Réseaux de neurones

IFT 780

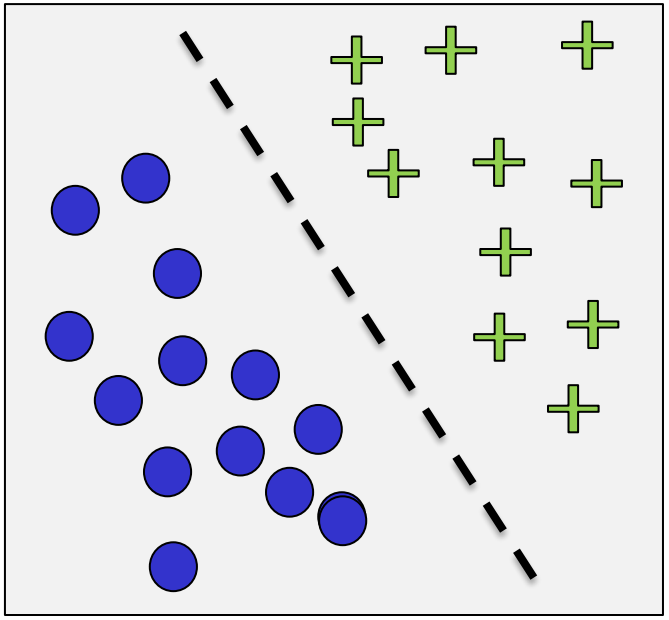
Modèles génératifs

Par

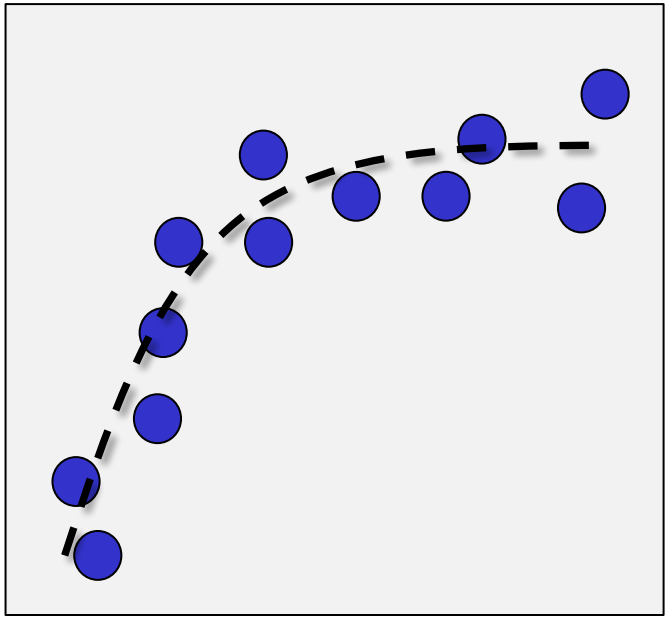
Pierre-Marc Jodoin, Antoine Théberge

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

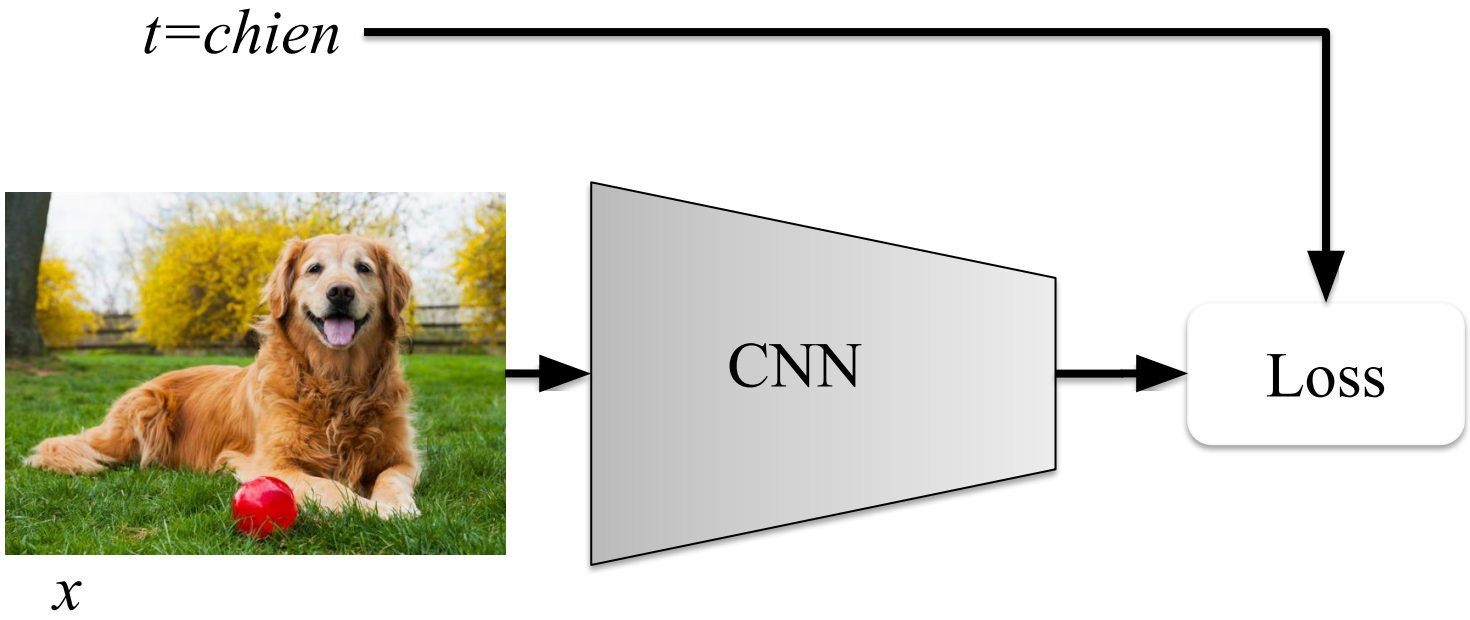
Apprentissage supervisé

