

Réseaux de neurones

# IFT 780

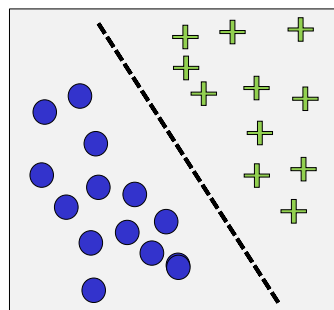
Modèles génératifs

Par  
Pierre-Marc Jodoin

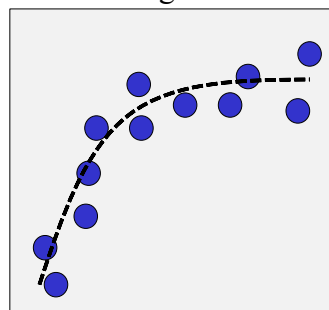
1

Jusqu'à présent : apprentissage supervisé

Classification



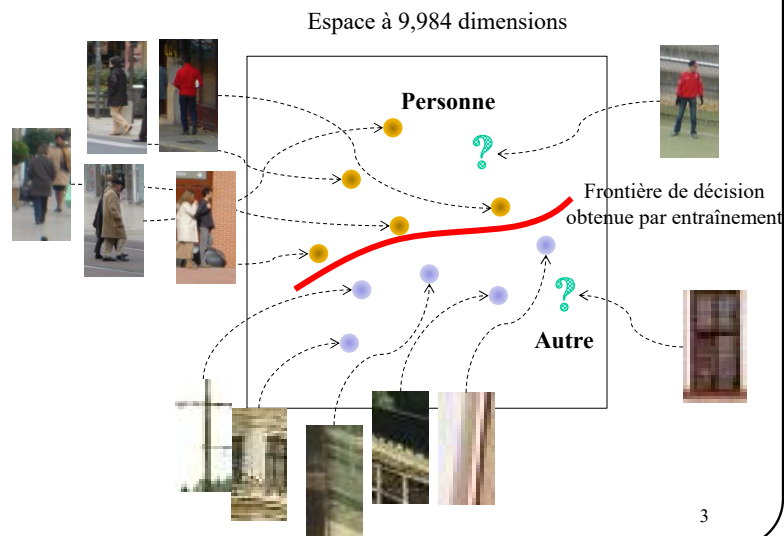
Régression



2

2

### *Inria person dataset*



3

## Apprentissage supervisé

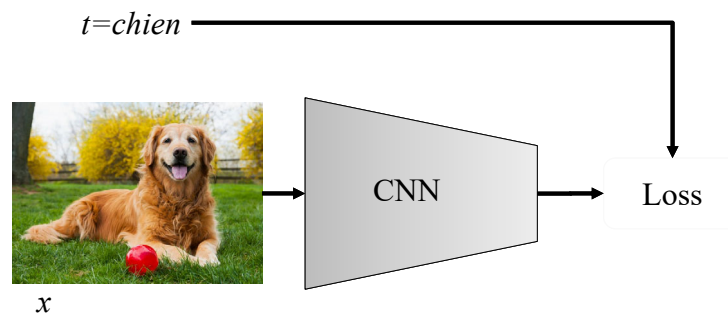
Deux grandes familles d'applications

- **Classification** : la cible est un indice de classe  $t \in \{1, \dots, K\}$ 
  - Exemple : reconnaissance de caractères
    - ✓  $\vec{x}$  : 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
    - ✓  $\vec{x}$  : vecteur contenant l'information sur l'activité économique de la journée
    - ✓  $t$  : valeur d'une action à la bourse le lendemain

4

4

## Apprentissage supervisé avec CNN



5

## Supervisé vs non supervisé

**Apprentissage supervisé** : il y a une cible

$$D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \dots, (\vec{x}_N, t_N)\}$$

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

$$D = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N\}$$

6

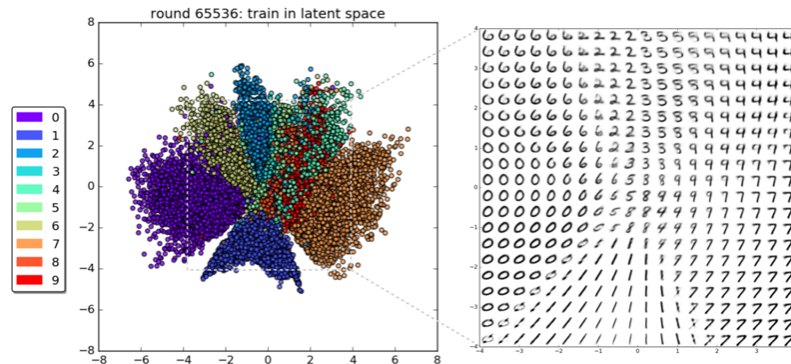
6

# Apprentissage non supervisé

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

**Applications** : clustering, visualization, comprehension, etc.

**Exemple** : visualization de la distribution des images MNIST



<https://blog.fastforwardlabs.com/2016/08/12/introducing-variational-autoencoders-in-prose-and.html>

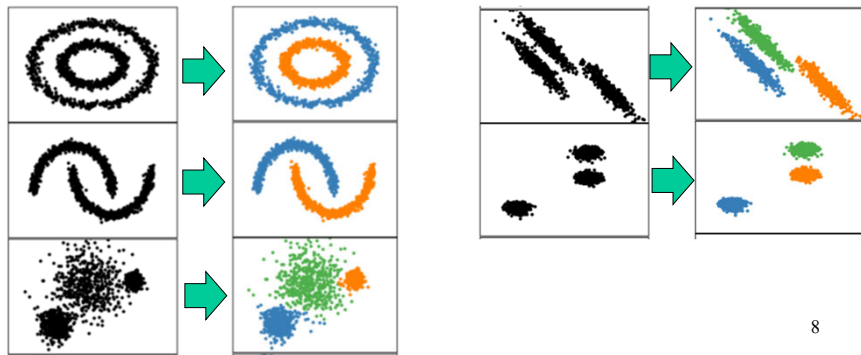
7

# 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"



8

8

Pourquoi une variable latente?

Plus facile de représenter  $p(\vec{x}, y), p(\vec{x} | y), p(y)$   
que  $p(\vec{x})$

**Plus d'info au tableau.**

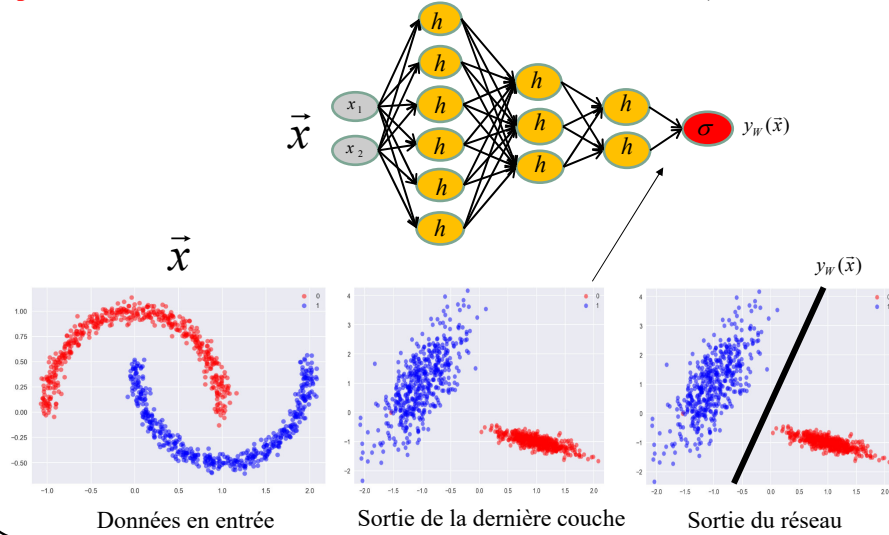
9

L'apprentissage non-supervisé par  
**réseaux de neurones**  
s'appuie sur  
**2 propriétés**

10

## Propriété 1

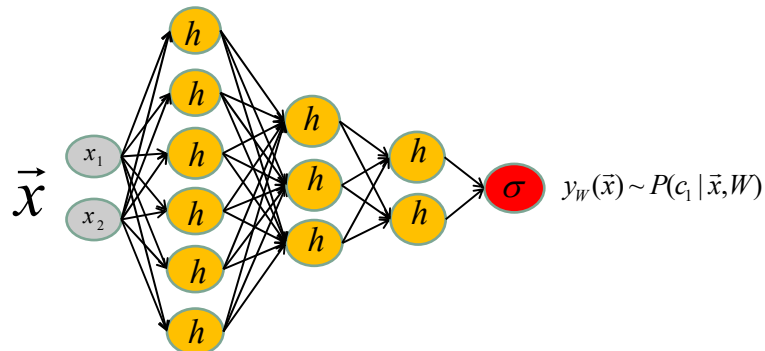
Les réseaux de neurones sont d'excellentes machines pour projeter des données brutes vers un **espace dimensionnel plus faible** dont les propriétés dépendent de la *loss* (ici **espace linéaire car la sortie du réseau est un classifieur linéaire**).



11

## Propriété 2

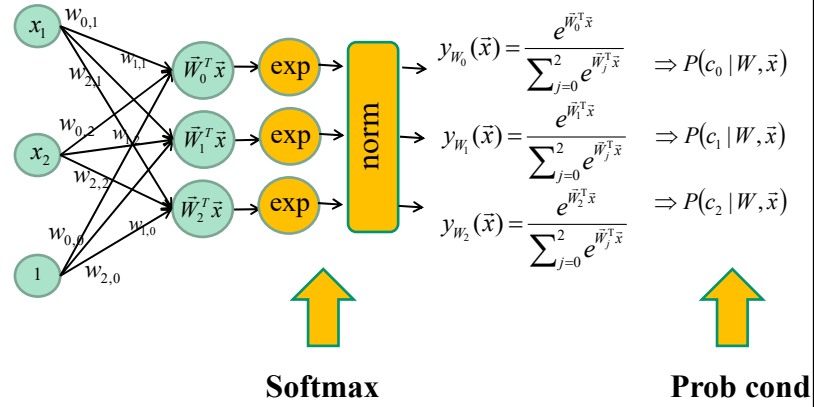
Les réseaux de neurones sont d'excellentes machines pour estimer des **probabilités conditionnelles**.



12

## Propriété 2

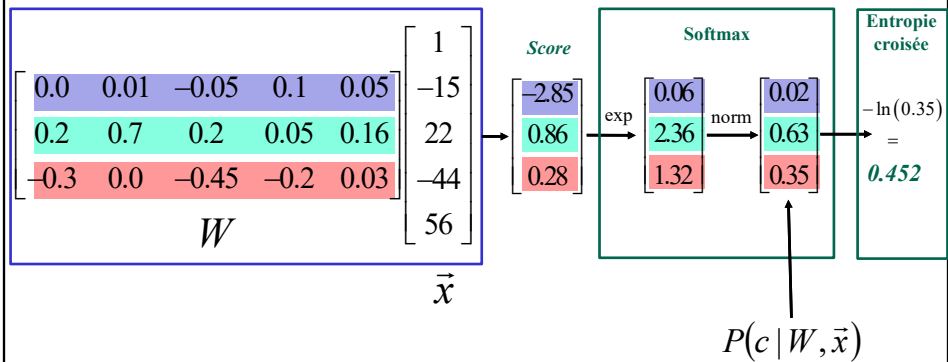
Les réseaux de neurones sont d'excellentes machines pour estimer des **probabilités conditionnelles**.



