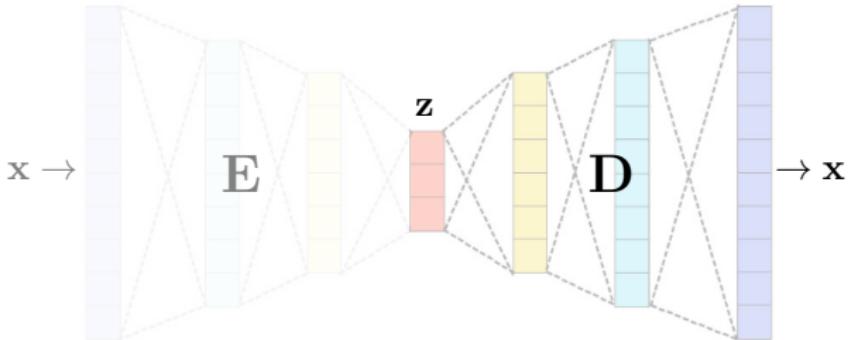
- The **bottleneck** prevents trivial solutions. *forza la rete a comprimere le informazioni*
- The **task** is not important here: it is always reconstruction.  
Once the AE is trained, we are interested in the structure of the **latent space** and in using  $E, D$  for new tasks.

# Manifold hypothesis



The decoder performs a mapping from a low-dimensional **latent space** to a high-dimensional **embedding** of observed data.

# Manifold hypothesis



The decoder performs a mapping from a low-dimensional **latent space** to a high-dimensional **embedding** of observed data.

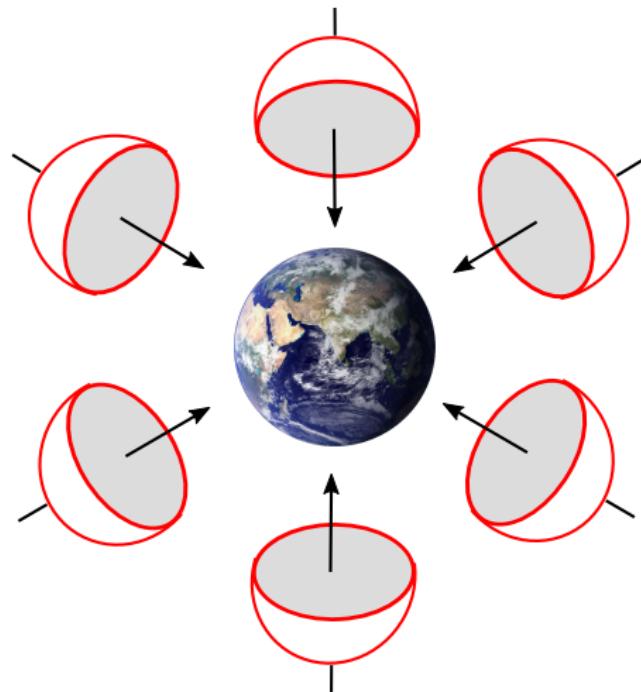
The latent space is Euclidean. **quindi è uno spazio vettoriale**

The data embedding space is curved (**manifold hypothesis**).

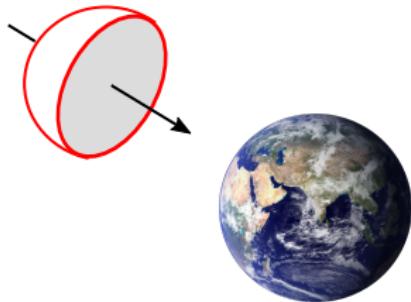
ovvero non è uno spazio vettoriale, ma contiene molte dimensioni.  
Quindi quello che si sta cercando di fare è ricostruire un dato posto su uno spazio vettoriale, portandolo ad alta dimensionalità

# Manifolds

Manifolds are unions of **charts**:



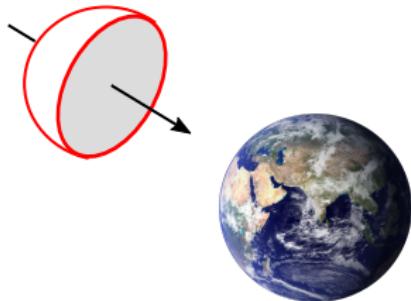
## 2D manifolds (surfaces)



chart

Each **chart** can be seen as a mapping  $\phi : \mathbb{R}^2 \rightarrow \mathcal{S} \subset \mathbb{R}^3$ .

## 2D manifolds (surfaces)

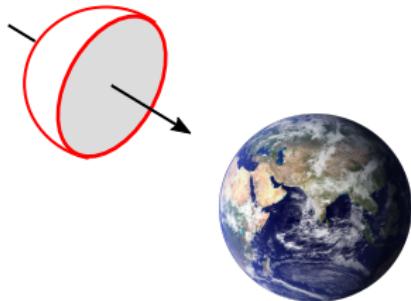


chart

Each **chart** can be seen as a mapping  $\phi : \mathbb{R}^2 \rightarrow \mathcal{S} \subset \mathbb{R}^3$ .

We require  $\phi$  to be **smooth** and **invertible** (diffeomorphism).

## 2D manifolds (surfaces)



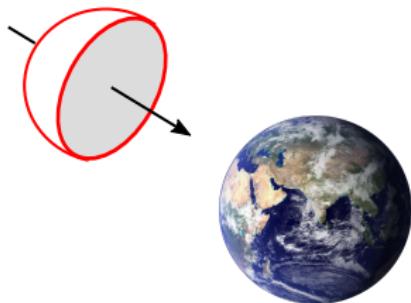
chart

Each **chart** can be seen as a mapping  $\phi : \mathbb{R}^2 \rightarrow \mathcal{S} \subset \mathbb{R}^3$ .

We require  $\phi$  to be **smooth** and **invertible** (diffeomorphism).

- The domain of  $\phi$  is the **parametric space** and is Euclidean.

## 2D manifolds (surfaces)



chart

Each **chart** can be seen as a mapping  $\phi : \mathbb{R}^2 \rightarrow \mathcal{S} \subset \mathbb{R}^3$ .

We require  $\phi$  to be **smooth** and **invertible** (diffeomorphism).

- The domain of  $\phi$  is the **parametric space** and is Euclidean.
- The image of  $\phi$  is the **embedding** and is a surface.

# Manifolds

Manifolds can be  $k$ -dimensional, meaning that we have charts:

$$\phi : \mathbb{R}^k \rightarrow \mathcal{M} \subset \mathbb{R}^d \quad \text{with } k < d$$

# Manifolds

Manifolds can be  $k$ -dimensional, meaning that we have charts:

$$\phi : \mathbb{R}^k \rightarrow \mathcal{M} \subset \mathbb{R}^d \quad \text{with } k < d$$

noi conosciamo  $d$  ma non  $k$   
(sarebbe la dimensione del  
latent code)

The parametrization is **not unique**:

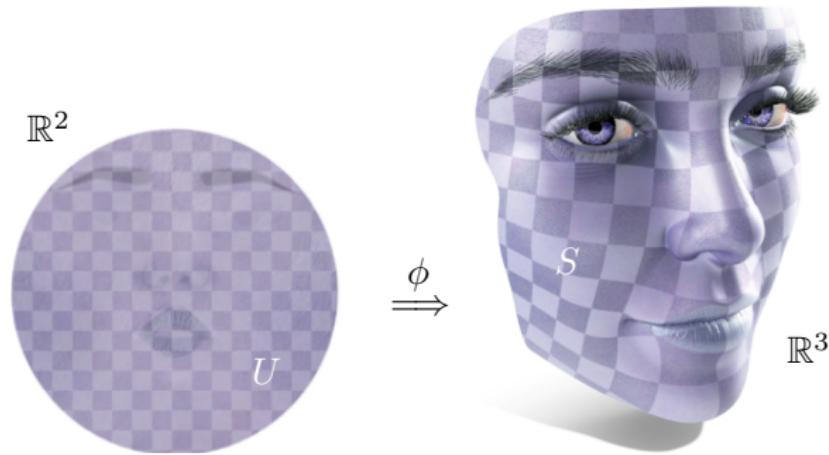


Figure by Keenan Crane

# Manifolds

Manifolds can be  $k$ -dimensional, meaning that we have charts:

$$\phi : \mathbb{R}^k \rightarrow \mathcal{M} \subset \mathbb{R}^d \quad \text{with } k < d$$