Deux grandes familles d'applications

- **Classification :** la cible est un indice de classe $t \in \{1, \ldots, K\}$
  - 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



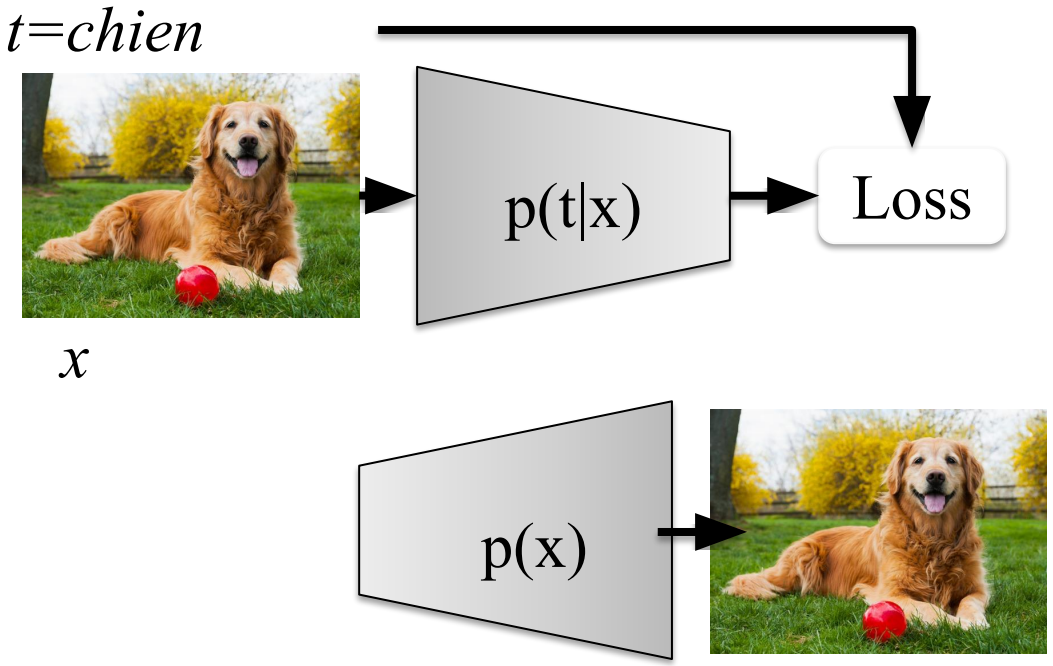
Modèles discriminatifs vs. génératifs

p(t|x)

probabilité de la classe t en sachant x

p(x)