13

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

Rappel

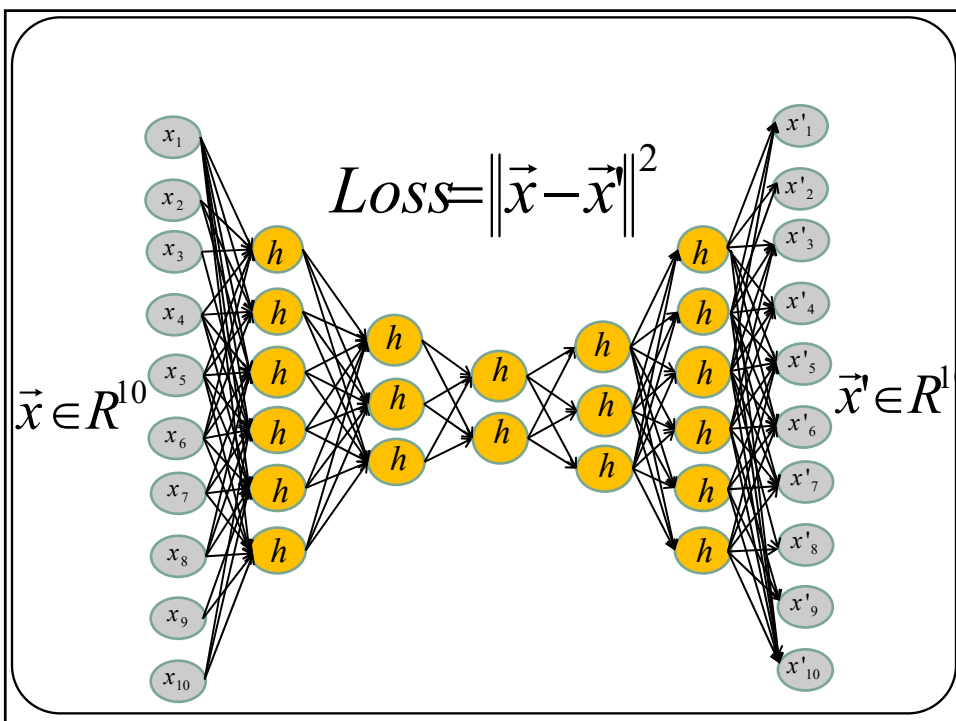


14

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

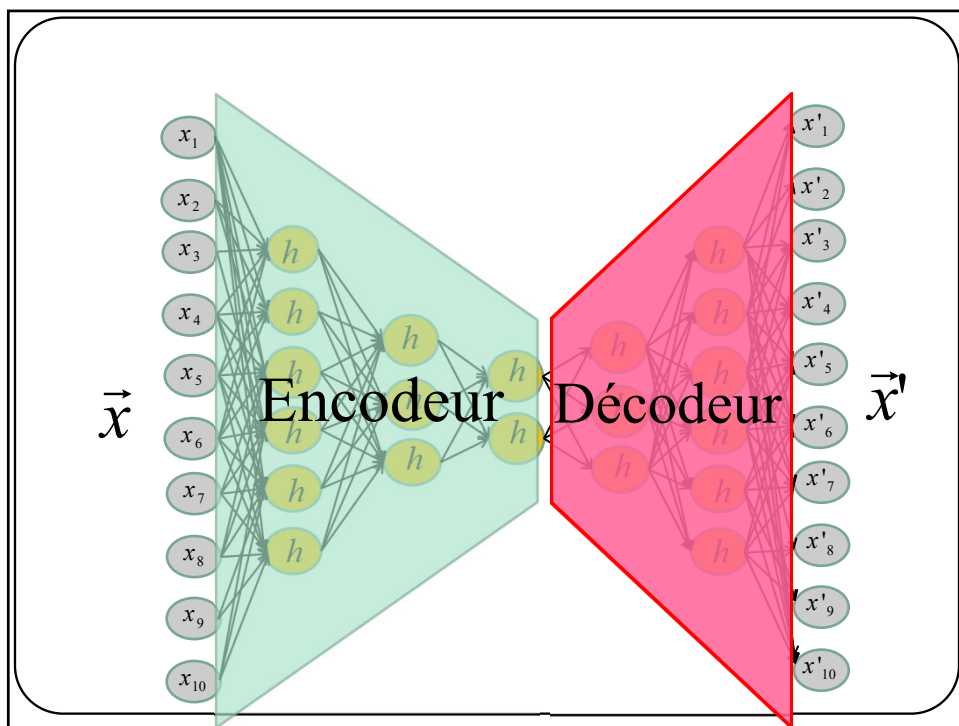


15

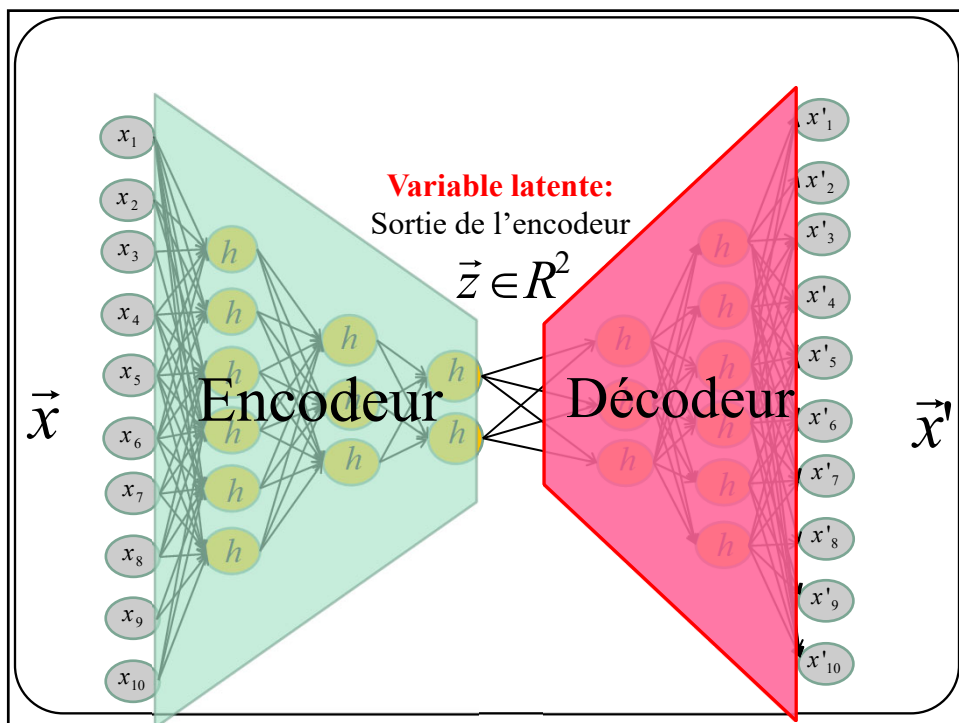


16

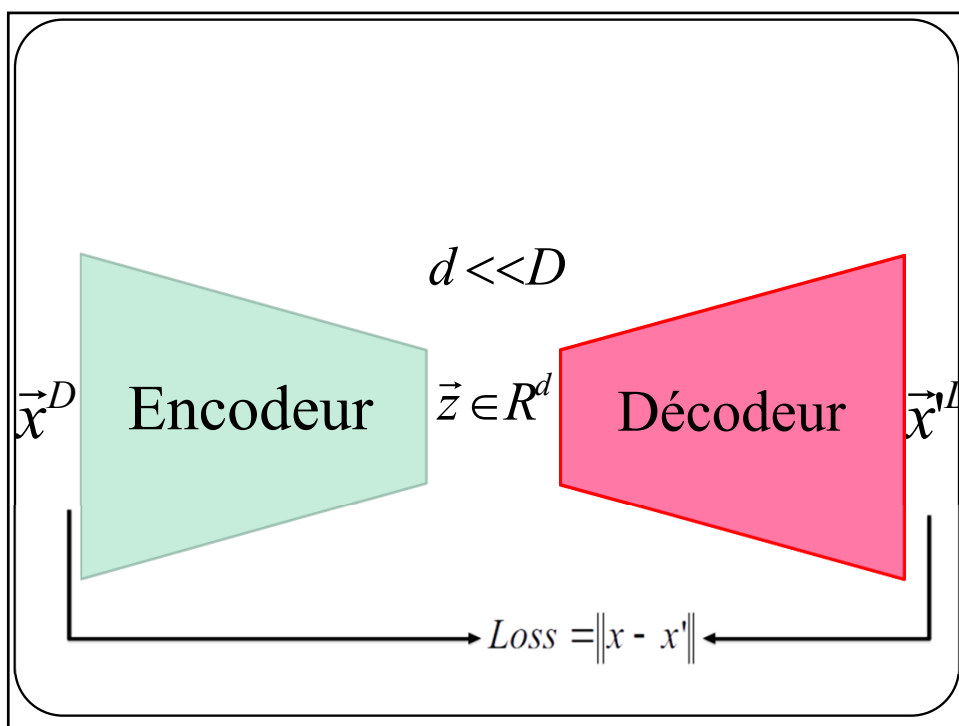




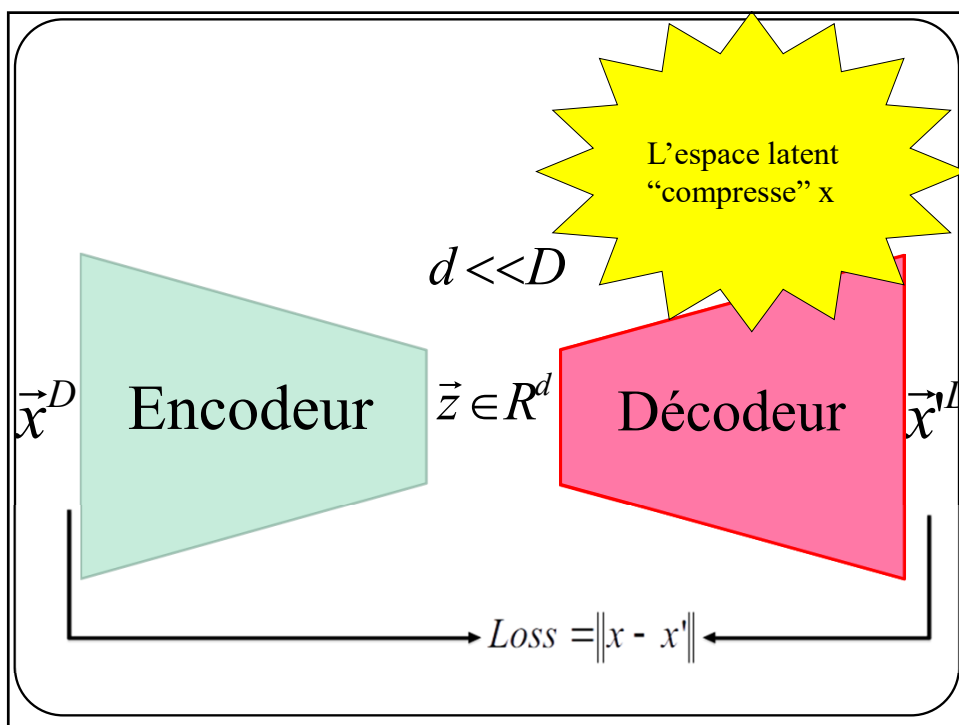
17



18

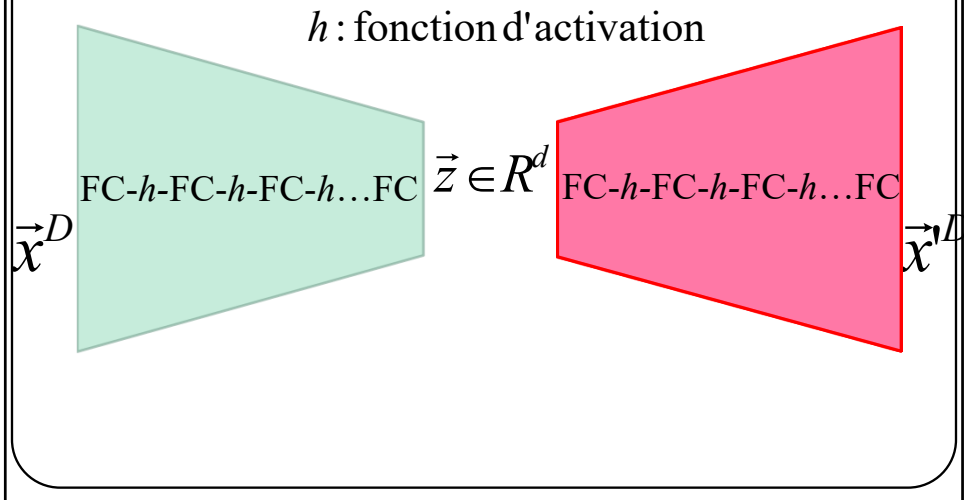


19

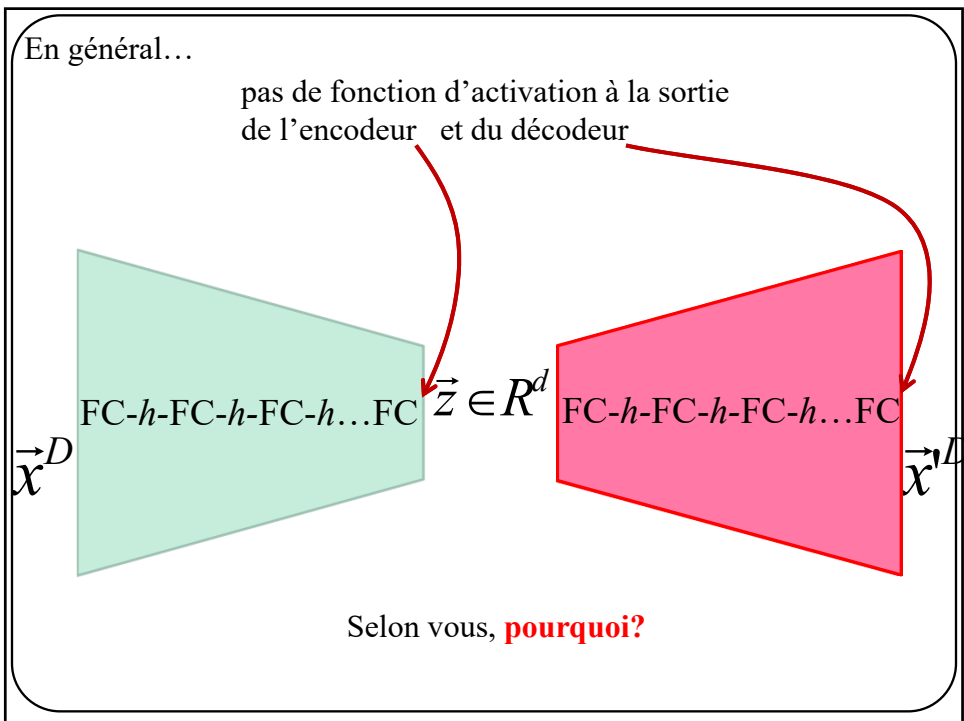


20

# Couches pleinement connectées

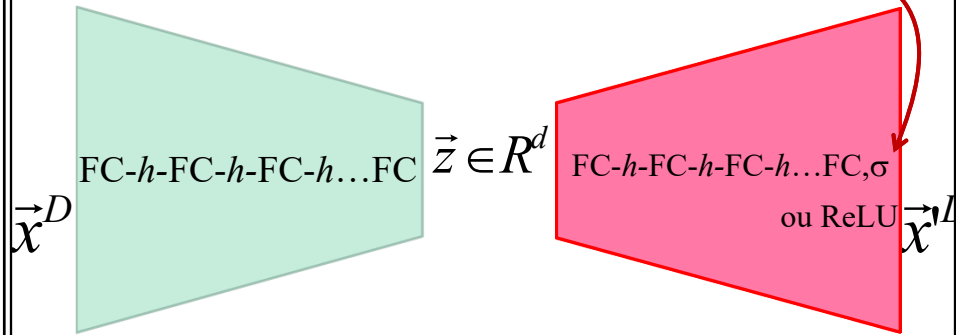


21



22

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 positifs et jamais négatifs

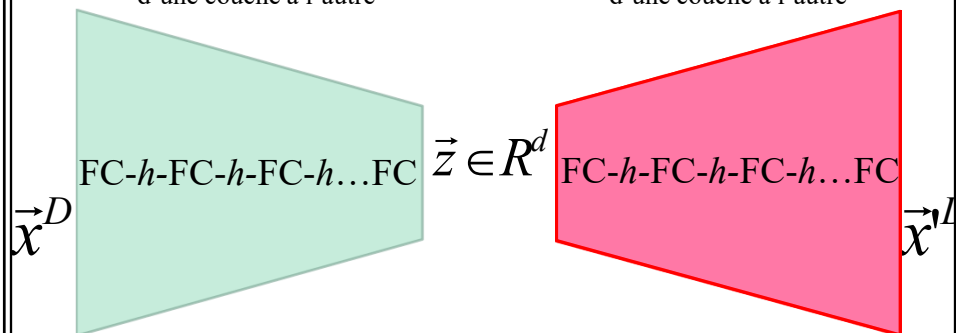


23

En général...

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

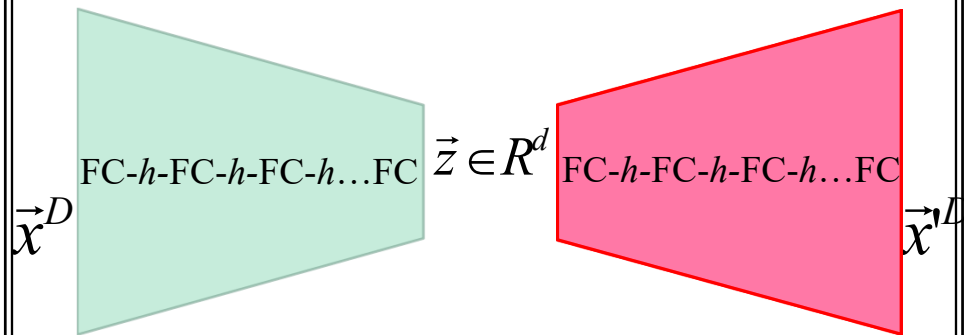


24

En général...

Très souvent...

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



25

### 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

26

## Autoencodeur jouet de MNIST

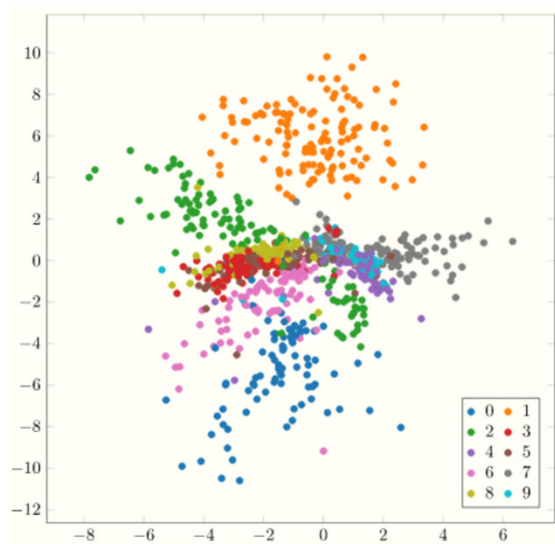
symétrie

```
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
```