The parametrization is **not unique**:

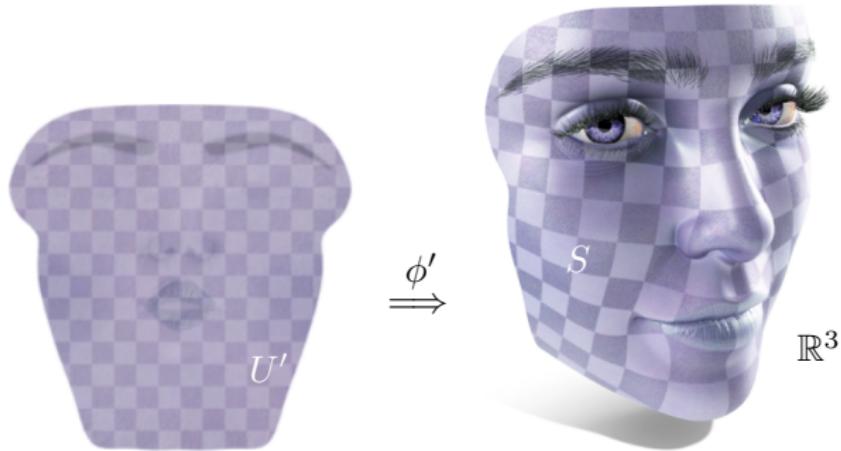


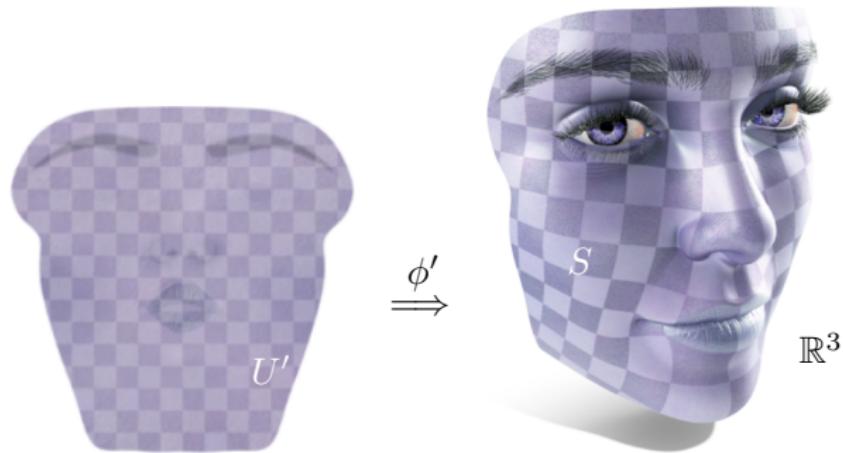
Figure by Keenan Crane

# Manifolds

Manifolds can be  $k$ -dimensional, meaning that we have charts:

$$\phi : \mathbb{R}^k \rightarrow \mathcal{M} \subset \mathbb{R}^d \quad \text{with } k < d$$

The parametrization is **not unique**:

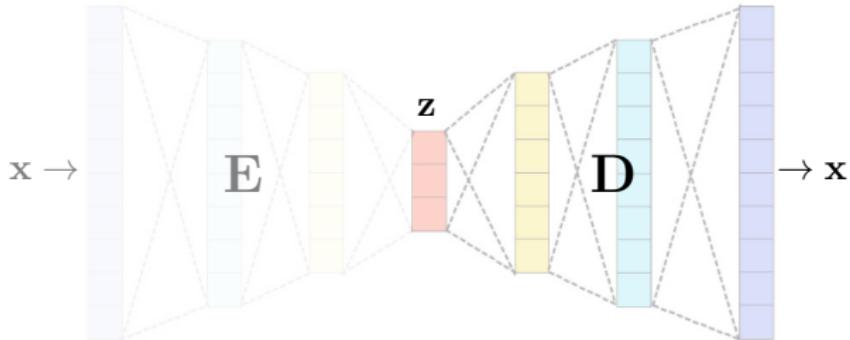


However, all encode exactly **the same** geometric information.



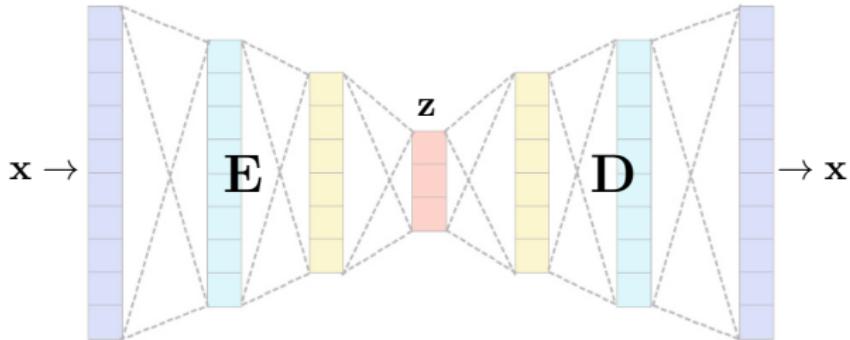
Figure by Keenan Crane

# Manifolds and generative models



The decoder  $D : \mathbb{R}^k \rightarrow \mathbb{R}^d$  is a chart from the latent space spanned by the codes  $z$  to the data space of the inputs  $x$ .

# Manifolds and generative models

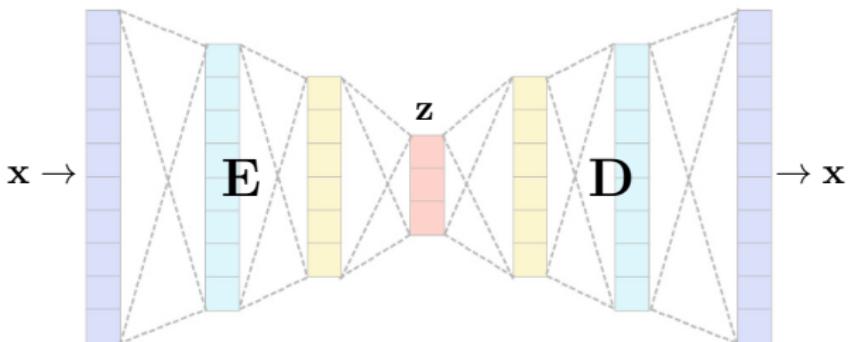


The decoder  $D : \mathbb{R}^k \rightarrow \mathbb{R}^d$  is a chart from the latent space spanned by the codes  $\mathbf{z}$  to the data space of the inputs  $\mathbf{x}$ .

It is differentiable.

It is invertible via the encoder  $E$ .