probabilité de x



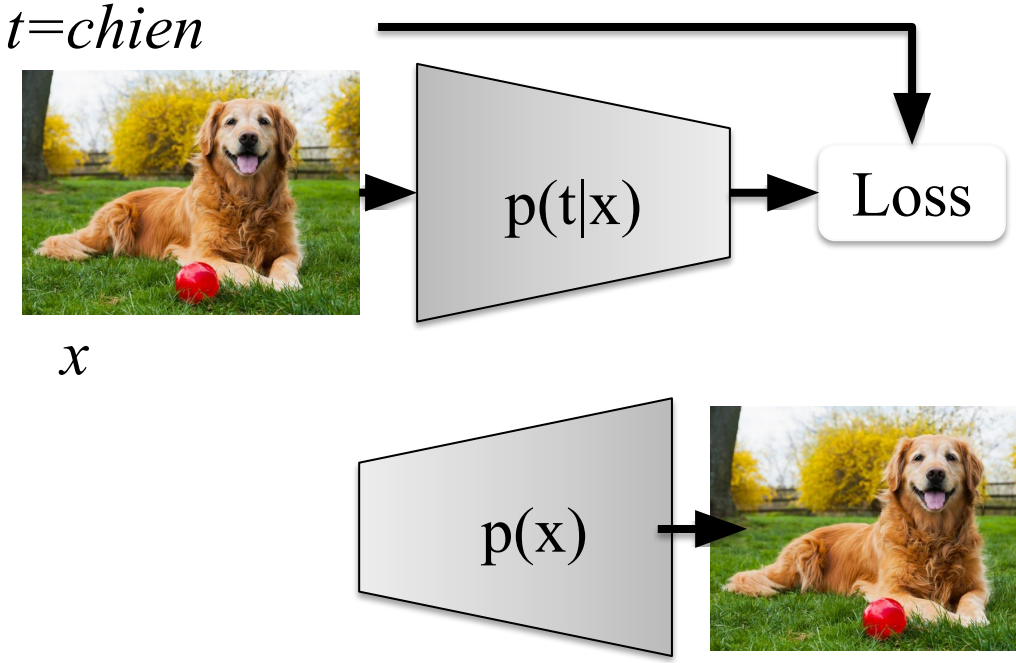
Modèles discriminatifs vs. génératifs

p(t|x) -> p(chien| )

probabilité de la classe t en sachant x

p(x) -> p()

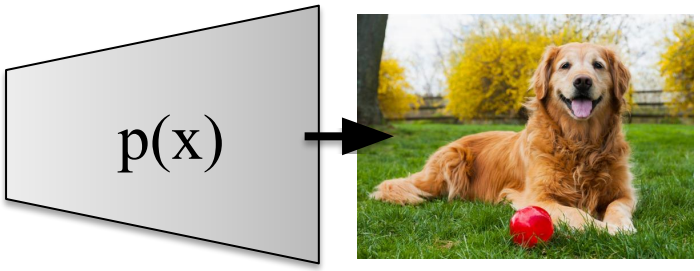
probabilité de x



Modèles génératifs

Comment apprendre p(x) ?

p(x) -> p()
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 $D = \{\vec{x_1}, \vec{x_2}, ..., \vec{x_N}\}$

