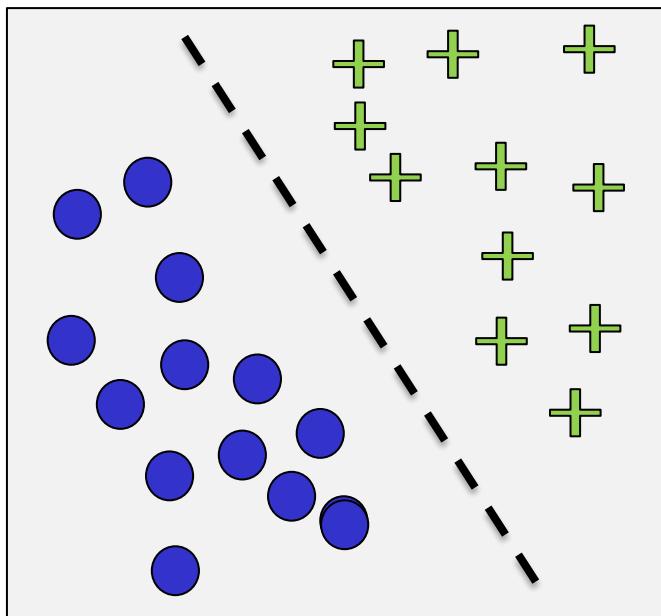


Réseaux de neurones
IFT 780

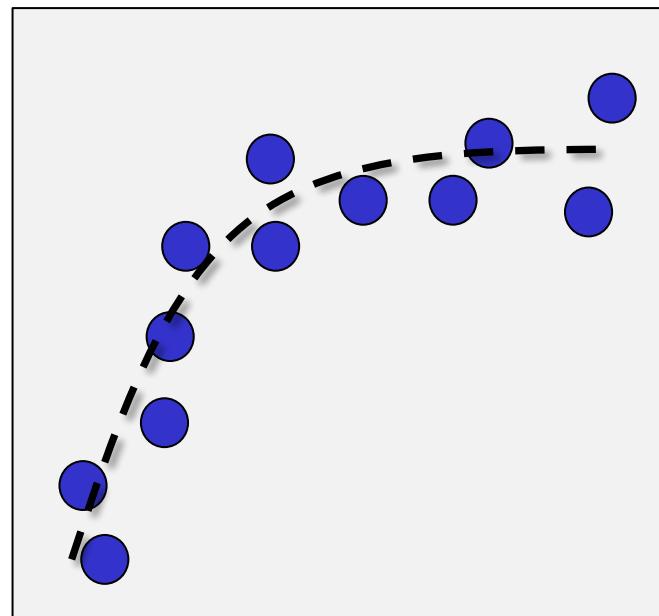
Modèles génératifs
Par
Pierre-Marc Jodoin

Jusqu'à présent : apprentissage supervisé

Classification



Régression



Jusqu'à présent : apprentissage supervisé

Segmentation



Description



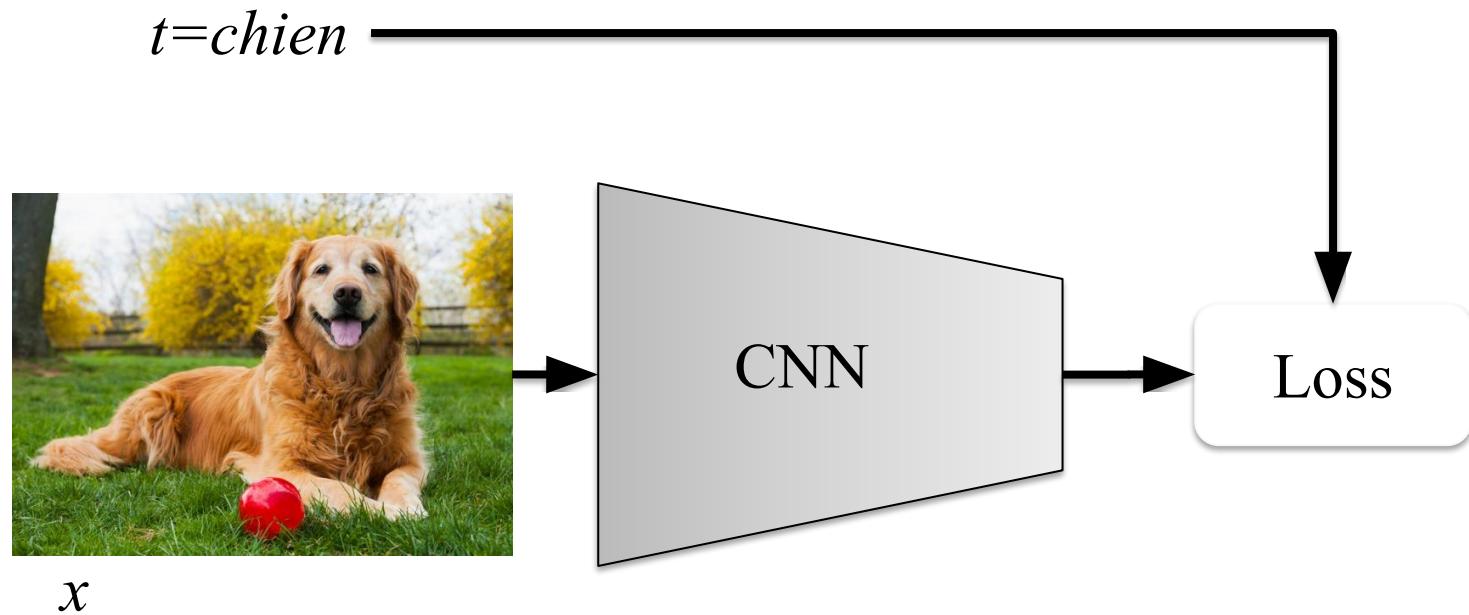
an elephant standing in a grassy field with
trees in the background

Apprentissage supervisé

Deux grandes familles d'applications

- **Classification** : la cible est un indice de classe $t \in \{1, \dots, K\}$
 - Exemple : reconnaissance de caractères
 - ✓ x : vecteur des intensités de tous les pixels de l'image
 - ✓ t : identité du caractère
- **Régression** : la cible est un nombre réel $t \in \mathbb{R}$
 - Exemple : prédiction de la valeur d'une action à la bourse
 - ✓ x : vecteur contenant l'information sur l'activité économique de la journée
 - ✓ t : valeur d'une action à la bourse le lendemain

Apprentissage supervisé avec CNN



Modèles discriminatifs vs. génératifs

$p(t|x)$

probabilité de la classe t
en sachant x

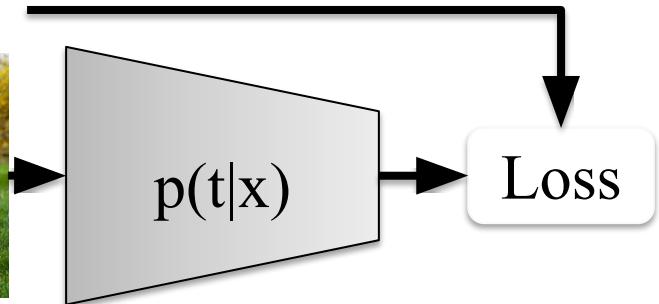
$p(x)$

probabilité de x

$t=chien$

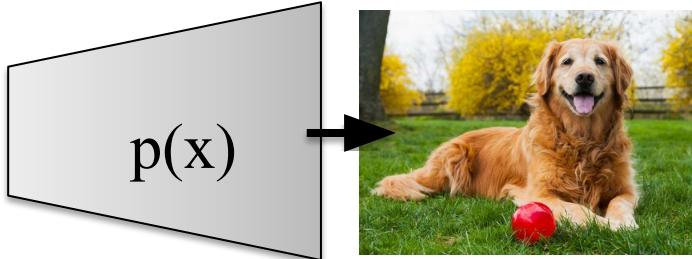


x



$p(t|x)$

Loss



$p(x)$

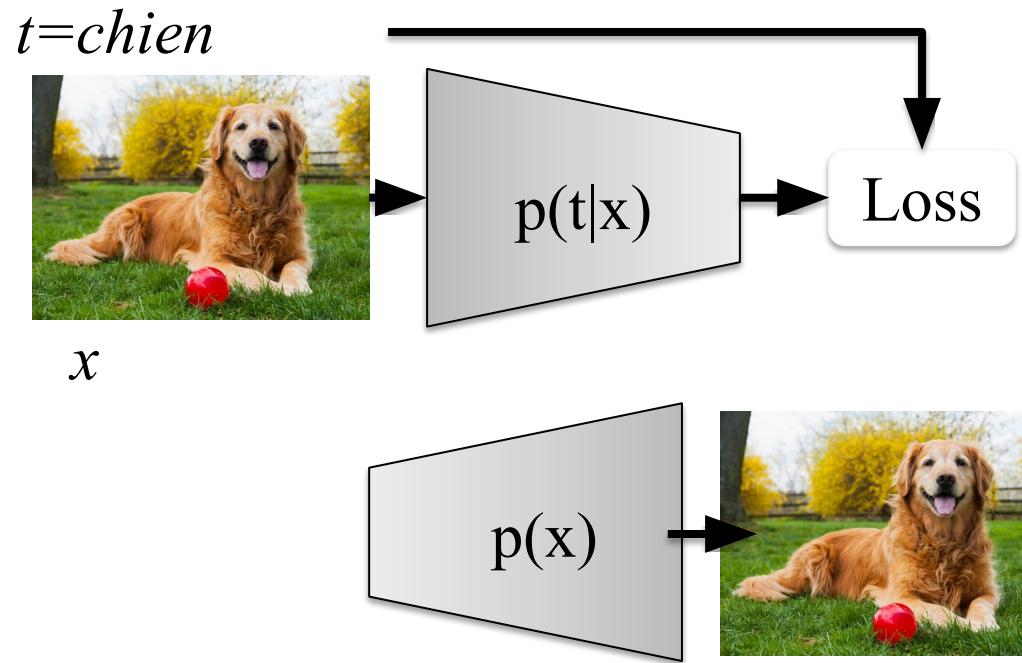
Modèles discriminatifs vs. génératifs

$p(t|x) \rightarrow p(\text{chien} | \text{})$

probabilité de la classe t
en sachant x

$p(x) \rightarrow p(\text{})$

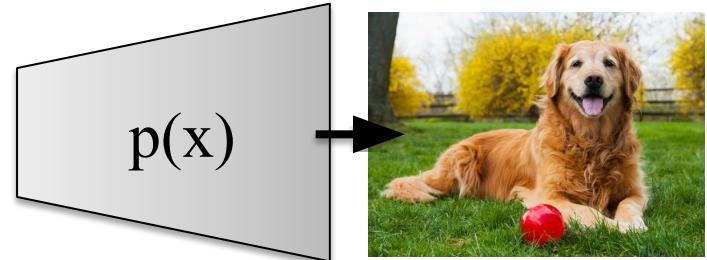
probabilité de x



Modèles génératifs

Comment apprendre $p(x)$?

$p(x) \rightarrow p(\text{dog})$
probabilité de x



Nous allons formuler $p(x)$ par un réseau de neurones $p(\vec{x}) = p_\theta(\vec{x})$

Selon un jeu de données $D = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N\}$

$$\theta^* = \operatorname{argmax}_\theta \prod_{i=1}^N p_\theta(\vec{x}_i)$$

$$= \operatorname{argmax}_\theta \sum_{i=1}^N \log p_\theta(\vec{x}_i)$$

Supervisé vs non supervisé

Apprentissage supervisé : il y a une cible

$$D = \{(x_1, t_1), (x_2, t_2), \dots, (x_N, t_N)\}$$

Apprentissage non-supervisé : la cible n'est pas fournie

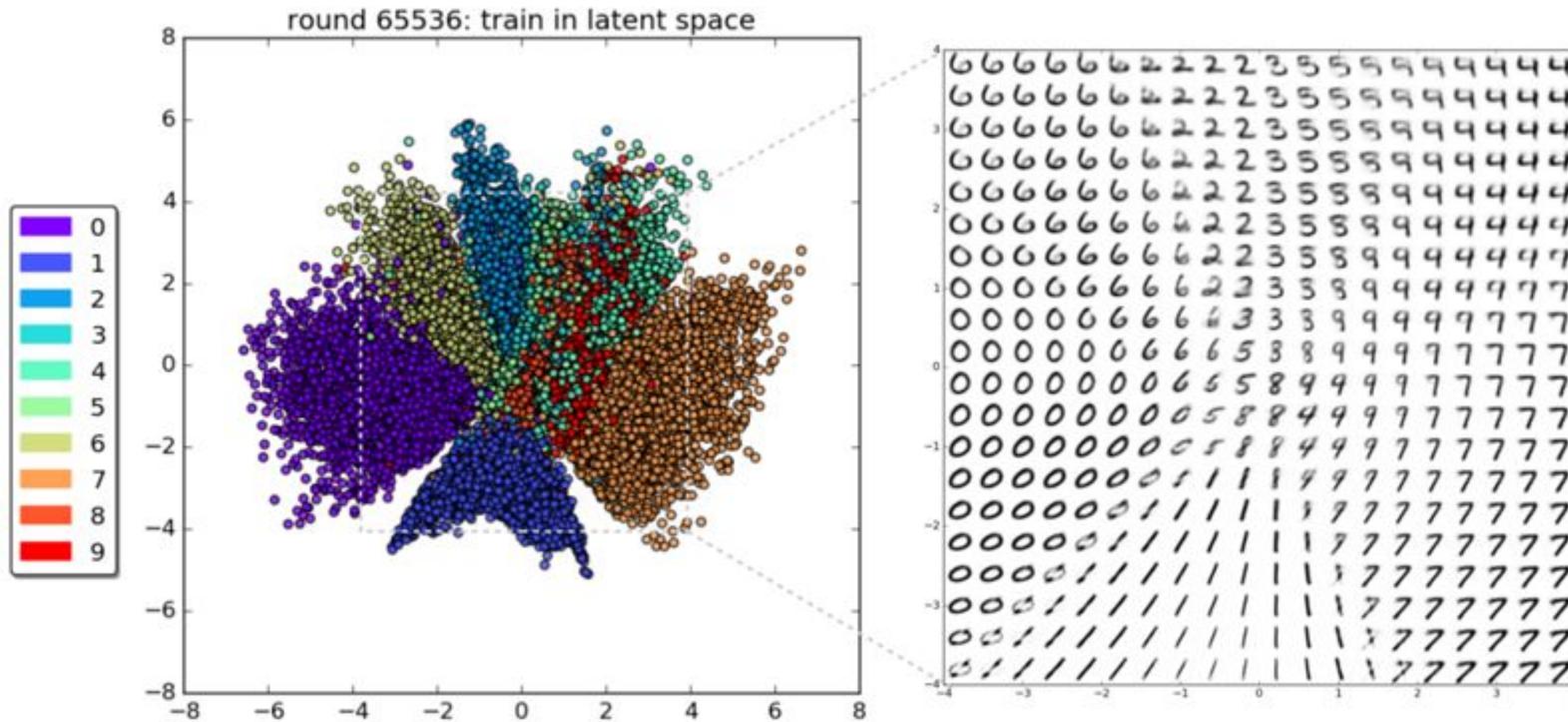
$$D = \{x_1, x_2, \dots, x_N\}$$

Apprentissage non supervisé

Comprendre la distribution sous-jacente de données **non-étiquetées**

Applications : clustering (p.e. k-means), visualization, comprehension, etc.

Exemple : visualization de la distribution des images MNIST

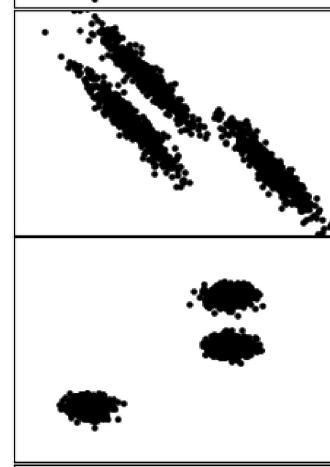
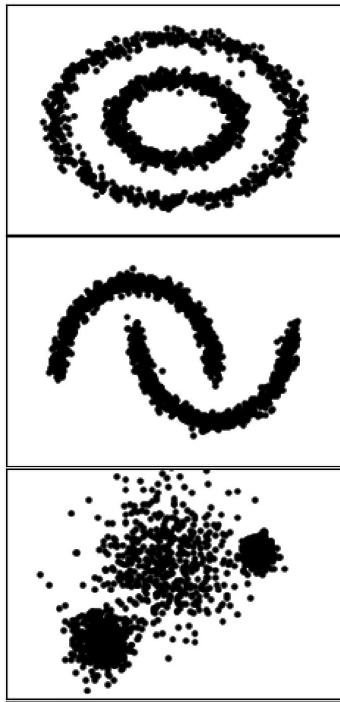


Apprentissage non supervisé

Souvent, l'apprentissage non-supervisé inclut un (ou des) **variables latentes**.

Variable latente: variable aléatoire non observée mais sous-jacente à la distribution des données

Ex: clustering = retrouver la variable latente “cluster”

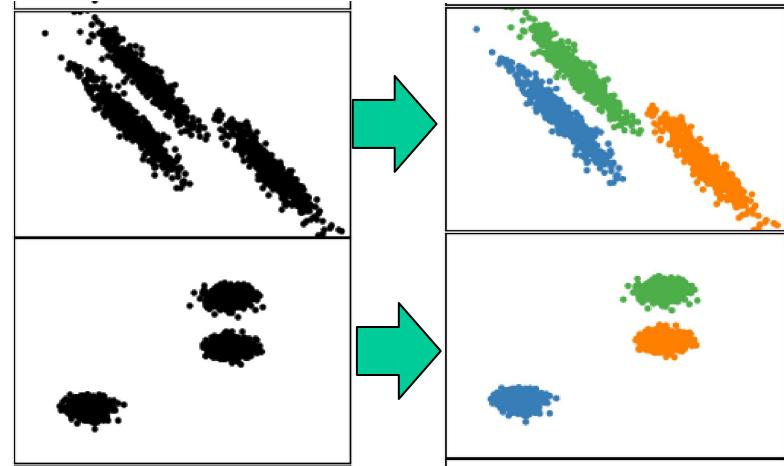
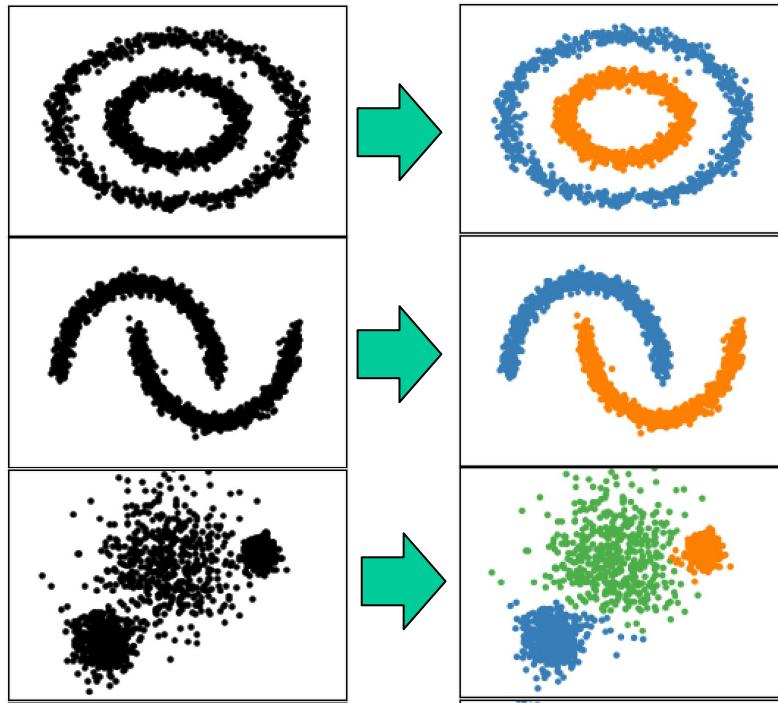


Apprentissage non supervisé

Souvent, l'apprentissage non-supervisé inclut un (ou des) **variables latentes**.

Variable latente: variable aléatoire non observée mais sous-jacente à la distribution des données

Ex: clustering = retrouver la variable latente “cluster”



Pourquoi une variable latente?

Plus facile de représenter $p(x, y)$, $p(x | y)$, $p(y)$
que $p(x)$

Apprentissage non supervisé

Variable latente: variable aléatoire non observée mais sous-jacente à la distribution des données

Rappel:

- si x_1 et x_2 sont des variables aléatoires indépendantes:
 $p(x_1, x_2) = p(x_1)(x_2)$ et $p(x_1|x_2) = p(x_1)$
- si x_1 est dépendant de x_2 , mais pas l'inverse:
 $p(x_1|x_2) \neq p(x_1)$ MAIS $p(x_2|x_1) = p(x_2)$
 $p(x_1, x_2) = p(x_1|x_2)p(x_2)$

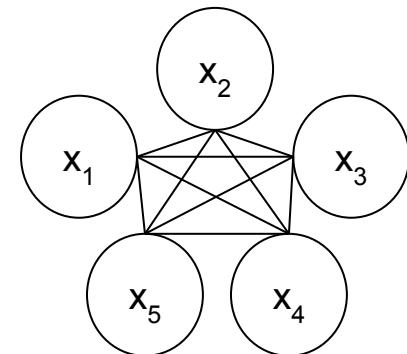
Exemple: x_1 = lire dans le jardin
 x_2 = il pleut dehors

Apprentissage non supervisé

Variable latente: variable aléatoire non observée mais sous-jacente à la distribution des données

Rappel:

si $\vec{x} = (x_1, x_2, x_3, x_4, x_5)$ sont des variables aléatoires dépendantes



$p(\vec{x}) = p(x_1, x_2, x_3, x_4, x_5)$, compliqué à calculer car il faut considérer chaque combinaison

Règle en chaîne des probabilités: $p(x_1, x_2, \dots, x_n) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$

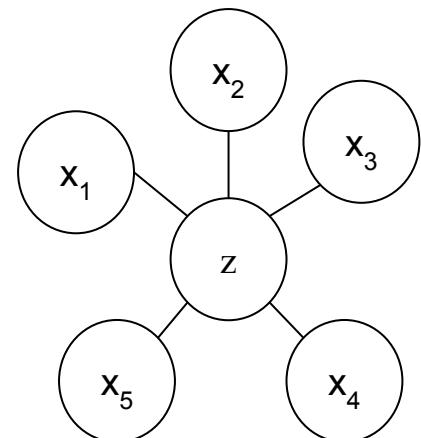
Pas si mal pour 5 variables, mais dans une image, chaque pixel est une variable !

Apprentissage non supervisé

Variable latente: variable aléatoire non observée mais sous-jacente à la distribution des données

Pour simplifier le problème on présuppose que les variables x_i dépendent d'une variable latente z et sont indépendantes entre elles

$$\begin{aligned} p(\vec{x}, z) &= p(x_1, x_2, x_3, x_4, x_5, z) \\ &= p(z)p(x_1|z)\dots p(x_5|z) \\ &= p(z) \prod_{i=1}^5 p(x_i|z) \\ &= p(z)p(\vec{x}|z) \end{aligned}$$



On peut toujours présumer que z existe !

$$\begin{aligned} p(\vec{x}) &= \int p(\vec{x}, z) dz \quad \text{=< marginalisation} \\ &= \int p(\vec{x}|z)p(z)dz \end{aligned}$$

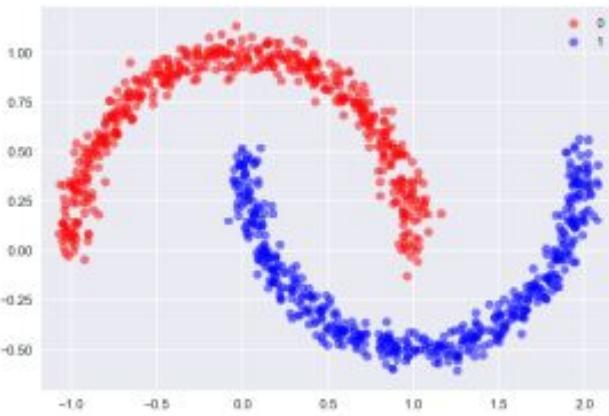
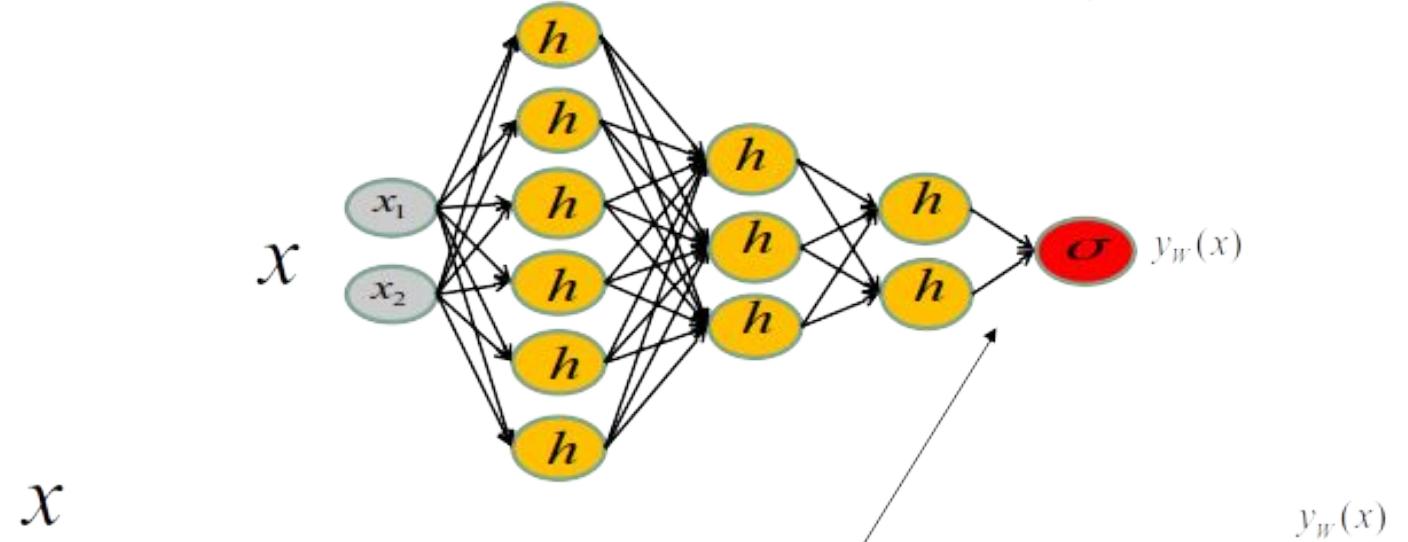
En apprentissage non-supervisé

nous nous appuierons sur

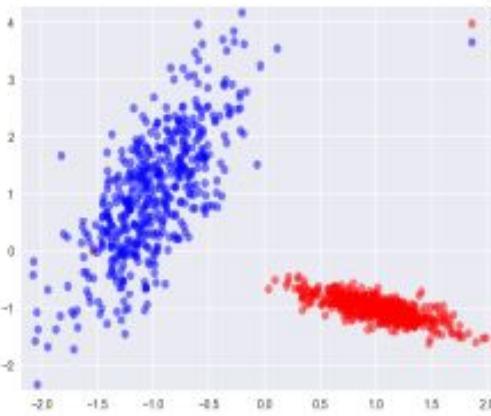
2 propriétés des réseaux de neurones

Propriété 1

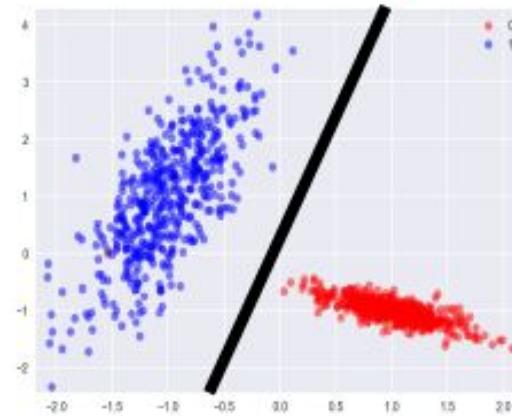
Les réseaux de neurones sont d'excellentes machines pour projeter des données brutes vers un **espace dimensionnel plus faible** dont les propriétés dépendent de la *loss* (ici **espace linéaire car la sortie du réseau est un classifieur linéaire**).