# Manifolds and generative models



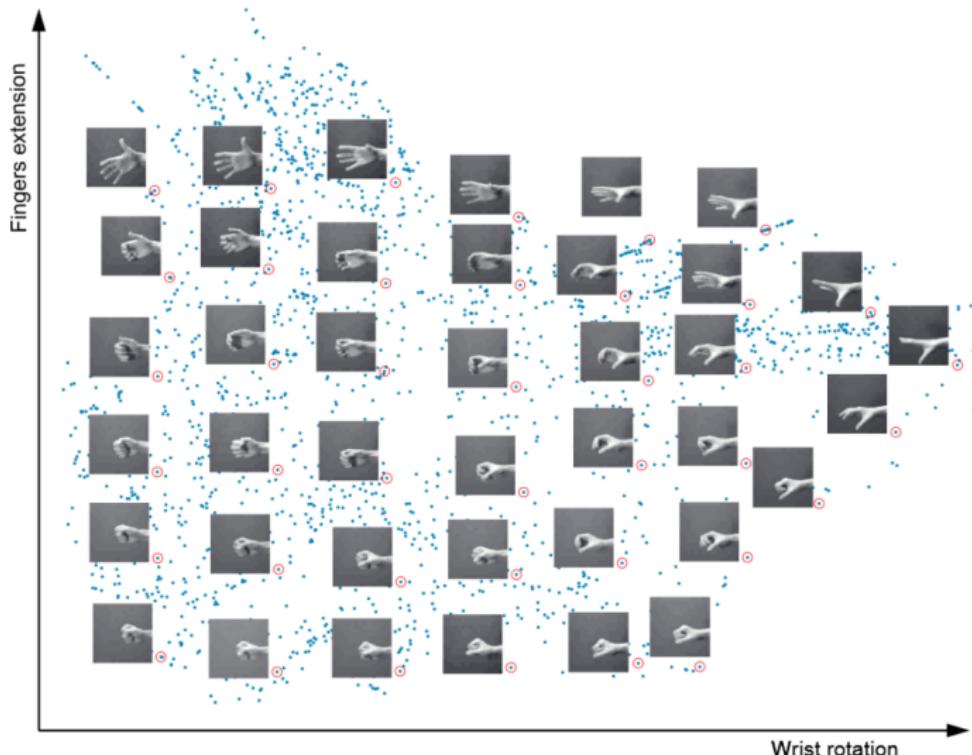
The decoder  $D : \mathbb{R}^k \rightarrow \mathbb{R}^d$  is a chart from the latent space spanned by the codes  $\mathbf{z}$  to the data space of the inputs  $\mathbf{x}$ .

**It is differentiable.**

**It is invertible via the encoder  $E$ .**

PCA puts the data on a **linear** (flat) manifold, since  $D$  simply performs a linear combination of orthogonal vectors.

# Manifolds and generative models



# Variational autoencoders (VAE)

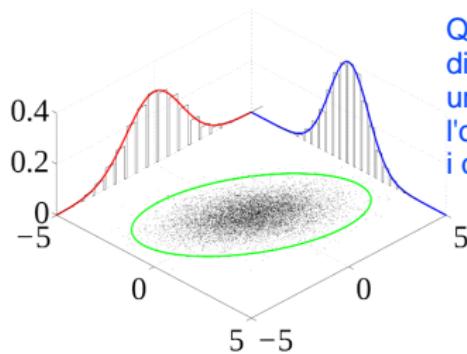
Autoencoders can map similar inputs to far regions of the latent space.

# Variational autoencoders (VAE)

Autoencoders can map similar inputs to far regions of the latent space.

A **variational** autoencoder constructs a probability distribution on the latent space. **quindi un autoencoder deterministico non produce nessuna distribuzione**

- The data is seen as a **sampling** of the learned distribution.
- The distribution is fixed and decided **a priori** (e.g. Gaussian).



Quindi, invece di cercare una distribuzione, ne viene imposta una nel latent code con l'obiettivo di imparare a mappare i dati in input nella distribuzione

# Entropy and divergence

The information carried by an event  $\mathbf{x}$  can be quantified as:

$$I(\mathbf{x}) = -\log p(\mathbf{x})$$

# Entropy and divergence

The information carried by an event  $\mathbf{x}$  can be quantified as:

$$I(\mathbf{x}) = -\log p(\mathbf{x})$$

The average information encoded in  $p$  is its [entropy](#):

$$H(p) = - \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x})$$

# Entropy and divergence

The information carried by an event  $x$  can be quantified as:

$$I(x) = -\log p(x)$$

indica la capacità informativa di un  
evento. Più l'evento è raro ( $P(x) \approx 0$ )  
più l'informazione fa notizia

The average information encoded in  $p$  is its **entropy**:

$$H(p) = - \sum_x p(x) \log p(x)$$

la media pesata  
dell'informazione data dagli  
eventi

Given two distributions  $p$  and  $q$ , the **Kullback-Leibler divergence**:

$$KL(p\|q) \approx H(q) - H(p)$$

misura quanto la  
distribuzione  $q(x)$  si discosta  
dall'altra distribuzione  $p(x)$

measures their dissimilarity in terms of their entropy.

# Entropy and divergence

The information carried by an event  $\mathbf{x}$  can be quantified as:

$$I(\mathbf{x}) = -\log p(\mathbf{x})$$

The average information encoded in  $p$  is its **entropy**:

$$H(p) = - \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x})$$

Given two distributions  $p$  and  $q$ , the **Kullback-Leibler divergence**:

$$KL(p\|q) \approx - \sum_{\mathbf{x}} q(\mathbf{x}) \log q(\mathbf{x}) + \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x})$$

measures their dissimilarity in terms of their entropy.

# Entropy and divergence

The information carried by an event  $\mathbf{x}$  can be quantified as:

$$I(\mathbf{x}) = -\log p(\mathbf{x})$$

The average information encoded in  $p$  is its **entropy**:

$$H(p) = - \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x})$$

Given two distributions  $p$  and  $q$ , the **Kullback-Leibler divergence**:

$$KL(p\|q) = - \sum_{\mathbf{x}} p(\mathbf{x}) \log q(\mathbf{x}) + \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x}) \geq 0$$

measures their dissimilarity in terms of their entropy.

rappresenta l'entropia incrociata tra le distribuzioni

rappresenta l'entropia della distribuzione  $p(\mathbf{x})$

# Entropy and divergence

The information carried by an event  $\mathbf{x}$  can be quantified as:

$$I(\mathbf{x}) = -\log p(\mathbf{x})$$

The average information encoded in  $p$  is its **entropy**:

$$H(p) = - \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x})$$

Given two distributions  $p$  and  $q$ , the **Kullback-Leibler divergence**:

$$KL(p\|q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \geq 0$$

measures their dissimilarity in terms of their entropy.

# Entropy and divergence

The information carried by an event  $\mathbf{x}$  can be quantified as:

$$I(\mathbf{x}) = -\log p(\mathbf{x})$$

The average information encoded in  $p$  is its **entropy**:

$$H(p) = - \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x})$$

Given two distributions  $p$  and  $q$ , the **Kullback-Leibler divergence**:

$$KL(p\|q) = - \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \geq 0$$

measures their dissimilarity in terms of their entropy.

## Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

## Variational inference

In our scenario:  $\textcolor{green}{x}$  is a given data point, and  $\textcolor{red}{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\textcolor{red}{z}|\textcolor{green}{x})$$

## Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})}$$

## Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

quindi l'encoder di un VAE



non produce un mapping,

ma una distribuzione

di probabilità dei possibili

latent codes



$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

teorema di Bayes

However, computing:

$$p_{\theta}(\mathbf{x}) = \int p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})d\mathbf{z}$$

is a high-dimensional **intractable** integral over the entire **latent space**.

ergo, non se po fa.  
Quindi...

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation:

$$q_{\phi}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

Quindi in pratica, suggerisce di allenare una rete a trovare una distribuzione simile a quella che cerchiamo...