$$\theta^* = \text{argmax}_\theta \prod_{i=1}^{N} p_\theta(\vec{x_i})$$

$$= \text{argmax}_\theta \sum_{i=1}^{N} \log p_\theta(\vec{x_i})$$

Supervisé vs non supervisé

Apprentissage supervisé : il y a une cible

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

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

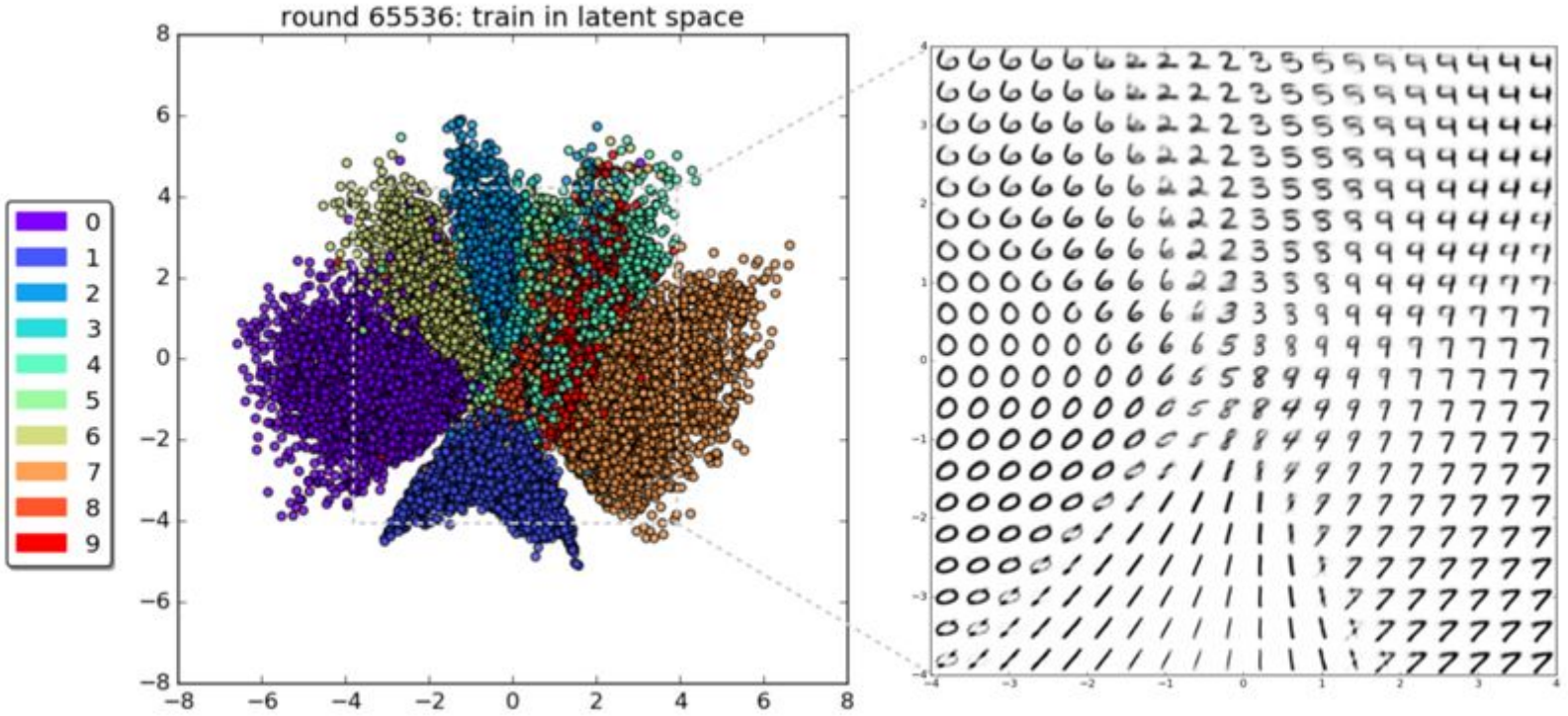
$$D = \{x_1, x_2, \ldots, x_N\}$$

Apprentissage non supervisé

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

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

Exemple : visualization de la distribution des images MNIST



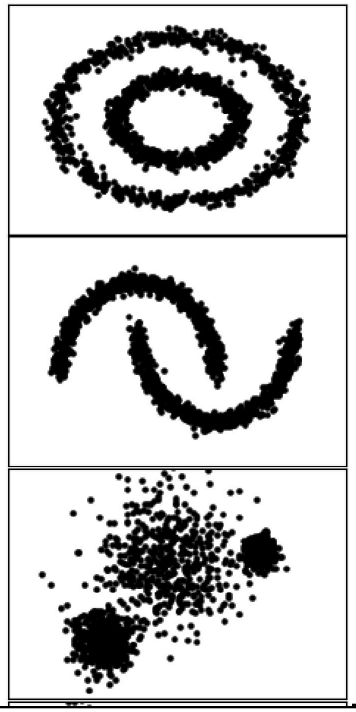
https://blog.fastforwardlabs.com/2016/08/12/introducing-variational-autoencoders-in-prose-and.html