Données en entrée



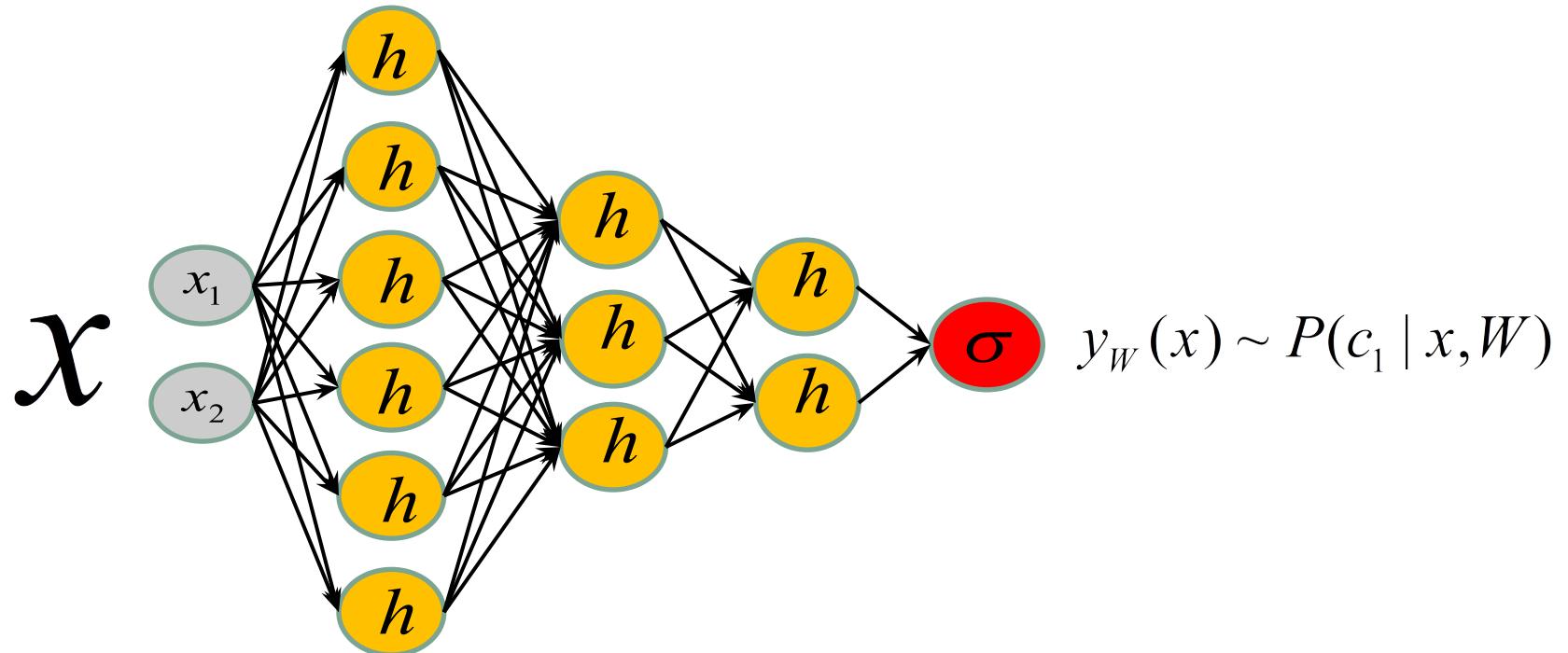
Sortie de la dernière couche



Sortie du réseau

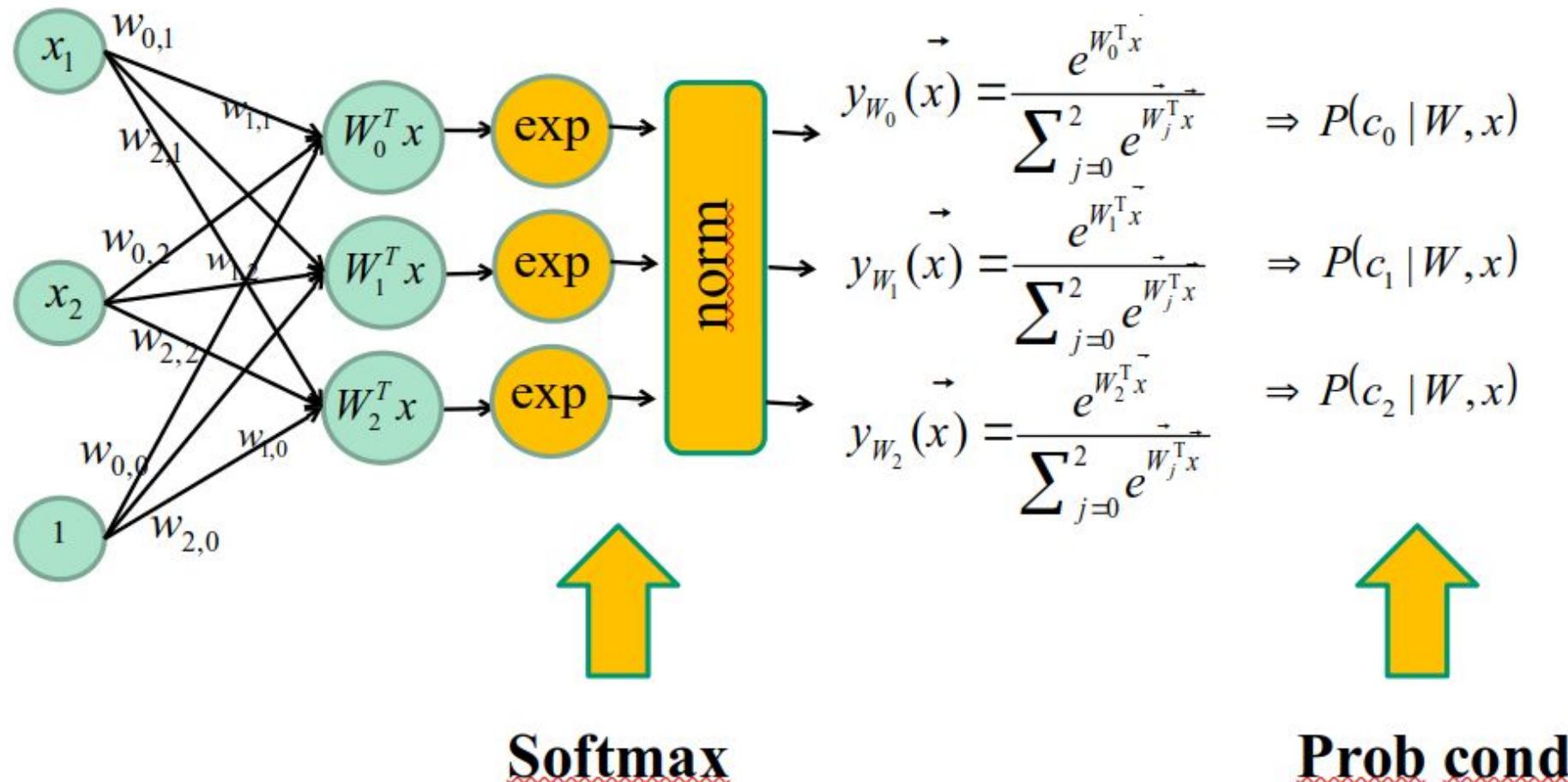
Propriété 2

Les réseaux de neurones sont d'excellentes machines pour estimer des **probabilités conditionnelles**.



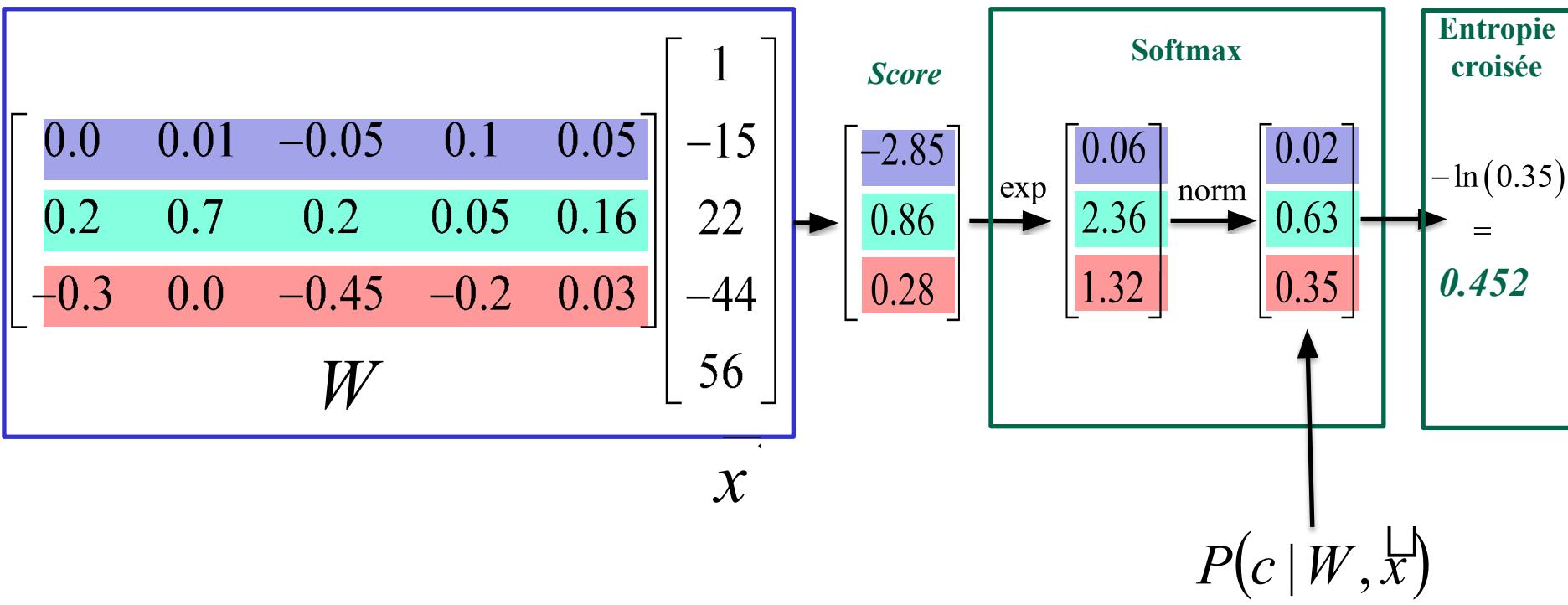
Propriété 2

Les réseaux de neurones sont d'excellentes machines pour estimer des **probabilités conditionnelles**.



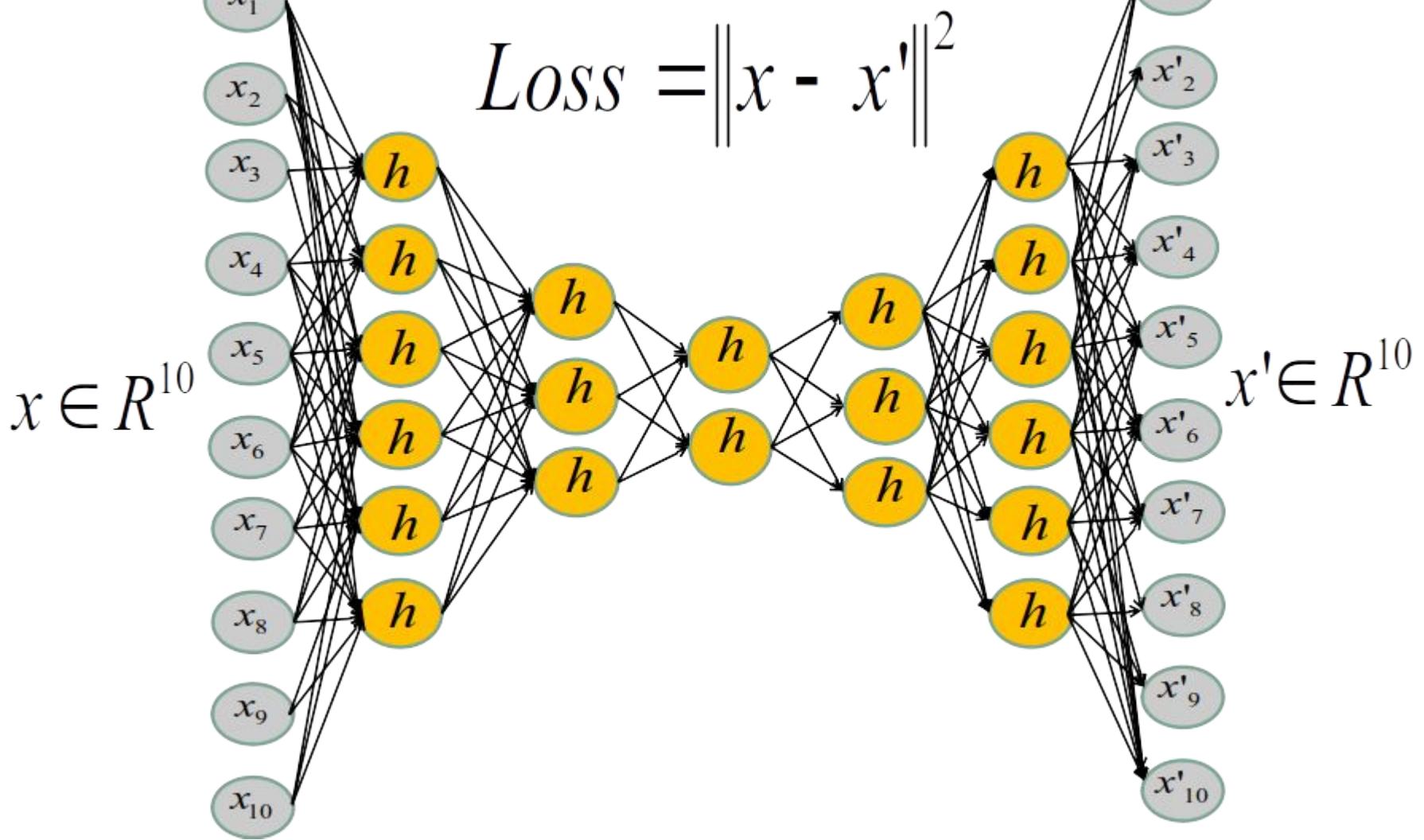
$$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 2$$

Rappel

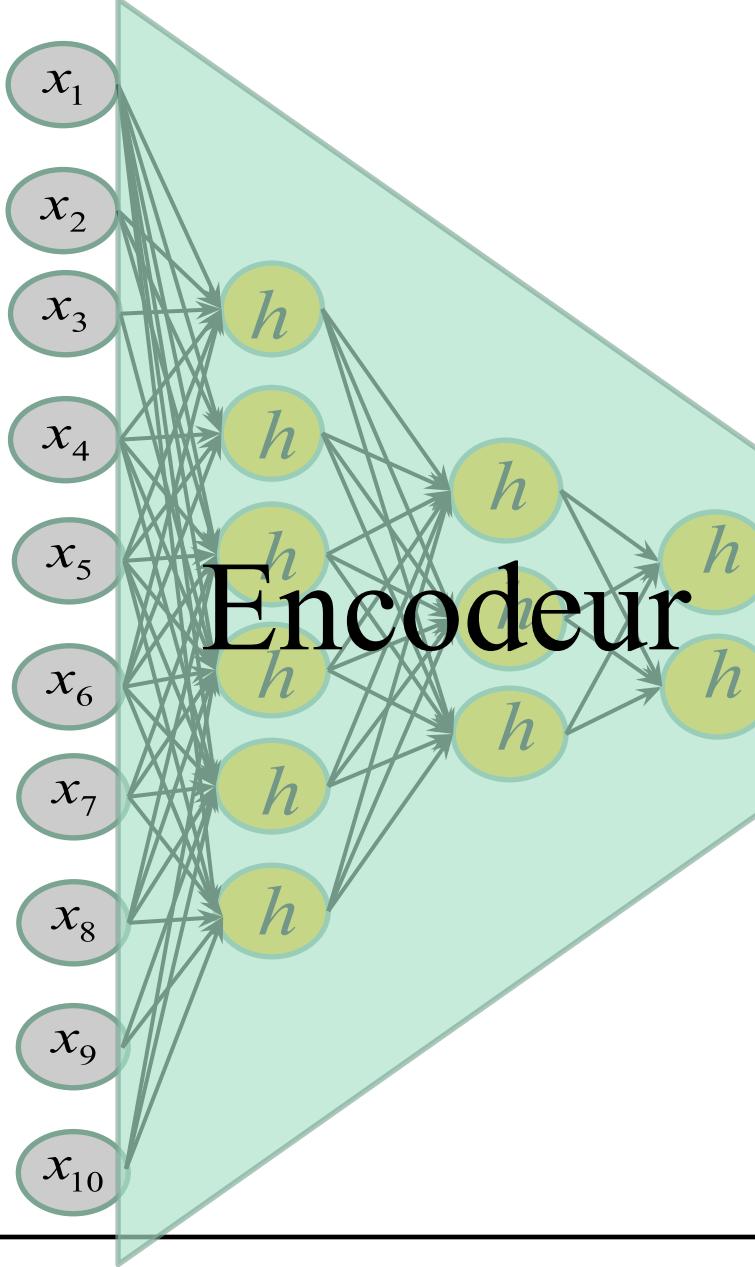


Comment utiliser un réseau de neurones pour apprendre
la **configuration sous-jacente** de données non étiquetés?





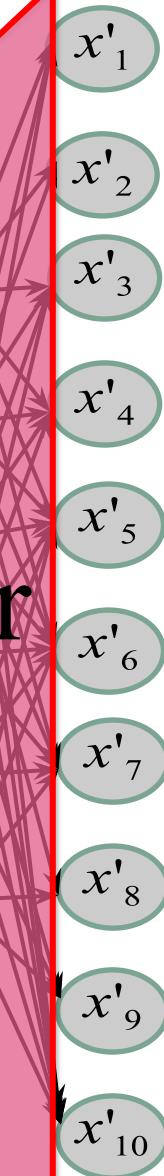
x

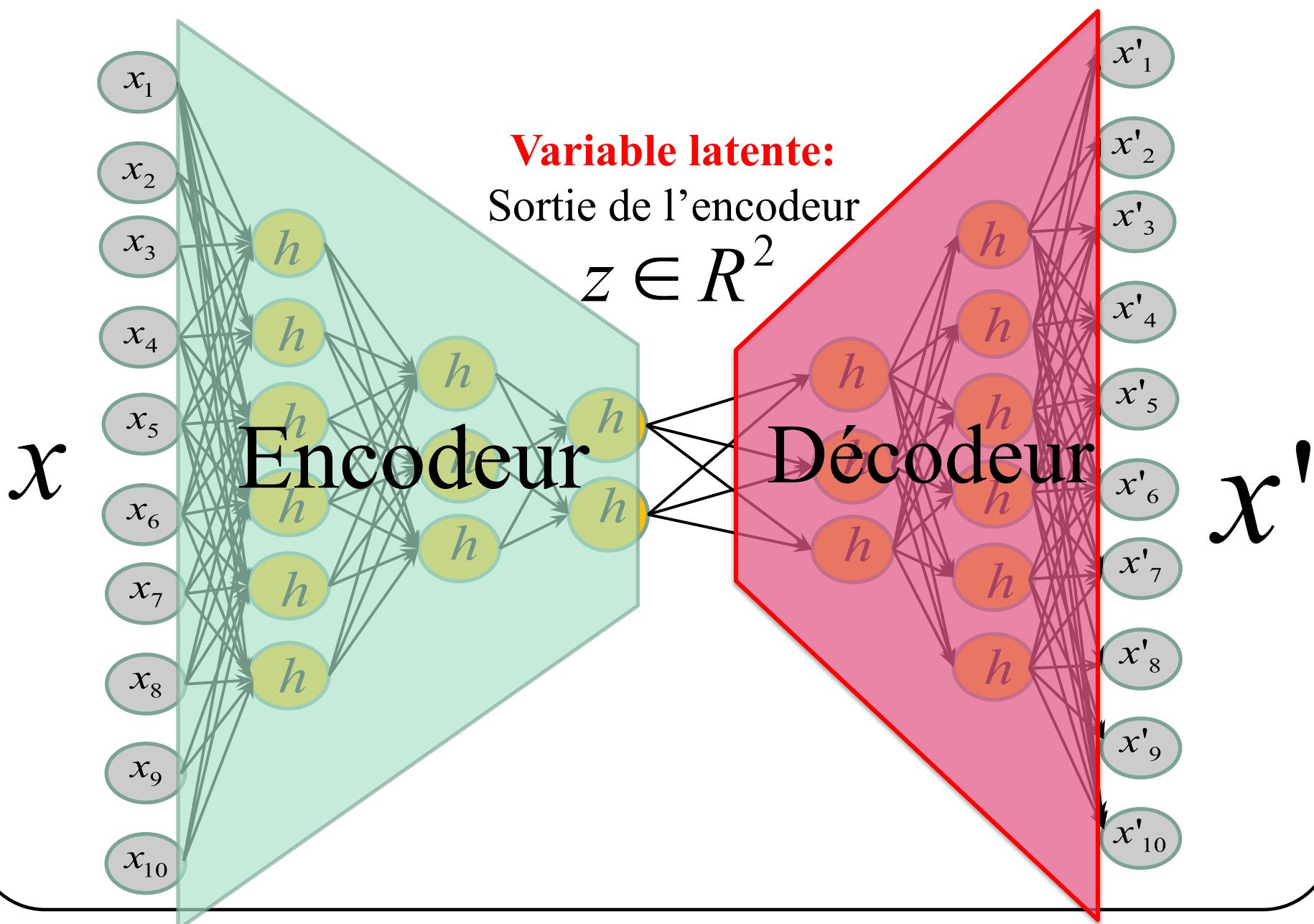


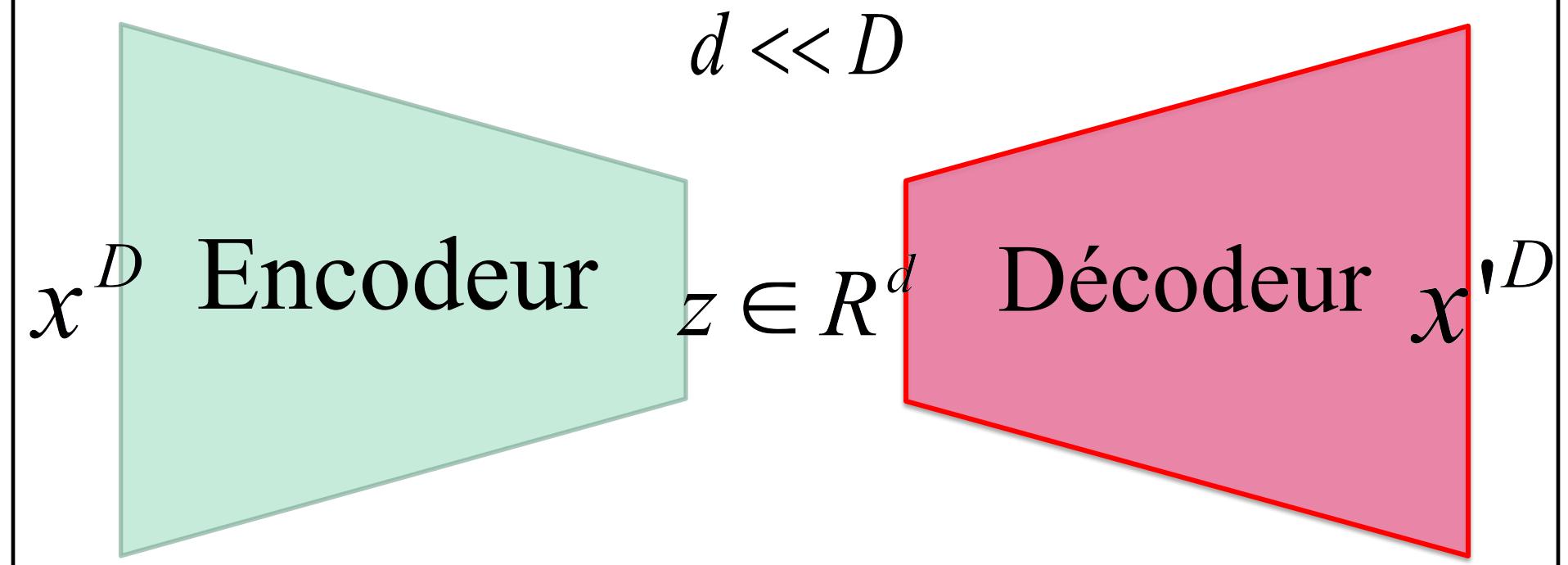
Encodeur

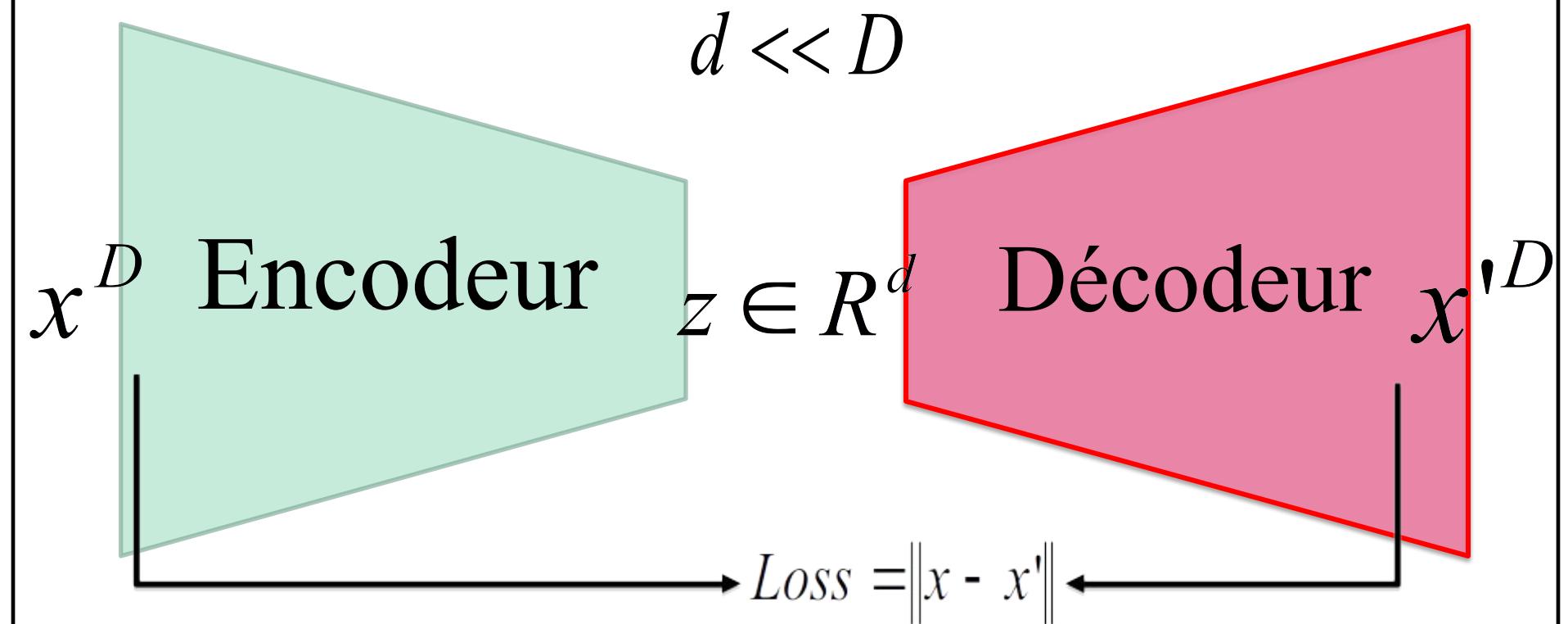
Décodeur

x'