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}|\mathbf{x}))$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

...supponendo di avere la distribuzione data dalla rete neurale, possiamo calcolare una loss con la KL divergency

## Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} - \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{z}|\mathbf{x})}{q_{\phi}(\mathbf{z}|\mathbf{x})}$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} - \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})/p_{\theta}(\mathbf{x})}{q_{\phi}(\mathbf{z}|\mathbf{x})}$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} - \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} \frac{1}{p_{\theta}(\mathbf{x})}$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} - \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \left( \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} + \log \frac{1}{p_{\theta}(\mathbf{x})} \right)$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} - \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \left( \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} - \log p_{\theta}(\mathbf{x}) \right)$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} - \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} + \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log p_{\theta}(\mathbf{x})$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

## Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} - \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} + \log p_{\theta}(\mathbf{x}) \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x})$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} - \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} + \log p_{\theta}(\mathbf{x}) \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x})$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

## Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \min_{\phi, \theta} - \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} + \log p_{\theta}(\mathbf{x})$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \max_{\phi, \theta} \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} - \log p_{\theta}(\mathbf{x})$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

# Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \max_{\phi, \theta} \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} - \cancel{\log p_{\theta}(\mathbf{x})}$$

relax 

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$ .

## Variational inference

In our scenario:  $\mathbf{x}$  is a given data point, and  $\mathbf{z}$  is a latent code.

We define a parametric **probabilistic encoder** as the distribution:

$$p_{\theta}(\mathbf{z}|\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{x})} = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{p_{\theta}(\mathbf{x})}$$

Instead, compute an approximation  $q_{\phi^*}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$  with:

$$\phi^* = \arg \max_{\phi, \theta} \text{ELBO}_{\phi, \theta}(\mathbf{x}) - \cancel{\log p_{\theta}(\mathbf{x})}^{\text{relax}}$$

where  $q_{\phi}(\mathbf{z}|\mathbf{x})$  is a neural network with weights  $\phi$  and:

$$\text{ELBO}_{\phi, \theta}(\mathbf{x}) = \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} \leq \log p_{\theta}(\mathbf{x})$$

is called the **Evidence variational Lower BOund**.

# Variational autoencoder (VAE)

$$\max_{\phi, \theta} ELBO_{\phi, \theta}(\mathbf{x})$$

# Variational autoencoder (VAE)

$$\max_{\phi, \theta} \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})}$$

# Variational autoencoder (VAE)

$$\max_{\phi, \theta} \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})}$$

# Variational autoencoder (VAE)

$$\max_{\phi, \theta} \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \left( \log p_{\theta}(\mathbf{x}|\mathbf{z}) + \log \frac{p_{\theta}(\mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} \right)$$

# Variational autoencoder (VAE)

$$\max_{\phi, \theta} \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log p_{\theta}(\mathbf{x}|\mathbf{z}) + \sum_{\mathbf{z}} q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})}$$

# Variational autoencoder (VAE)

$$\max_{\phi, \theta} \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z}) - KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))$$

# Variational autoencoder (VAE)

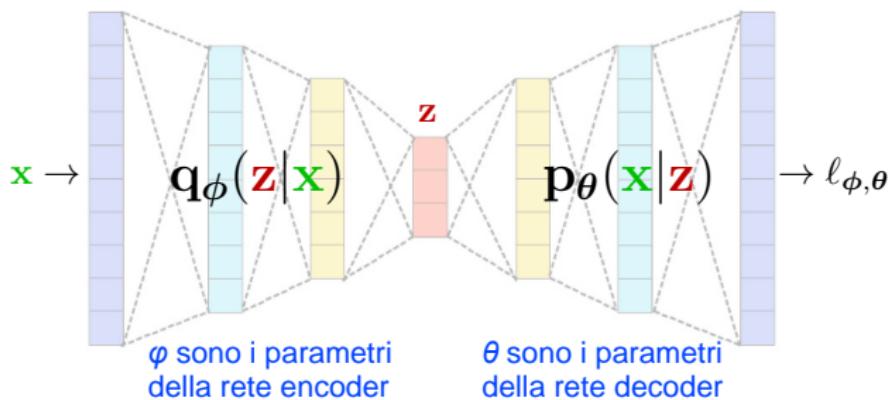
$$\max_{\phi, \theta} \underbrace{\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{likelihood of observing } \mathbf{x} \text{ given } \mathbf{z}} - KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))$$

# Variational autoencoder (VAE)

$$\max_{\phi, \theta} \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})}_{\text{likelihood of observing } \mathbf{x} \text{ given } \mathbf{z}} - \underbrace{KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))}_{\text{ensures } q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z})}$$

# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{likelihood of observing } \mathbf{x} \text{ given } \mathbf{z}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{ensures } q_{\phi}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z})} \text{ che cos'è } p_{\theta}(z) \dots$$



**ATTENZIONE:** uno stesso sample ogni volta che viene dato in pasto all'encoder produrrà una distribuzione probabilistica differente

# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{likelihood of observing } \mathbf{x} \text{ given } \mathbf{z}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{ensures } q_{\phi}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z})}$$

The prior over the latent variables is Gaussian and has **no free parameters**:

$$p(\mathbf{z}) = \mathcal{N}_{\mathbf{0}, \mathbf{I}}(\mathbf{z})$$

# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{likelihood of observing } \mathbf{x} \text{ given } \mathbf{z}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{ensures } q_{\phi}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z})}$$

The prior over the latent variables is Gaussian and has **no free parameters**:

$$p(\mathbf{z}) = \mathcal{N}_{\mathbf{0}, \mathbf{I}}(\mathbf{z})$$

The **probabilistic encoder** also generates a Gaussian distribution:

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}_{\boldsymbol{\mu}, \boldsymbol{\sigma}}(\mathbf{z})$$

# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{likelihood of observing } \mathbf{x} \text{ given } \mathbf{z}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{ensures } q_{\phi}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z})}$$

The prior over the latent variables is Gaussian and has **no free parameters**:

$$p(\mathbf{z}) = \mathcal{N}_{\mathbf{0}, \mathbf{I}}(\mathbf{z})$$

The **probabilistic encoder** also generates a Gaussian distribution:

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}_{\boldsymbol{\mu}, \boldsymbol{\sigma}}(\mathbf{z})$$



# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{likelihood of observing } \mathbf{x} \text{ given } \mathbf{z}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{ensures } q_{\phi}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z})}$$

The prior over the latent variables is Gaussian and has **no free parameters**:

$$p(\mathbf{z}) = \mathcal{N}_{\mathbf{0}, \mathbf{I}}(\mathbf{z})$$

The **probabilistic encoder** also generates a Gaussian distribution:

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}_{\boldsymbol{\mu}, \boldsymbol{\sigma}}(\mathbf{z})$$

where  $\boldsymbol{\mu}, \boldsymbol{\sigma}$  are functions of the input  $\mathbf{x}$  and the network parameters  $\phi$ .

The probabilistic encoder outputs  $\boldsymbol{\mu}, \boldsymbol{\sigma}$ , not  $\mathbf{z}$ .

# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{likelihood of observing } \mathbf{x} \text{ given } \mathbf{z}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{ensures } q_{\phi}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z})}$$

The prior over the latent variables is Gaussian and has **no free parameters**

$$p(\mathbf{z}) = \mathcal{N}_{\mathbf{0}, \mathbf{I}}(\mathbf{z})$$

**IMPORTANTE**

The **probabilistic encoder** also generates a Gaussian distribution:

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}_{\mu, \sigma}(\mathbf{z})$$

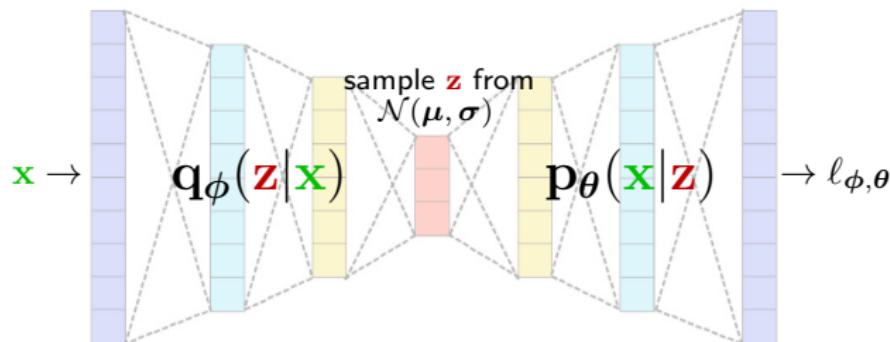
where  $\mu, \sigma$  are functions of the input  $\mathbf{x}$  and the network parameters  $\phi$ .

The probabilistic encoder outputs  $\mu, \sigma$ , not  $\mathbf{z}$ .

**Using Gaussians, the KL term has a closed form.**

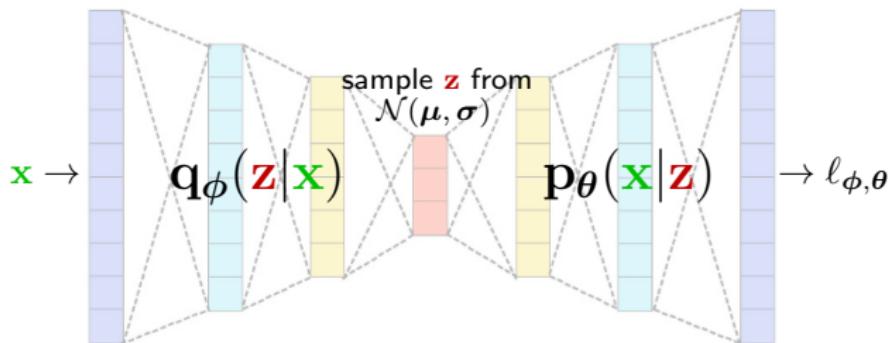
# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{likelihood of observing } \mathbf{x} \text{ given } \mathbf{z}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{ensures } q_{\phi}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z})}$$



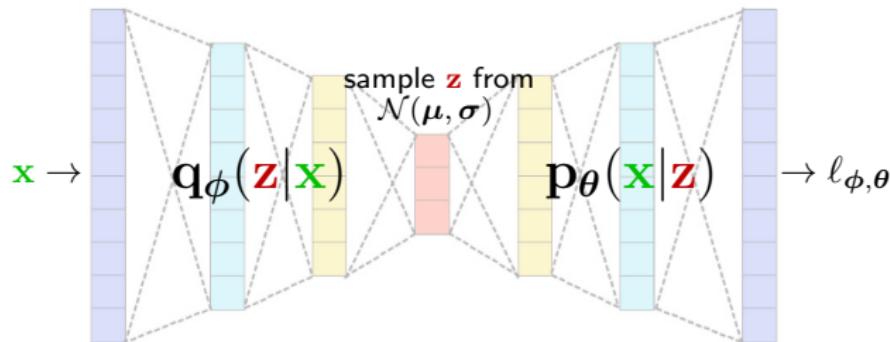
# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{reconstruction loss} \atop (\text{choose your own})} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{regularizer}}$$



# Variational autoencoder (VAE)

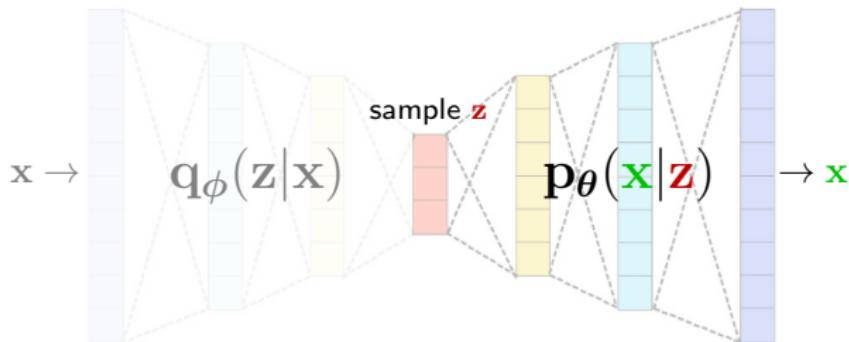
$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{reconstruction loss} \atop \text{(choose your own)}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{regularizer}}$$



To distinguish from VAE, classical AE are also called **deterministic AE**.

# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{reconstruction loss} \atop \text{(choose your own)}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{regularizer}}$$

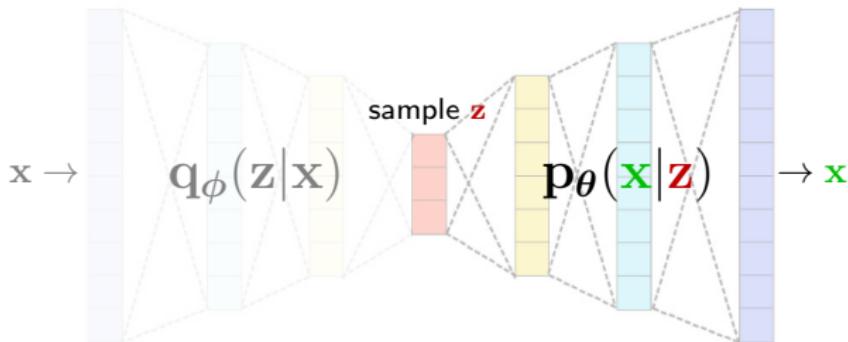


To distinguish from VAE, classical AE are also called **deterministic AE**.

**Generation:** Sample  $\mathbf{z}$  according to the learned distribution.

# Variational autoencoder (VAE)

$$\ell_{\phi, \theta} = \underbrace{-\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \log p_{\theta}(\mathbf{x}|\mathbf{z})}_{\text{reconstruction loss} \atop \text{(choose your own)}} + \underbrace{KL(q_{\phi}(\mathbf{z}|\mathbf{x}) \| p_{\theta}(\mathbf{z}))}_{\text{regularizer}}$$



To distinguish from VAE, classical AE are also called **deterministic AE**.

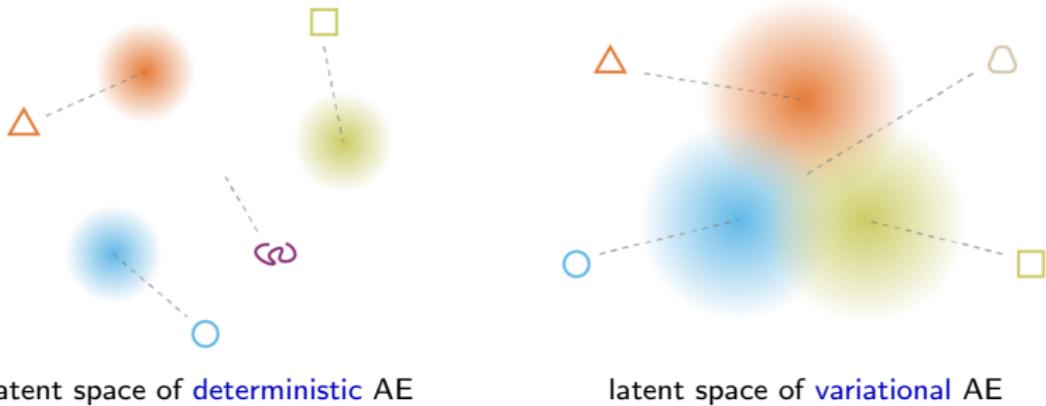
**Generation:** Sample  $\mathbf{z}$  according to the learned distribution.