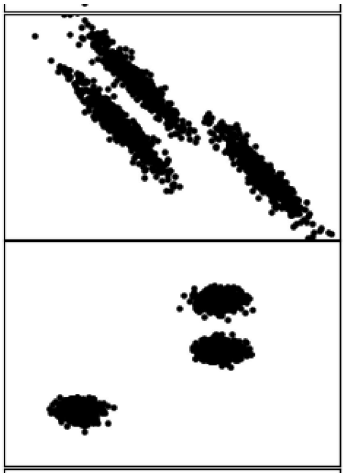
Apprentissage non supervisé

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"



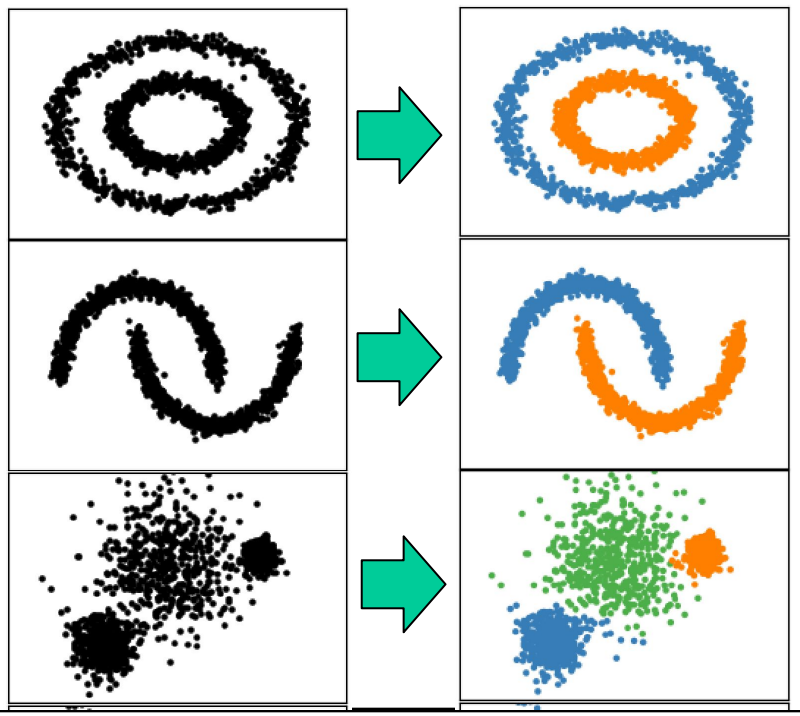


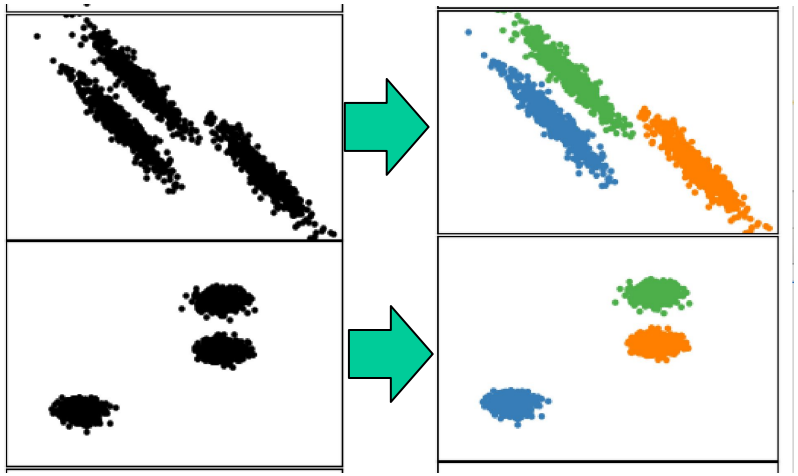
Apprentissage non supervisé

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 $p(x, y), p(x \mid y), p(y)$ 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 $p(x_1|x_2) = p(x_1)$

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 $p(x_2|x_1) = p(x_2)$
  $p(x_1,x_2) = p(x_1|x_2)p(x_2)$

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 $p(x_2|x_1) = p(x_2)$
  $p(x_1,x_2) = p(x_1|x_2)p(x_2)$