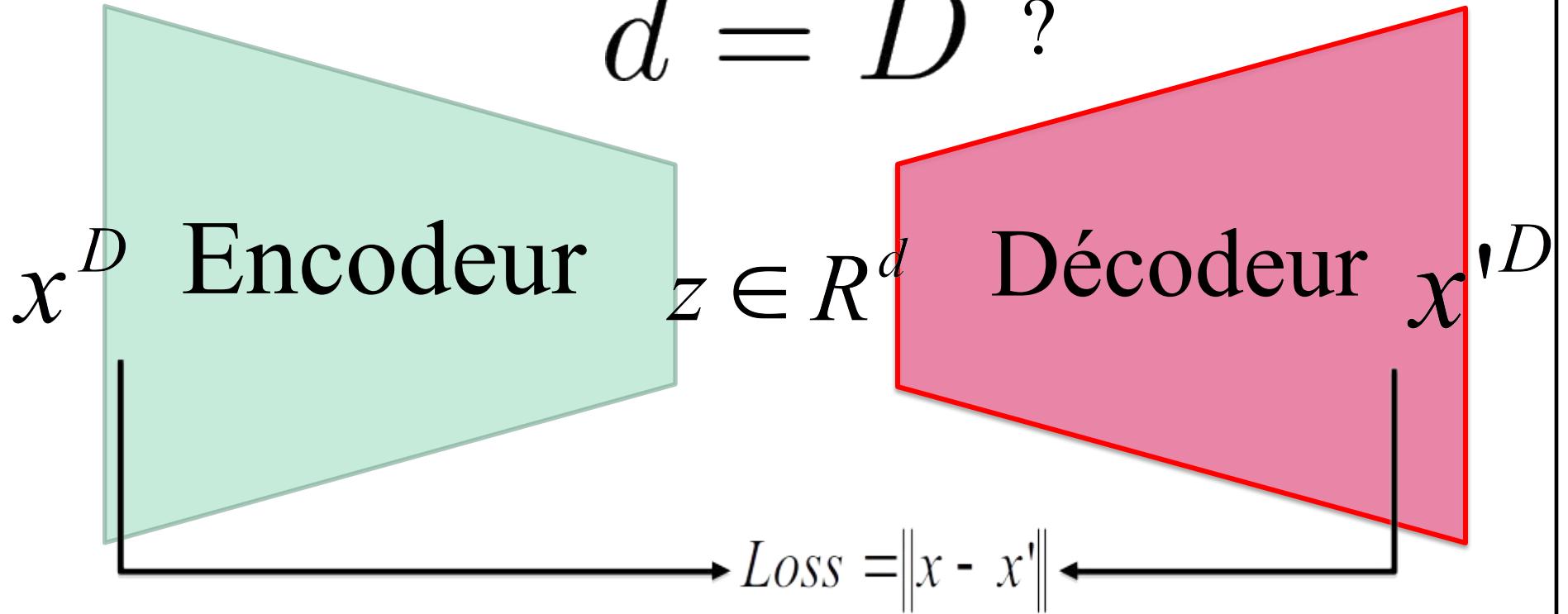


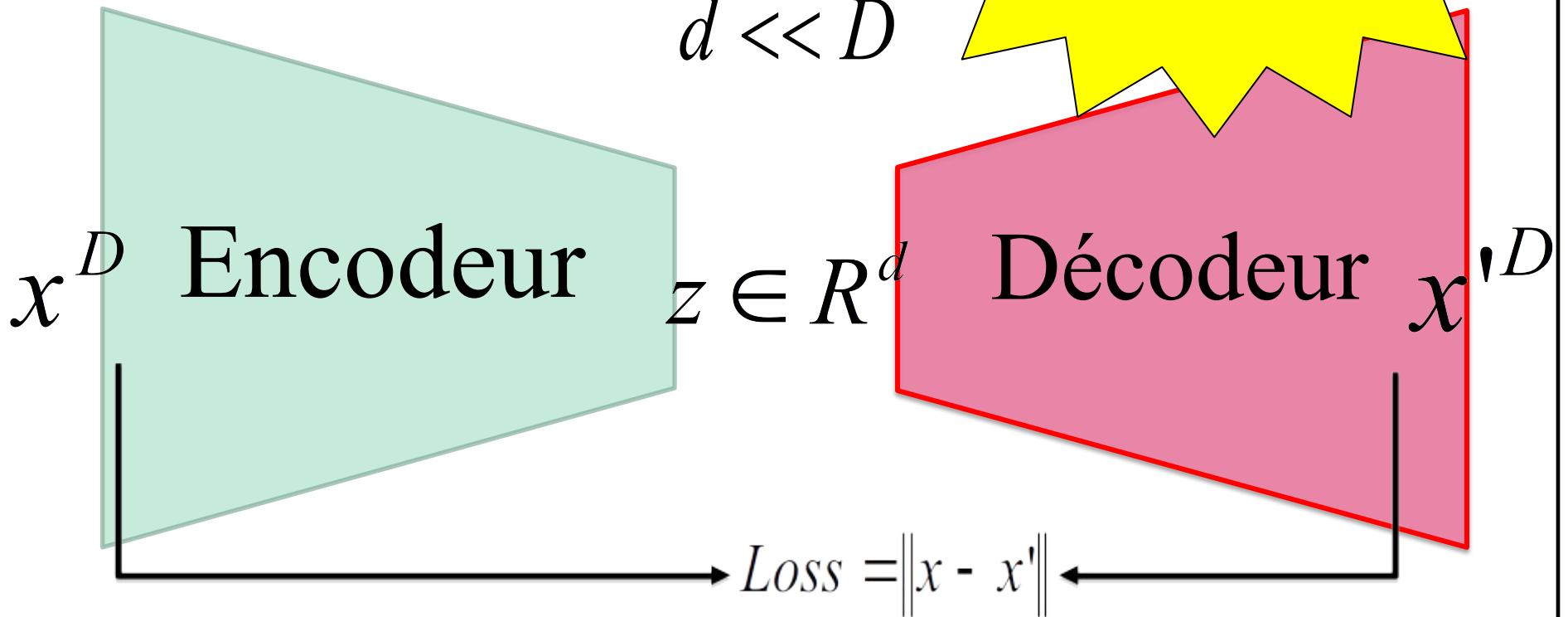




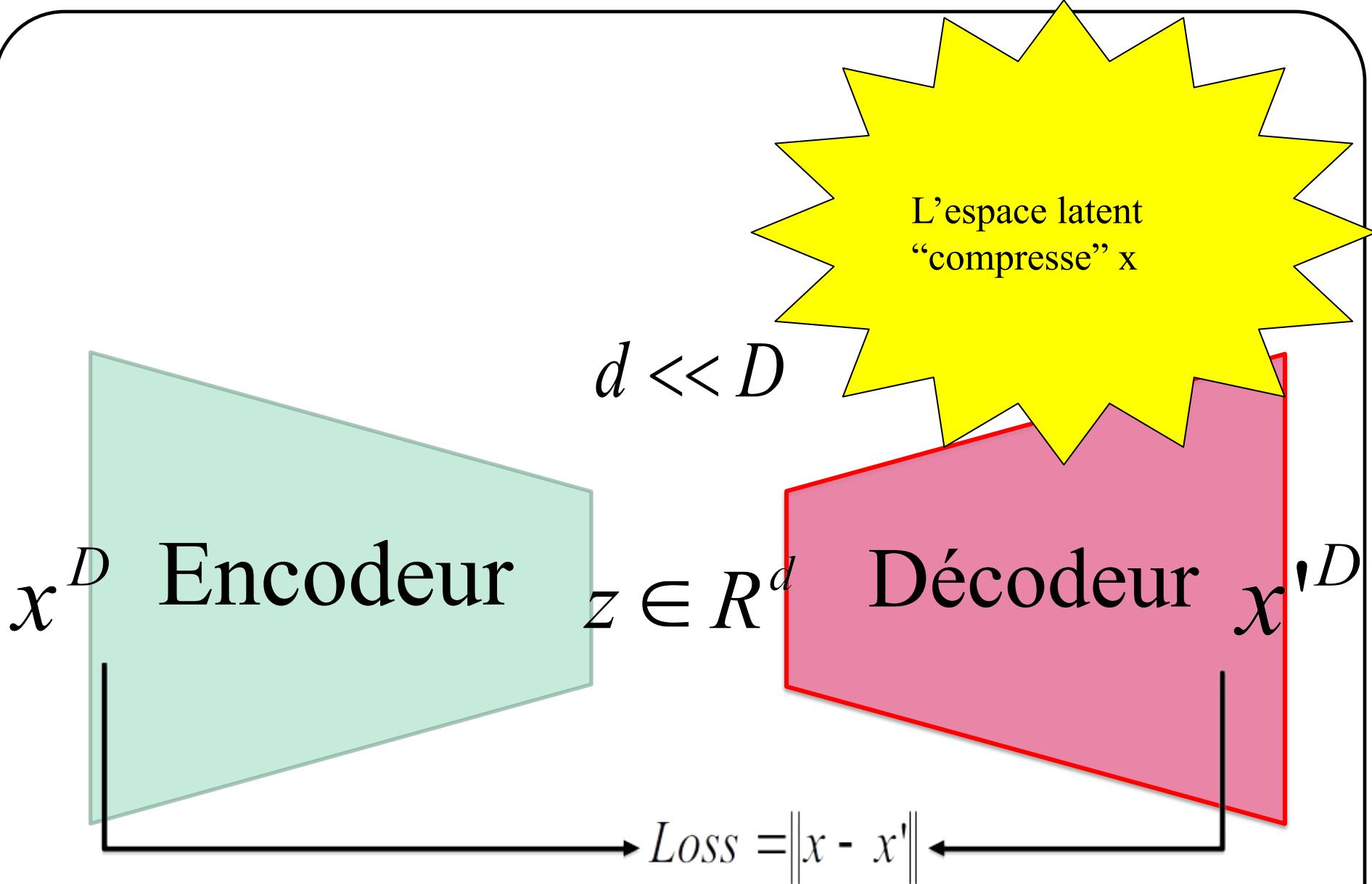


$$d = D ?$$





N'apprend pas la
fonction identité !



L'espace latent
“comprime” x

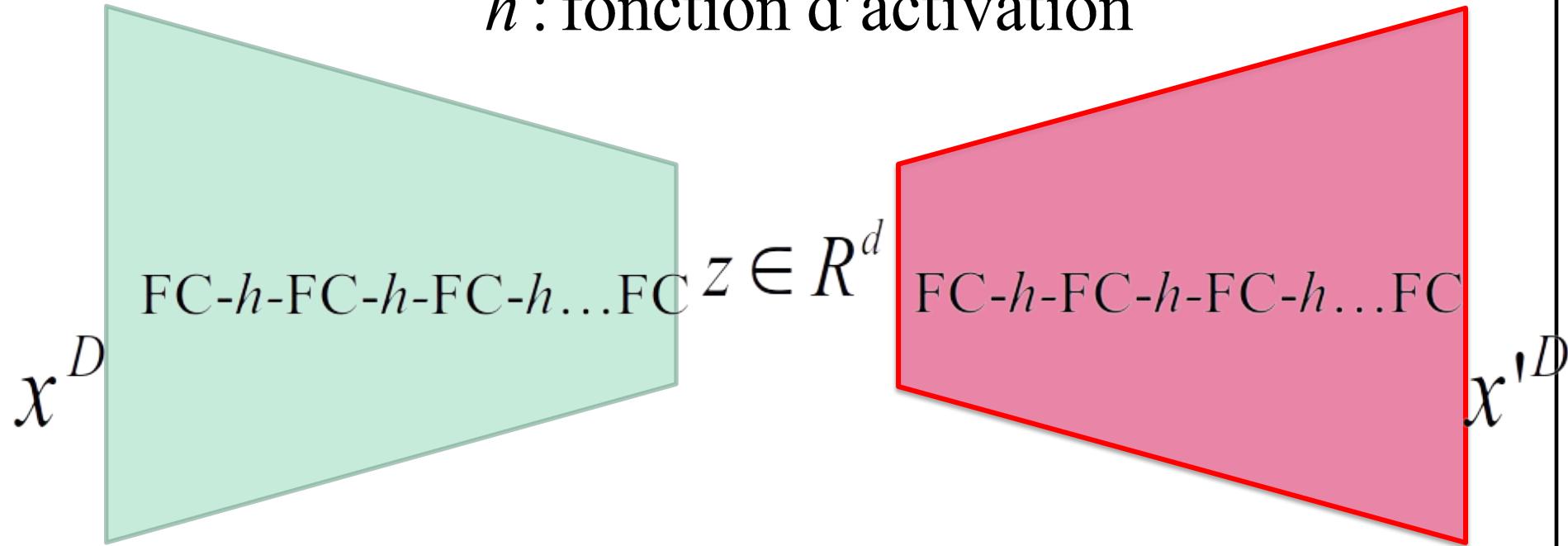
$$d \ll D$$

Décodeur x'^D

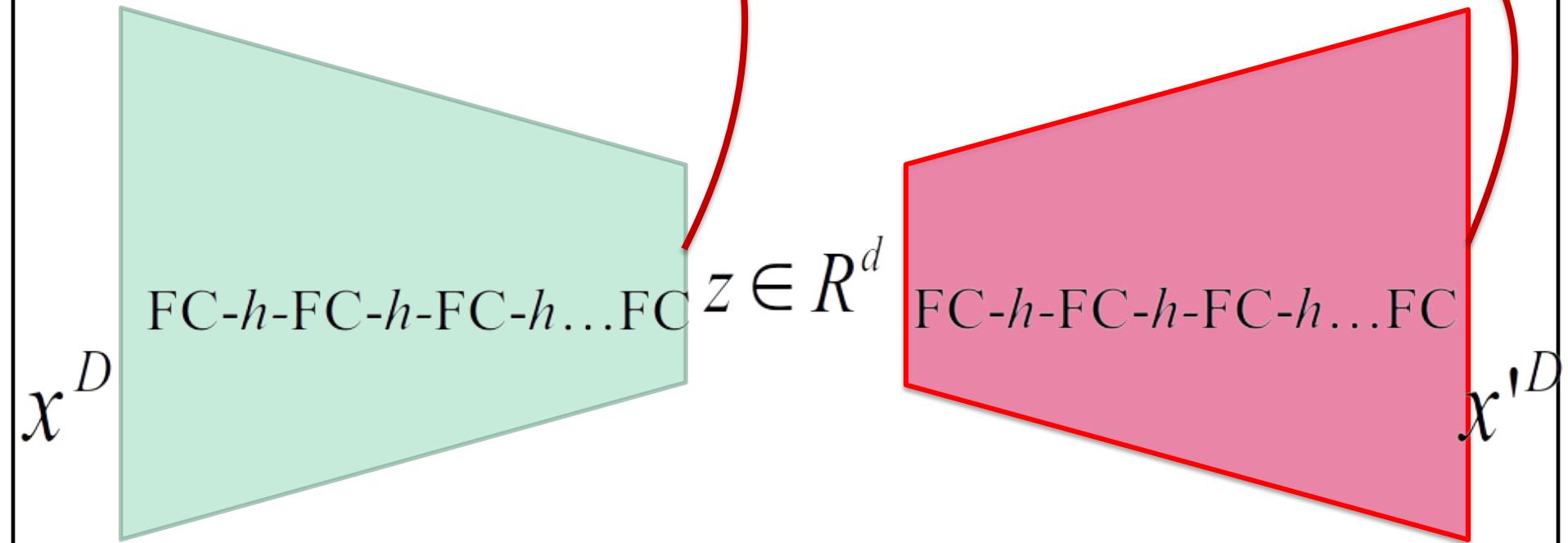
$$\text{Loss} = \|x - x'\|$$

Couches pleinement connectées

h : fonction d'activation

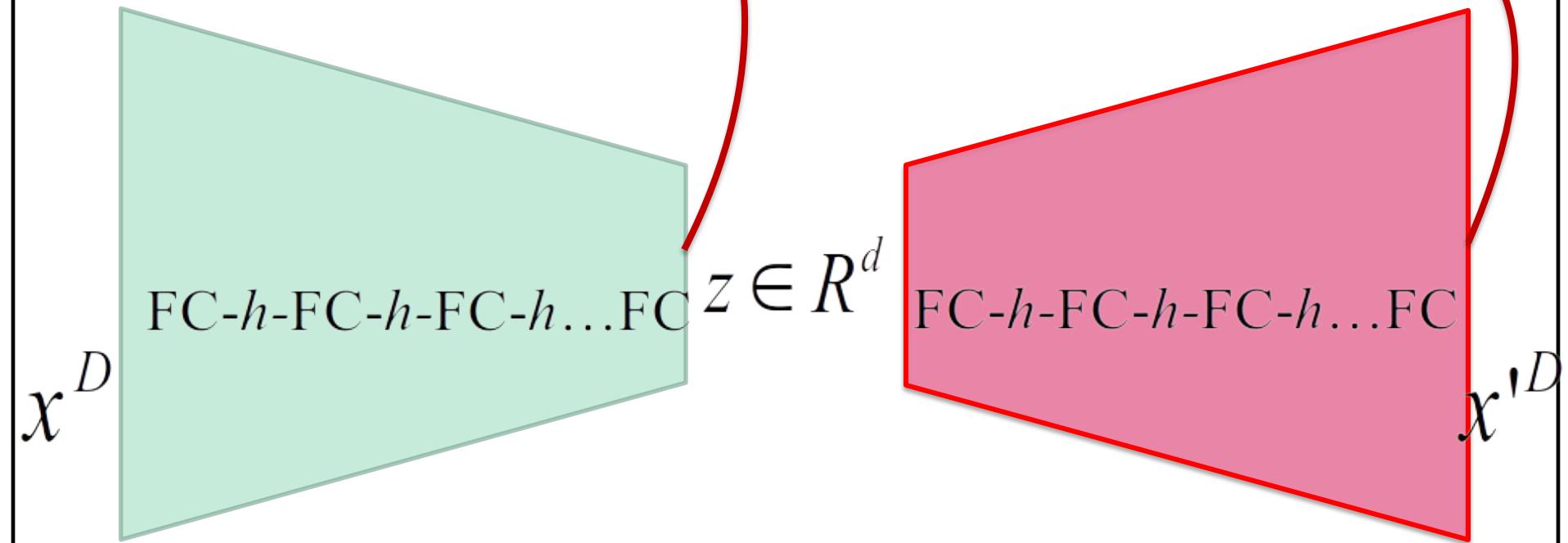


En général...
pas de fonction d'activation à la sortie
de l'encodeur et du décodeur



Selon vous, **pourquoi?**

En général...
pas de fonction d'activation à la sortie
de l'encodeur et du décodeur

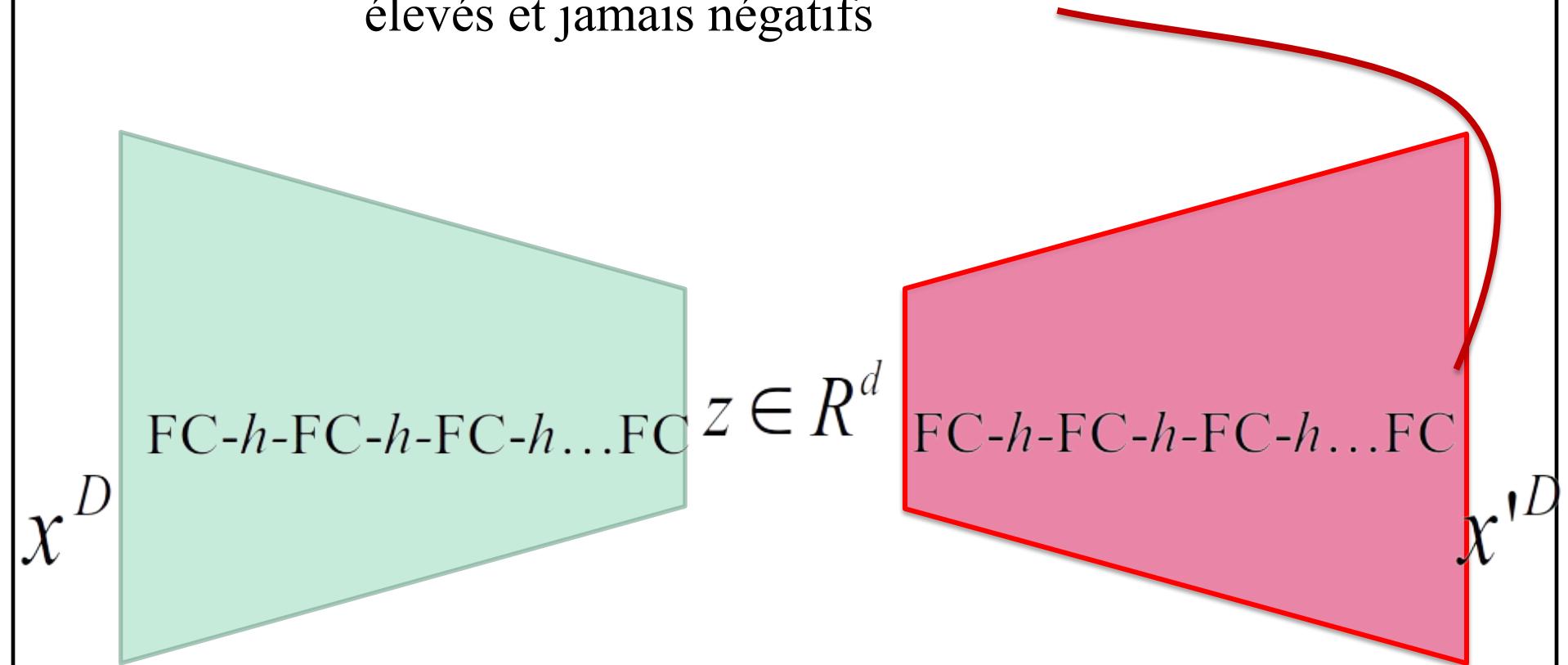


Selon vous, **pourquoi?**

Parce qu'on veut que le x encodé soit réellement un point
dans l'espace latent

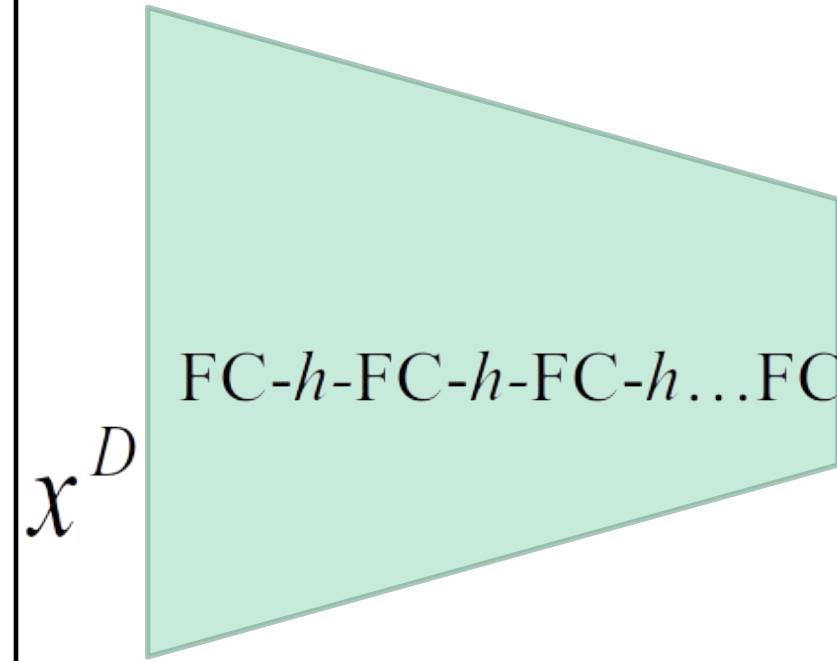
Parfois sigmoïde en sortie lorsque les pixels ont des niveaux de gris entre 0 et 1.

ou ReLU lorsque les niveaux de gris peuvent être élevés et jamais négatifs



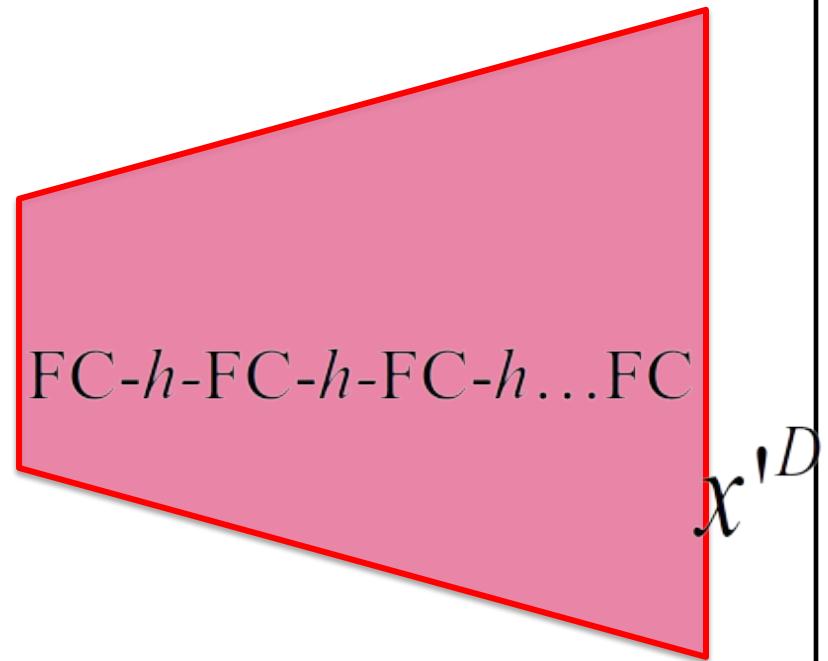
Le nombre de neurones

Décroît ou se maintient
d'une couche à l'autre



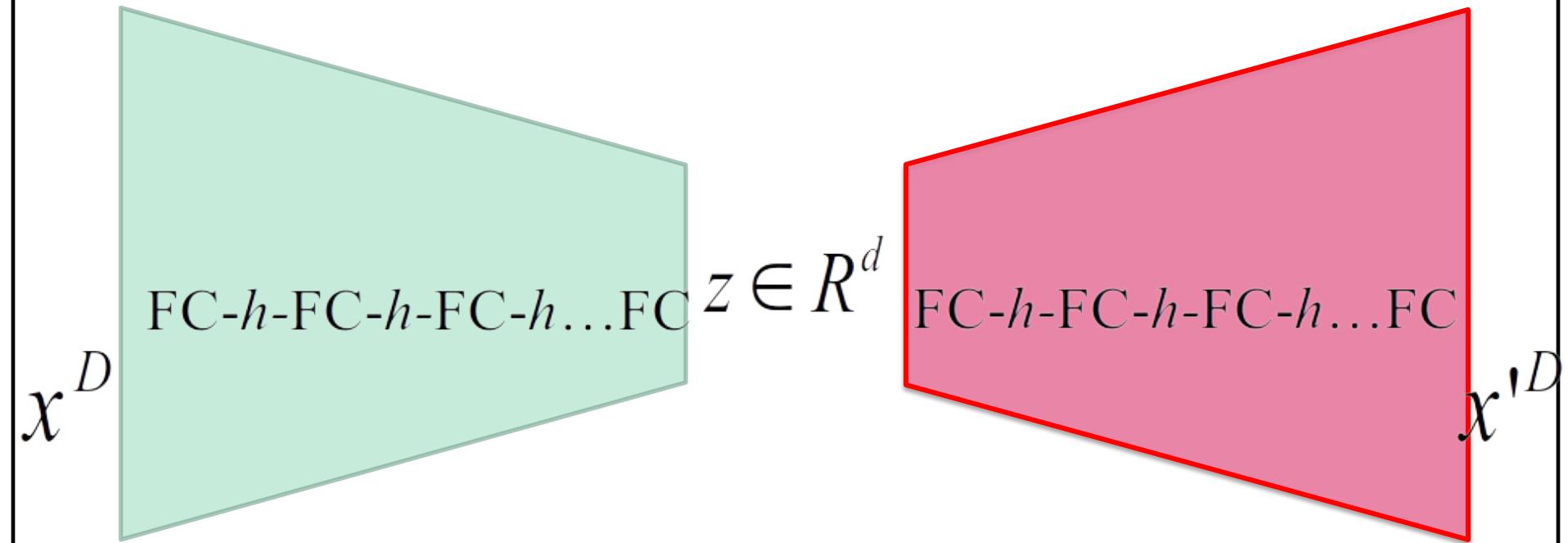
Le nombre de neurones

Augmente ou se maintient
d'une couche à l'autre



Très souvent...