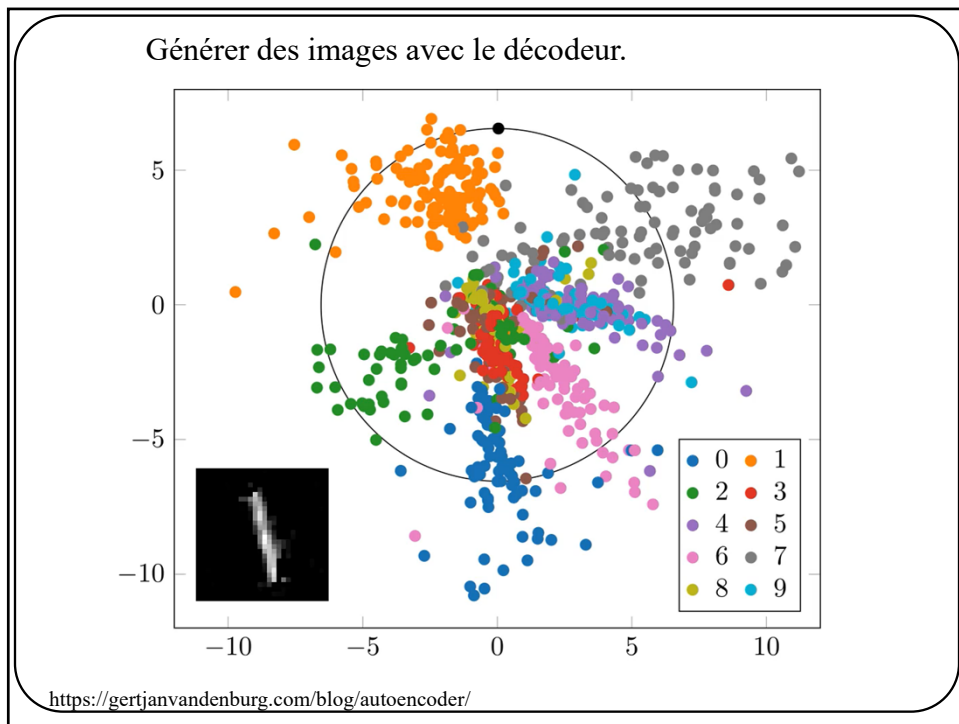
27

## Visualisation de l'espace latent pour 1000 images MNIST chaque image correspond à 1 point 2D

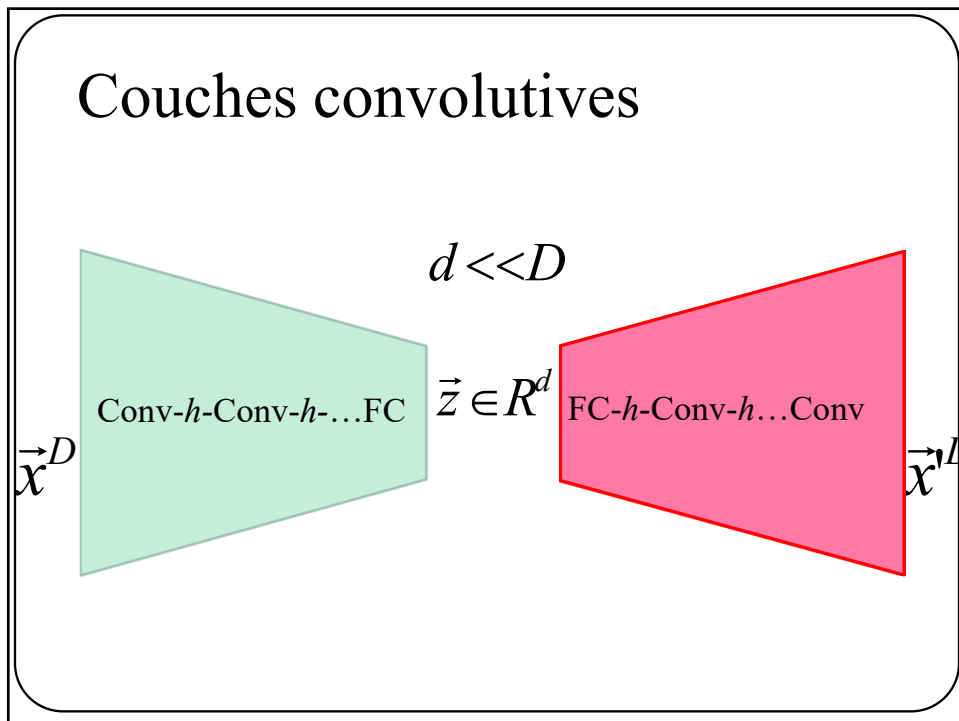


<https://gertjanvandenburger.com/blog/autoencoder/>

28

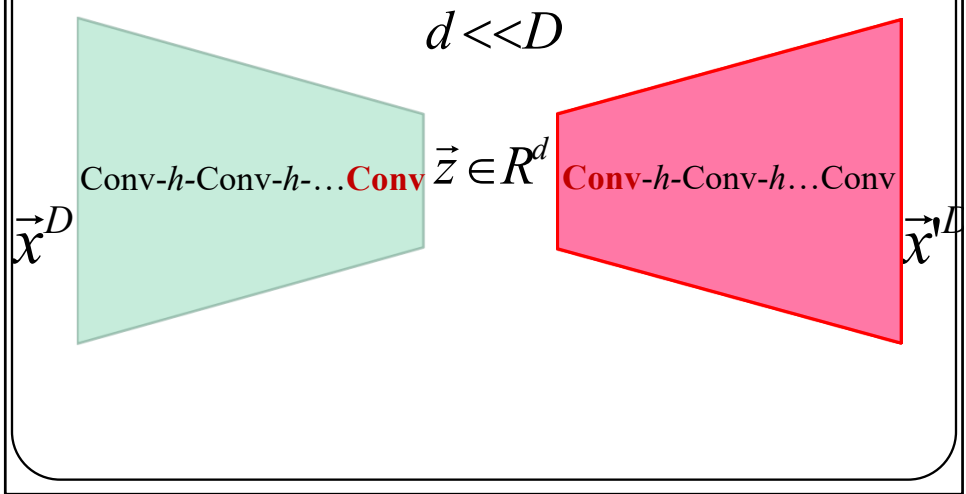


29



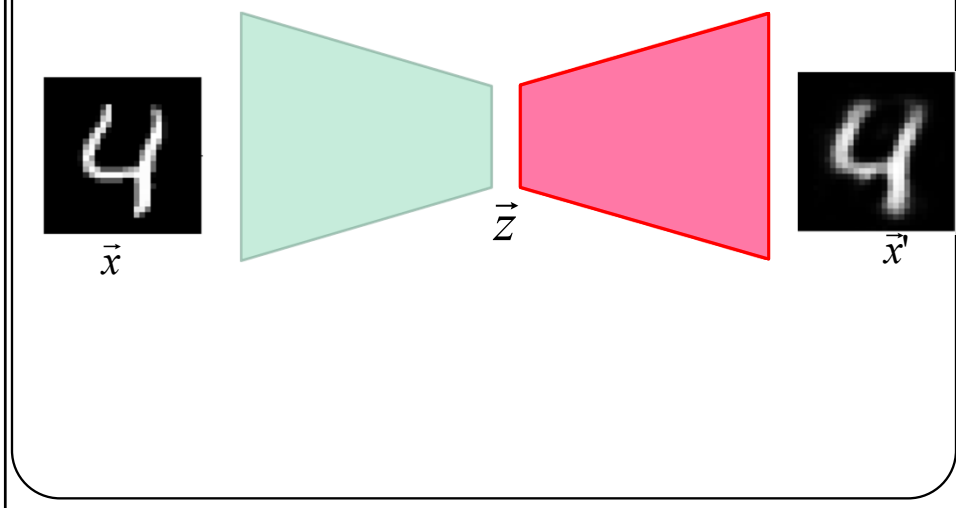
30

# Autoencodeur pleinement convolutif



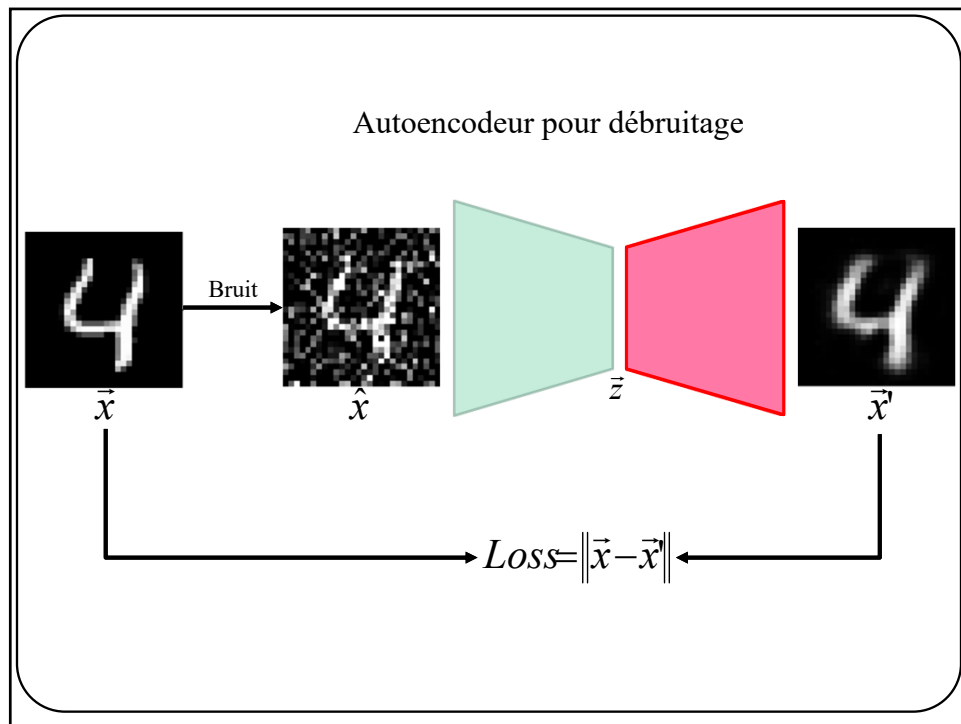
31

## Autoencodeur de base

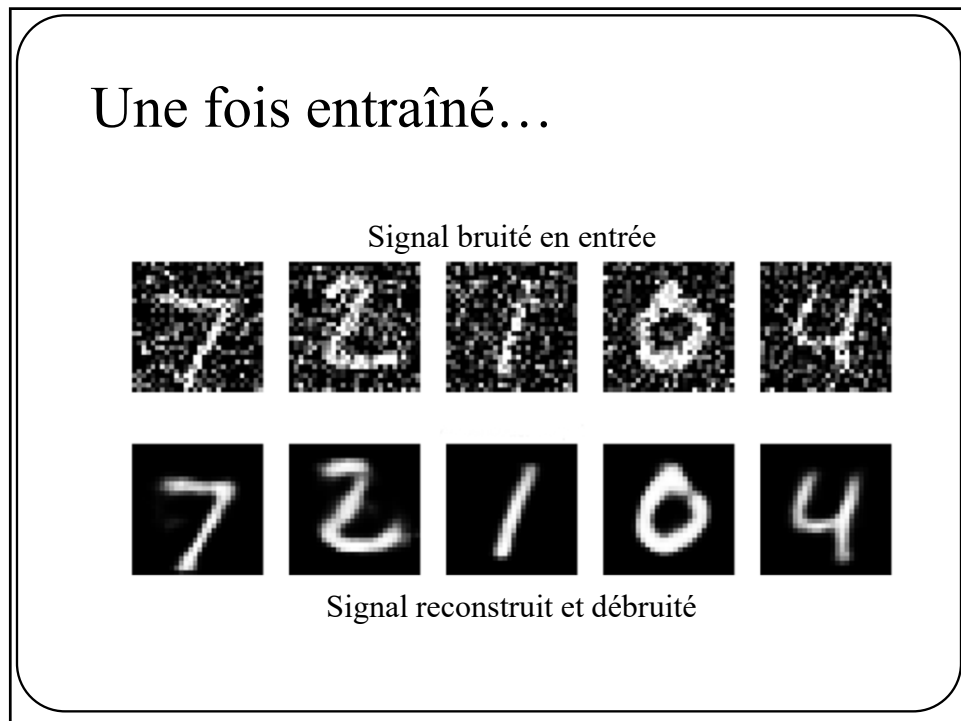


32

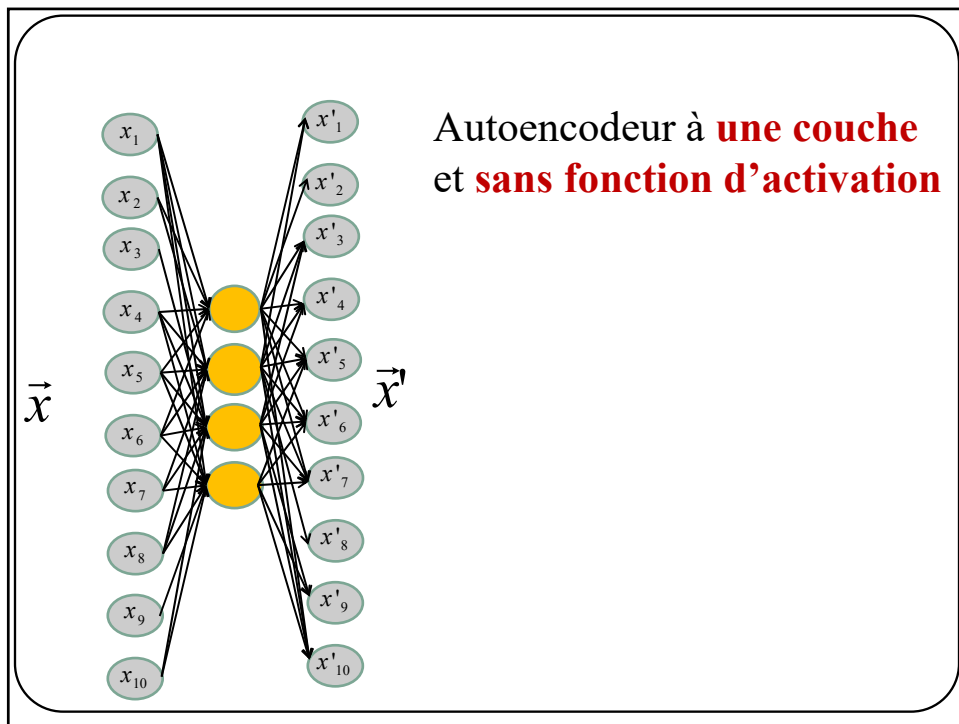




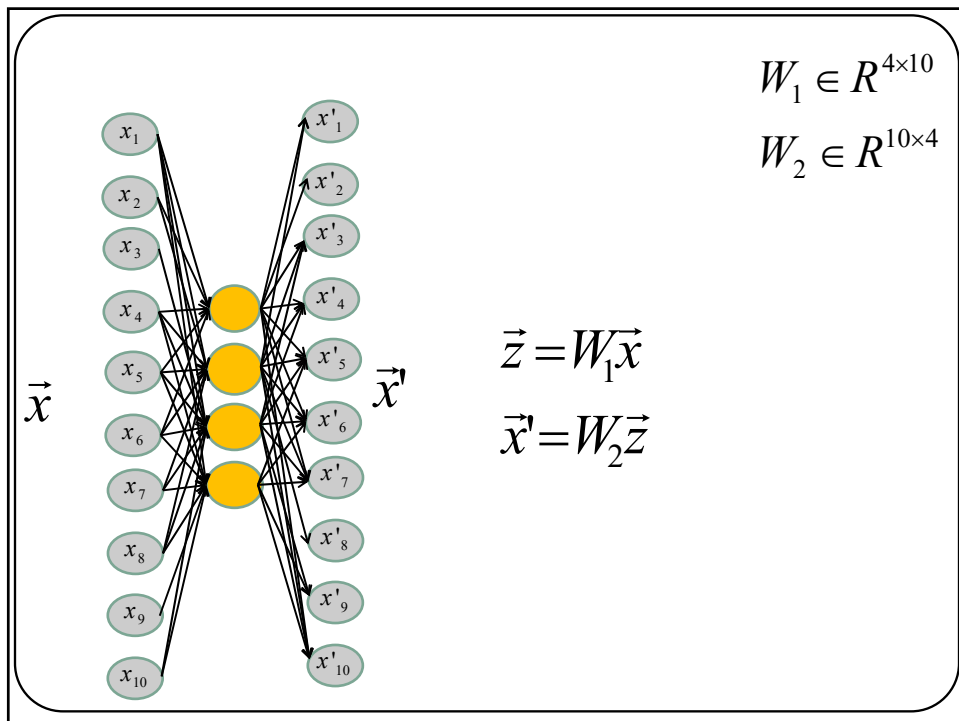
33



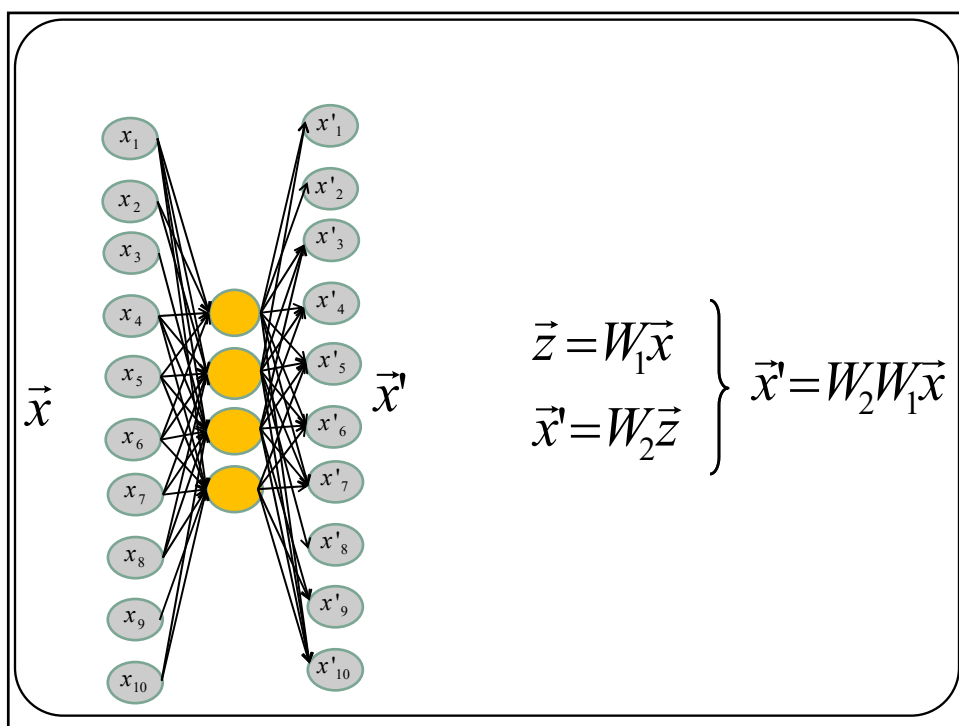
34



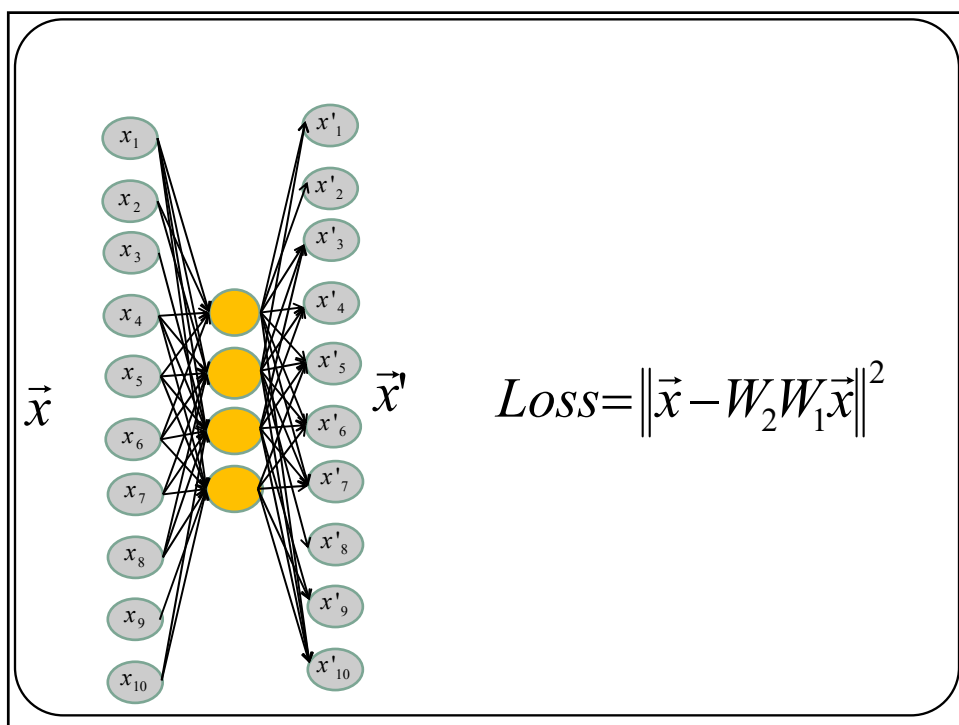
35



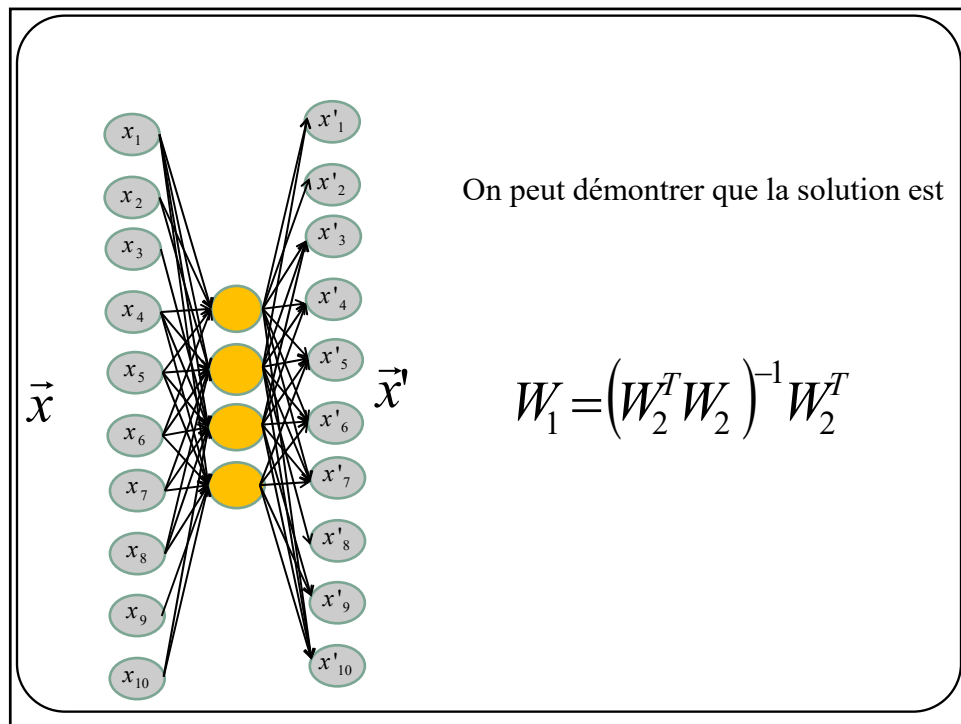
36



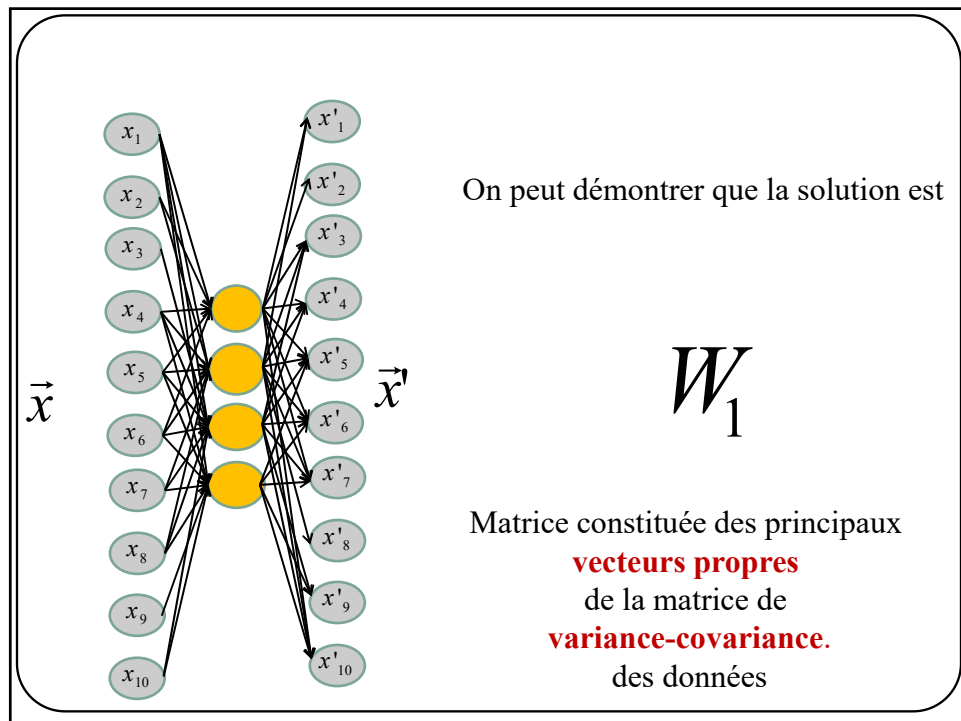
37



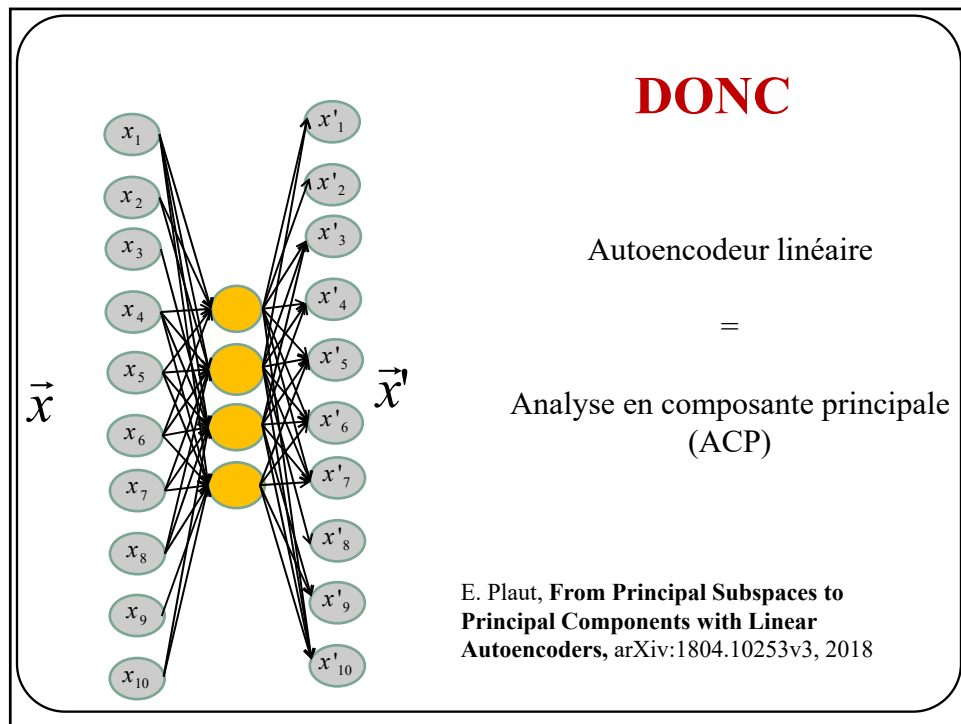
38



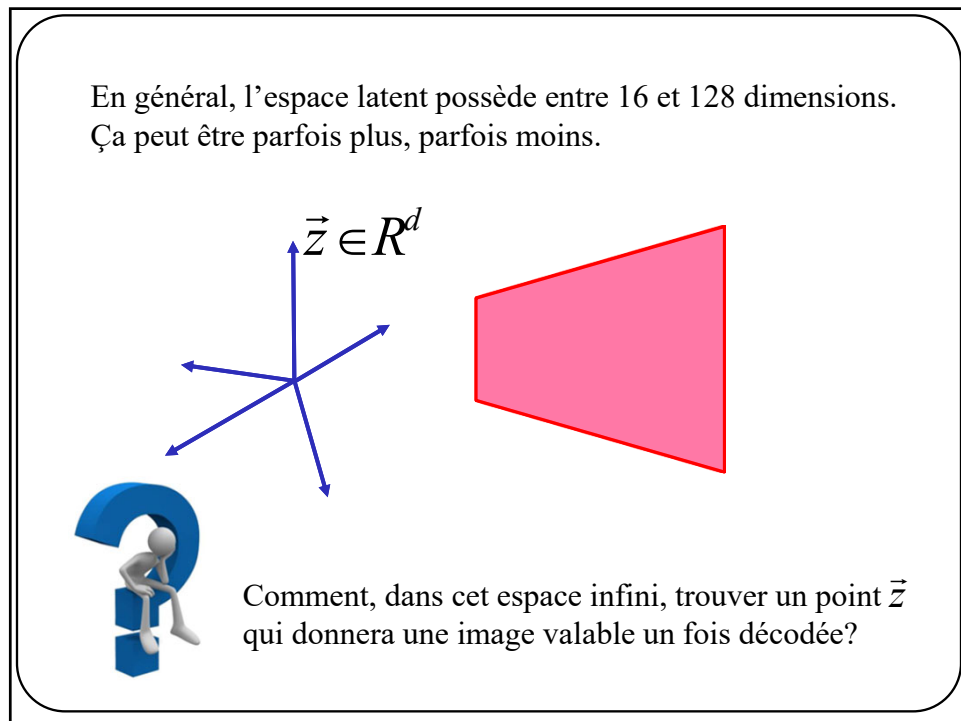
39



40

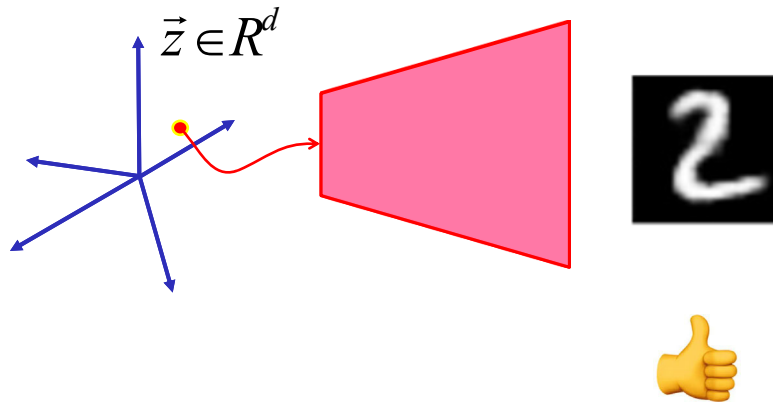


41



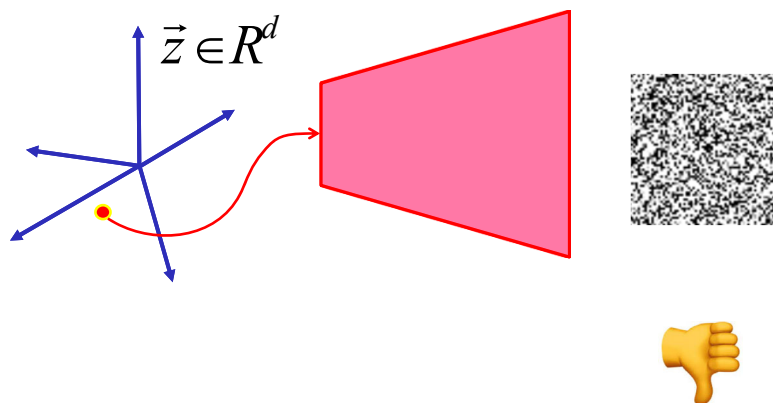
42

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



43

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