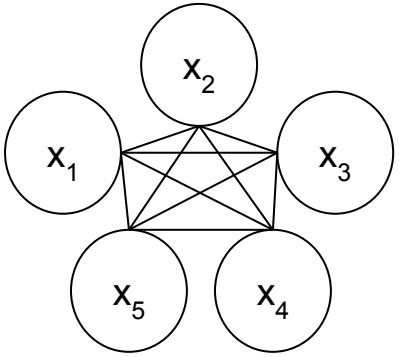
  Exemple: x_1 = lire dans le jardin
  x_2 = il pleut dehors

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

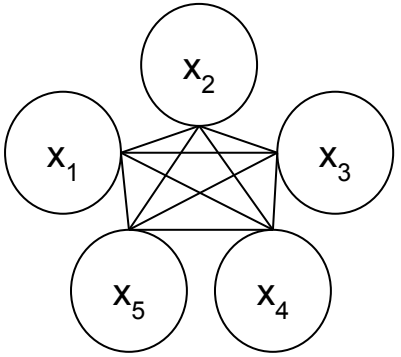


Apprentissage non supervisé

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



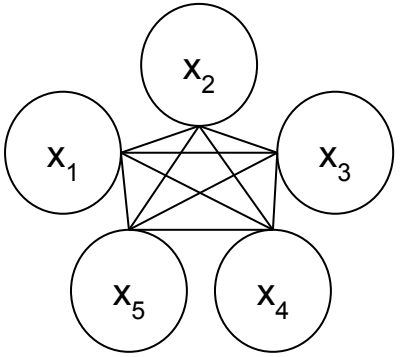
$p(\vec{x}) = p(x_1, x_2, x_3, x_4, x_5)$, compliqué à calculer car il faut considérer chaque combinaison

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

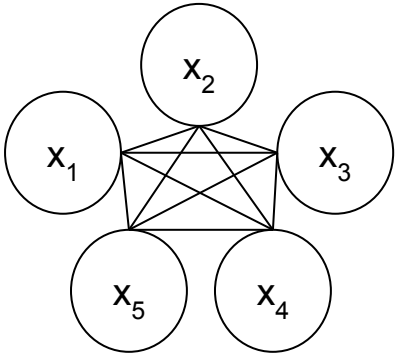
Règle en chaîne des probabilités: $p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1})$

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: $p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1})$

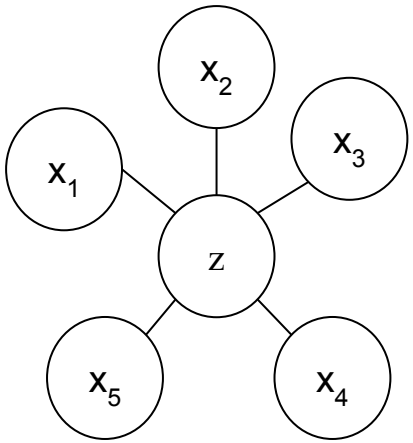
Pas si mal pour 5 variables, mais dans une image, chaque pixel est une variable !