La structure de l'encodeur est le dual de celle du décodeur



Autoencodeur jouet de MNIST

```
class autoencoder(nn.Module):

    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128), nn.ReLU(True),
            nn.Linear(128, 64), nn.ReLU(True),
            nn.Linear(64, 12), nn.ReLU(True),
            nn.Linear(12, 2))

        self.decoder = nn.Sequential(
            nn.Linear(2, 12), nn.ReLU(True),
            nn.Linear(12, 64), nn.ReLU(True),
            nn.Linear(64, 128), nn.ReLU(True),
            nn.Linear(128, 28 * 28))

    def forward(self, x):
        z = self.encoder(x)
        x_prime = self.decoder(z)
        return x_prime
```

Espace latent 2D

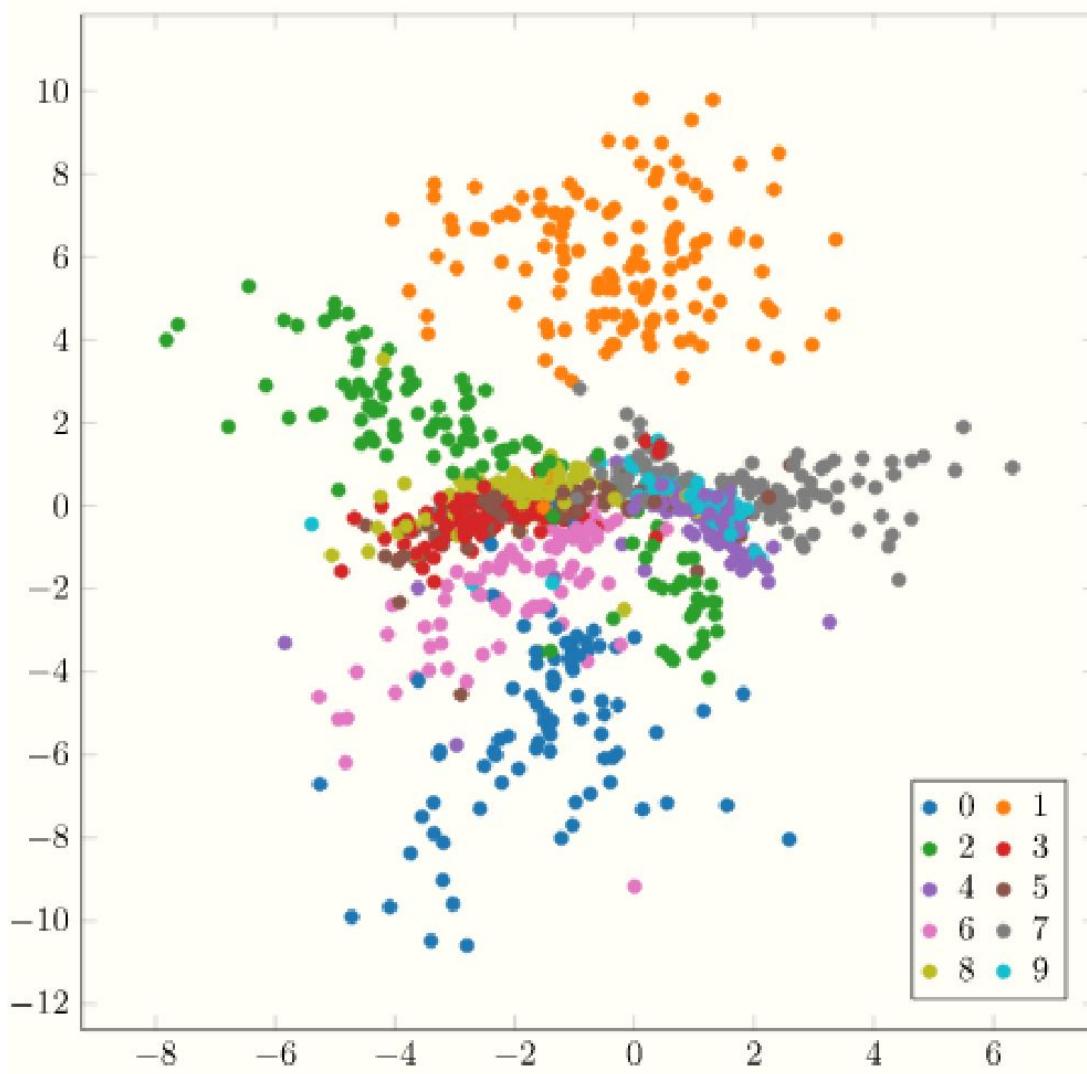
Autoencodeur jouet de MNIST

```
class autoencoder(nn.Module):  
  
    def __init__(self):  
        super(autoencoder, self).__init__()  
  
        self.encoder = nn.Sequential(  
            nn.Linear(28 * 28, 128), nn.ReLU(True),  
            nn.Linear(128, 64), nn.ReLU(True),  
            nn.Linear(64, 12), nn.ReLU(True),  
            nn.Linear(12, 2))  
  
        self.decoder = nn.Sequential(  
            nn.Linear(2, 12), nn.ReLU(True),  
            nn.Linear(12, 64), nn.ReLU(True),  
            nn.Linear(64, 128), nn.ReLU(True),  
            nn.Linear(128, 28 * 28))  
  
    def forward(self, x):  
        z = self.encoder(x)  
        x_prime = self.decoder(z)  
  
        return x_prime
```

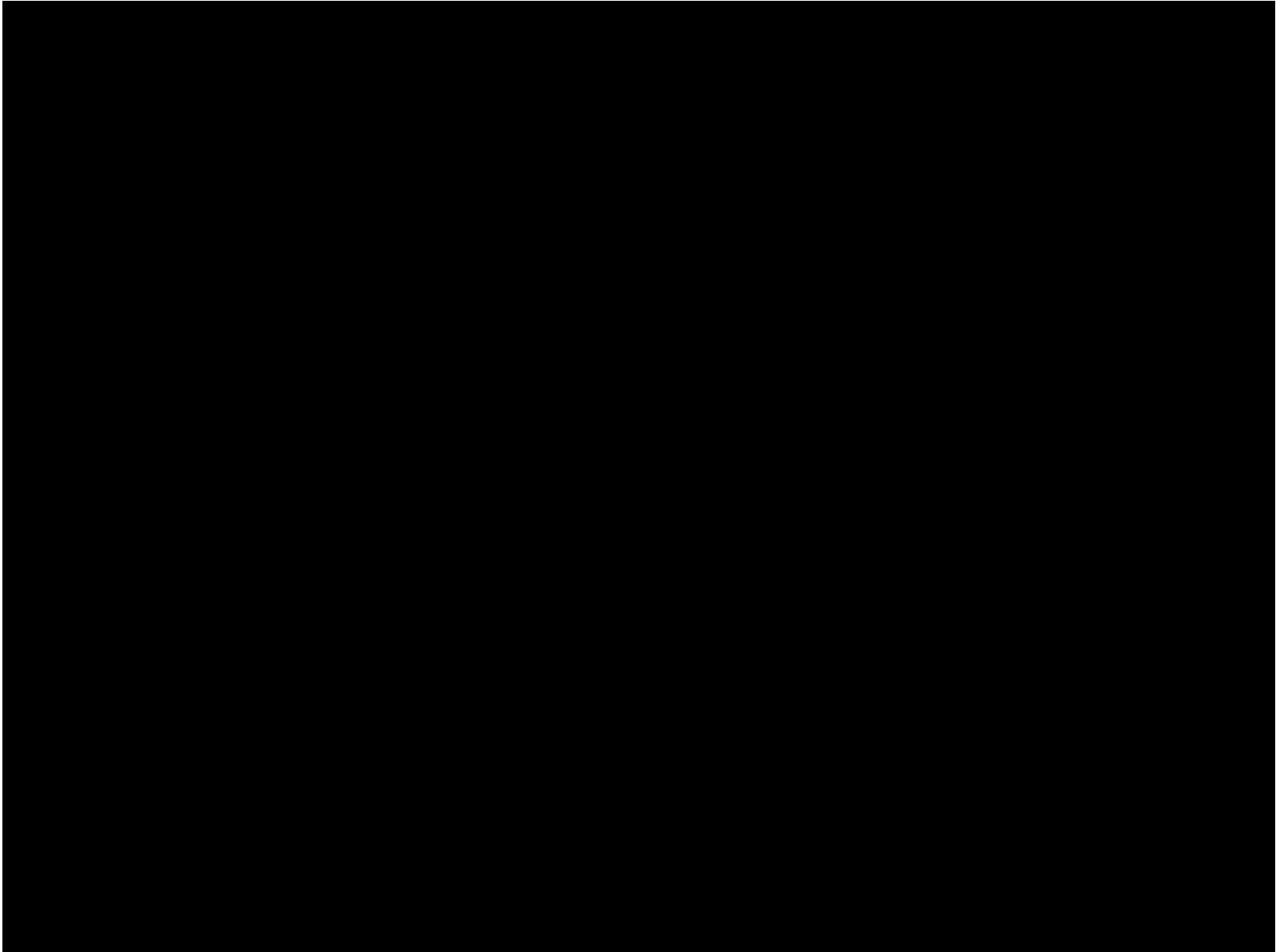
symétrie

Visualisation de l'espace latent pour 1000 images MNIST

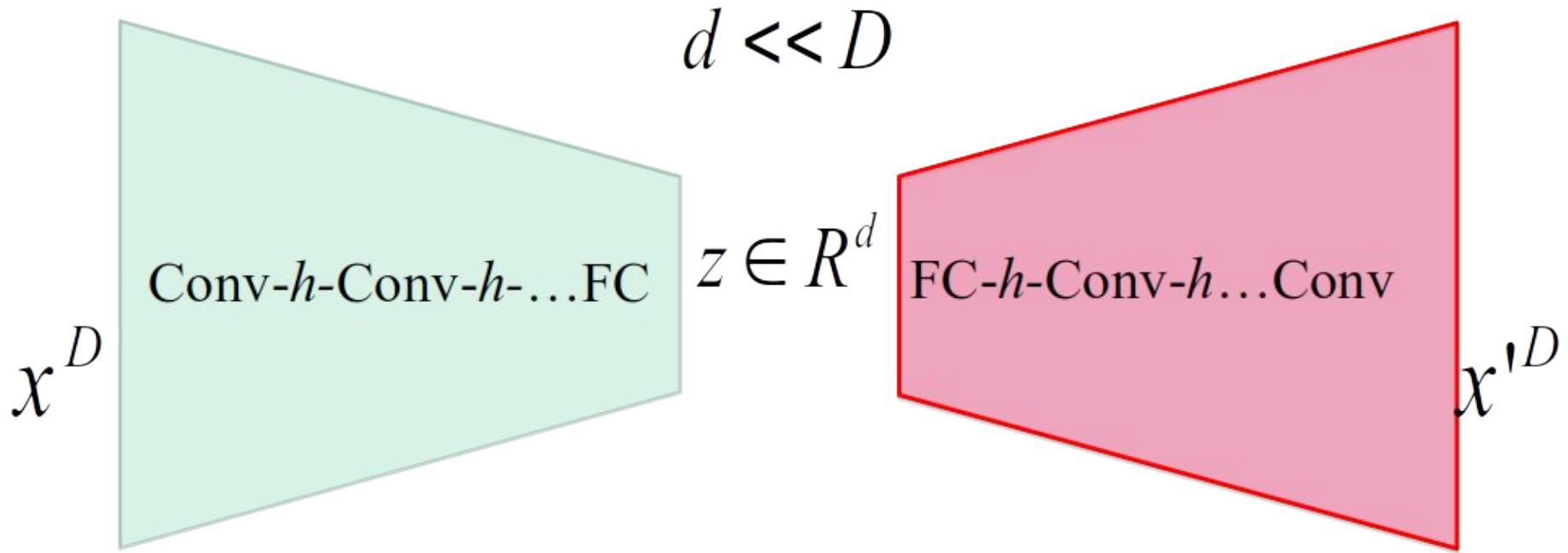
chaque image correspond à 1 point 2D



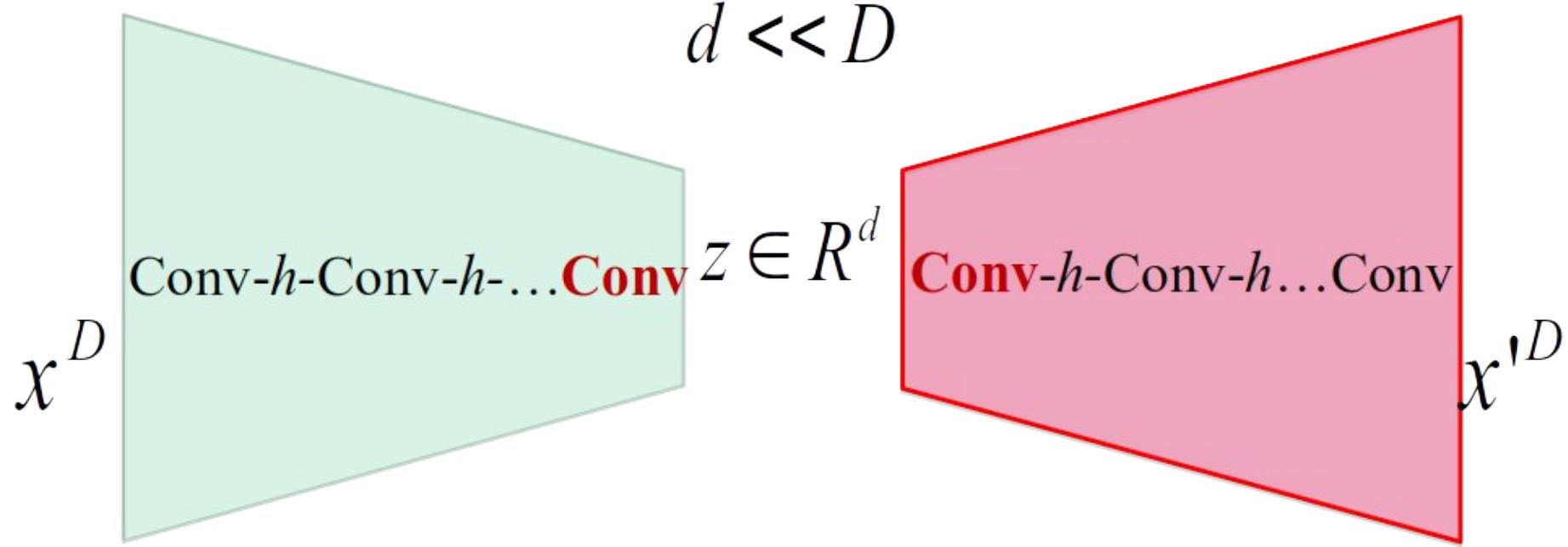
Générer des images avec le décodeur.



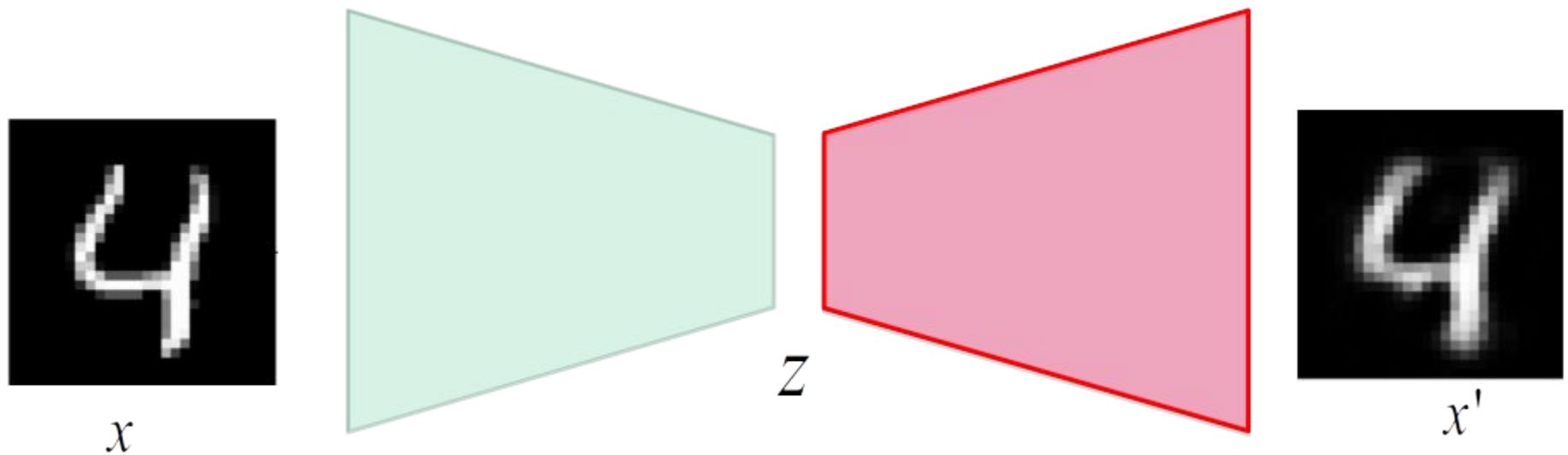
Couches convolutives



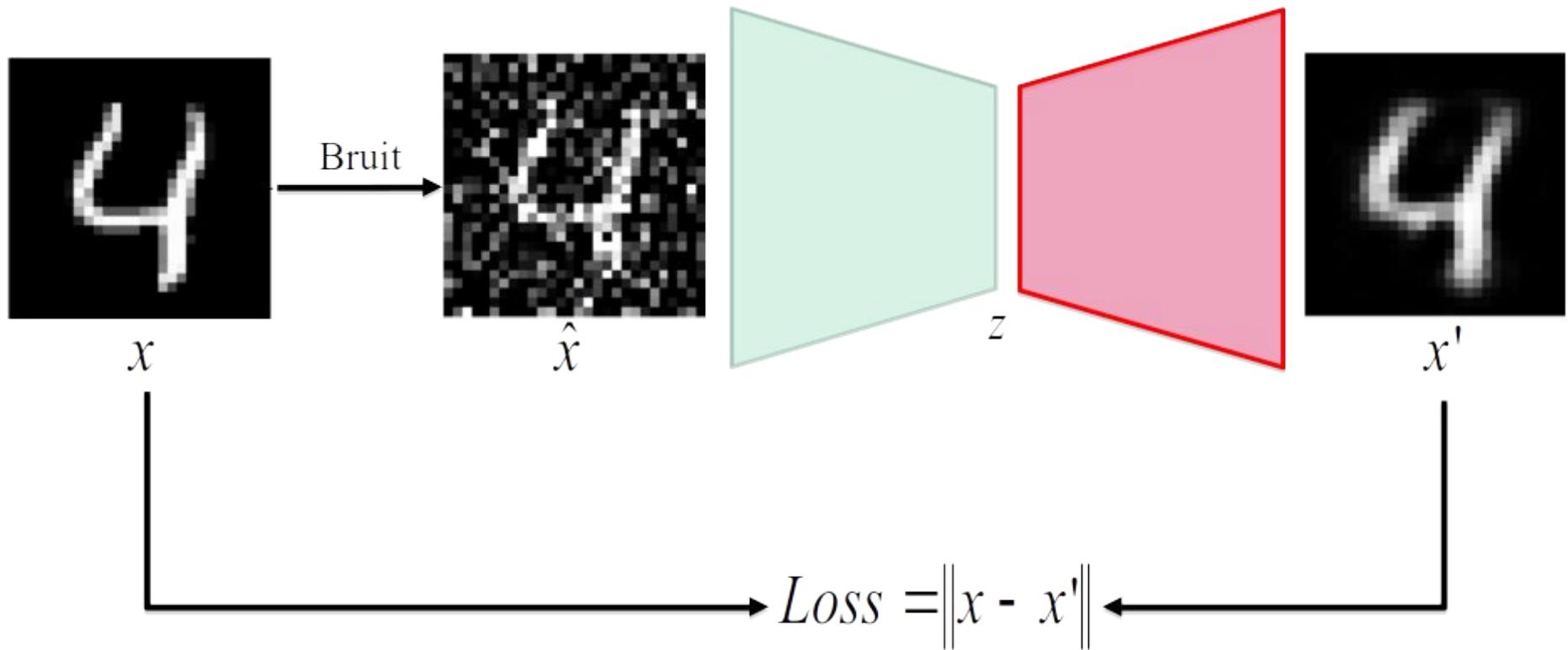
Autoencodeur pleinement convolutif



Autoencodeur de base

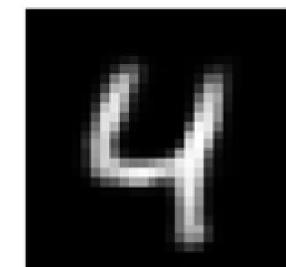
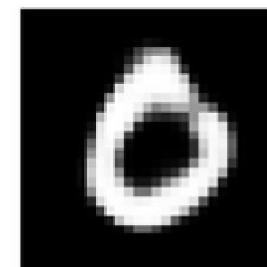
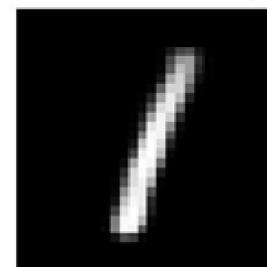
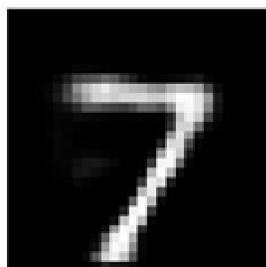
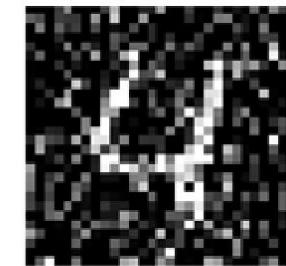
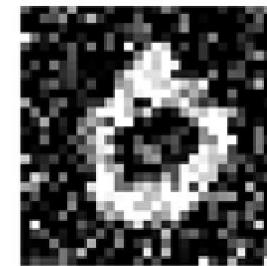
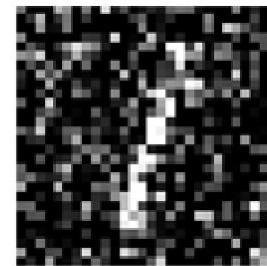
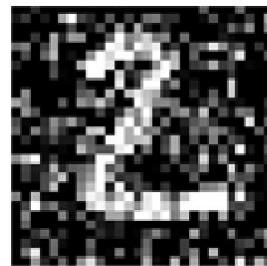
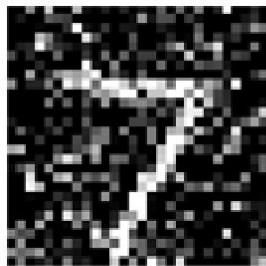


Autoencodeur pour débruitage



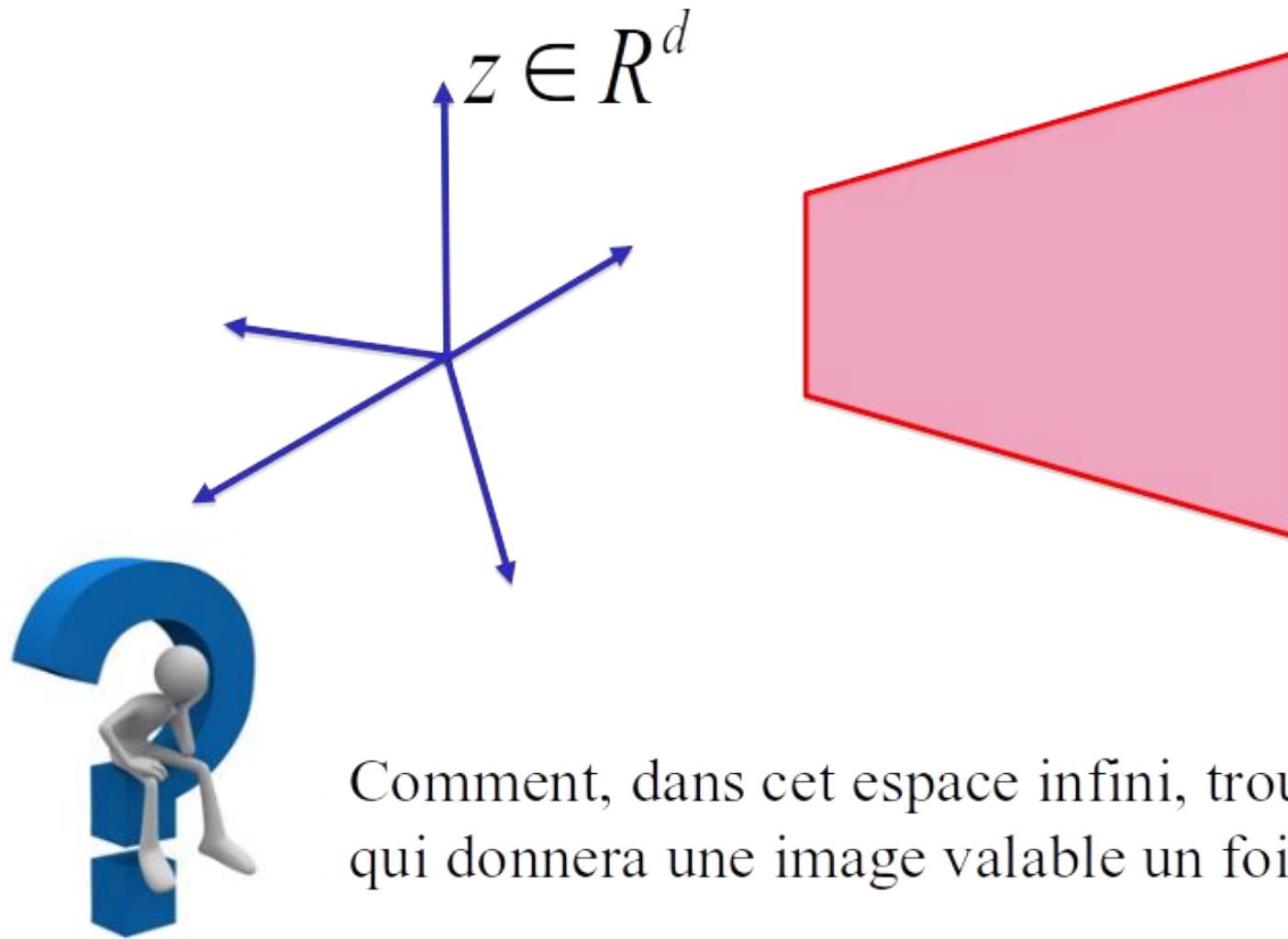
Une fois entraîné...