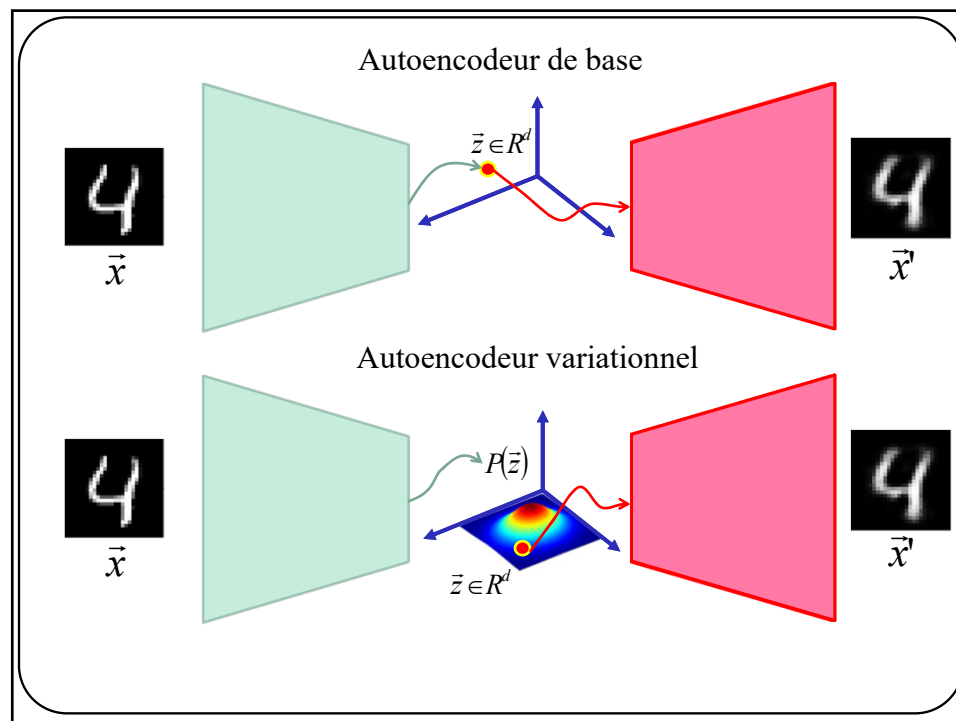
44

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

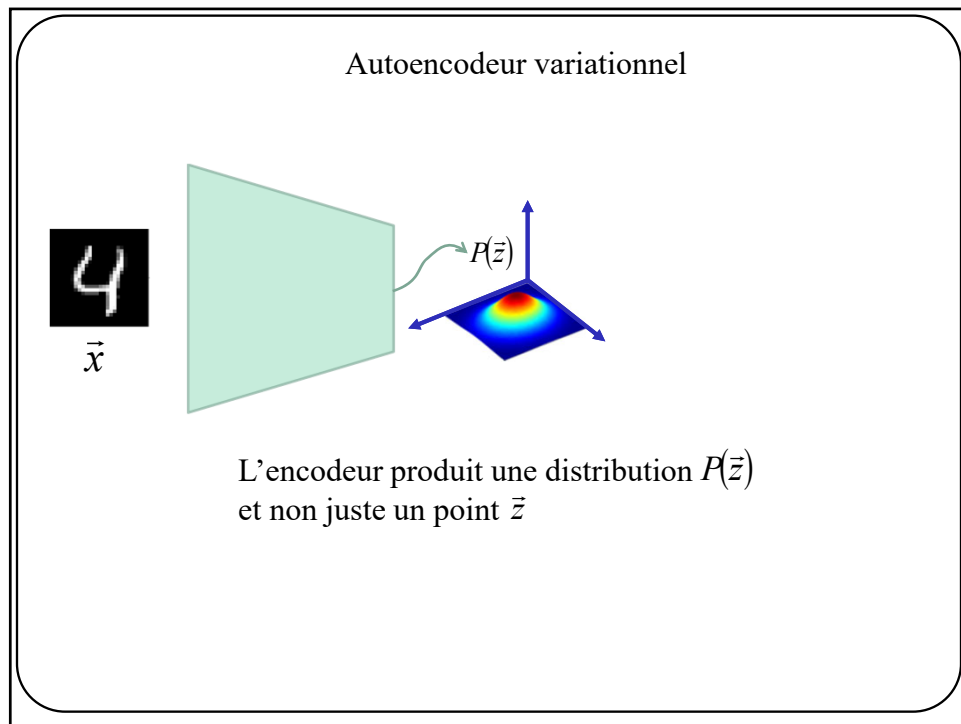


Apprendre à reproduire une **distribution**  $p(\vec{z})$   
**connue** de sorte qu'un **point échantillonné**  
**et décodé** de cette distribution correspond à  
un signal reconstruit valable

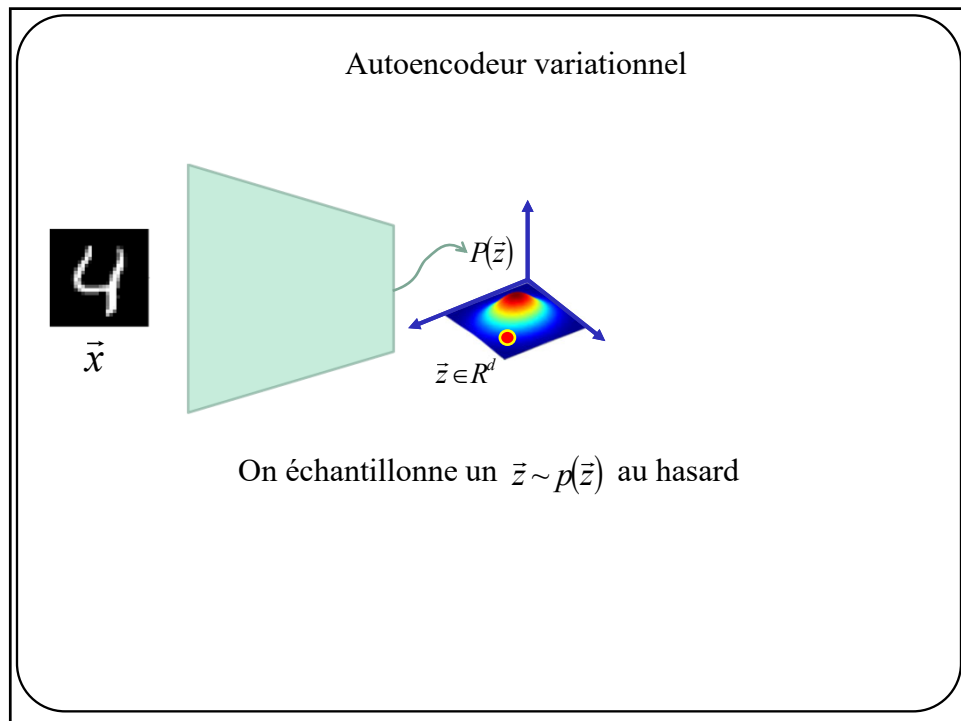
45



46

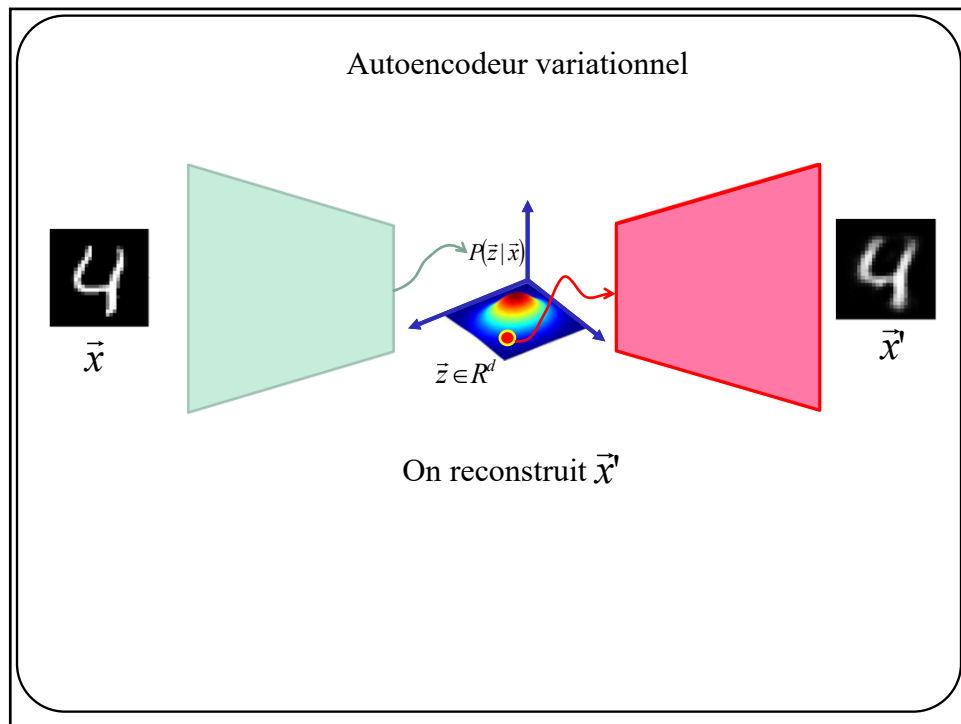


47

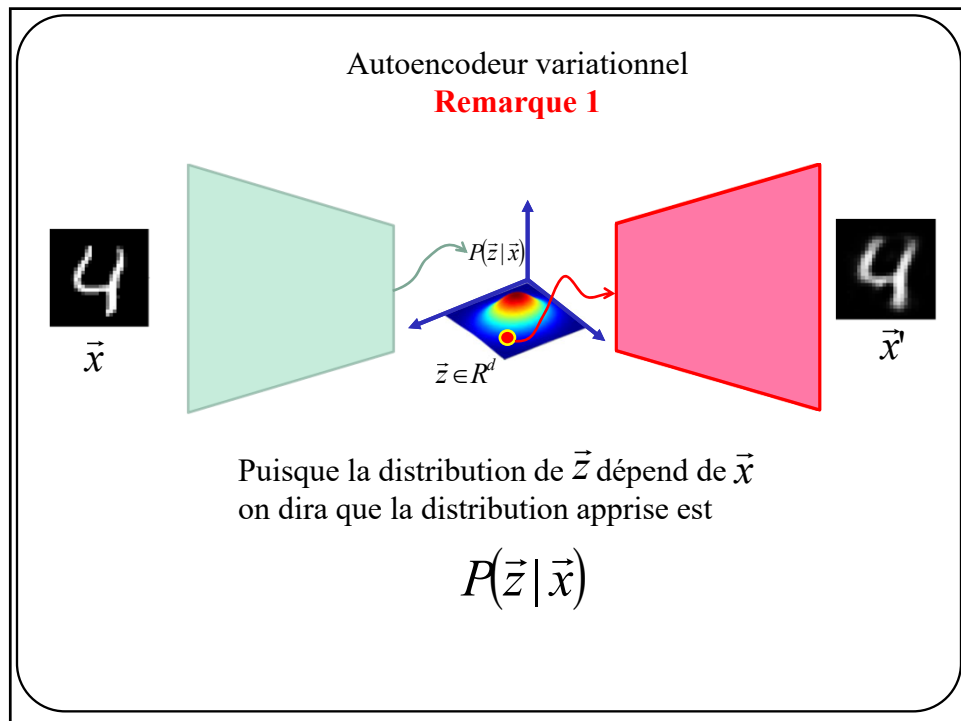


48





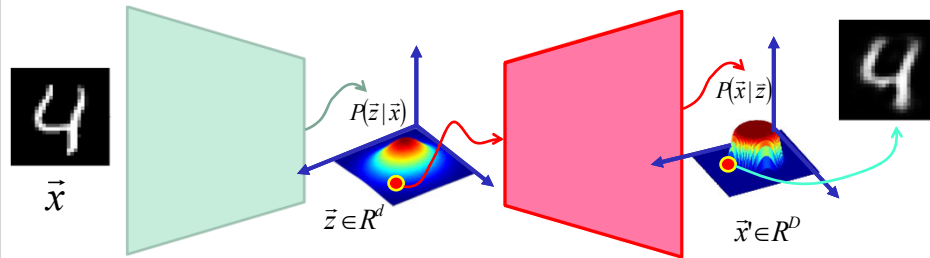
49



50

# Autoencodeur variationnel

## Remarque 2



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

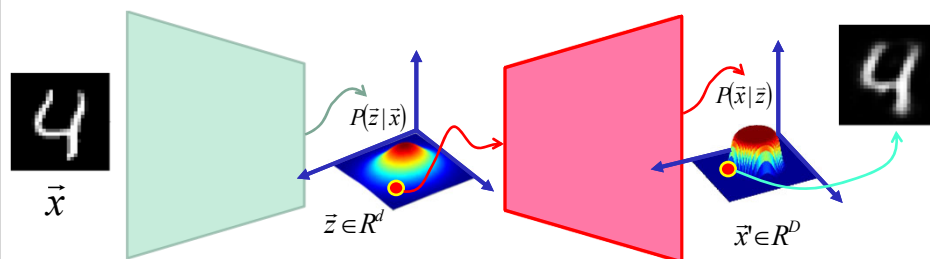
$$P(\vec{x}|\vec{z})$$

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

51

# Autoencodeur variationnel

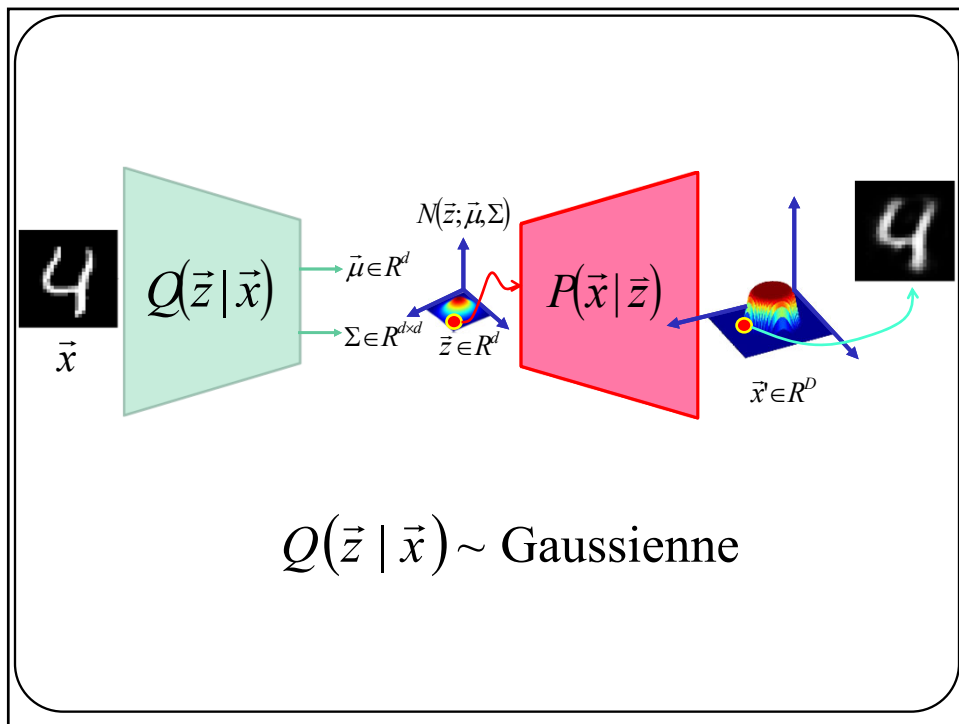
## Remarque 3



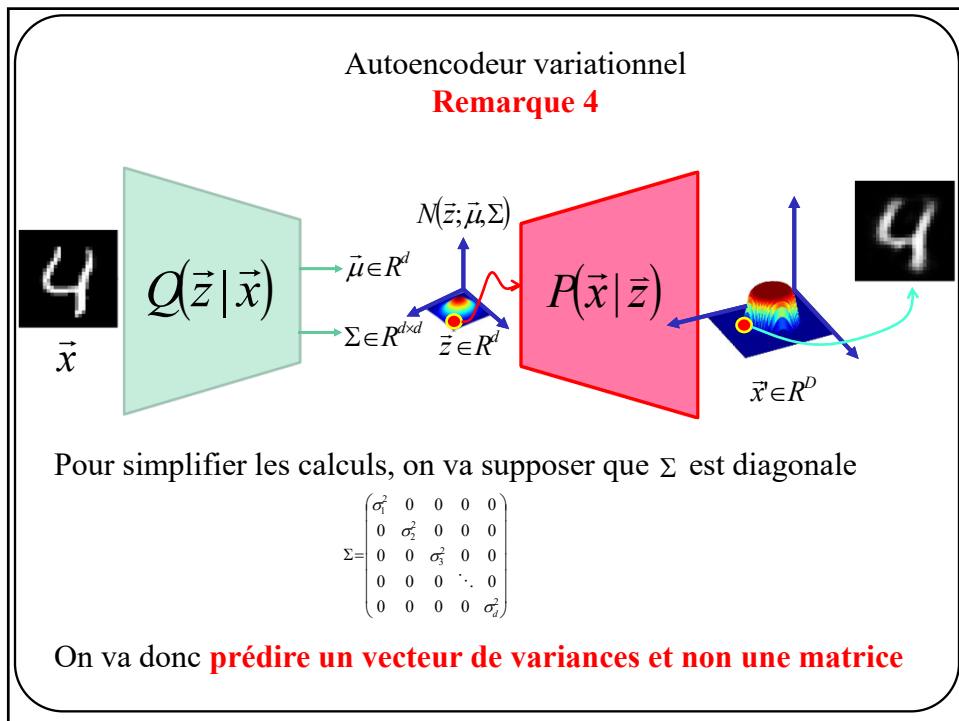
La distribution  $P(\vec{z}|\vec{x})$  peut être très complexe et difficile à échantillonner, on va donc l'approximer par une distribution plus simple... une **gaussienne**