In principle, **also the decoder outputs a probability distribution.** reparametrization trick

If the reconstruction loss does not need a distribution, take  $\mathbf{x} := \mu$ .

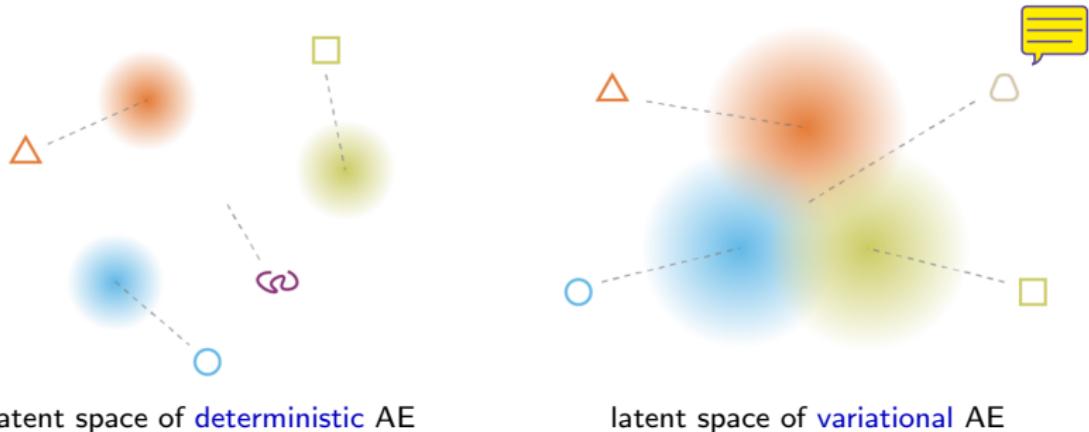
# Regularizing effect



The requirement  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  leads to a compact latent space.

Small distances  $\|\mathbf{z}_1 - \mathbf{z}_2\|_2 \Rightarrow$  small changes in the decoded output.

# Regularizing effect



latent space of deterministic AE

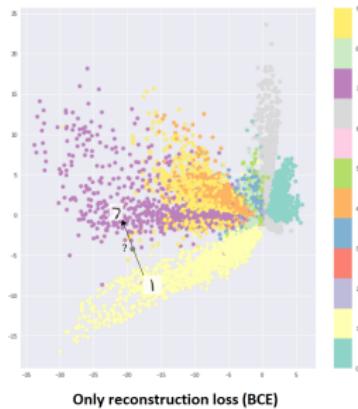
latent space of variational AE

The requirement  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  leads to a compact latent space.

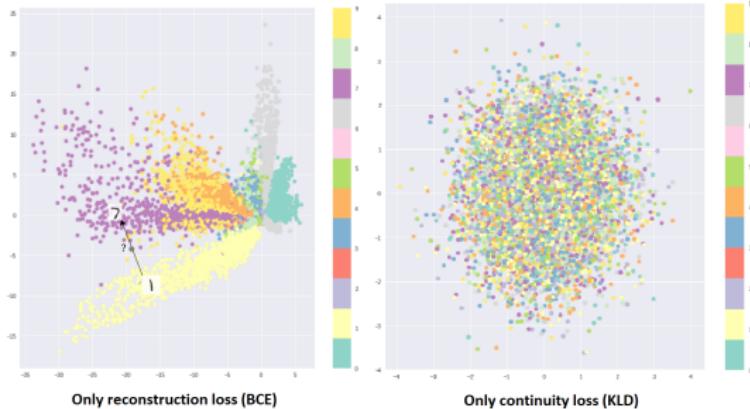
Small distances  $\|\mathbf{z}_1 - \mathbf{z}_2\|_2 \Rightarrow$  small changes in the decoded output.

A deterministic AE might generate arbitrary noise.

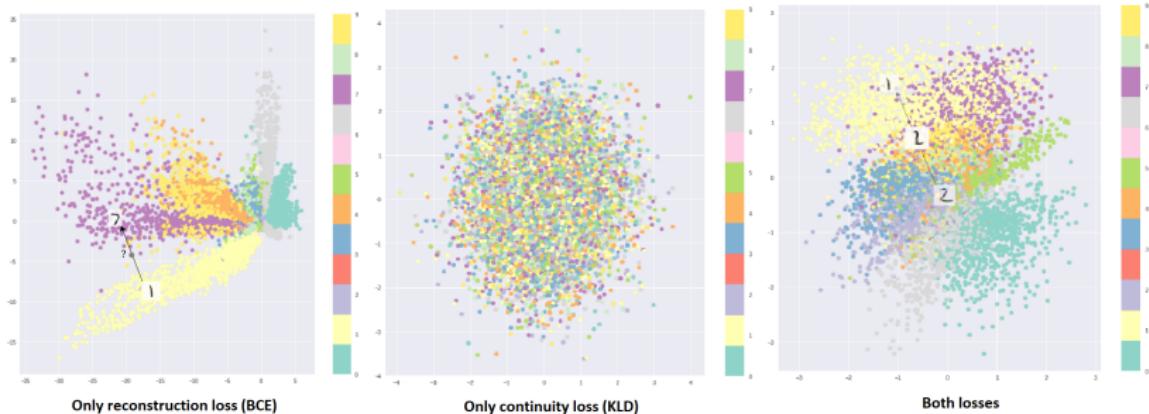
# Regularizing effect



# Regularizing effect



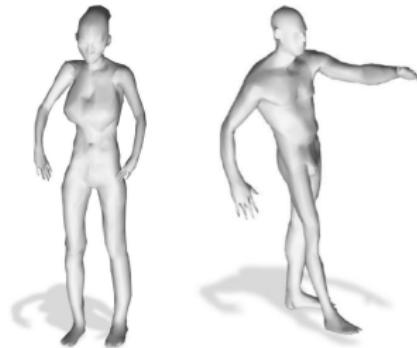
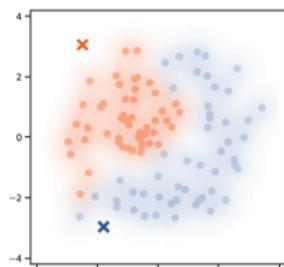
# Regularizing effect



The distributional regularizer brings **smoothness** in the learned space.

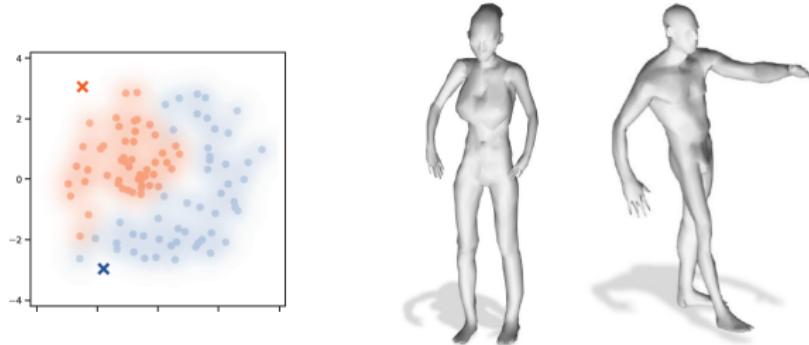
## Unlikely samples

Sampling on the **boundaries** of the learned distribution leads to the generation of **unlikely** samples:



## Unlikely samples

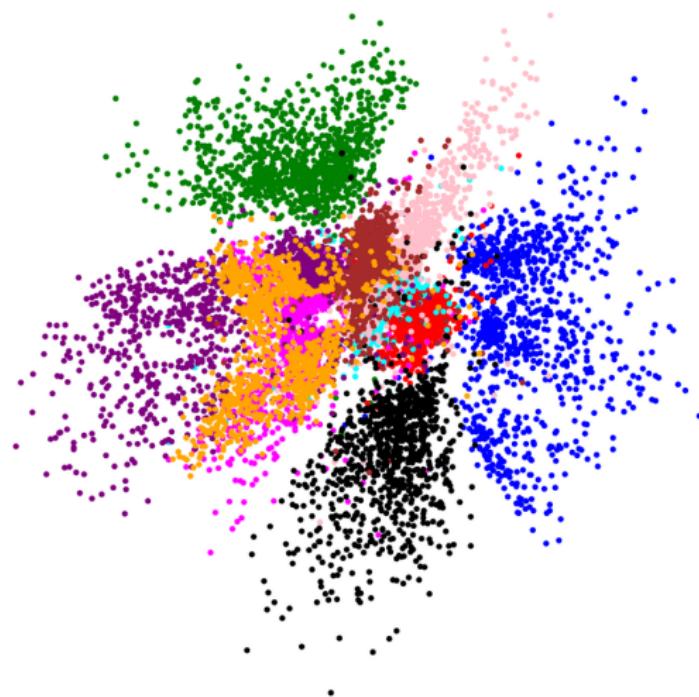
Sampling on the **boundaries** of the learned distribution leads to the generation of **unlikely** samples:



Another relevant phenomenon is **posterior collapse**, where the latents are ignored if the decoder is powerful enough to model the data perfectly.

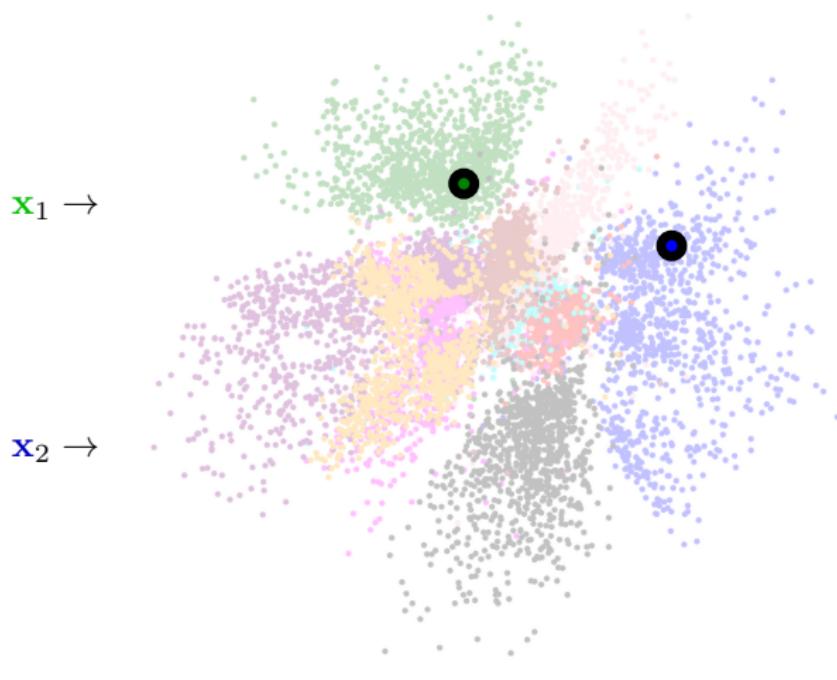
## VAE interpolation

Assume we have a trained VAE, colors correspond to different classes:



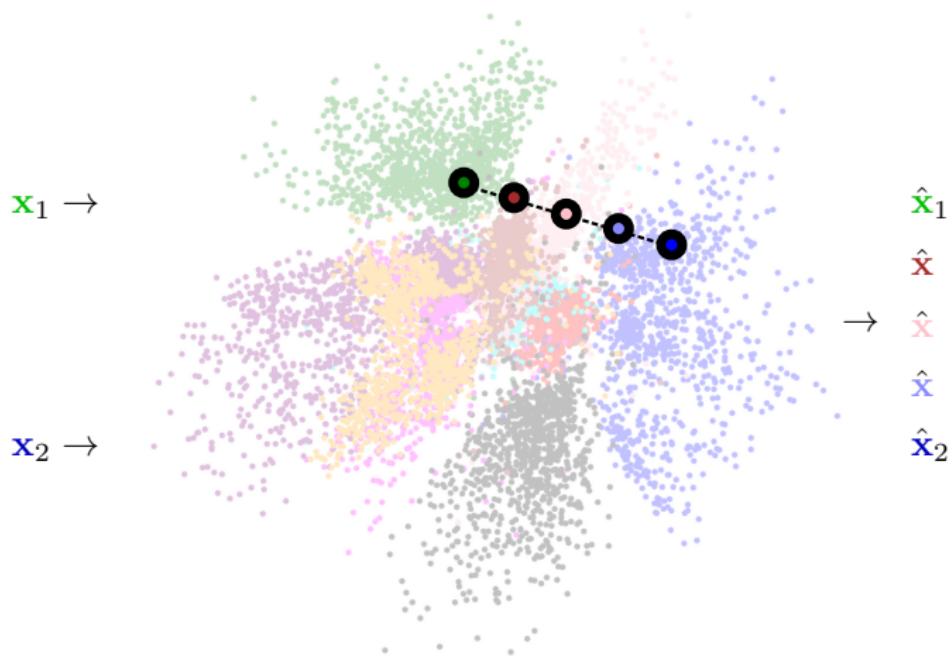
## VAE interpolation

Assume we have a trained VAE, colors correspond to different classes:



# VAE interpolation

Assume we have a trained VAE, colors correspond to different classes:



The interpolated samples are **not** part of the training set. quindi si produce un risultato nuovo

# VAE interpolation

7

1

# VAE interpolation

interpolazione da 7  
a 1 col dataset mnist



# VAE interpolation

6

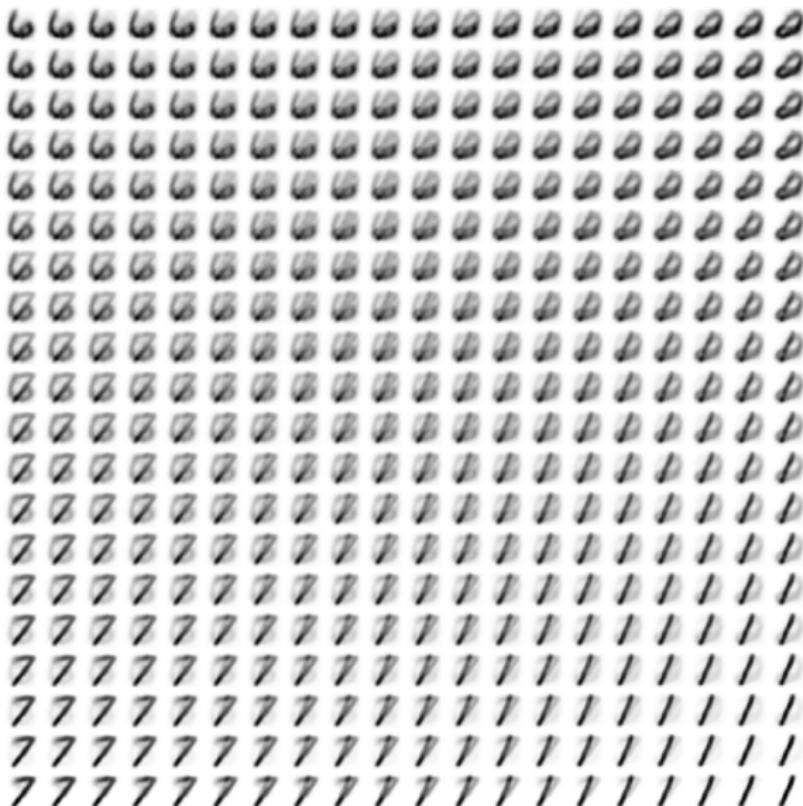
7 7 7 7 7 7 1 1 1 1 1 1 1 1 1 1 1 1

## VAE interpolation

## interpolazione da 7 a 6

## VAE interpolation

## Euclidean interpolation



questa è  
l'interpolazione nel  
vector space, che  
porta a risultati  
peggiori (AE)