Signal bruité en entrée



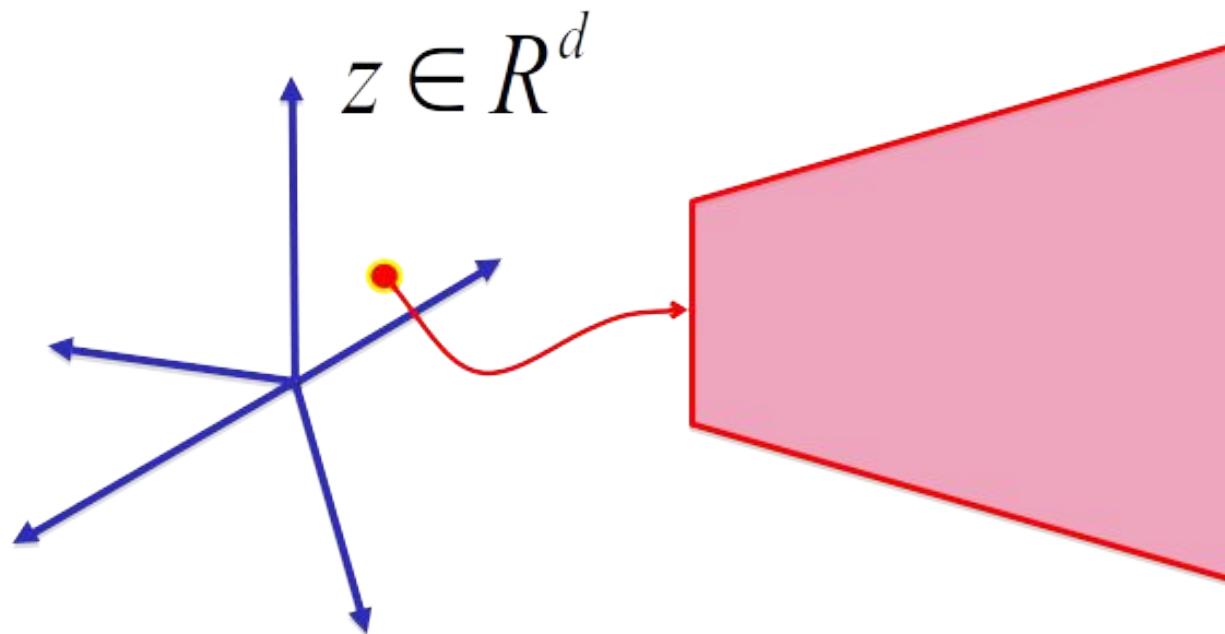
Signal reconstruit et débruité

En général, l'espace latent possède entre 16 et 128 dimensions.
Ça peut être parfois plus, et parfois moins.

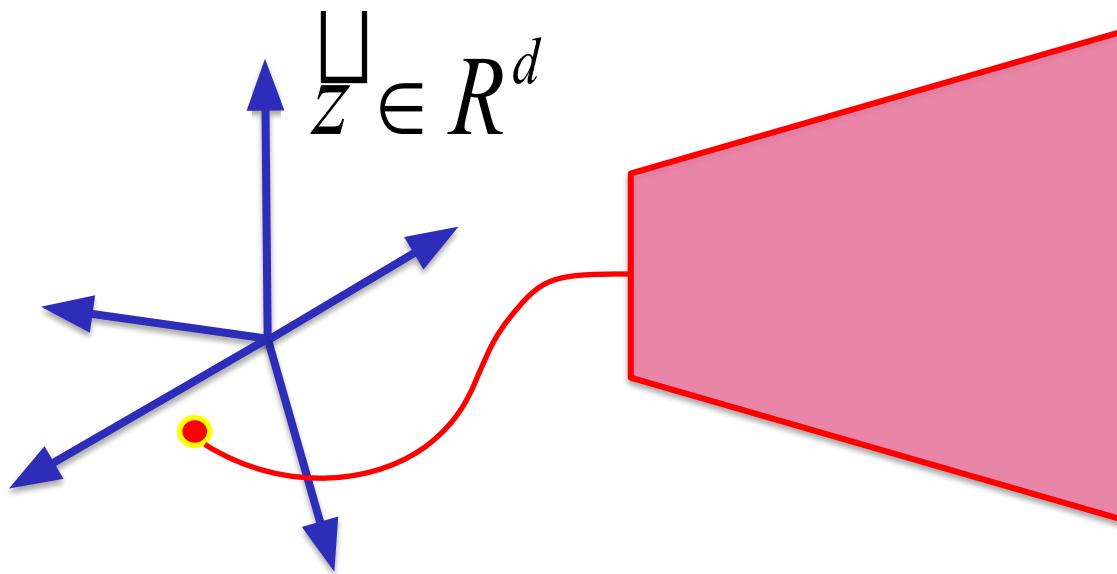


Comment, dans cet espace infini, trouver un point z qui donnera une image valable un fois décodée?

Avec de la chance, on peut sélectionner un point au hasard et reproduire une « bonne » image (ici une image « MNIST »)



Malheureusement, la vaste majorité du temps, on reproduira du bruit



Au lieu d'apprendre à **reproduire**
un signal d'entrée...

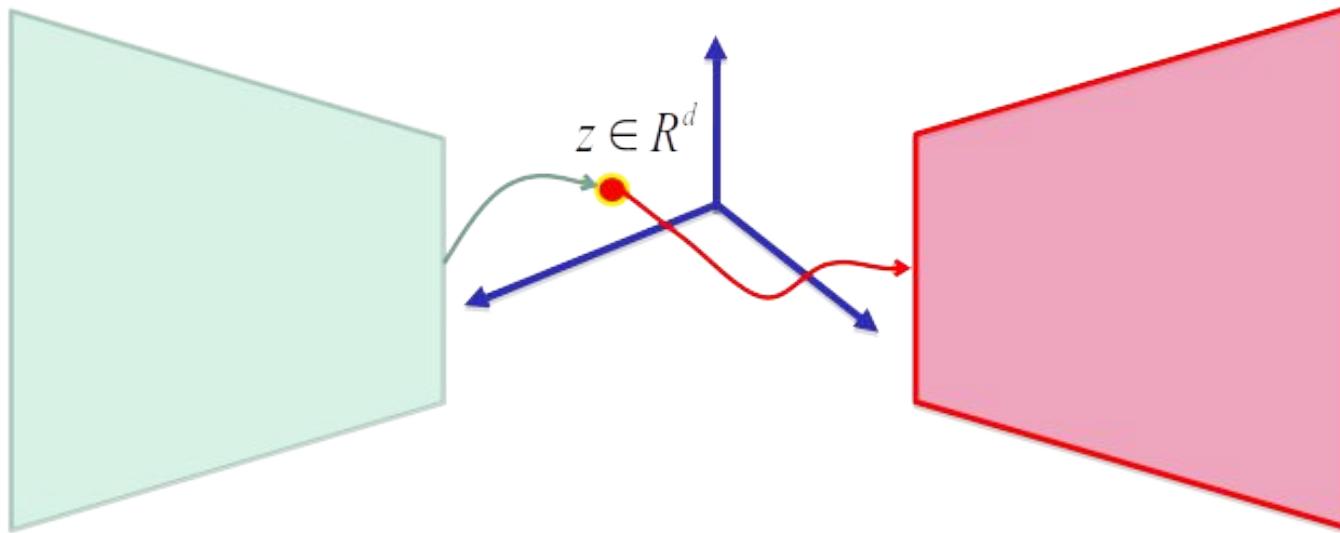


Apprendre à reproduire une **distribution** $p(\mathbb{z})$
connue de sorte qu'un **point échantillonné**
et décodé de cette distribution correspond à
un signal reconstruit valable

Autoencodeur de base



x

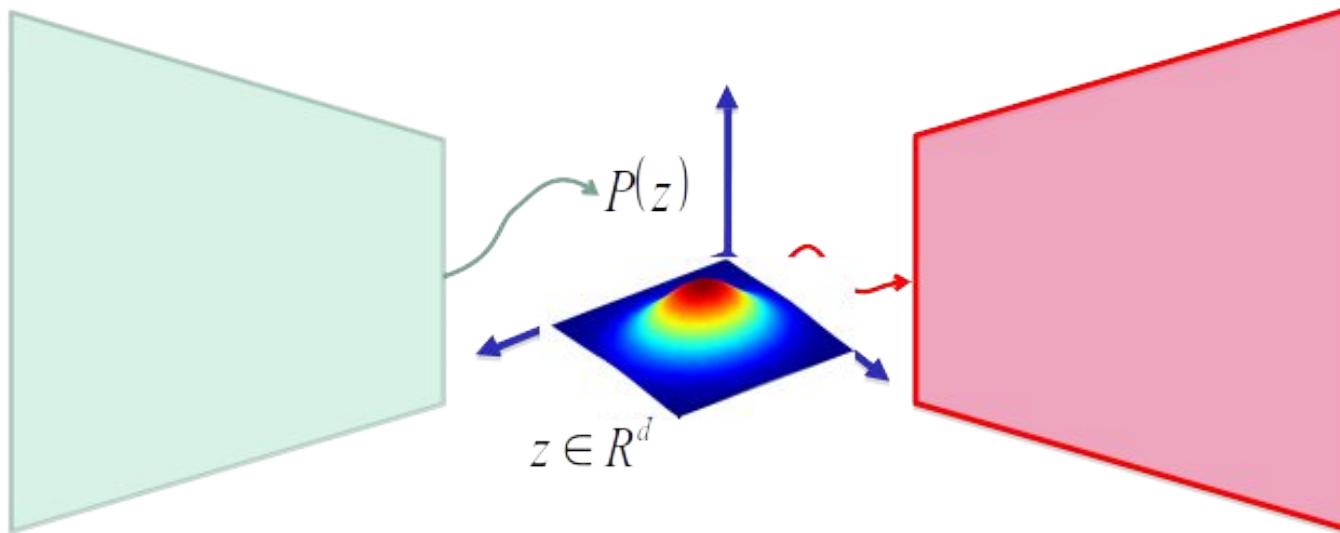


x'

Autoencodeur variationnel



x

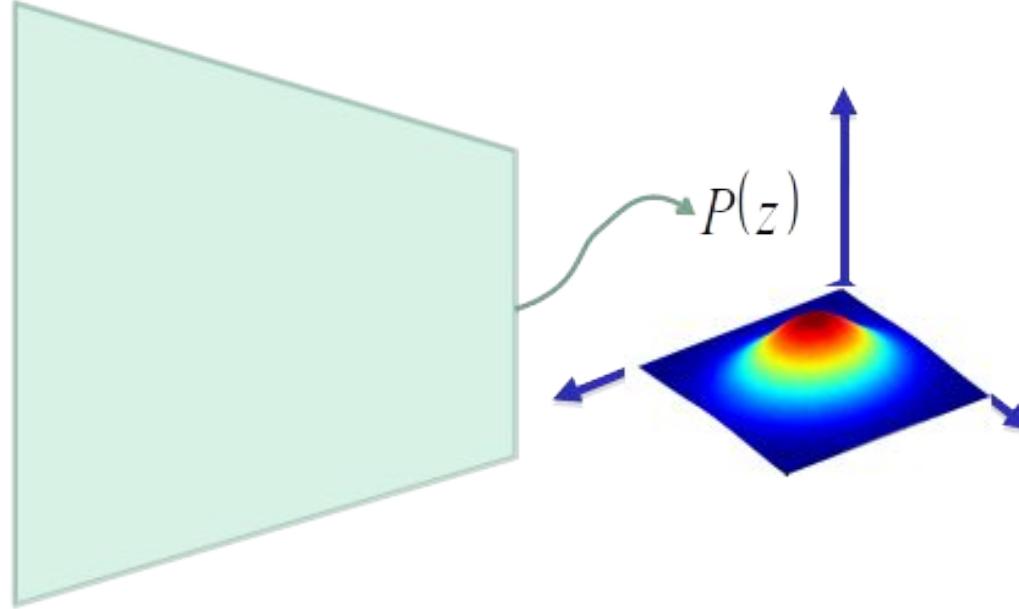


x'

Autoencodeur variationnel



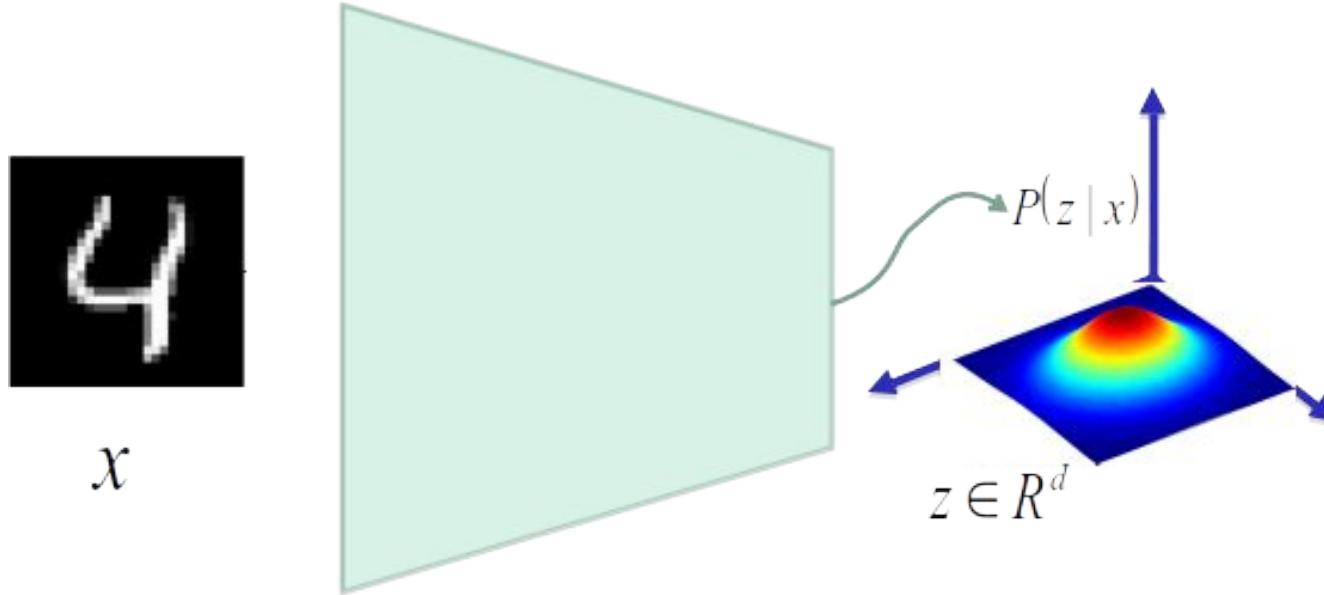
x



L'encodeur produit une distribution $P(z)$
et non juste un point z

Autoencodeur variationnel

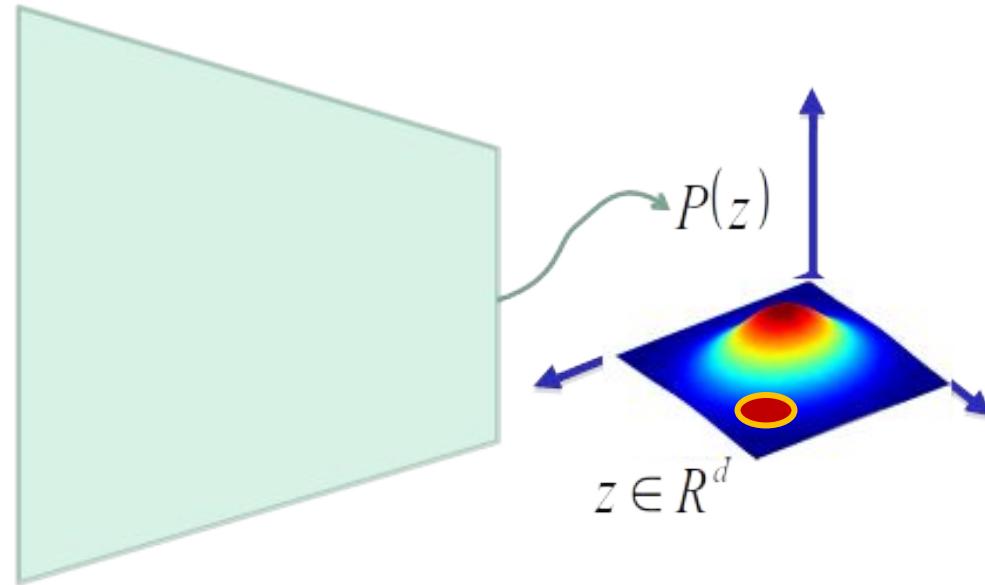
Remarque 1



Puisque la distribution de z dépend de x
on dira que la distribution apprise est

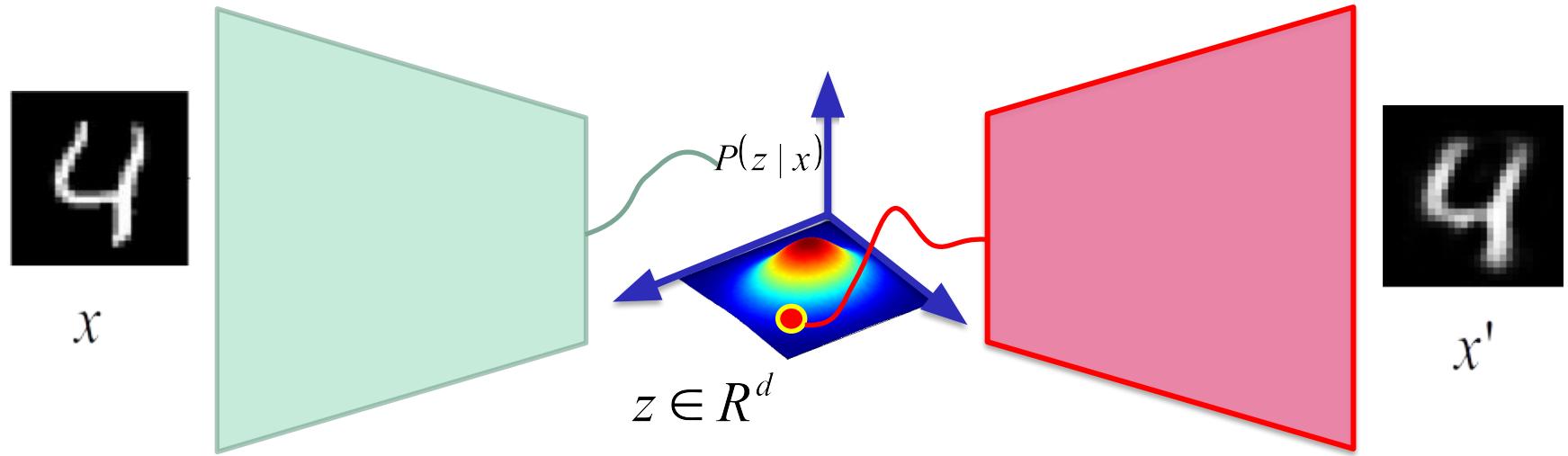
$$P(z | x)$$

Autoencodeur variationnel



On échantillonne un $z \sim p(z)$ au hasard

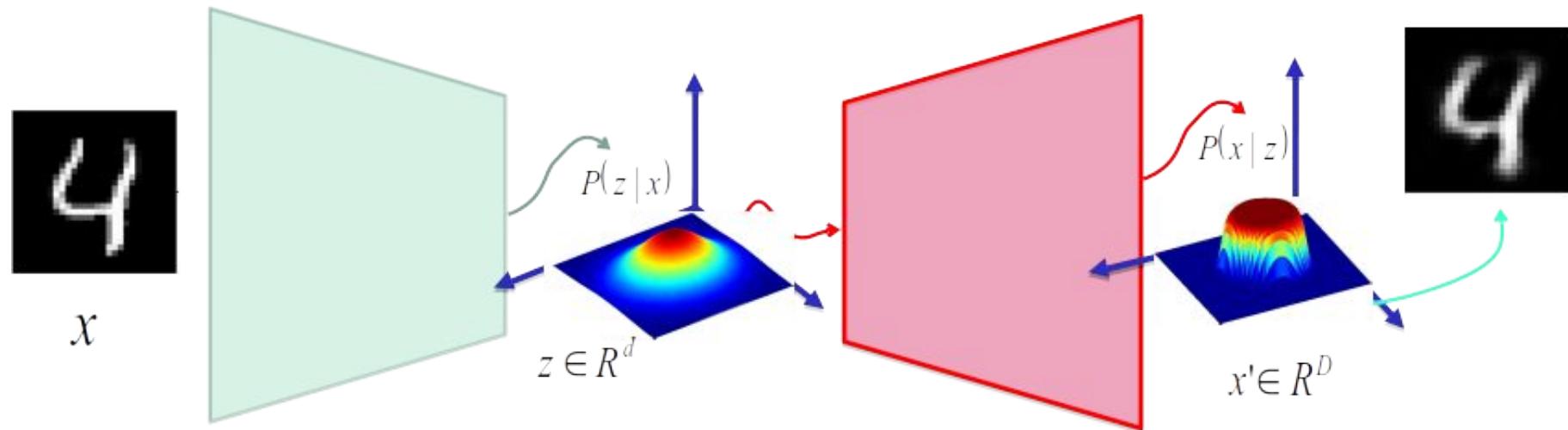
Autoencodeur variationnel



On reconstruit x'

Autoencodeur variationnel

Remarque 2



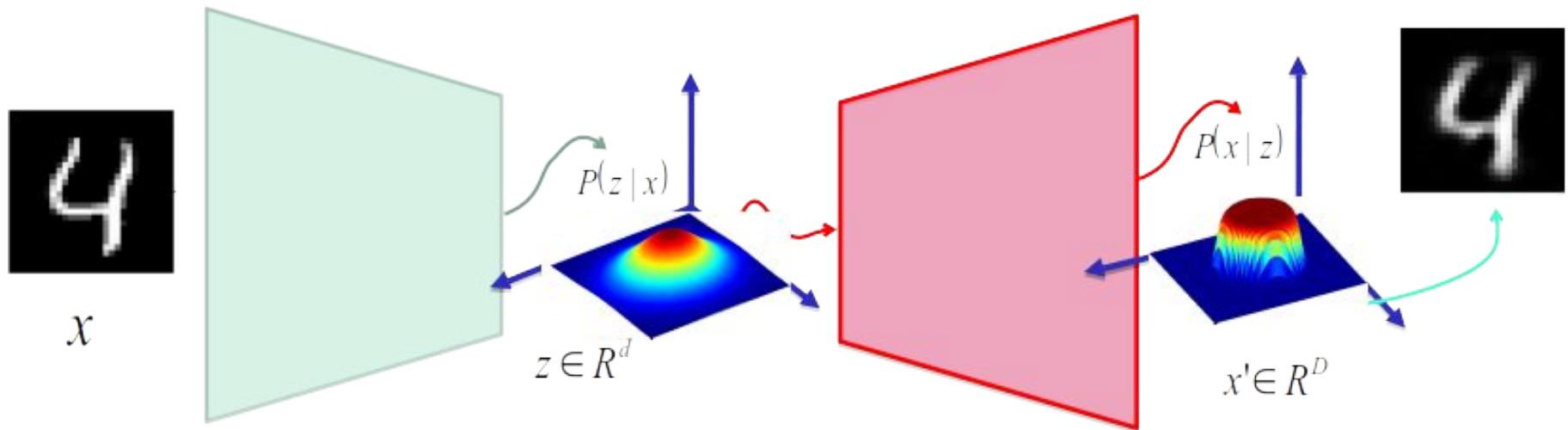
Le **décodeur** peut également produire une distribution de probabilités