$$Q(\vec{z}|\vec{x}) \approx P(\vec{z}|\vec{x})$$

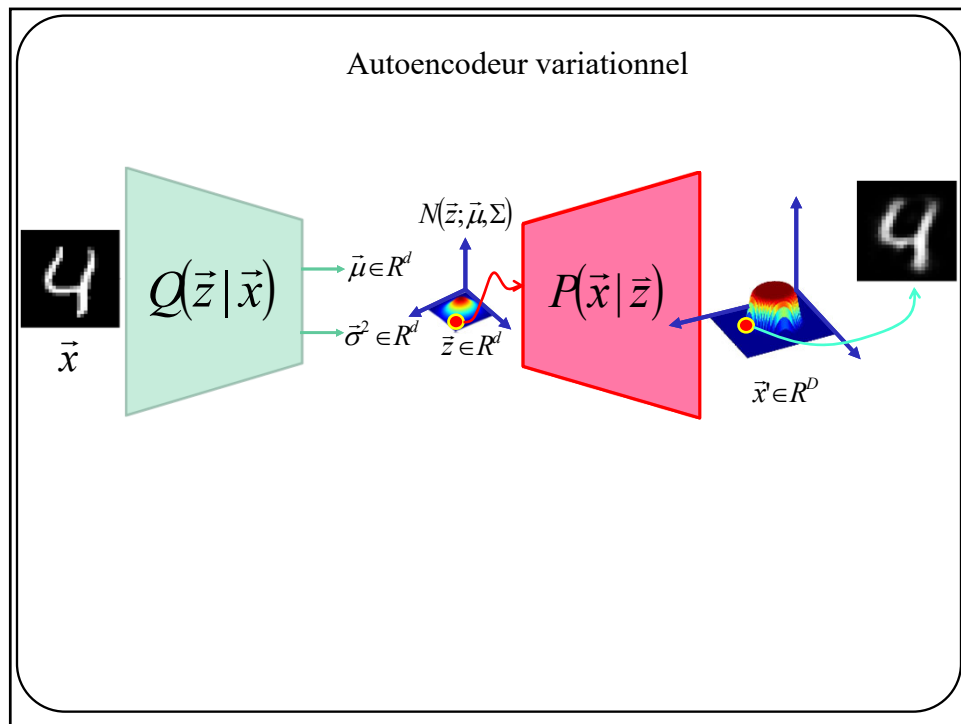
52



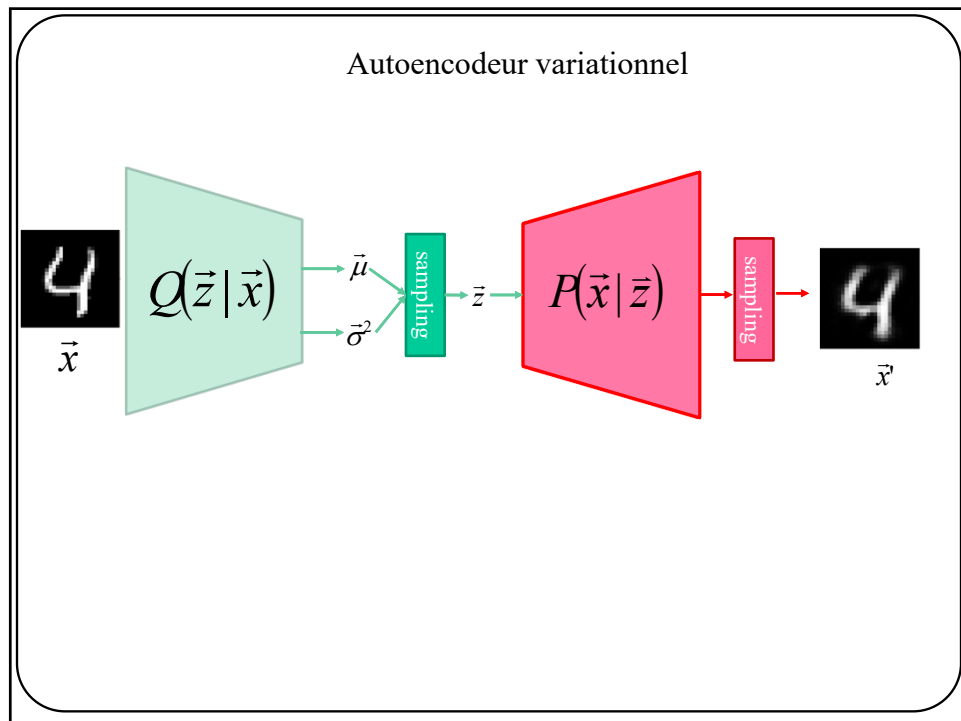
53



54

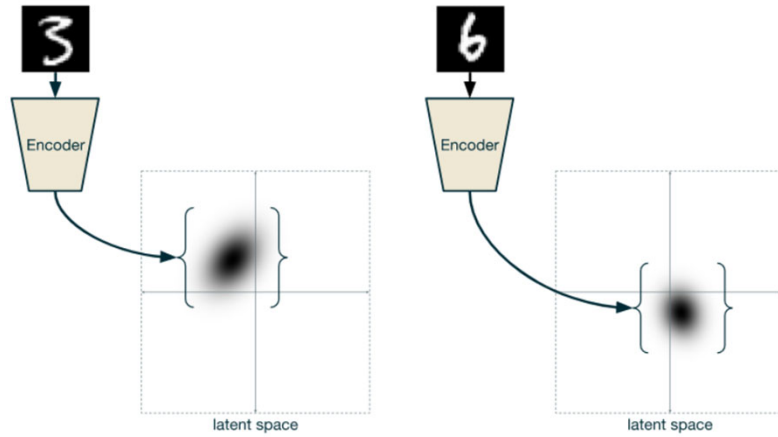


55



56

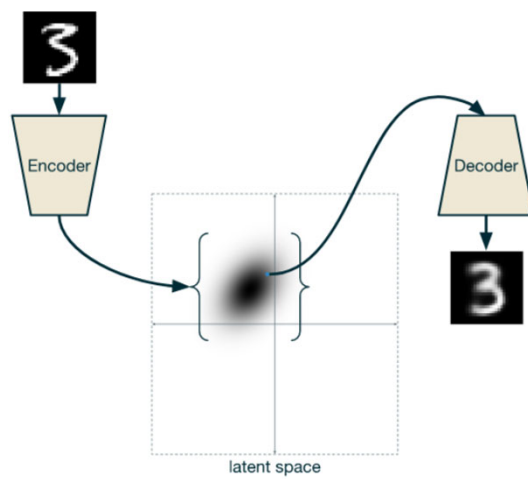
Autre façon de voir les choses...



<https://ijdykeman.github.io/ml/2016/12/21/cvae.html>

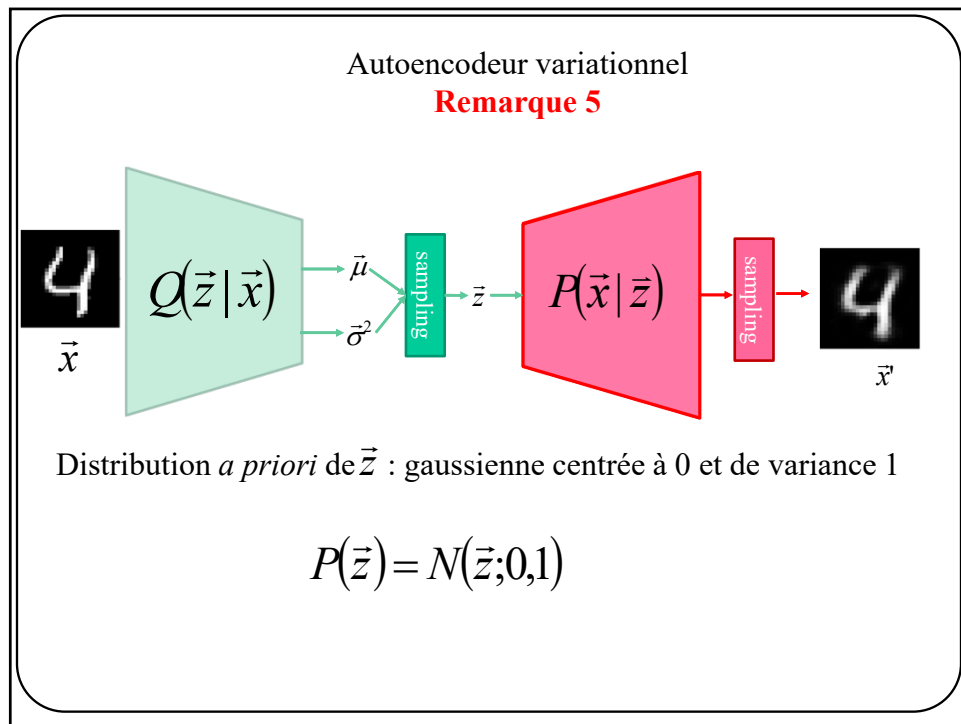
57

Autre façon de voir les choses...