## Example: Deformable 3D shapes



Cosmo, Norelli, Halimi, Kimmel, Rodolà, "LIMP: Learning Latent Shape Representations with Metric Preservation Priors", 2020

## Example: Deformable 3D shapes



Cosmo, Norelli, Halimi, Kimmel, Rodolà, "LIMP: Learning Latent Shape Representations with Metric Preservation Priors", 2020

## Example: Deformable 3D shapes



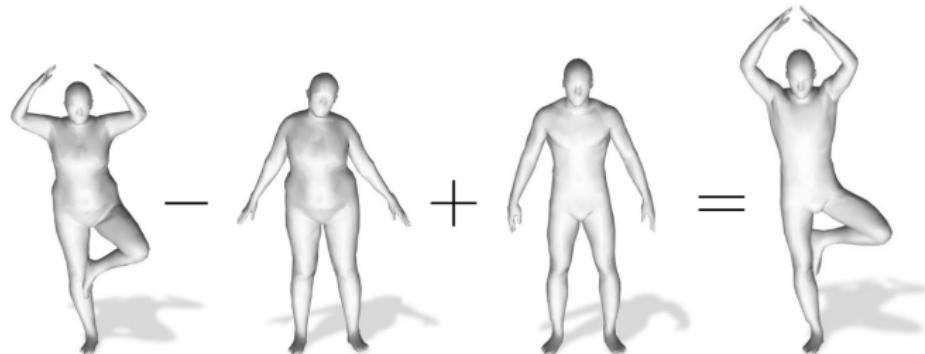
The latent space can be explored along different dimensions  
(disentanglement):



Cosmo, Norelli, Halimi, Kimmel, Rodolà, "LIMP: Learning Latent Shape Representations with Metric Preservation Priors", 2020

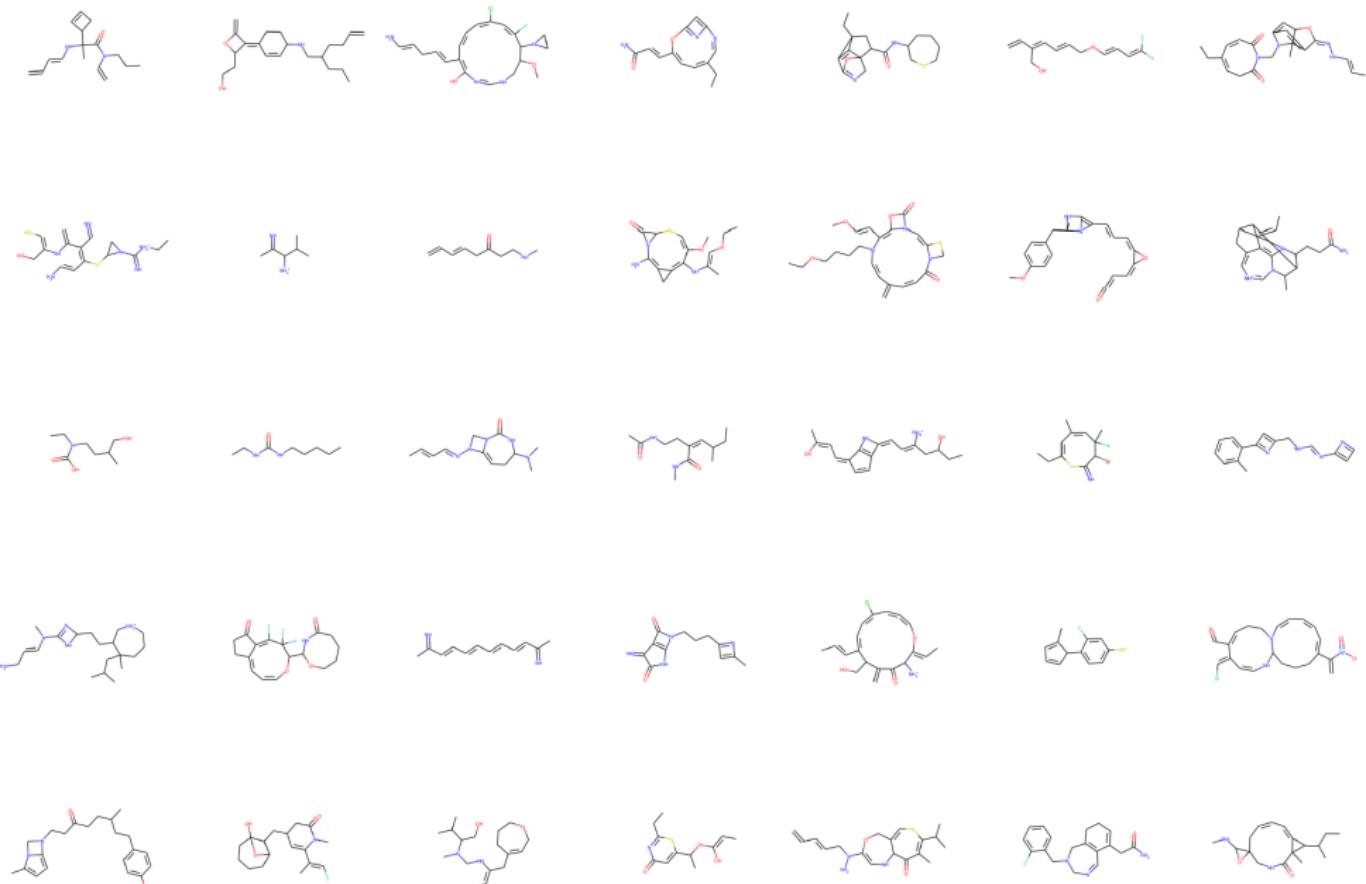
# Analogies

Algebraic manipulation of codes leads to the following:

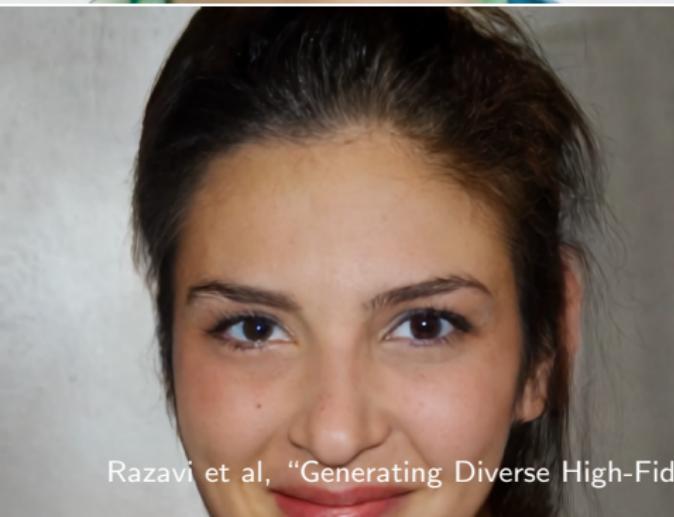


In NLP this is done by manipulating [word embeddings](#):

King – Man + Woman = Queen



Bresson, "A Two-Step Graph Convolutional Decoder for Molecule Generation", 2019



Razavi et al, "Generating Diverse High-Fidelity Images with VQ-VAE-2", 2019