$$P(x|z)$$

et x' est un point échantillonné au hasard de $P(x|z)$

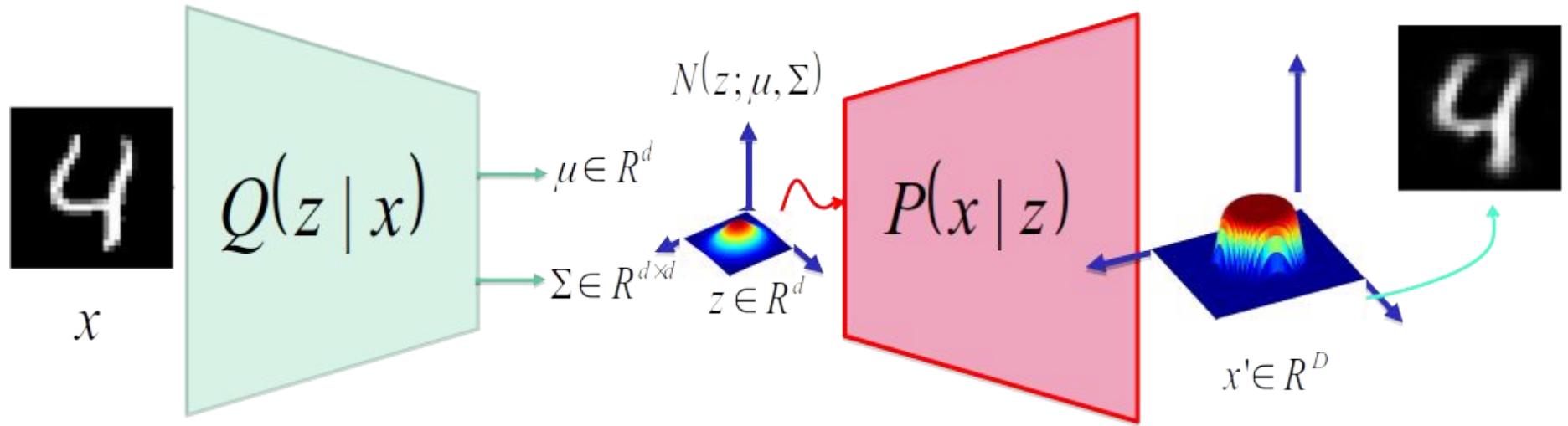
Autoencodeur variationnel

Remarque 3



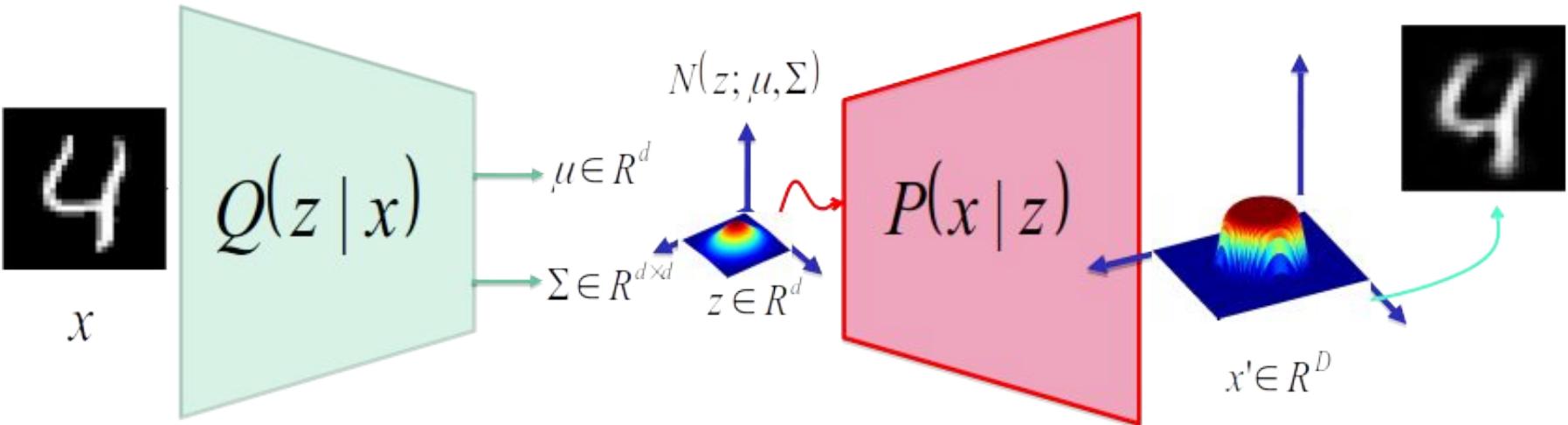
La distribution $P(z | x)$ peut être très complexe et difficile à échantillonner, on va donc l'approximer par une distribution plus simple... une gaussienne

$$Q(z | x) \approx P(z | x)$$



$Q(z | x) \sim \text{Gaussienne}$

Autoencodeur variationnel Remarque 4

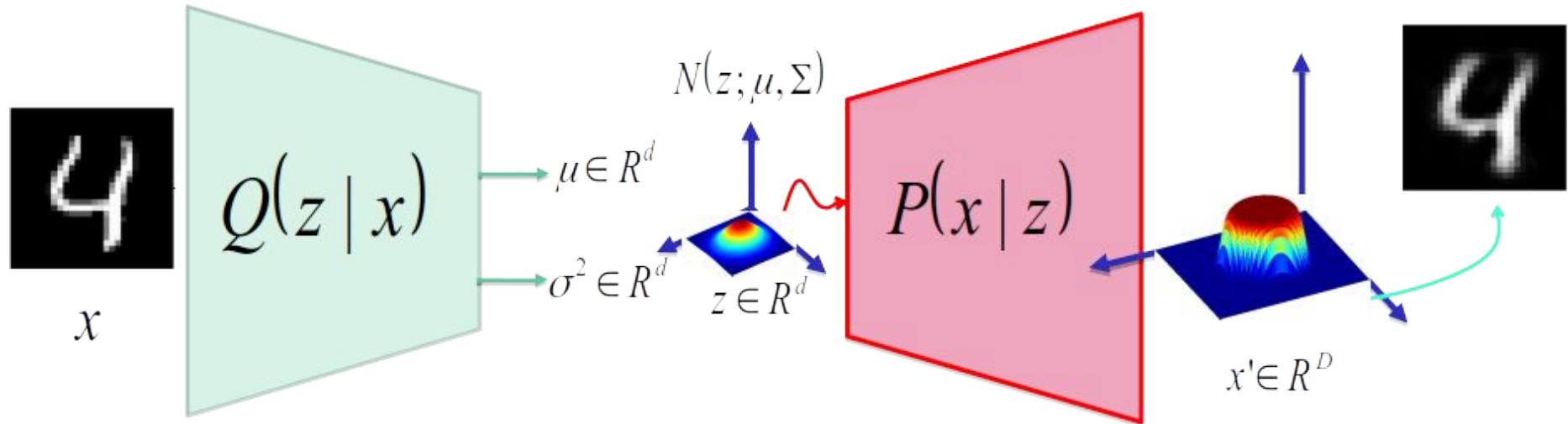


Pour simplifier les calculs, on va supposer que Σ est diagonale

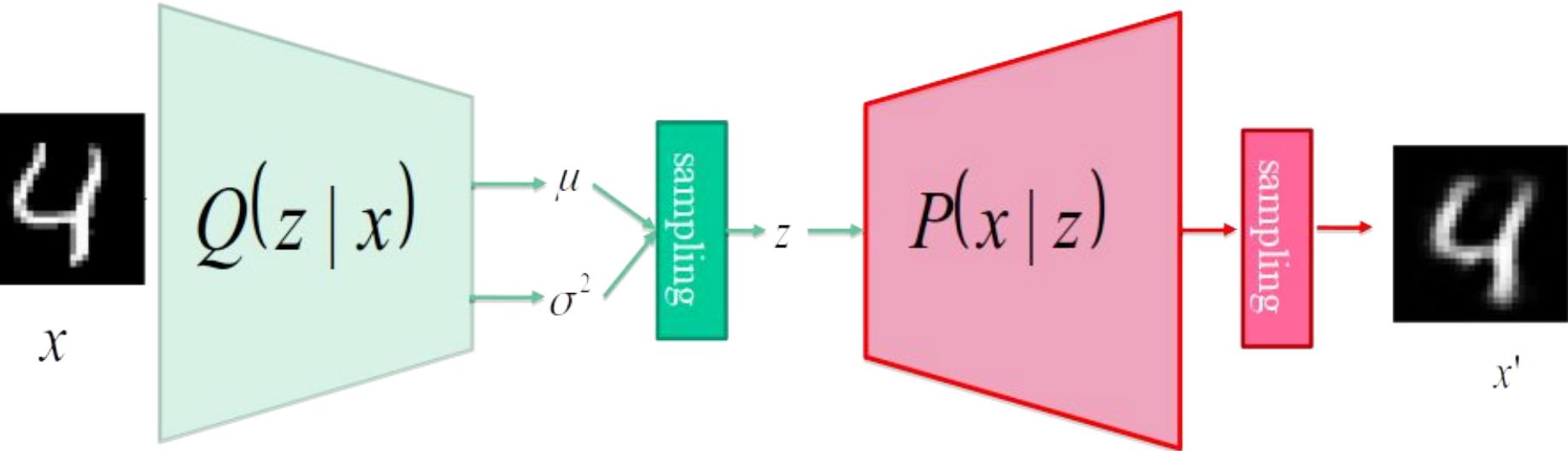
$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3^2 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \sigma_d^2 \end{pmatrix}$$

On va donc **prédirer un vecteur de variances et non une matrice**
Présume que les pixels sont dépendants de z , mais indépendants entre eux

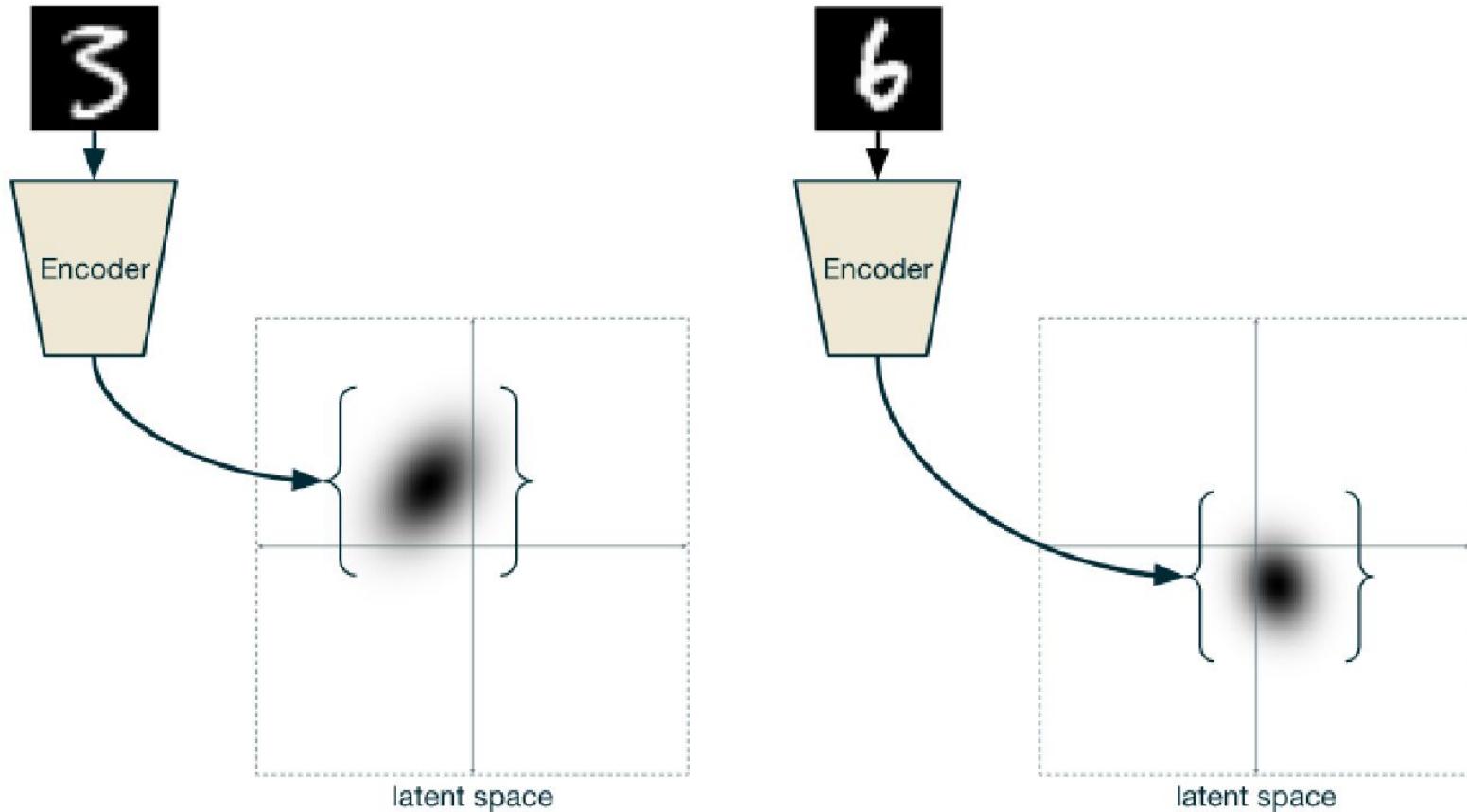
Autoencodeur variationnel



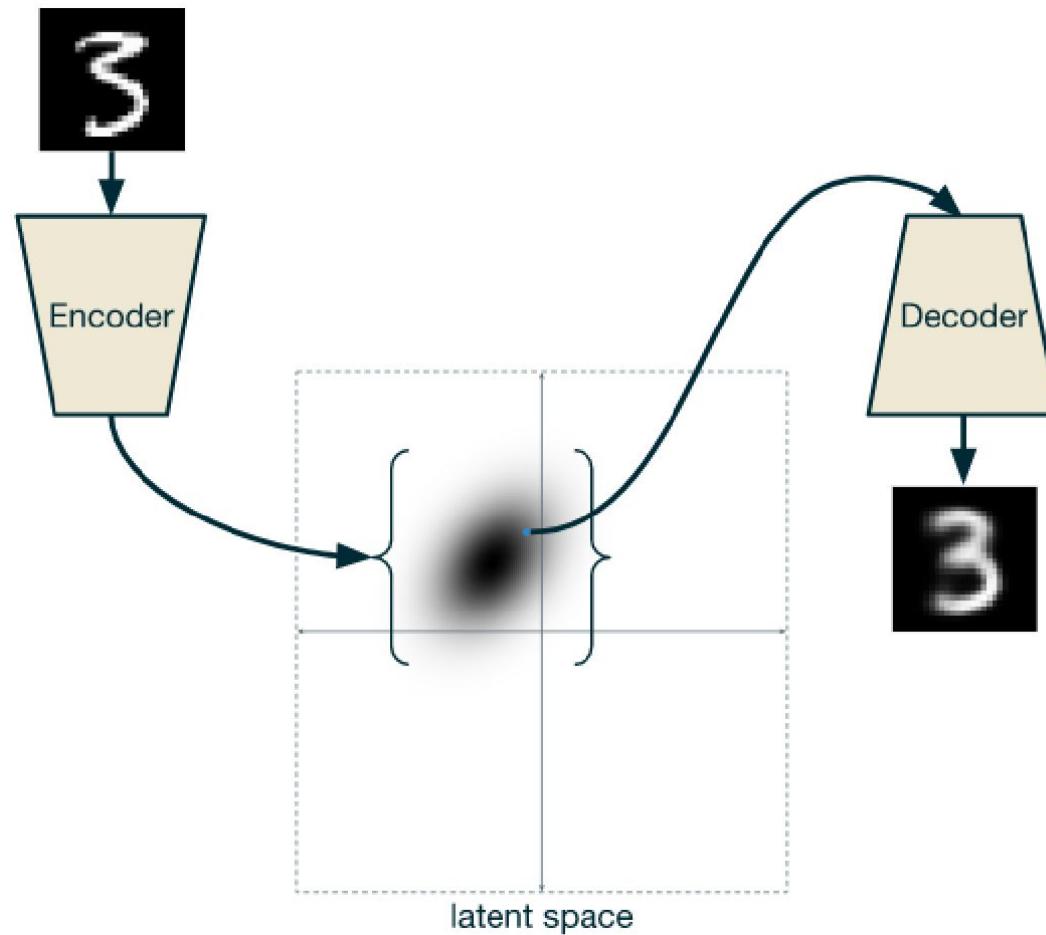
Autoencodeur variationnel



Autre façon de voir les choses...

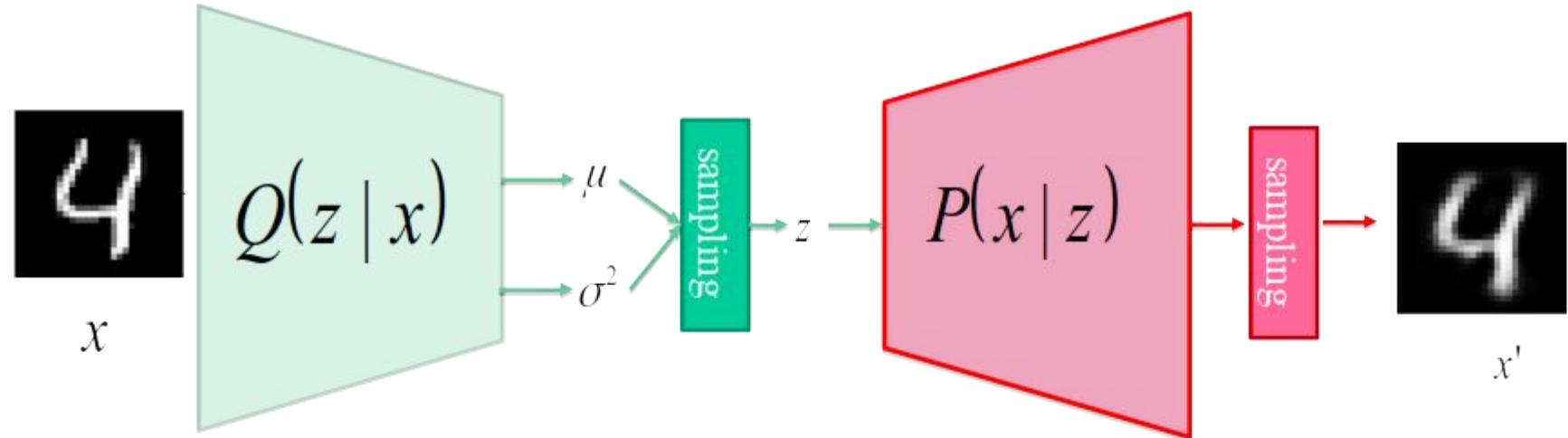


Autre façon de voir les choses...



Autoencodeur variationnel

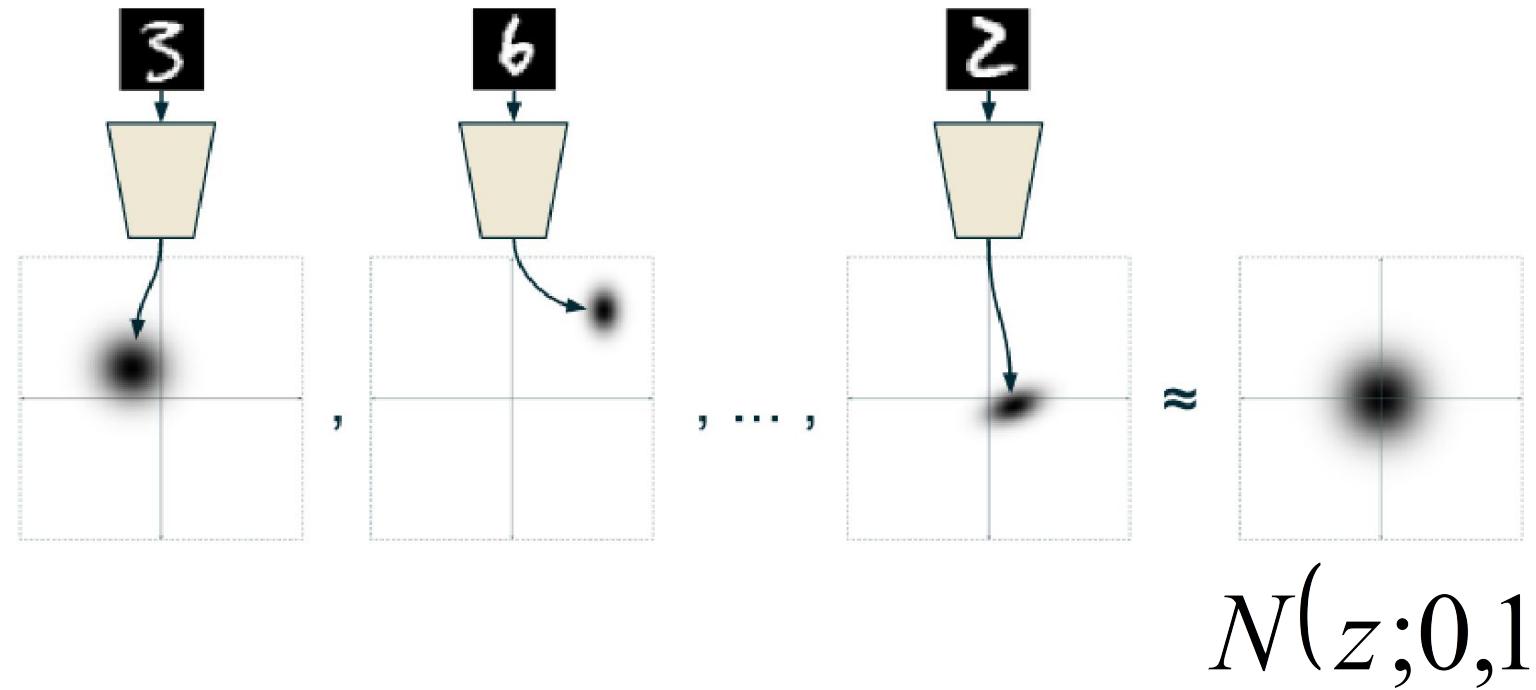
Remarque 5



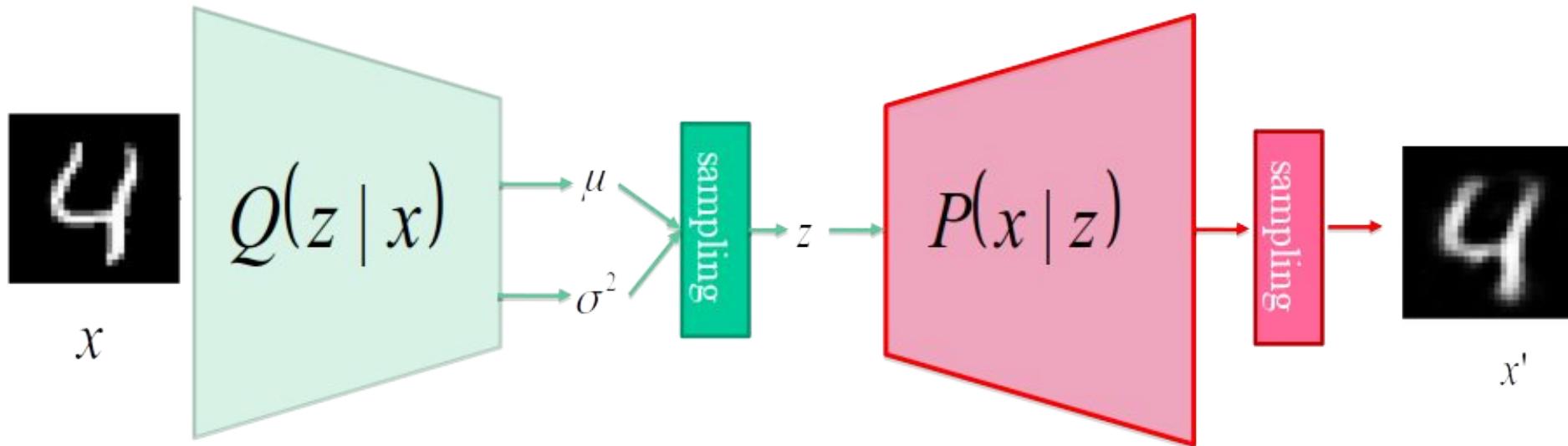
Distribution *a priori* de z : gaussienne centrée à 0 et de variance 1

$$P(z) = N(z; 0, 1)$$

Autre façon de voir les choses...



Autoencodeur variationnel



ELBO loss : Evidence Lower Bound

$$Loss = KL(N(z; 0, 1), N(z; \mu, \Sigma)) - \log(P(x | z))$$

Perte encodeur

Perte décodeur

Autoencodeur variationnel

D.Kingma, M.Welling, Auto-Encoding Variational Bayes, arXiv:1312.6114v10 ([Annexe B](#))

ELBO loss : Evidence Lower Bound

$$Loss = \underbrace{KL(N(z;0,1), N(z; \mu, \Sigma))}_{\text{Perte encodeur}} - \underbrace{\log(P(x | z))}_{\text{Perte décodeur}}$$

Si on suppose que $P(x|z)$ est gaussien

Autoencodeur variational

D.Kingma, M.Welling, Auto-Encoding Variational Bayes, and

ELBO loss : Evidence Lower Bound

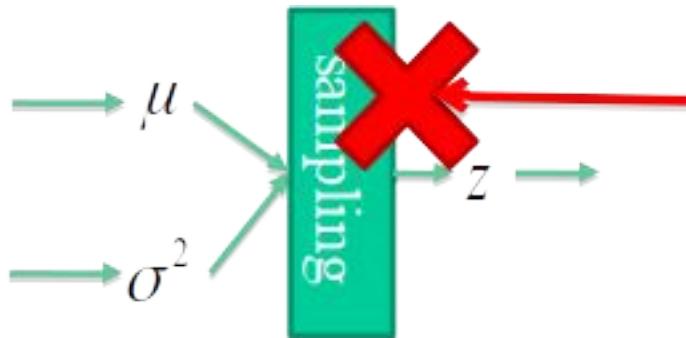
À voir au prochain
cours

Si on suppose que $P(x|z)$ est gaussien

$$Loss = \frac{1}{2} \sum_{i=1}^d (1 + \log(\sigma_i^2) + \mu_i^2 - \sigma_i^2) - \lambda ||\vec{x} - \vec{x}'||^2$$


Autoencodeur variationnel

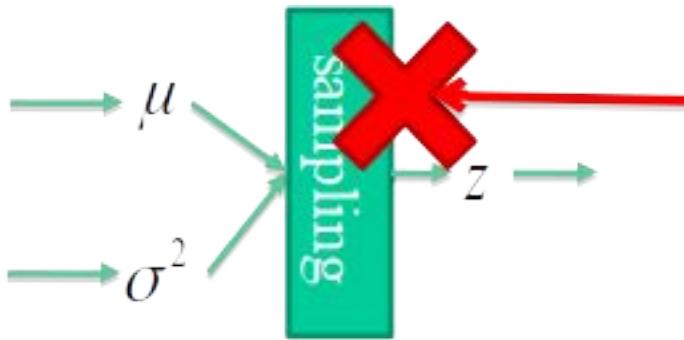
Remarque 6



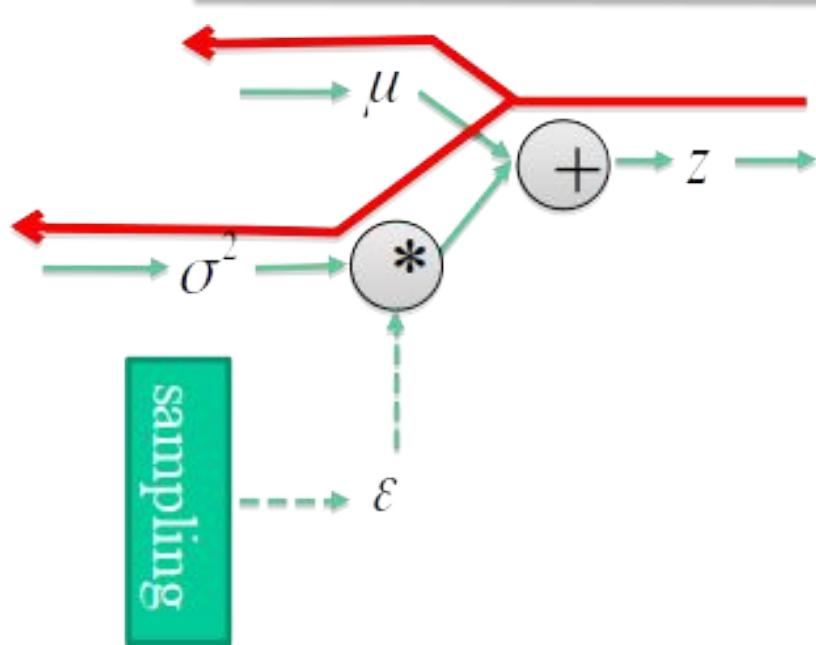
Pas de rétro-propagation à travers
un processus d'échantillonage
 $z \sim N(z; \mu, \Sigma)$

Autoencodeur variationnel

Remarque 6



Pas de rétro-propagation à travers
un processus d'échantionnage
 $z \sim N(z; \mu, \Sigma)$