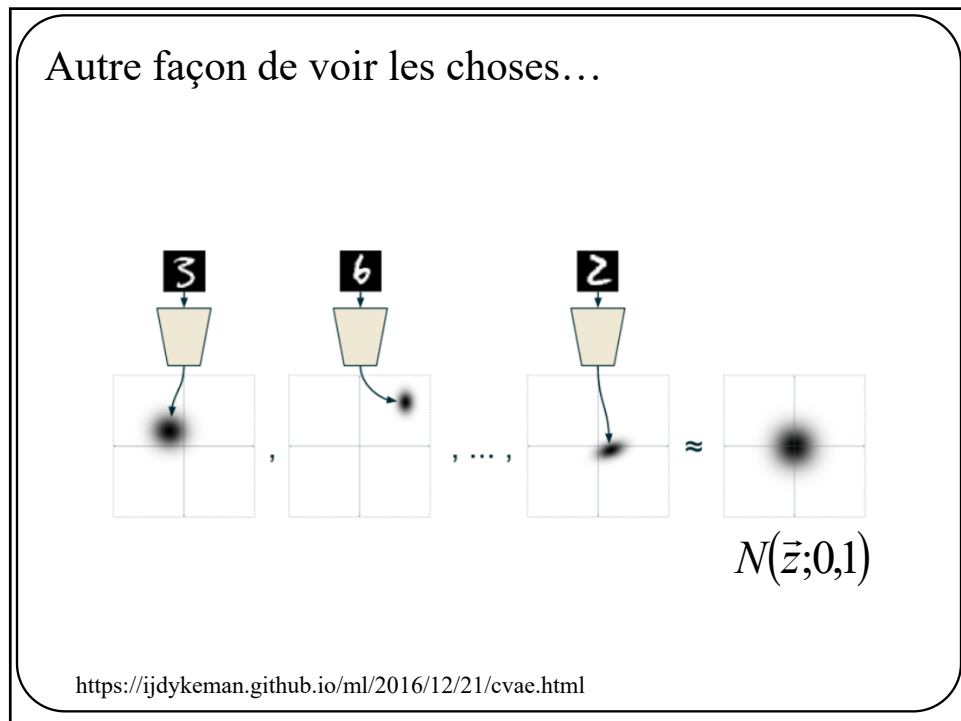


<https://ijdykeman.github.io/ml/2016/12/21/cvae.html>

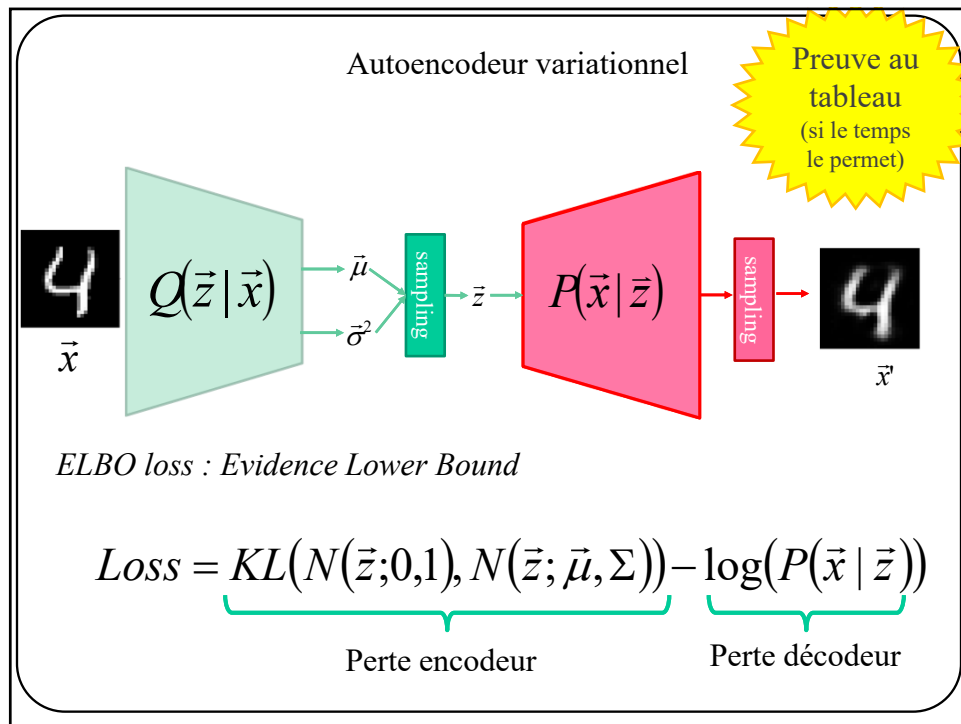
58



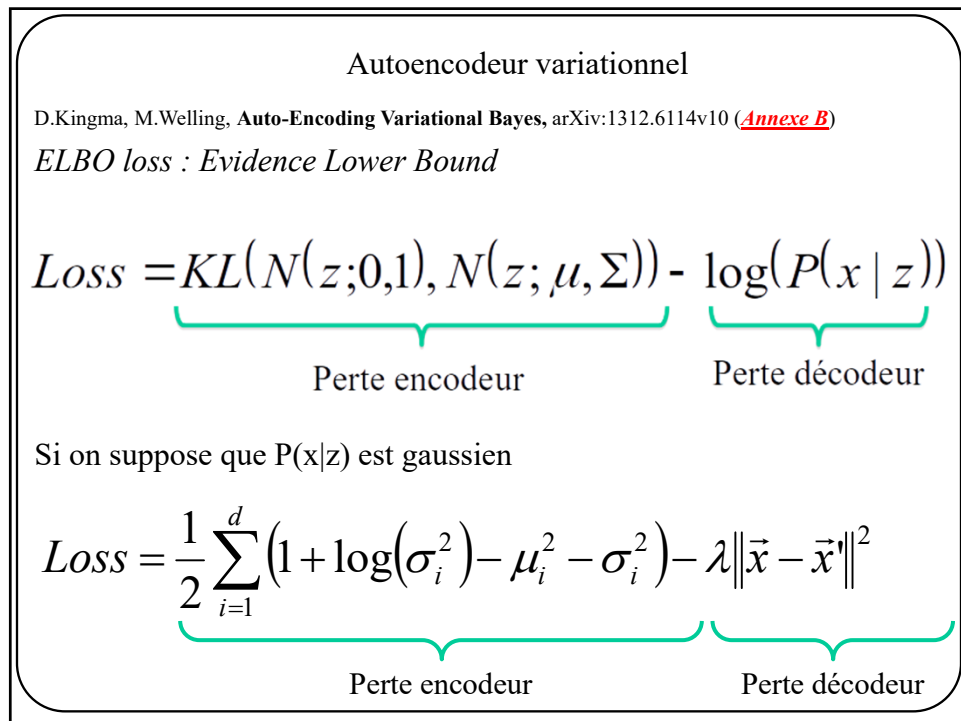
59



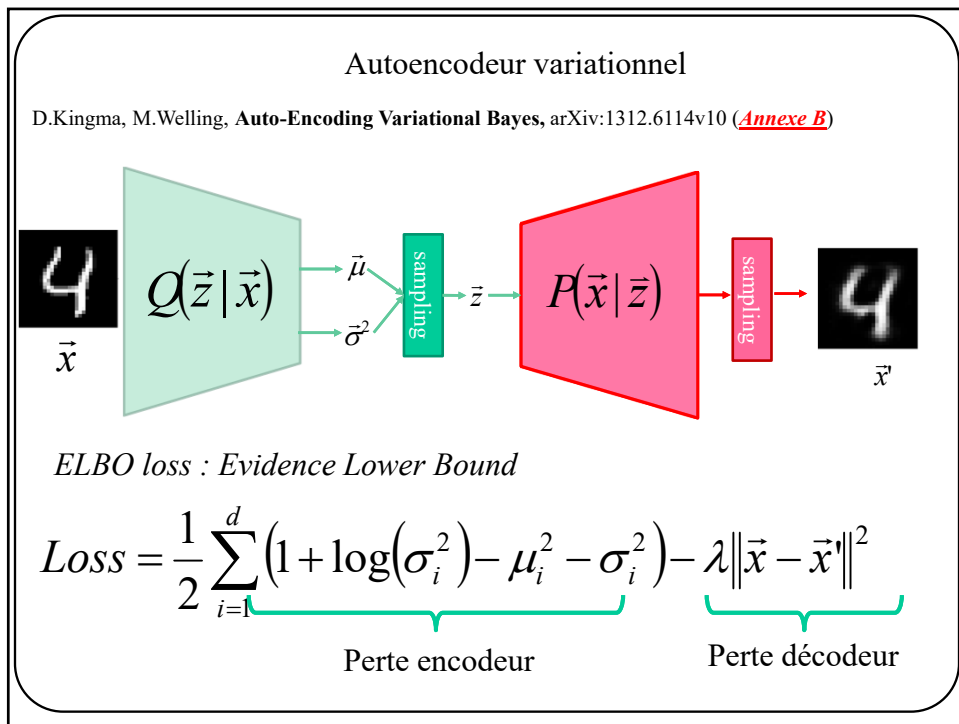
60



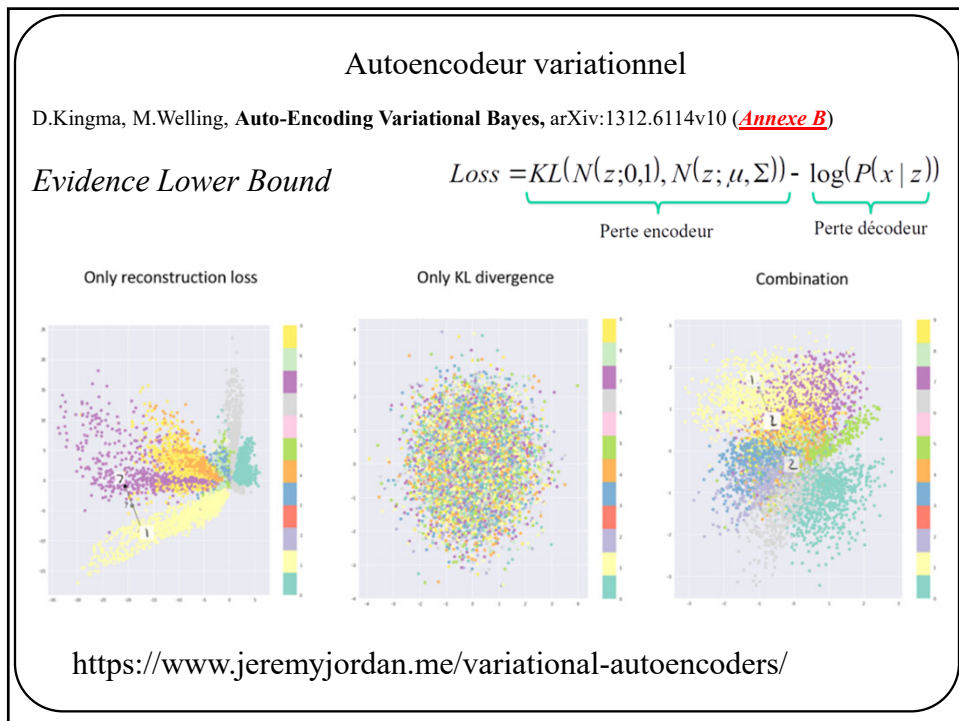
61



62

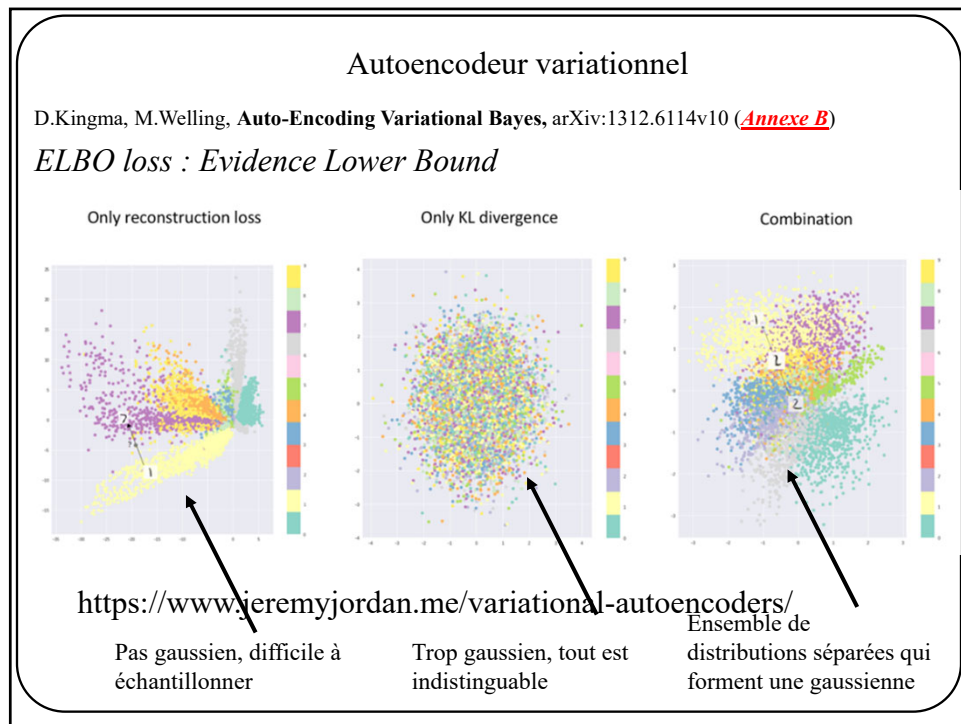


63

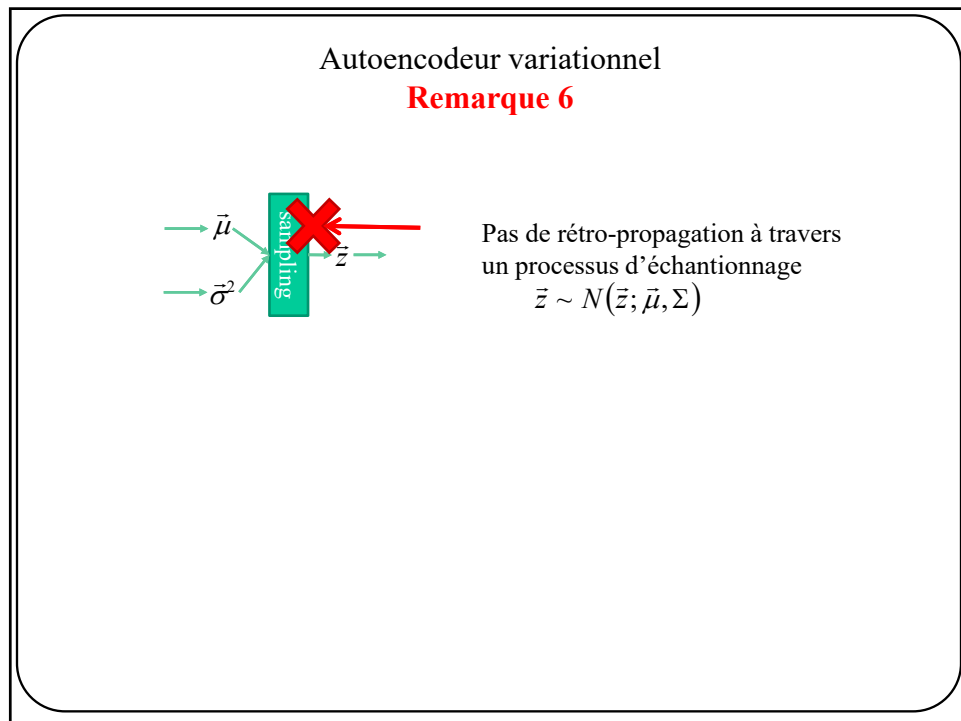


64





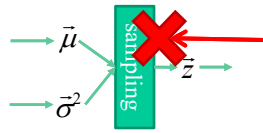
65



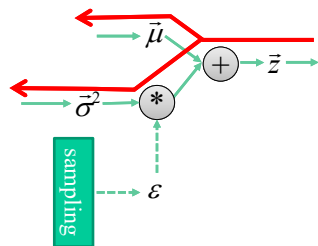
66

## Autoencodeur variationnel

### Remarque 6



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



*Reparameterization trick*

$$\bar{z} = \bar{\mu} + \epsilon \bar{\sigma}^2$$

67

## 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*

68

### Autoencodeur variationnel jouet MNIST : d=32 dim

```
def loss_function(recon_x, x, mu, logvar):
    L2 = nn.MSELoss()(recon_x, x)

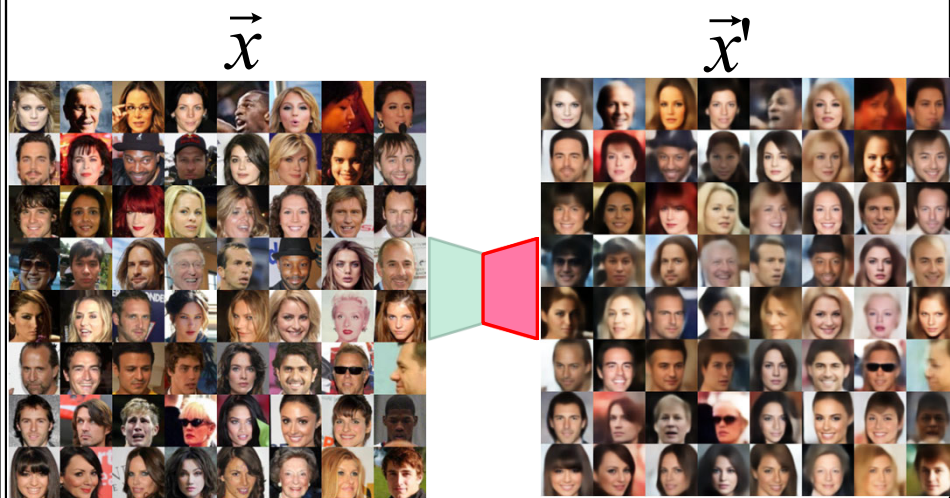
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    return KLD + self.lambda*L2
```

$$Loss = \underbrace{\frac{1}{2} \sum_{i=1}^d (1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2)}_{\text{KLD}} - \underbrace{\lambda \|\vec{x} - \vec{x}'\|^2}_{\text{L2}}$$

69

### Ex.: base de données *CelebA*

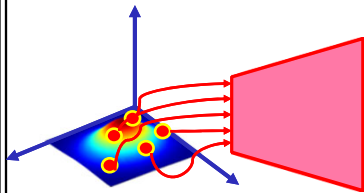


<https://github.com/vzwxx/vae-celebA>

70

## Ex.: base de données *CelebA*

Décodage d'échantillons aléatoires  $\bar{z}$



<https://github.com/vzwxx/vae-celebA>

71

## Ex.: base de données *CelebA*

Images floues.  
Pourquoi ?

$x'$



<https://github.com/vzwxx/vae-celebA>

72

## 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

73

# GAN

Generative Adversarial Nets

74

On voudrait générer des images  $\vec{x}$  en échantillonnant  $P(\vec{x})$

=> **TROP DIFFICILE** car  $P(\vec{x})$  trop complexe



75

Comme précédemment, pour simplifier le problème, on pourrait introduire une variable latente  $\vec{z}$  et ainsi modéliser

$$P(\vec{x}, \vec{z}) = P(\vec{x} | \vec{z}) P(\vec{z})$$

Modèle génératif

Distribution *a priori*

76

Comme pour les VAE, on utilisera une **distribution *a priori*** facile à échantillonner : une **gaussienne**!

$$P(\vec{z}) = N(\vec{z}; 0, 1)$$

77

Comment estimer  $P(\vec{x} | \vec{z})$  ?

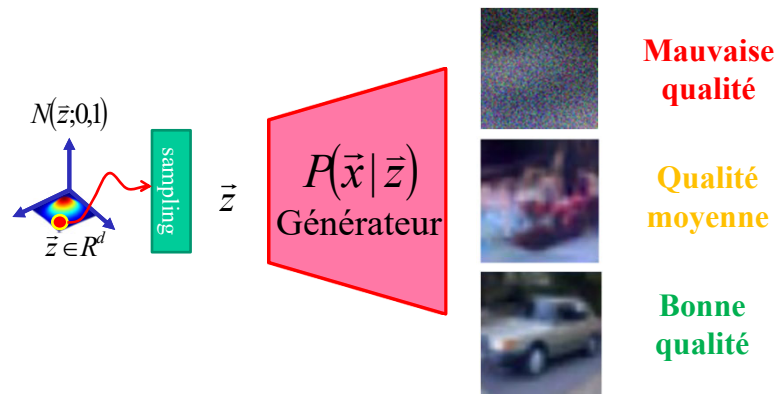
À l'aide d'un réseau de neurones car ce sont **d'excellentes machines pour estimer des probabilités conditionnelles**



78

## VAE

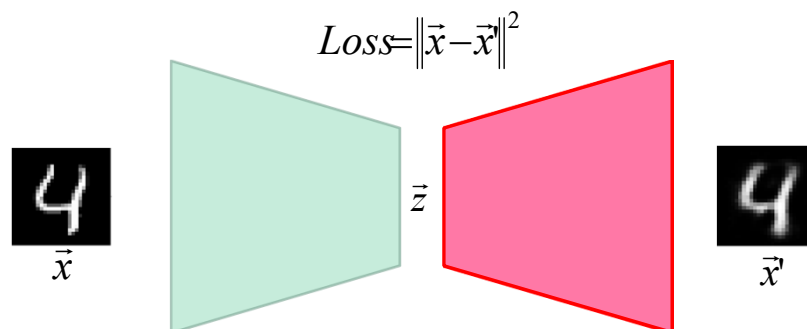
Dépendamment des performances du décodeur, les images générées **seront de qualité très variable.**



79

Pour entraîner un décodeur, il faut une **fonction de perte (loss)** qui **mesure la qualité (degré de réalisme) des images produites**

Pour un autoencodeur (variationnel ou non) c'est facile!  
car on a un encodeur et une image de référence

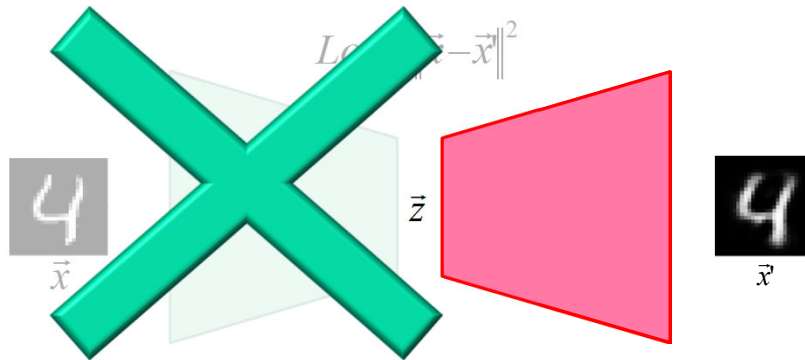


80

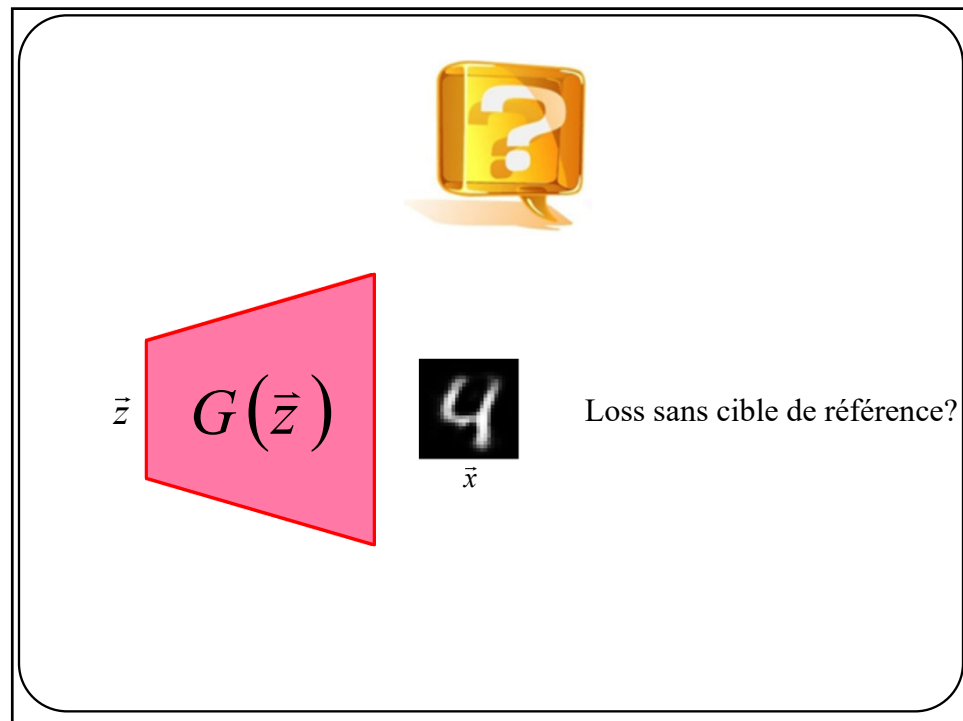


Pour entraîner un décodeur, il faut une **fonction de perte** (*loss*) qui **mesure la qualité (degré de réalisme) des images produites**

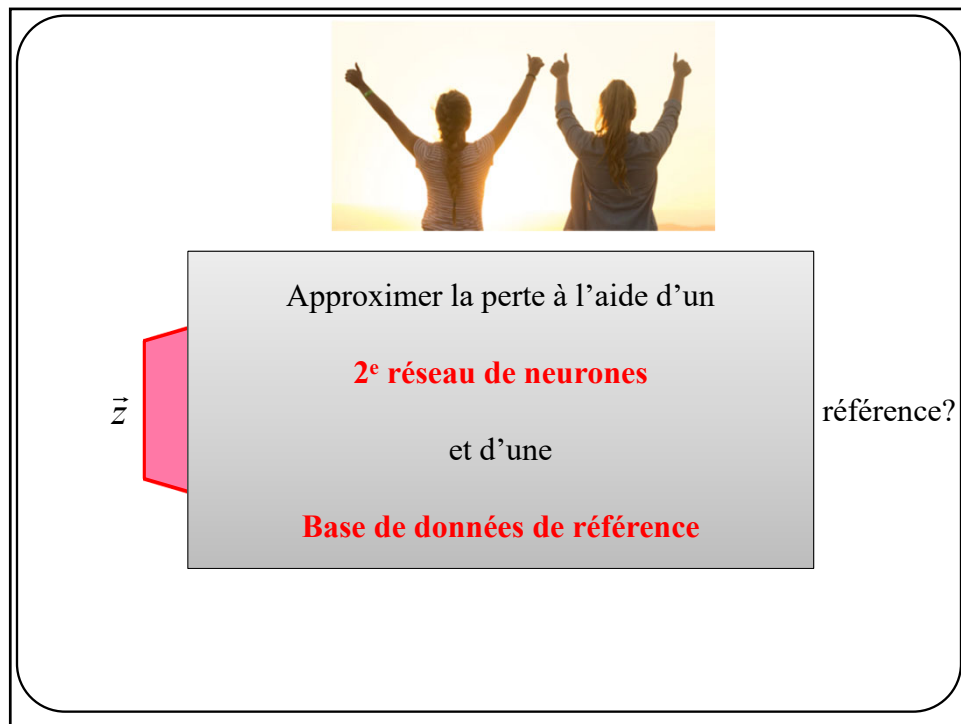
Comment faire pour un réseau **sans encodeur**?



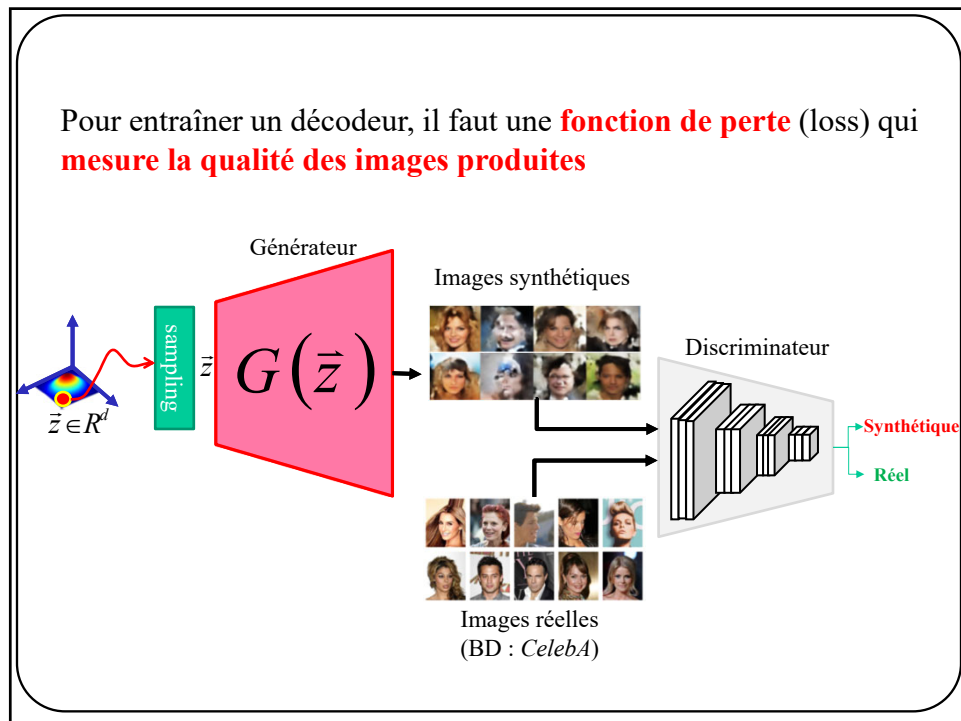
81



82

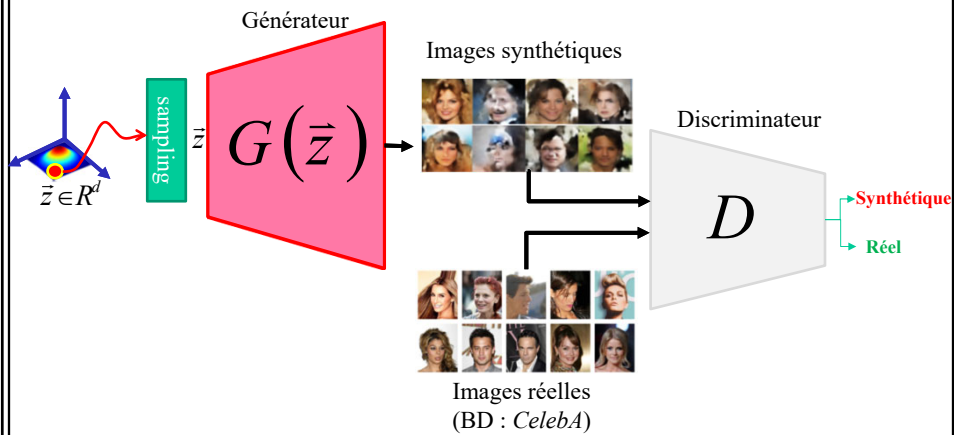


83



84

Pour entraîner un décodeur, il faut une **fonction de perte** (loss) qui **mesure la qualité des images produites**



85

Données étiquetées



$t = 0$  ('synthétique')

Produites par le générateur



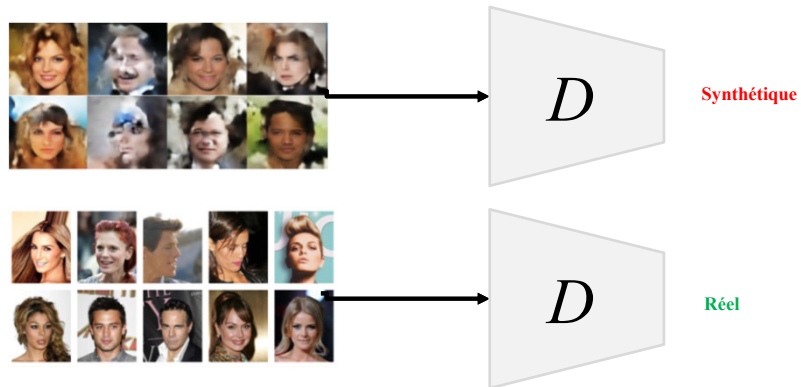
$t = 1$  ('réel')

Issues d'une vraie BD

86

Deux réseaux aux **objectifs différents** :

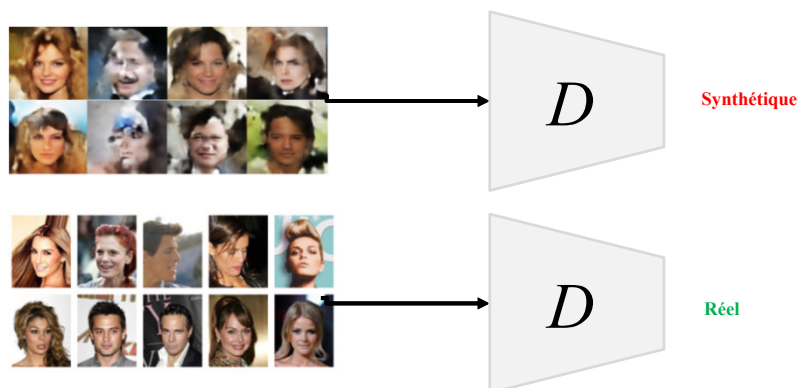
**Discriminateur** : différencie les images synthétiques des images réelles