Reparameterization trick

$$z = \mu + \varepsilon \sigma^2$$

Autoencodeur variationnel jouet MNIST : d=32 dim

```
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128), nn.ReLU(True),
            nn.Linear(128, 64), nn.ReLU(True),
            nn.Linear(64, 32*2)
        self.decoder = nn.Sequential(
            nn.Linear(32, 64), nn.ReLU(True),
            nn.Linear(64, 128), nn.ReLU(True),
            nn.Linear(128, 28 * 28))

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return mu + eps*std

    def forward(self, x):
        enc_x = self.encoder(x)
        mu = enc_x[:, :32]
        logvar = stats[:, 32:]
        z = self.reparameterize(mu, logvar)
        return self.decoder(z), mu, logvar
```

} Reparameterization
trick

Autoencodeur variationnel jouet MNIST : d=32 dim

```
def loss_function(recon_x, x, mu, logvar):
    L2 = torch.sum((recon_x-x).pow(2))

    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

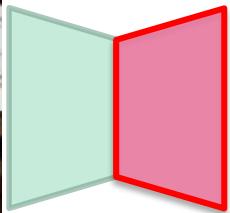
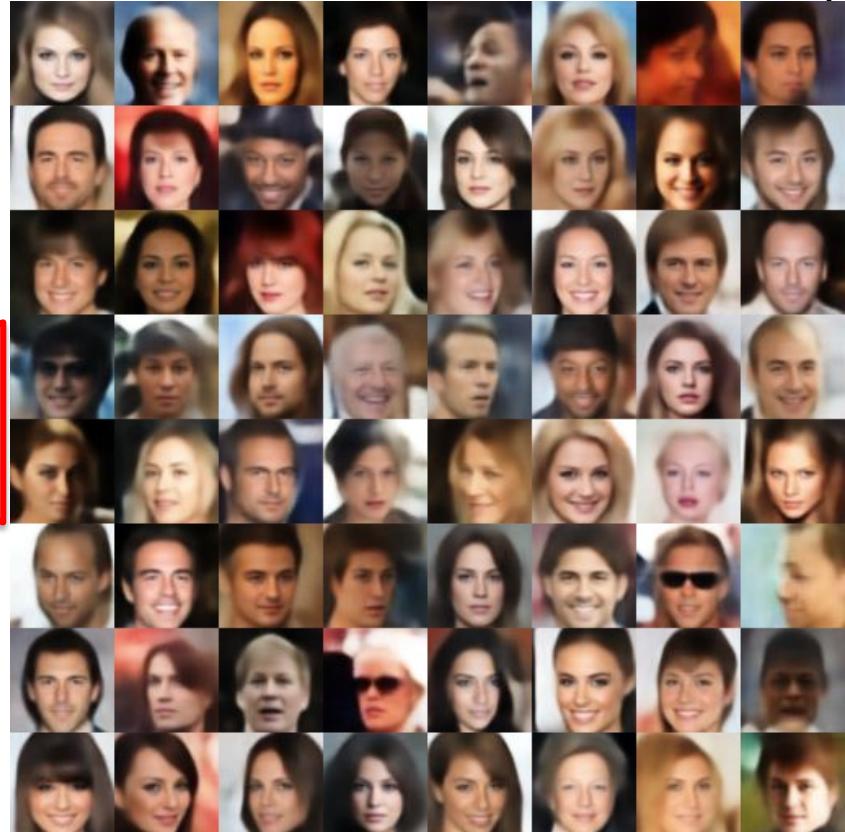
    return L2 + KLD
```

Ex.: base de données *CelebA*

x

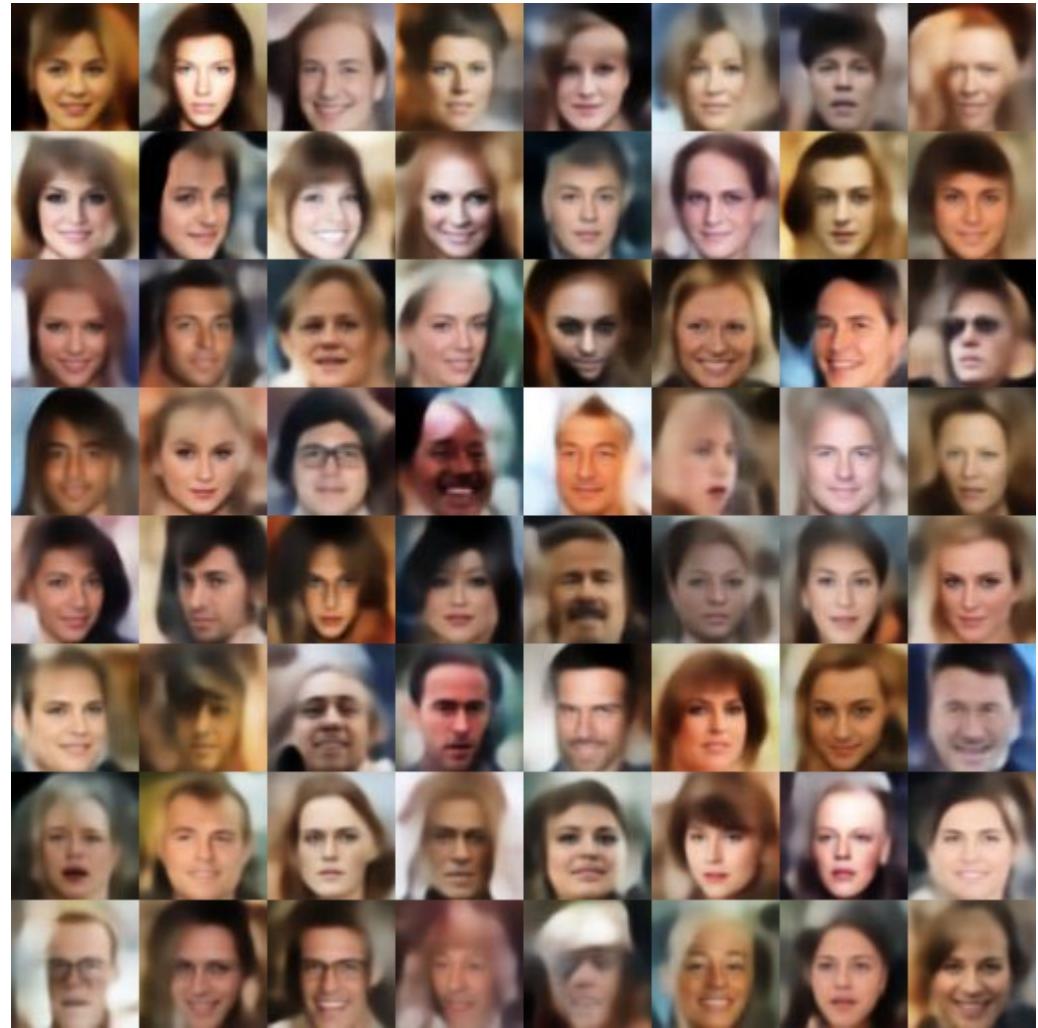
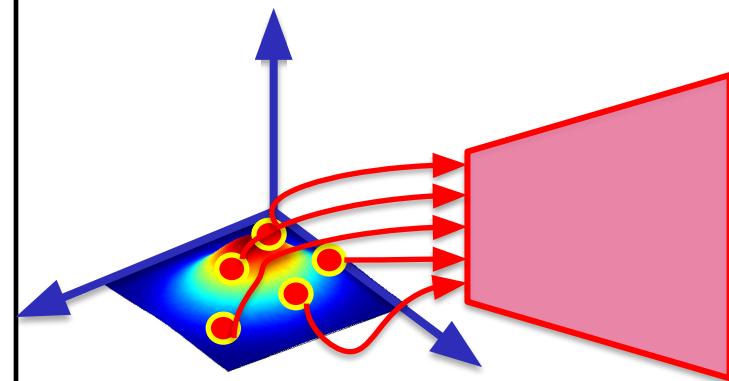


x'



Ex.: base de données *CelebA*

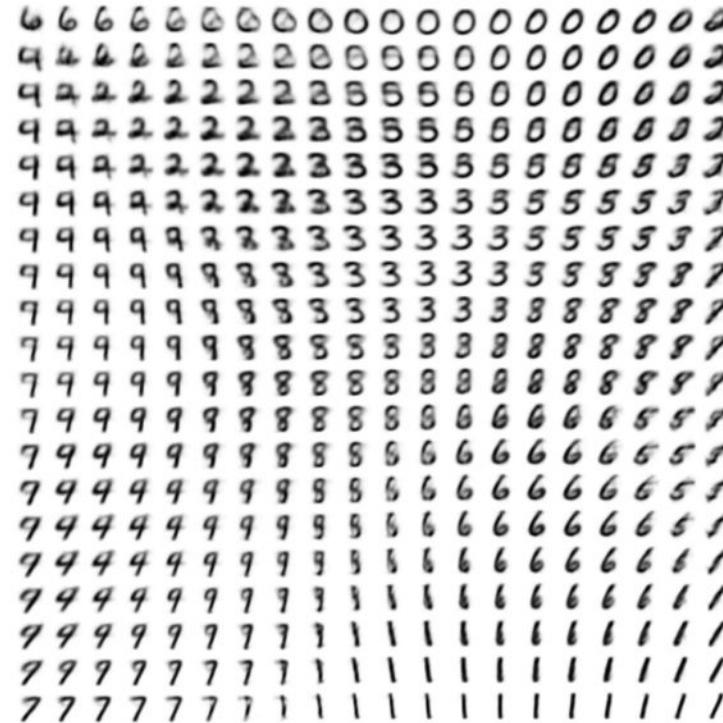
Décodage d'échantillons aléatoires z



Ex.: VAE



(a) Learned Frey Face manifold

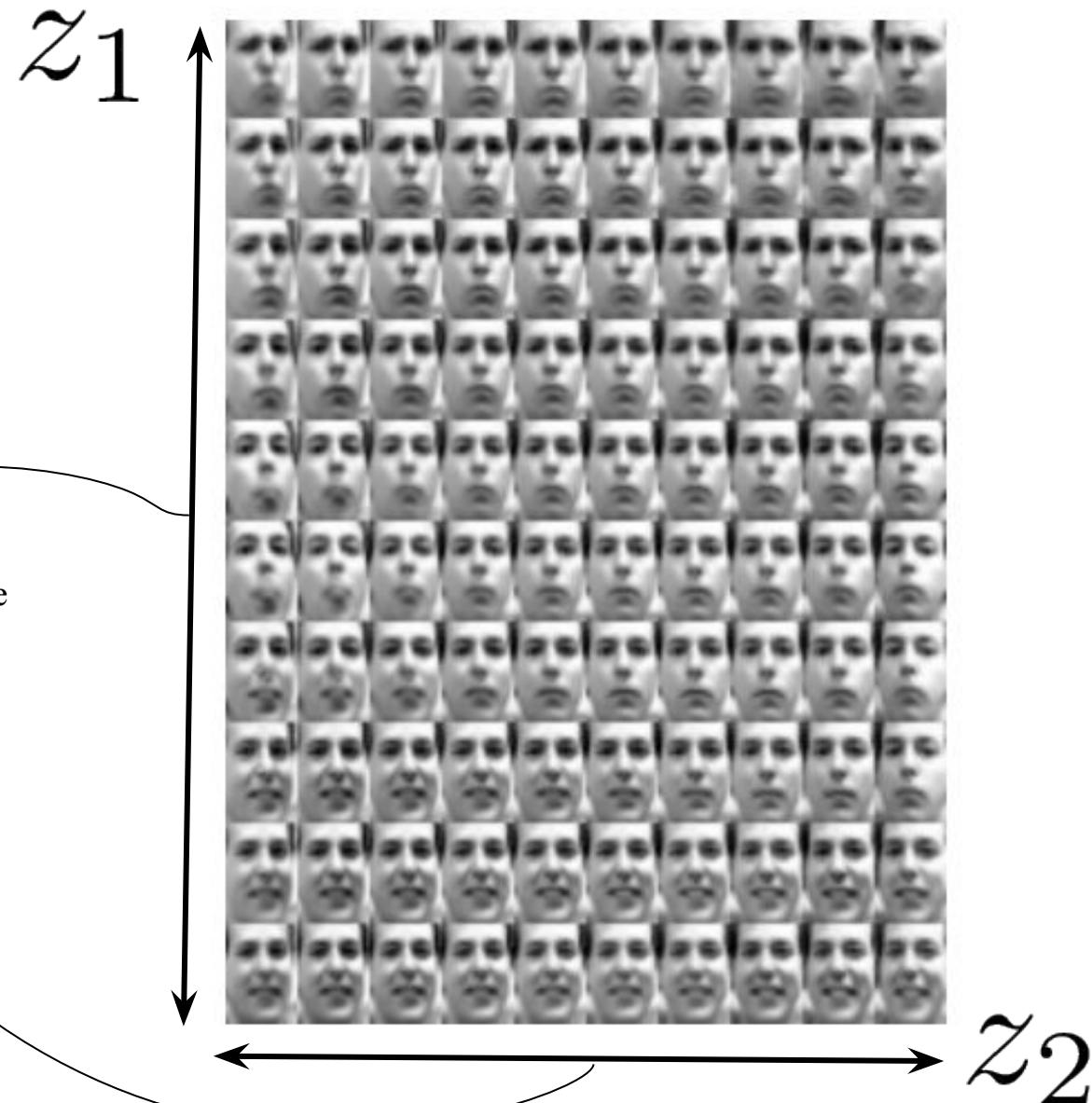


(b) Learned MNIST manifold

Figure 4: Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables \mathbf{z} . For each of these values \mathbf{z} , we plotted the corresponding generative $p_{\theta}(\mathbf{x}|\mathbf{z})$ with the learned parameters θ .

Ex.: VAE

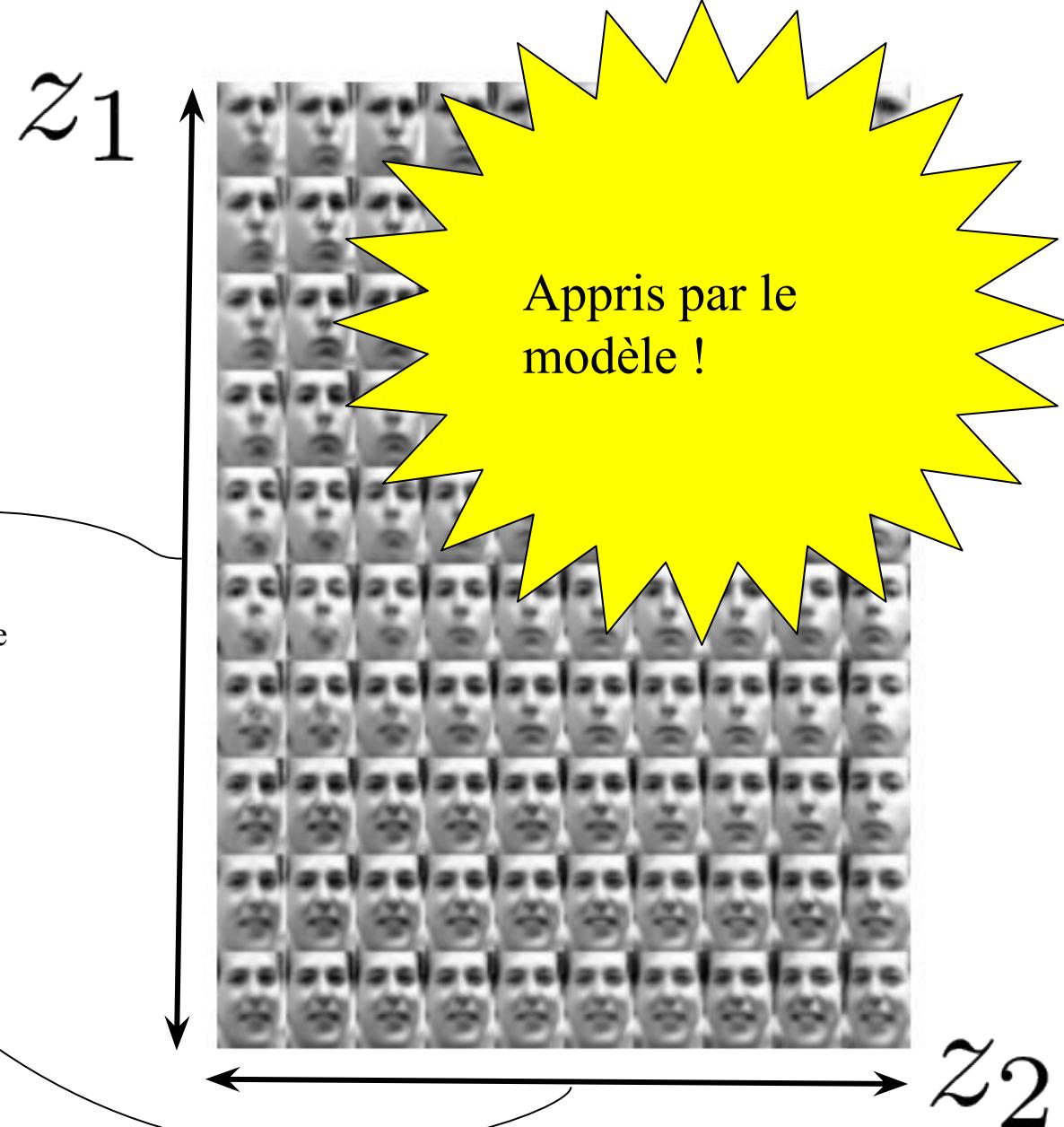
Dimension du “sourire”
Dimension de l’orientation de la tête



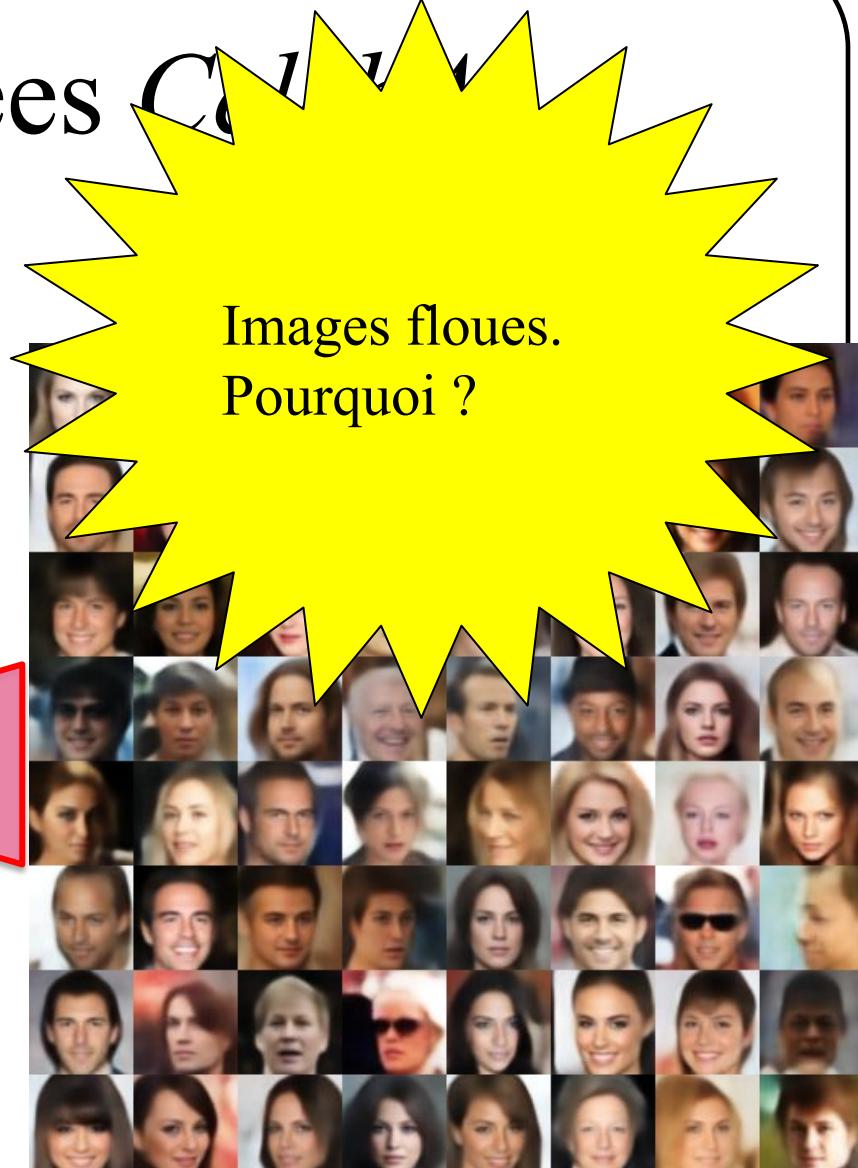
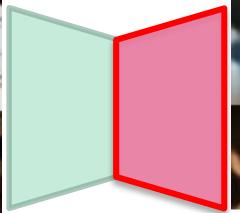
Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Ex.: VAE

Dimension du “sourire”
Dimension de l’orientation de la tête

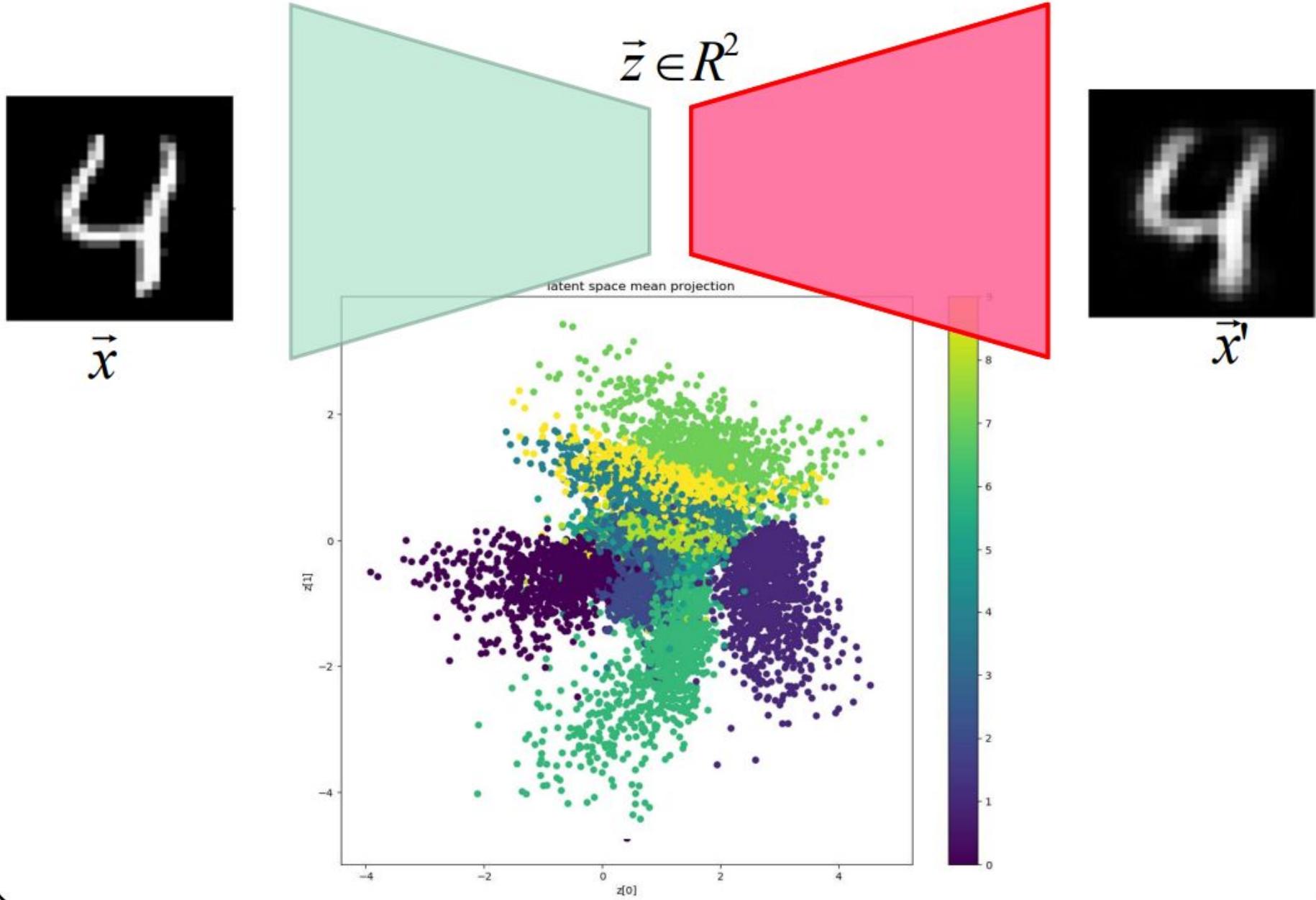


Ex.: base de données

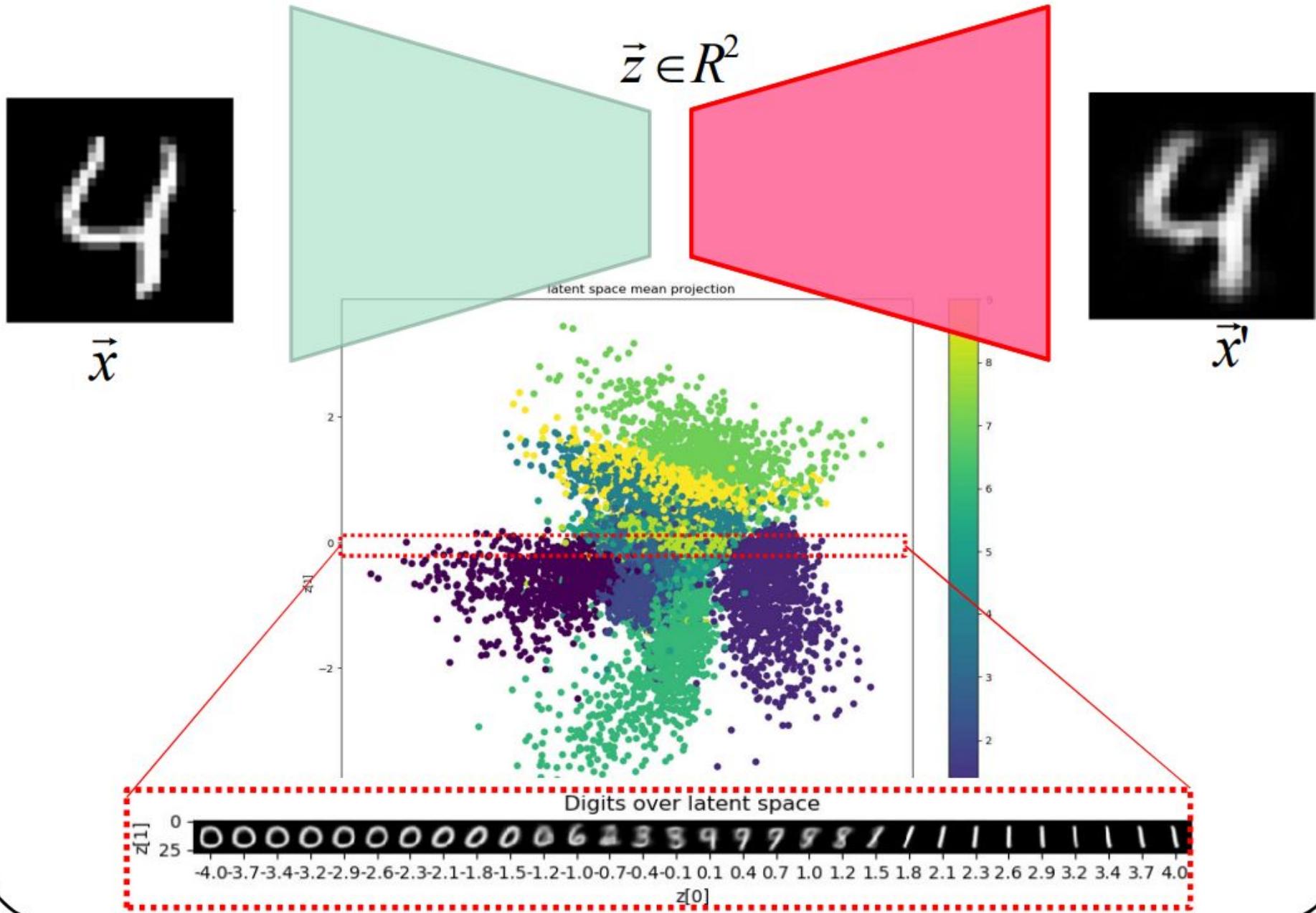
 χ 

NOTE: on peut également conditionner un espace latent en y
« **injectant les gradients d'un 2^e réseau de neurones** »

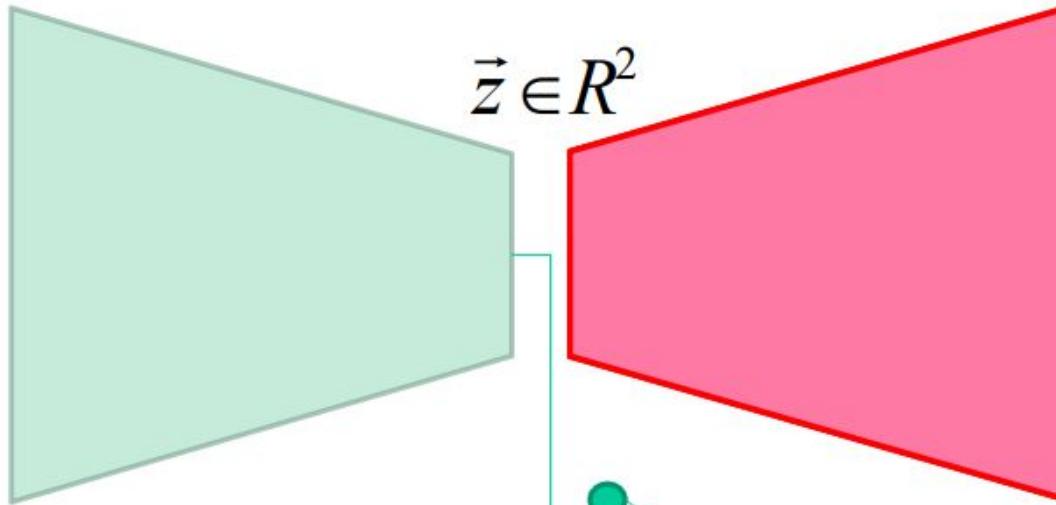
Autoencodeur de base



Autoencodeur de base



Autoencodeur contraint

 \vec{x} $t=4$  \vec{x}'

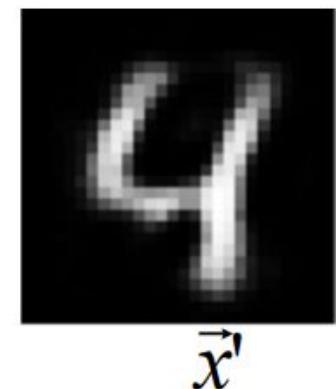
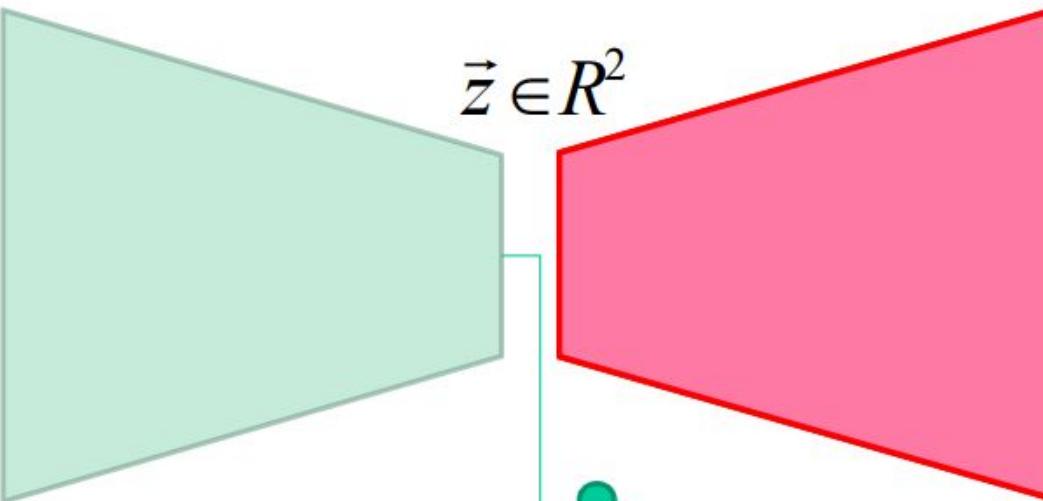
Régression linéaire
(perceptron sans couche cachée)

Autoencodeur contraint

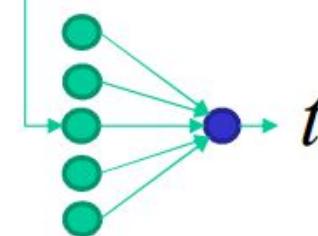


\vec{x}

$t=4$



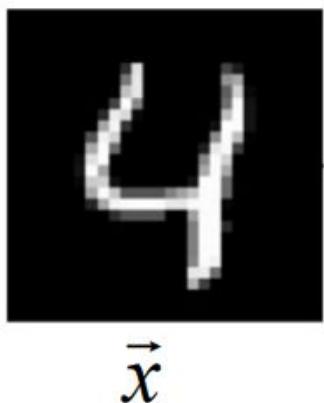
\vec{x}'



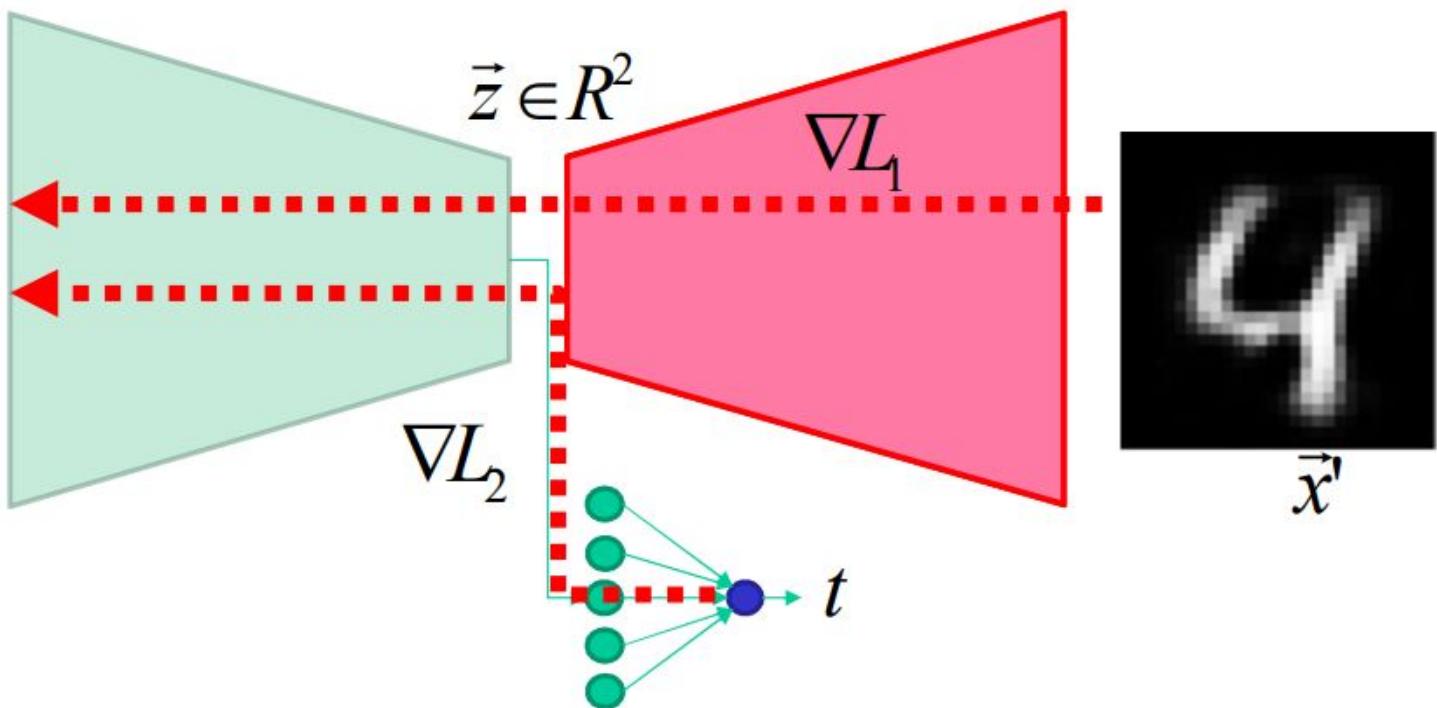
$$L_1 = \|\vec{x} - \vec{x}'\|^2$$

$$L_2 = \|\hat{t} - t\|^2$$

Autoencodeur contraint



\vec{x}
 $t=4$

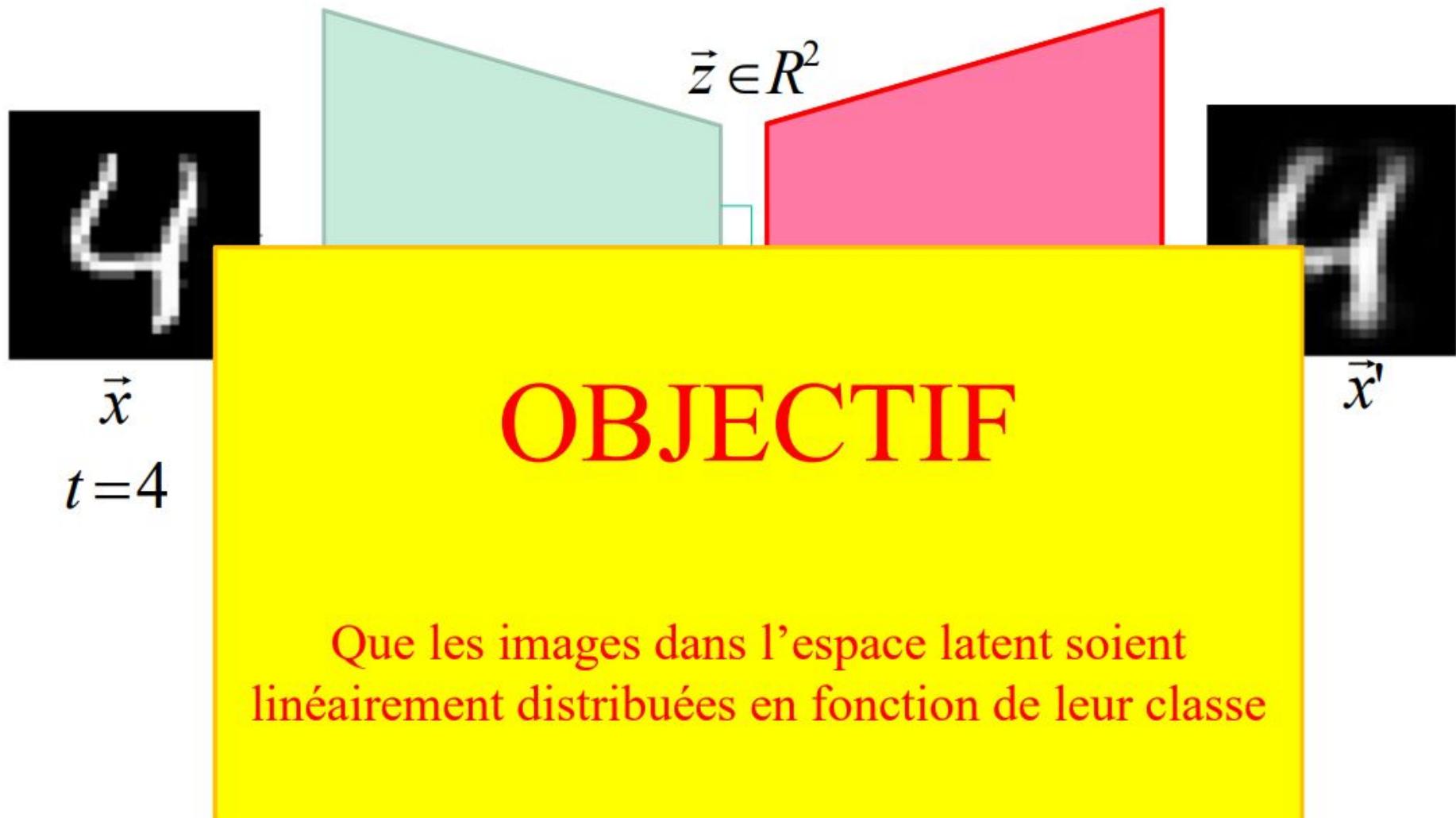


$$L_{\text{décodeur}} = L_1$$

$$L_{\text{reg}} = L_2$$

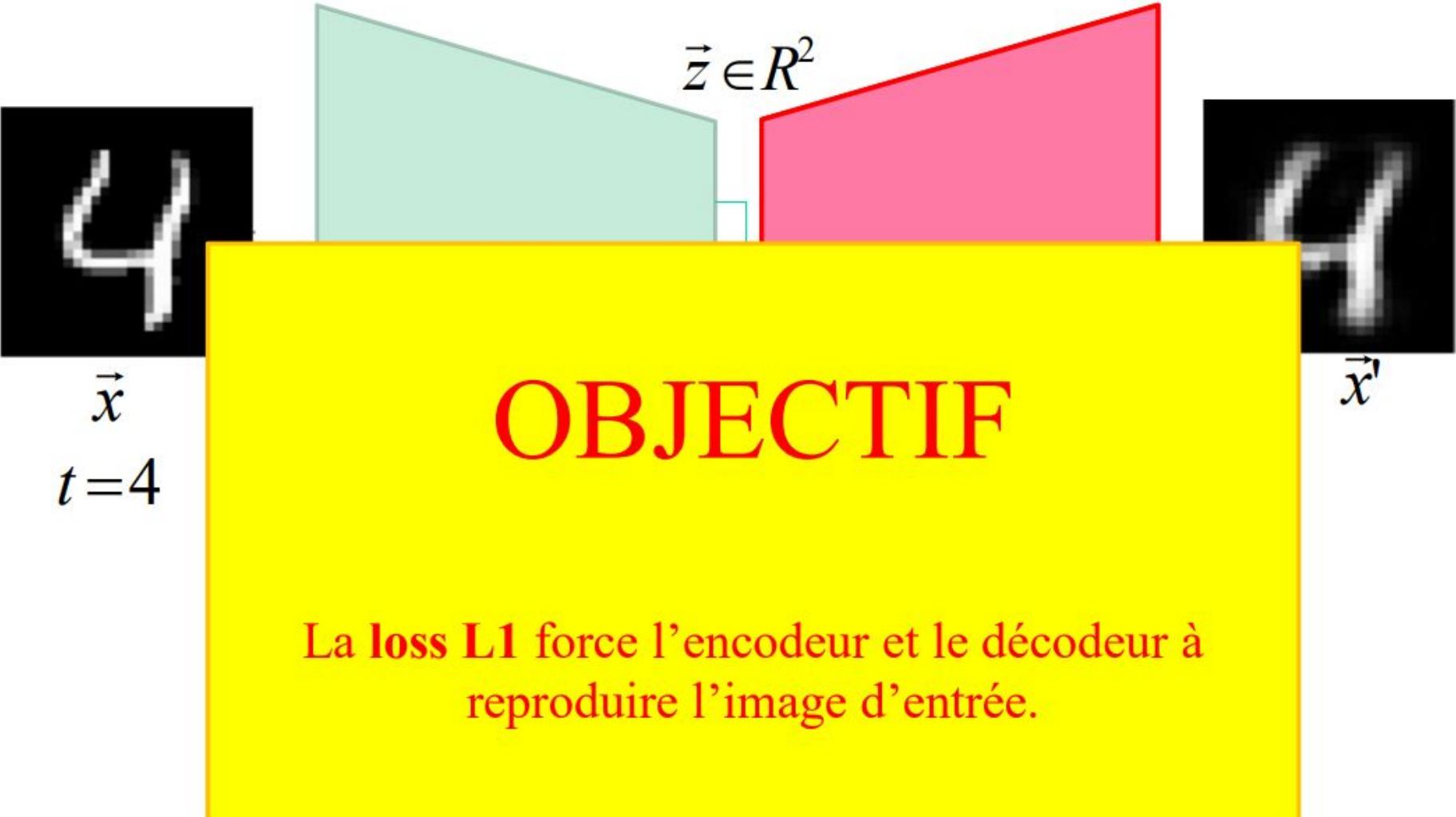
$$L_{\text{encodeur}} = L_1 + \lambda L_2$$

Autoencodeur contraint



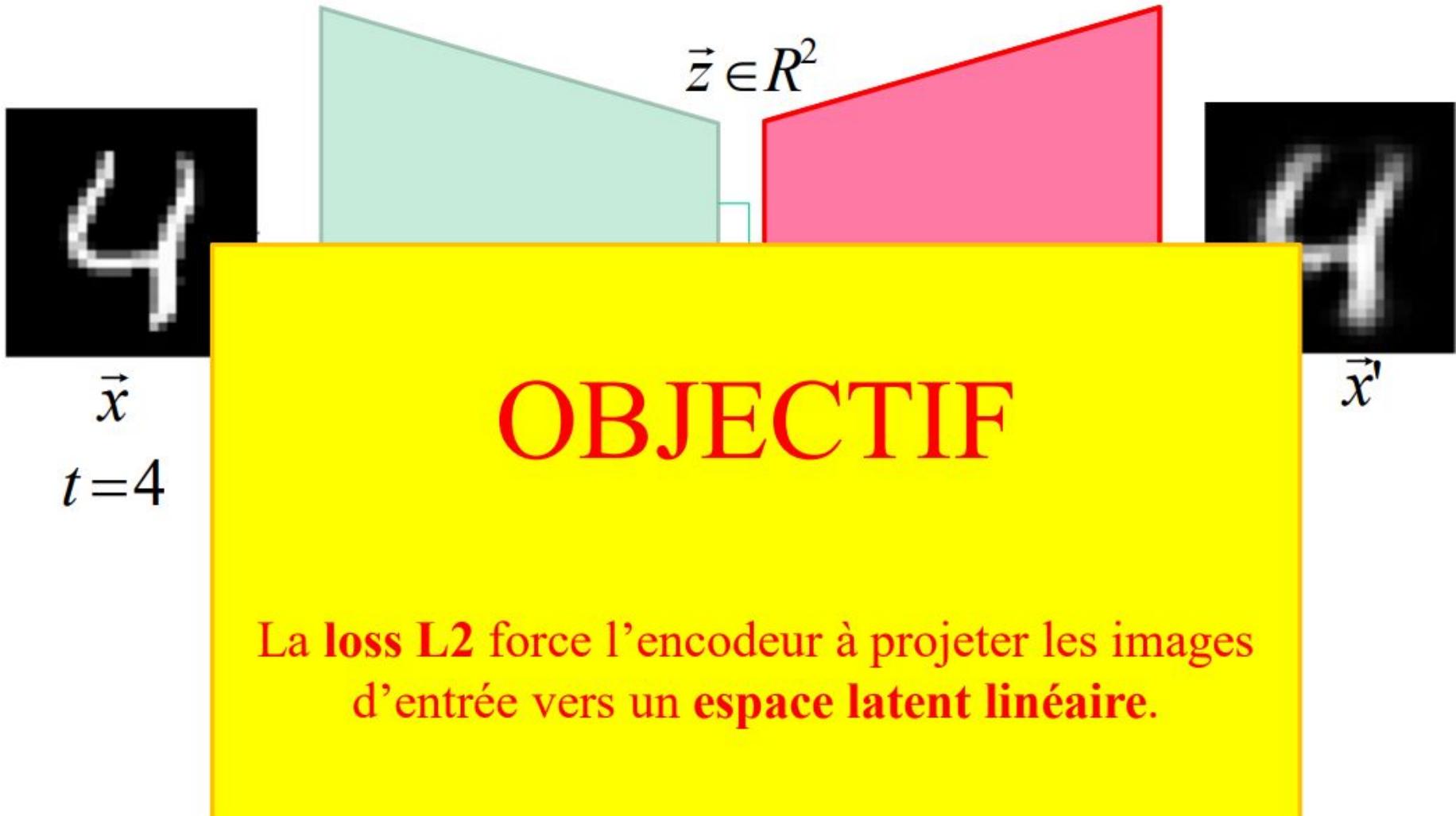
$$L_{encodeur} = L_1 + \lambda L_2$$

Autoencodeur contraint



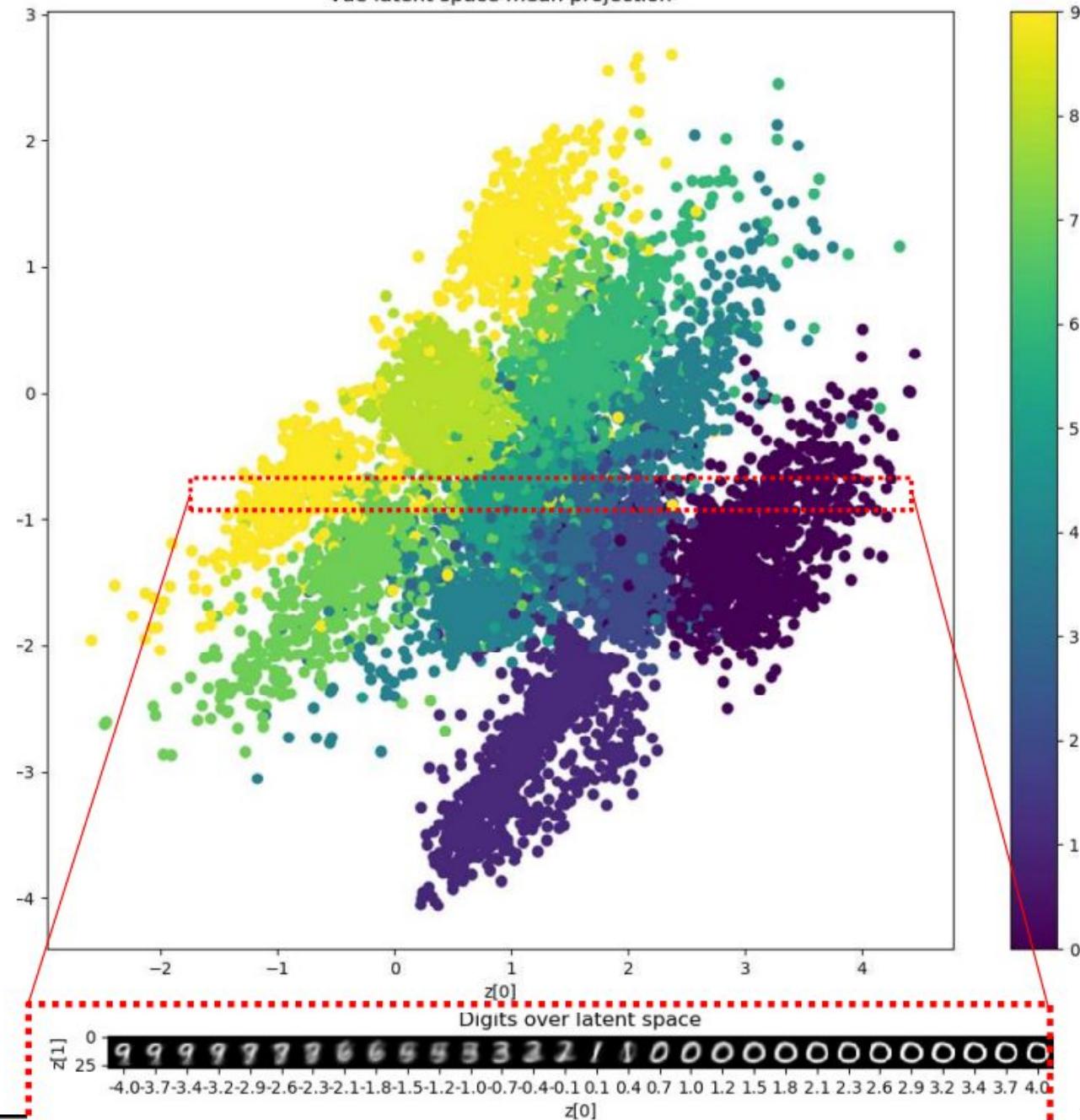
$$L_{encodeur} = L_1 + \lambda L_2$$

Autoencodeur contraint



$$L_{encodeur} = L_1 + \lambda L_2$$

Vae latent space mean projection

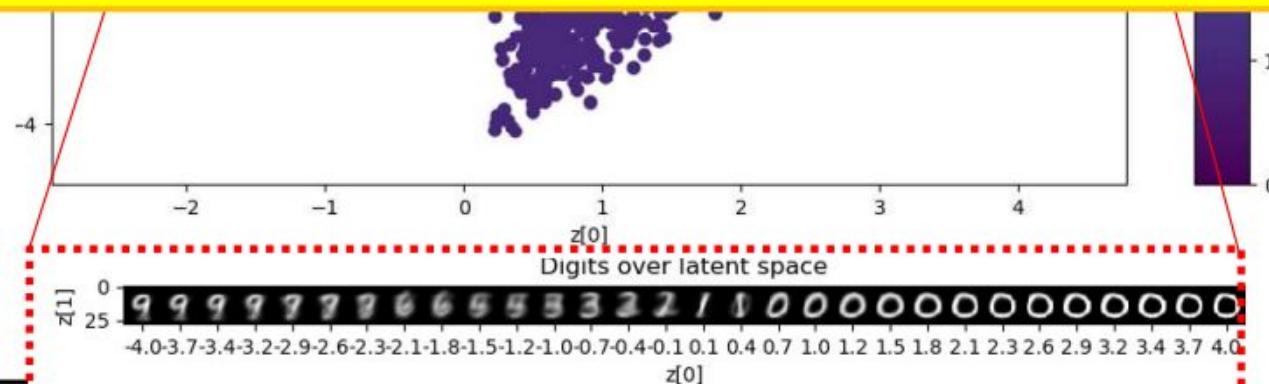


Vae latent space mean projection



NOTE:

D'autres réseaux de neurones et d'autres loss peuvent être utilisés! Tout dépend de l'application ... et **de votre créativité.**



Plusieurs tutoriels, VAE

- <https://ijdykeman.github.io/ml/2016/12/21/cvae.html>
- <https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/>
- <https://towardsdatascience.com/deep-latent-variable-models-unravel-hidden-structures-a5df0fd32ae2>
- C. Doersch, **Tutorial on Variational Autoencoders**, arXiv:1606.05908