87

**Discriminateur** : classifieur binaire (régression logistique)

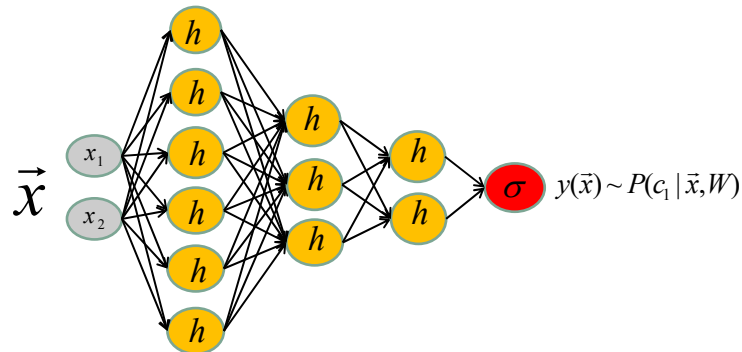
=> Perte l'entropie croisée



88

**Rappel, entropie croisée** pour une régression logistique binaire:

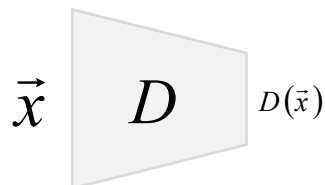
$$L_D = \frac{1}{N} \sum_i -t_i \ln(y(\vec{x}_i)) - (1 - t_i) \ln(1 - y(\vec{x}_i))$$



89

Le réseau discriminateur est représenté par la **lettre D**

$$L_D = \frac{1}{N} \sum_i -t_i \ln(D(\vec{x}_i)) - (1 - t_i) \ln(1 - D(\vec{x}_i))$$

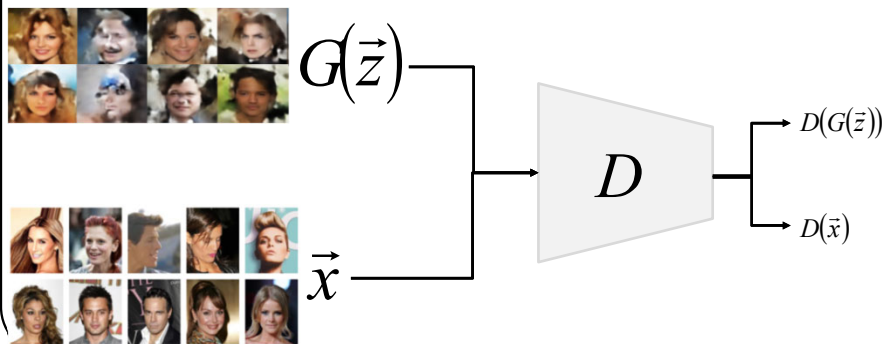


90

Puisque les images **synthétiques** ont été générées par le **générateur**

$$L_D = \frac{1}{N} \sum_i -t_i \ln(D(\tilde{x}_i)) - (1 - t_i) \ln(1 - D(G(\tilde{z}_i)))$$

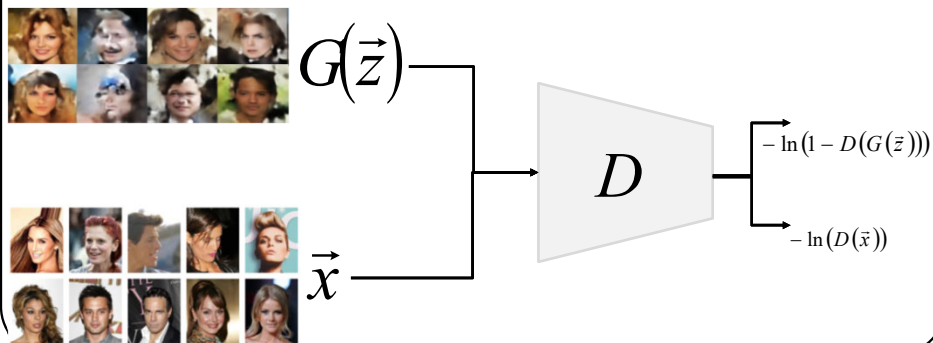
↑



91

Sans perte de généralité, séparer la loss des images réelles et synthétiques

$$L_D = \underbrace{-\frac{1}{N_{reel}} \sum_i \ln(D(\tilde{x}_i))}_{\text{Perte images réelles}} - \underbrace{\frac{1}{N_{syn}} \sum_j \ln(1 - D(G(\tilde{z}_j)))}_{\text{Perte images synthétiques}}$$



92

**Rappel: Espérance mathématique et approximation Monte Carlo**

$$IE[x] = \int xp(x)dx$$

$$IE[f(x)] = \int f(x)p(x)dx$$

93

**Rappel: Espérance mathématique et approximation Monte Carlo**

$$\begin{array}{l} IE[x] = \int xp(x)dx \\ \qquad \qquad \qquad \approx \frac{1}{N} \sum_{i=1}^N x_i \quad \text{où } x_i \sim p(x) \\ \text{approximation} \left\{ \begin{array}{l} IE[f(x)] = \int f(x)p(x)dx \\ \qquad \qquad \qquad \approx \frac{1}{N} \sum_{i=1}^N f(x_i) \quad \text{où } x_i \sim p(x) \end{array} \right. \\ \text{Monte Carlo} \end{array}$$

94

### Rappel: Espérance mathématique et estimateur Monte Carlo

$$L_D = - \underbrace{\frac{1}{N_{reel}} \sum_i \ln(D(\vec{x}_i))}_{\text{Perte images réelles}} - \underbrace{\frac{1}{N_{syn}} \sum_j \ln(1 - D(G(\vec{z}_j)))}_{\text{Perte images synthétiques}}$$

$$L_D = -IE_{\vec{x} \sim P_{reel}} [\ln(D(\vec{x}))] - IE_{\vec{z} \sim P_z} [\ln(1 - D(G(\vec{z})))]$$

(Loss de GAN dans la littérature)

95

### Objectif du discriminateur

#### Paramètres du discriminateur

$$\hat{W}_D = \arg \min_{W_D} - IE_{\vec{x} \sim P_{reel}} [\ln(D(\vec{x}))] - IE_{\vec{z} \sim P_z} [\ln(1 - D(G(\vec{z})))]$$

#### Ou encore, de façon équivalente (mult par -1)

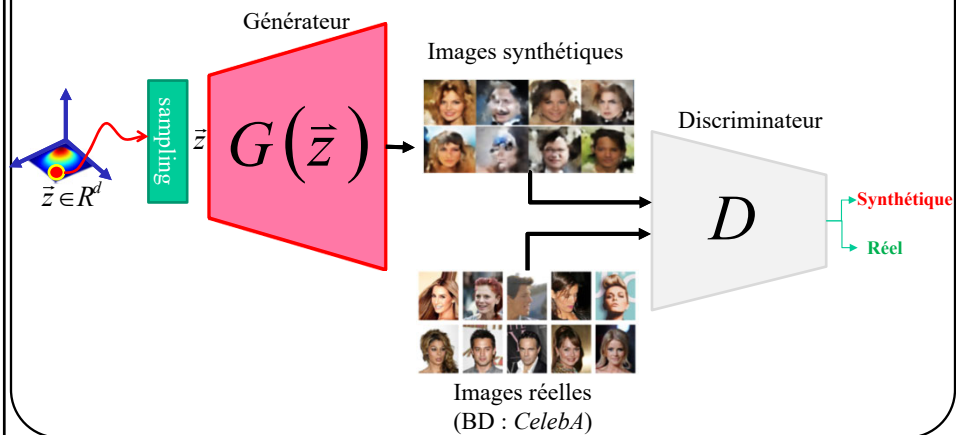
$$W_D = \arg \max_{W_D} IE_{\vec{x} \sim P_{reel}} [\ln(D(\vec{x}))] + IE_{\vec{z} \sim P_z} [\ln(1 - D(G(\vec{z})))]$$

96



### Objectif du générateur

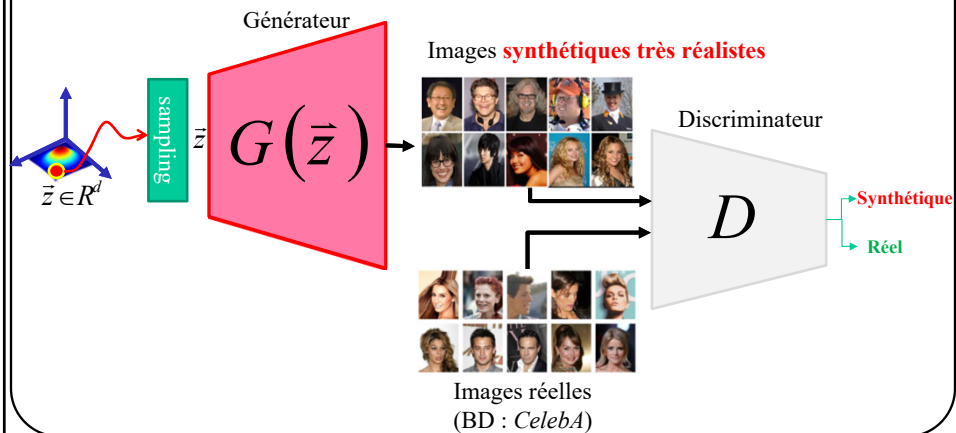
Produire des images aussi réalistes que celle de la BD de référence



97

### Objectif du générateur

S'il y parvient, le discriminateur ne pourra plus les distinguer des images réelles. La loss du discriminateur sera alors élevée.



98

**Objectif du discriminateur :**

bien distinguer les images réelles des images synthétiques

$$W_D = \arg \max_{W_D} \mathbb{E}_{\tilde{x} \sim P_{\text{reel}}} [\ln(D(\tilde{x}))] + \mathbb{E}_{\tilde{z} \sim P_z} [\ln(1 - D(G(\tilde{z})))]$$

**Objectif du générateur :**

produire des images synthétiques indistinguables des images réelles

$$W_G = \arg \min_{W_G} \mathbb{E}_{\tilde{z} \sim P_z} [\ln(1 - D(G(\tilde{z})))]$$

99

« Two player » mini-max game

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

100

## « Two player » mini-max game

Discriminateur veut  
 $D(x) = 1$  pour les vrais  
 données

Discriminateur veut  
 $D(G(x)) = 0$  pour les  
 données synthétiques

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))].$$

Générateur veut  
 $D(G(x)) = 1$  pour les  
 données synthétiques

Ian Goodfellow et al., "Generative  
 Adversarial Nets", NIPS 2014

101

### NOTE

dans les faits, on ne minimise pas cette loss

$$W_G = \arg \min_{W_G} \mathbb{E}_{\bar{z} \sim p_z} [\ln(1 - D(G(\bar{z})))]$$

on maximise plutôt celle-ci

$$W_G = \arg \max_{W_G} \mathbb{E}_{\bar{z} \sim p_z} [\ln(D(G(\bar{z})))]$$

102

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{\text{data}}(x)$ .
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)})))]$$

**end for**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

**end for**

103

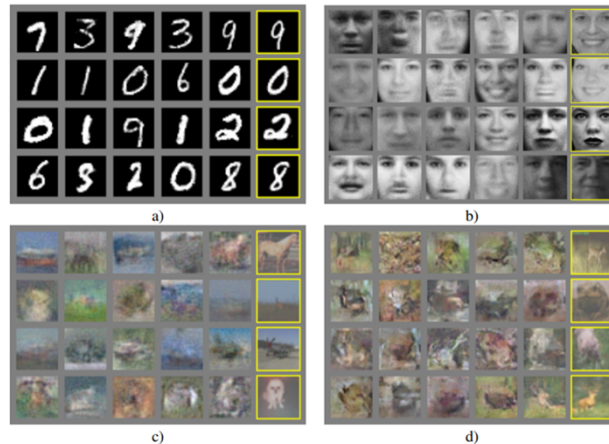
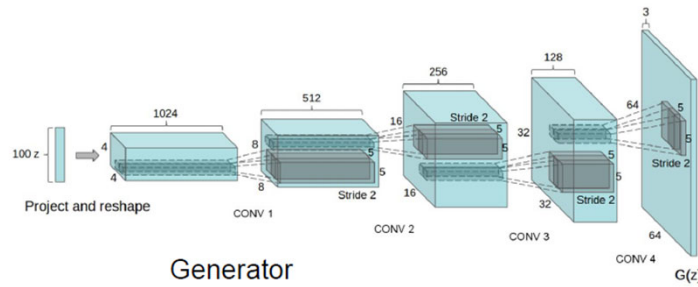


Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

104

## Deep Convolution Generative Adversarial Net (DCGAN)



Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

105

## Deep Convolution Generative Adversarial Net (DCGAN)

### Recommandations discriminateur

- Conv stride > 1 au lieu des couches de pooling
- ReLU partout sauf en sortie : tanh

### Recommandations générateur

- Conv transpose au lieu de upsampling
- LeakyReLU partout

### Autre recommandations

- BatchNorm partout
- Pas de FC, juste des conv

Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

106

## Deep Convolution Generative Adversarial Net (DCGAN)

### Recommandations discriminateur

- C
- E

<https://github.com/soumith/ganhacks>

### Reco

- C
- LeakyReLU partout

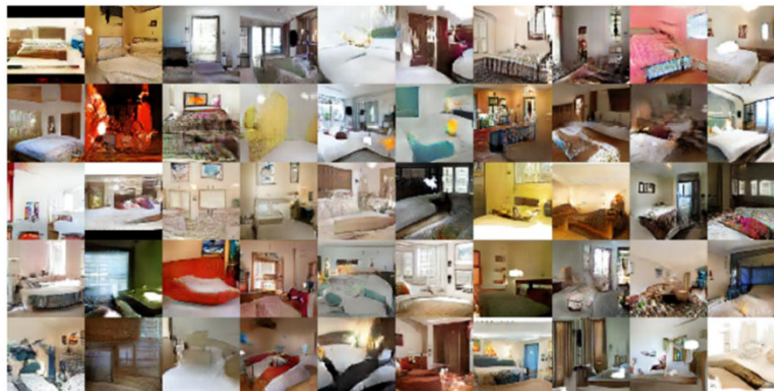
### Autre recommandations

- BatchNorm partout
- Pas de FC, juste des conv

Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

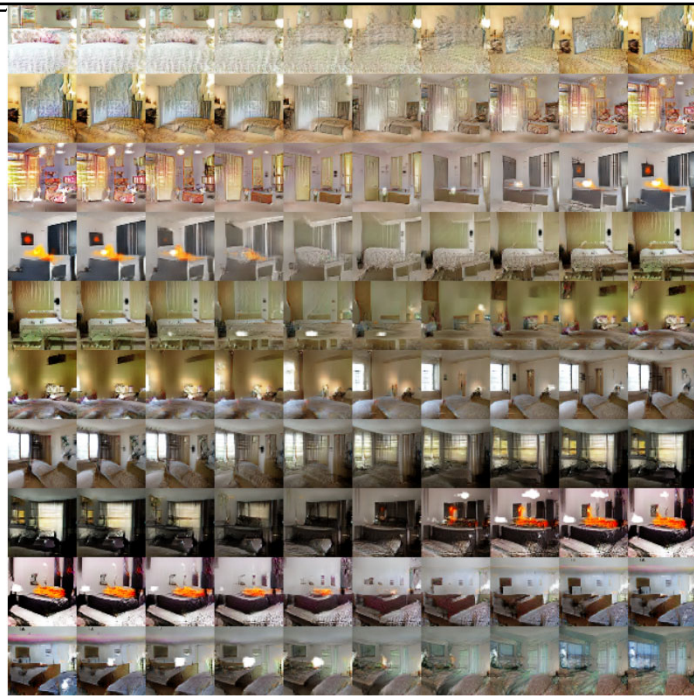
107

## Deep Convolution Generative Adversarial Net (DCGAN)



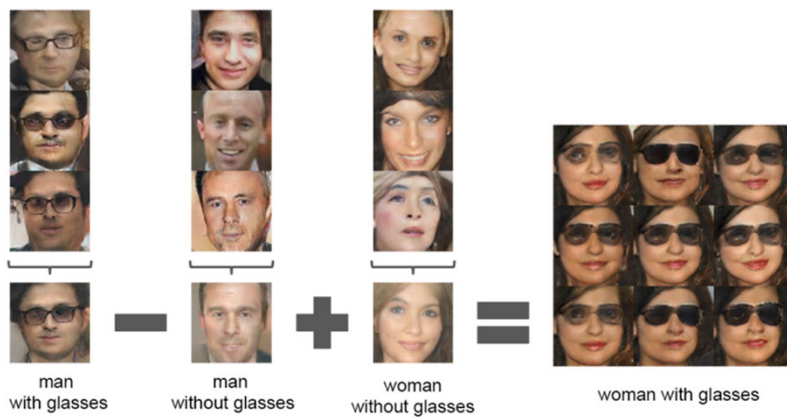
108

Interpolation  
entre 9 vecteurs  
latents aléatoires



109

*“Vector arithmetic for visual concepts”*



110

## Problèmes d'instabilité

- Si discriminateur et générateur et n'apprennent pas ensemble:
  - disparition des gradients
  - effondrement des modes
  - on ne peut générer d'images à haute résolution
- Plusieurs solutions proposées:
  - *Wasserstein GAN* (utilise "earth mover distance")
  - *Least Squares GAN* (utilise distance d'erreur quadratique)
  - *Progressive GAN*
  - ....

111

111

## Problèmes d'instabilité

- Si discriminateur et générateur et n'apprennent pas ensemble:
  - **disparition des gradients**
  - effondrement des modes
  - on ne peut générer d'images à haute résolution

Si le discriminateur apprend trop vite, le générateur sera systématiquement battu, et n'apprendra rien

112

112



## Problèmes d'instabilité

- Si discriminateur et générateur et n'apprennent pas ensemble:
  - disparition des gradients
  - **effondrement des modes**
  - on ne peut générer d'images à haute résolution

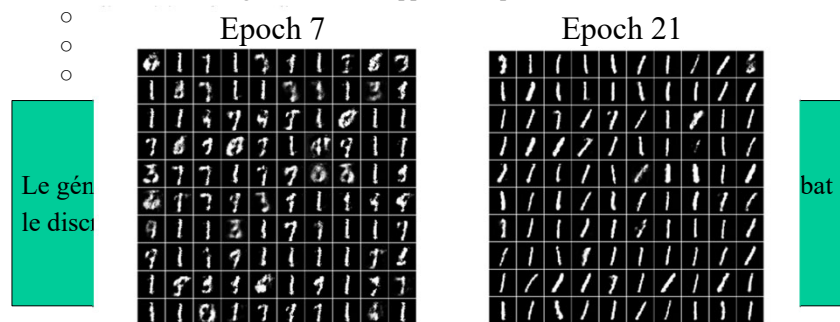
Le générateur peut apprendre à toujours générer la même image et ainsi battre le discriminateur

113

113

## Problèmes d'instabilité

- Si discriminateur et générateur et n'apprennent pas ensemble:



<https://datascience.stackexchange.com/questions/29485/gan-discriminator-converging-to-one-output>

114

114

## Problèmes d'instabilité

- Si discriminateur et générateur et n'apprennent pas ensemble:
  - disparition des gradients
  - effondrement des modes
  - **on ne peut générer d'images à haute résolution**

Solution: les progressive GANs

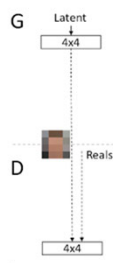
115

115

## progressive GAN

On veut générer des images à **haute résolution**

On commence avec des images de **faible résolution : 4x4 pixels**

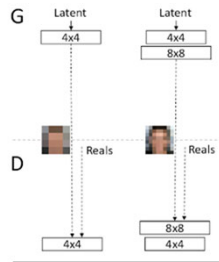


*"Progressive GAN" Karras et al. ICLR'18*

116

# progressive GAN

Et progressivement, on augmente la résolution de l'image



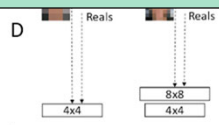
*“Progressive GAN” Karras et al. ICLR’18*

117

# progressive GAN

Et progressivement, on augmente la résolution de l'image

***“Progressive Growing GAN requires that the capacity of both the generator and discriminator model be expanded by adding layers during the training process”***

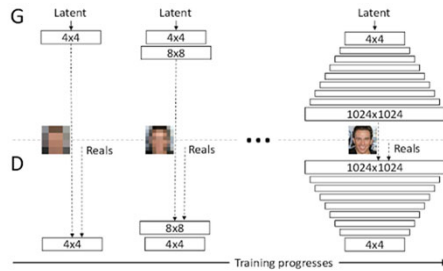


*“Progressive GAN” Karras et al. ICLR’18*

118

# progressive GAN

Et progressivement, chaque couche qu'on ajoute vient bonifier la couche précédente : cela se fait à l'aide d'une **opération « résiduelle »**.

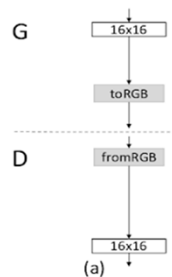


*“Progressive GAN” Karras et al. ICLR’18*

119

## Ajout de couches

Lorsque l'**entraînement** d'une couche de résolution RxR (ici 16x16) est **terminé**...

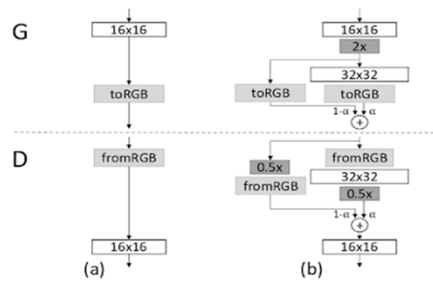


*“Progressive GAN” Karras et al. ICLR’18*

120

## Ajout de couches

... On **ajoute une nouvelle couche** de résolution 2Rx2D (ici 32x32) au générateur ET au discriminateur.



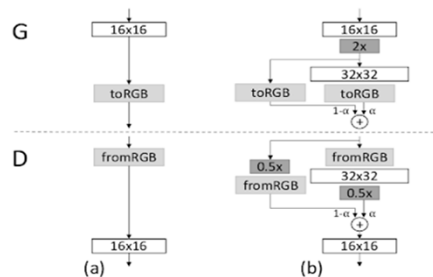
“Progressive GAN” Karras et al. ICLR’18

121

## Ajout de couches

... mais pour éviter un choc, on ajoute une **composante résiduelle** comprenant un facteur  $\alpha$

**Au début de l’entrainement,  $\alpha=0$  et progressivment  $\alpha$  augmente pour atteindre  $\alpha=1$  à la fin**

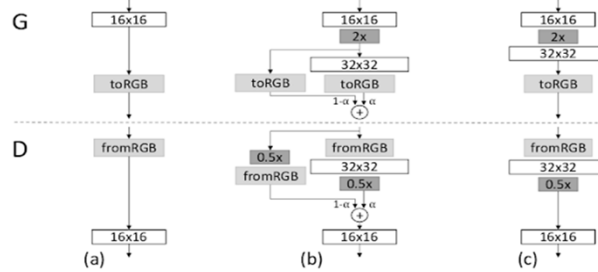


“Progressive GAN” Karras et al. ICLR’18

122

## Ajout de couches

Lorsque l'entraînement est terminé, on enlève la composante résiduelle.



“Progressive GAN” Karras et al. ICLR’18

123

“Progressive GAN” Karras et al. ICLR’18

Generator	Act.	Output shape	Params
Latent vector	—	512 × 1 × 1	—
Conv 4 × 4	LReLU	512 × 4 × 4	4.2M
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Upsample	—	512 × 8 × 8	—
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Upsample	—	512 × 16 × 16	—
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Upsample	—	512 × 32 × 32	—
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Upsample	—	512 × 64 × 64	—
Conv 3 × 3	LReLU	256 × 64 × 64	1.2M
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Upsample	—	256 × 128 × 128	—
Conv 3 × 3	LReLU	128 × 128 × 128	295k
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Upsample	—	128 × 256 × 256	—
Conv 3 × 3	LReLU	64 × 256 × 256	74k
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Upsample	—	64 × 512 × 512	—
Conv 3 × 3	LReLU	32 × 512 × 512	18k
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Upsample	—	32 × 1024 × 1024	—
Conv 3 × 3	LReLU	16 × 1024 × 1024	4.6k
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 1 × 1	linear	3 × 1024 × 1024	51
Total trainable parameters			<b>23.1M</b>

Discriminator	Act.	Output shape	Params
Input image	—	3 × 1024 × 1024	—
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	—	32 × 512 × 512	—
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	—	64 × 256 × 256	—
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Conv 3 × 3	LReLU	128 × 256 × 256	74k
Downsample	—	128 × 128 × 128	—
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Conv 3 × 3	LReLU	256 × 128 × 128	295k
Downsample	—	256 × 64 × 64	—
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Conv 3 × 3	LReLU	512 × 64 × 64	1.2M
Downsample	—	512 × 32 × 32	—
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Downsample	—	512 × 16 × 16	—
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Downsample	—	512 × 8 × 8	—
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Downsample	—	512 × 4 × 4	—
Minibatch stddev	—	513 × 4 × 4	—
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Conv 4 × 4	LReLU	512 × 1 × 1	4.2M
Fully-connected	linear	1 × 1 × 1	513
Total trainable parameters			<b>23.1M</b>

Table 2: Generator and discriminator that we use with CELEBA-HQ to generate 1024×1024 images.

124

“Progressive GAN” Karras et al. ICLR’18

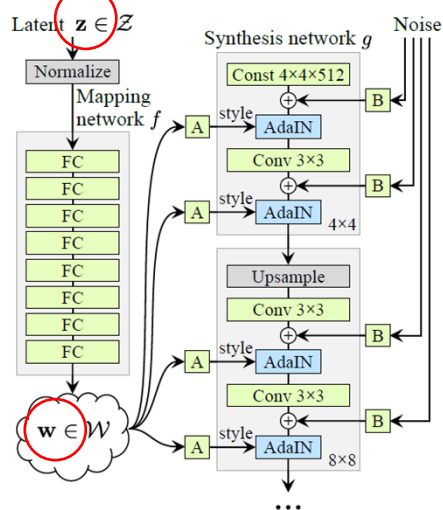


<https://youtu.be/XOxxPcy5Gr4>

125

125

## Style GAN



(b) Style-based generator

Le Générateur reçoit une version modifiée “w” par 8 couches FC du vecteur latent “z”

Karras et al. A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

126

# Style GAN

Latent  $\mathbf{z} \in \mathcal{Z}$

Mapping network  $f$

FC

$\mathbf{w} \in \mathcal{W}$

Synthesis network  $g$

Const  $4 \times 4 \times 512$

AdaIN

Conv  $3 \times 3$

4x4

Upsample

Conv  $3 \times 3$

8x8

Noise

Le Générateur reçoit une version modifiée “w” par 8 couches FC du vecteur latent “z”

Et ce, à **chaque couche** du réseau

(b) Style-based generator

Karas et al. A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

Et ce, à **chaque couche** du réseau

Karas et al. A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

128

# Style GAN

Latent  $\mathbf{z} \in \mathcal{Z}$

Noise

Synthesis network  $g$

Const  $4 \times 4 \times 512$

Mapping network  $f$

FC

AdaIN

Conv  $3 \times 3$

Upsample

style

A

B

$w \in \mathcal{W}$

(b) Style-based generator

L'entrée du réseau est un **tenseur constant** appris par entraînement

Karas et al. A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

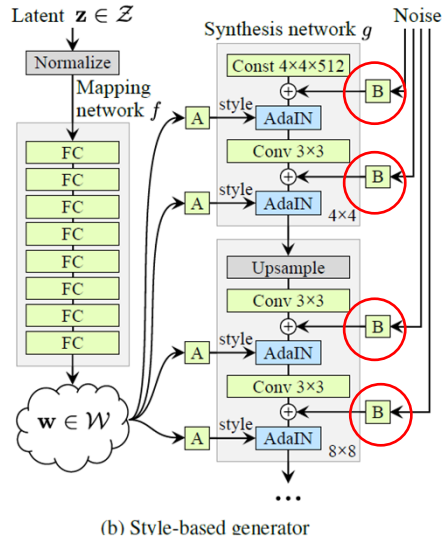
L'entrée du réseau est un **tenseur constant** appris par entraînement

Karas et al. A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

128



## Style GAN



Du **bruit** est multiplié à chaque carte d'activation de **chaque couche** du réseau

129

129

## Effet du bruit

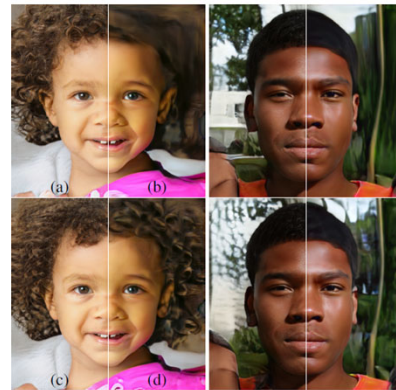
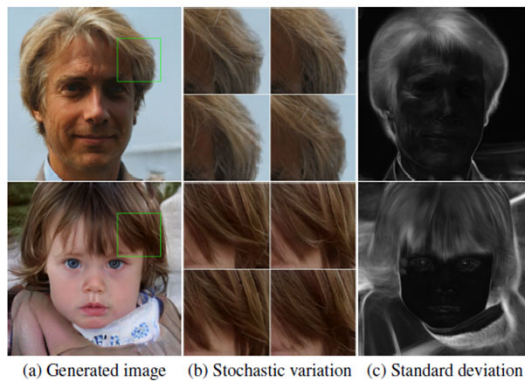
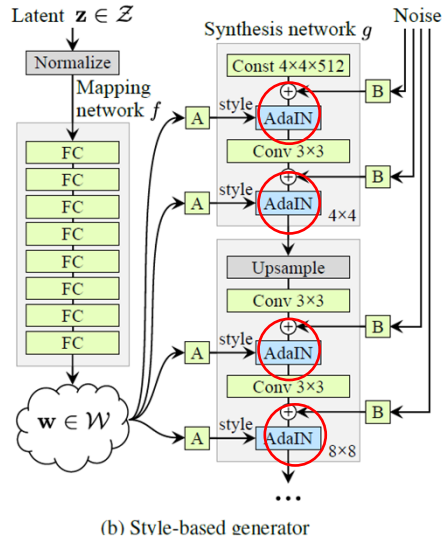


Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only ( $64^2 - 1024^2$ ). (d) Noise in coarse layers only ( $4^2 - 32^2$ ). We can see that the artificial omission of noise leads to featureless "painterly" look. Coarse noise causes large-scale curling of hair and appearance of larger background features, while the fine noise brings out the finer curls of hair, finer background detail, and skin pores.

Karas et al. A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

130

# Style GAN



AdaIN: adaptive instance normalization

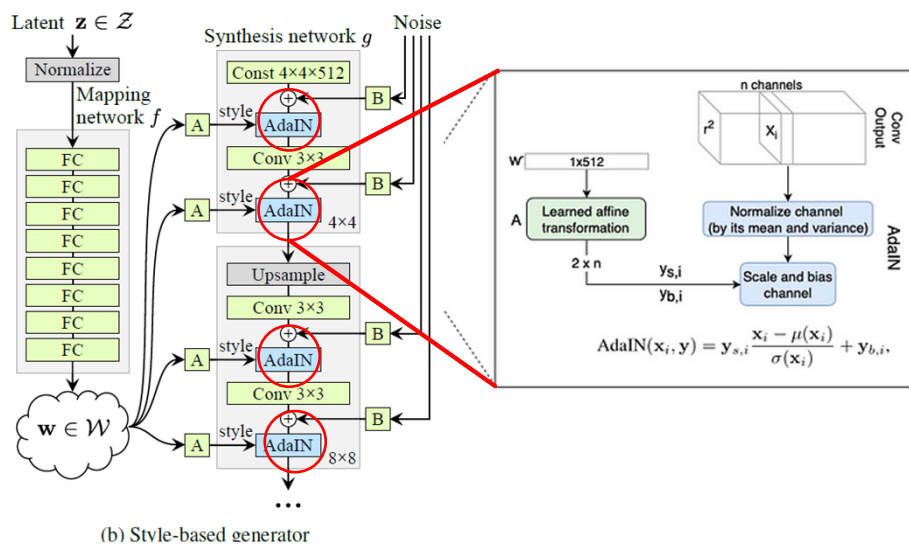
$$AdaIN(x, y) = \sigma(y) \frac{x - \mu(x)}{\sigma(x)} + \mu(y)$$

Comme du batchNorm, mais dont les 2 opérateurs sont fournis par w

131

# Style GAN

AdaIN: adaptive instance normalisation



<https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431>

132

## Style GAN

Entraînement progressif comme pour **progressive GAN**

133

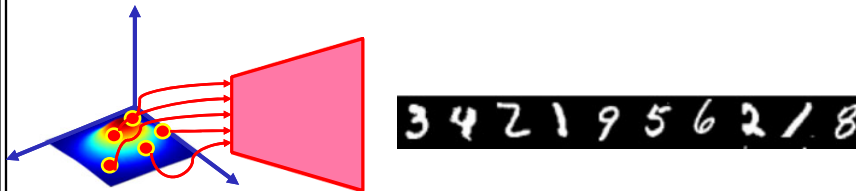
## Style GAN



134

## Défi avec les GAN

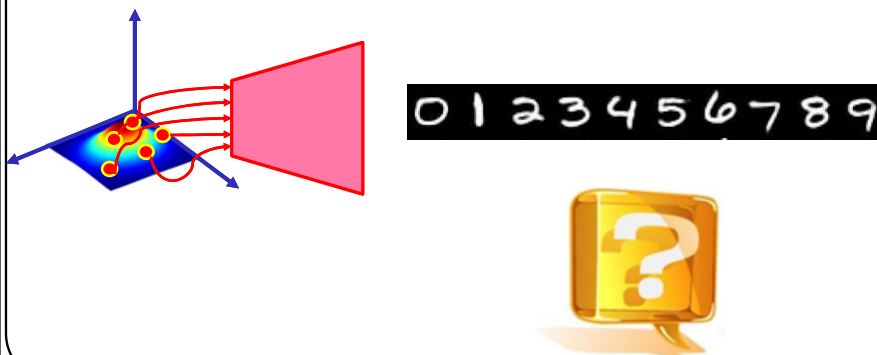
Soit un GAN entraîné sur MNIST, si je décode **10 vecteurs latents pris au hasard**, j'aurai les images de **10 caractères aléatoires**.



135

## Défi avec les GAN

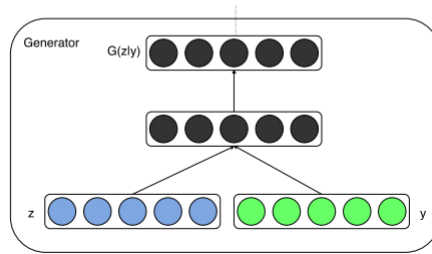
**Question:** comment générer des images de catégories prédéterminées? Ex. comment sélectionner 10 vecteurs latents qui produiront la séquence : 0,1,2,3,4,5,6,7,8,9?



136

## Gan conditionnel

L'idée est d'encoder un vecteur latent  $\vec{z}$  ainsi qu'un **vecteur de classe** « one-hot »  $\vec{y}$

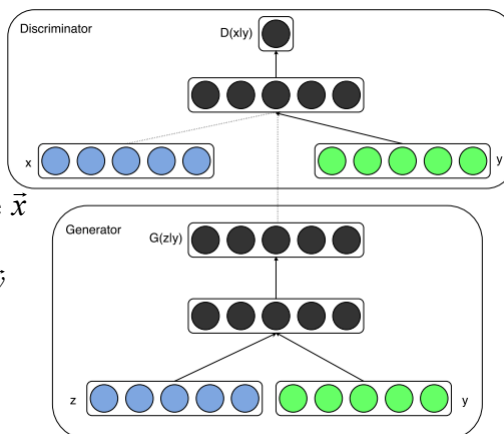


Mirza, Mehdi & Osindero, Simon. (2014). Conditional Generative Adversarial Nets. arXiv:1411.1784v1

137

## Gan conditionnel

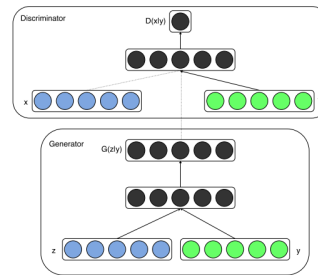
Et de discriminer une image  $\vec{x}$  avec **le même** « one-hot »  $\vec{y}$



Mirza, Mehdi & Osindero, Simon. (2014). Conditional Generative Adversarial Nets. arXiv:1411.1784v1

138

## Gan conditionnel



### GAN de base

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

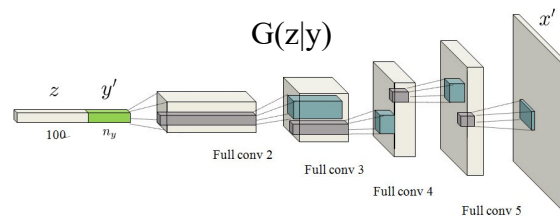
### GAN conditionnel

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z}|\mathbf{y})))]$$

Mirza, Mehdi & Osindero, Simon. (2014). Conditional Generative Adversarial Nets. arXiv:1411.1784v1

139

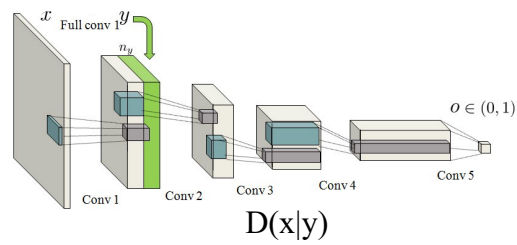
## Version convolutionnelle



<https://medium.com/@sam.maddrellmander/conditional-dcgan-in-tensorflow-336f8b03b7b6>

140

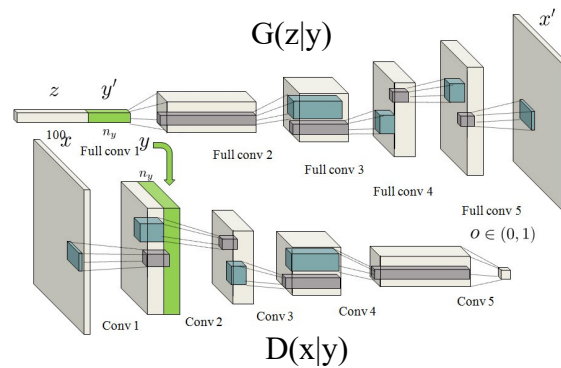
## Version convolutionnelle



<https://medium.com/@sam.maddrellmander/conditional-dcgan-in-tensorflow-336f8b03b7b6>

141

## Version convolutionnelle



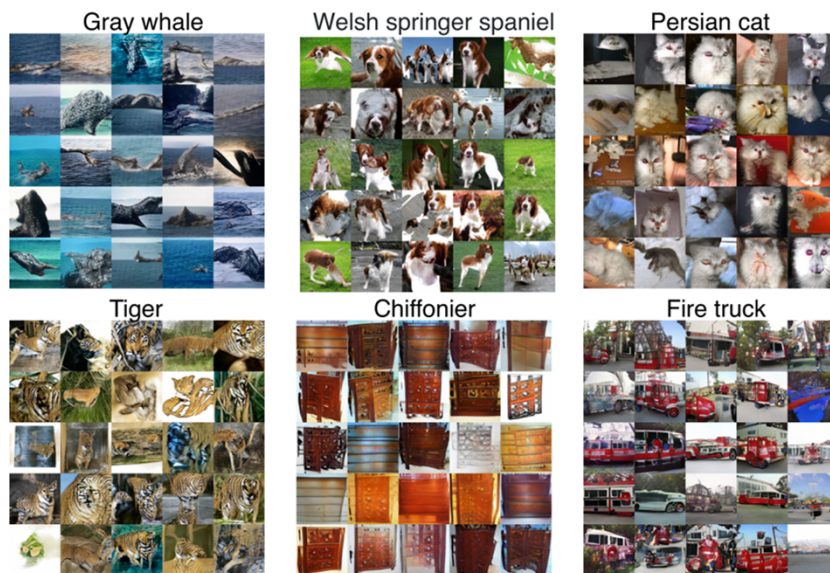
<https://medium.com/@sam.maddrellmander/conditional-dcgan-in-tensorflow-336f8b03b7b6>

142



"t-shirt", 'pants', 'pullover', 'dress', 'coat', 'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot'.

143



Miyato, T., Kataoka, T., Koyama, M., & Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*.

144



## Code pytorch pour plus de 30 modèles de GANs

<https://github.com/eriklindernoren/PyTorch-GAN>

145

Belle vidéo sur les GANs montrant  
comment on peut manipuler l'espace  
latent et comment certains les utilise  
pour produire des « *deep fake* »

<https://www.youtube.com/watch?v=dCKbRCUyop8>

146