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 entres elles

$$p(\vec{x}, z) = p(x_1, x_2, x_3, x_4, x_5, z)$$

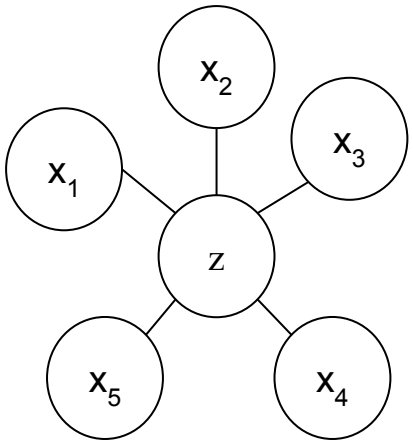


Apprentissage non supervisé

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 entres elles

$$p(\vec{x}, z) = p(x_1, x_2, x_3, x_4, x_5, z)$$
$$= p(z)p(x_1|z)...p(x_5|z)$$

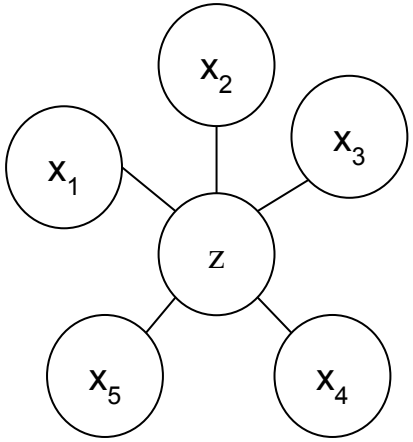


Apprentissage non supervisé

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 entres elles

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

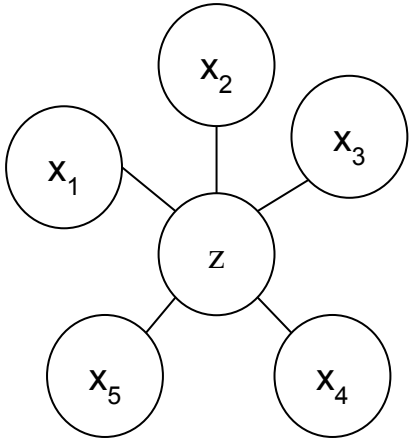


Apprentissage non supervisé

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 entres elles

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

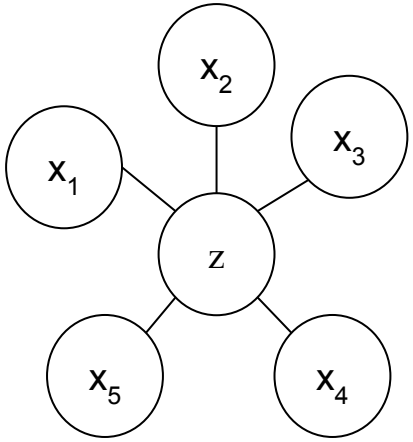


Apprentissage non supervisé

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 entres elles

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



On peut toujours présumer que z existe !

$$p(\vec{x}) = \int p(\vec{x}, z)dz \qquad \text{<= marginalisation}$$

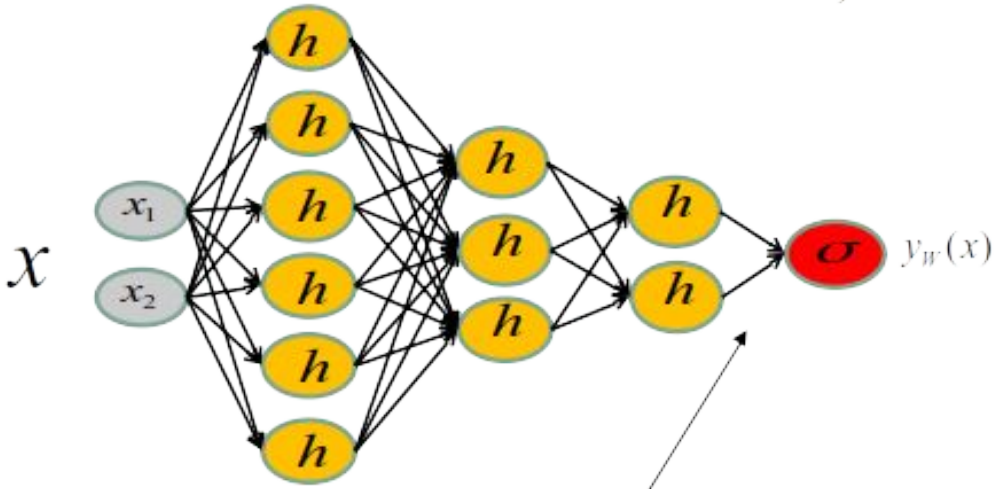
$$= \int p(\vec{x|z})p(z)dz$$

En apprentissage non-supervisé

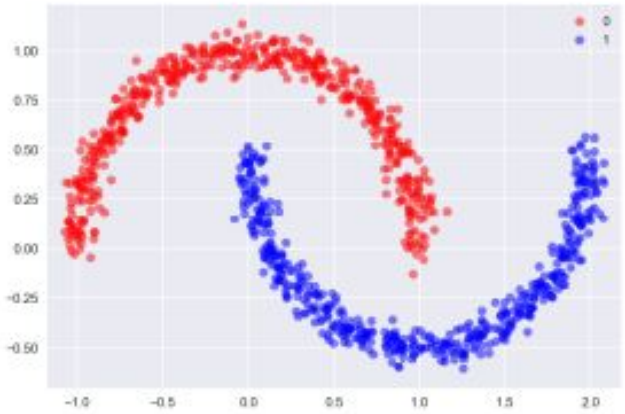
nous nous appuierons sur
2 propriétés des réseaux de neurones

Propriété 1

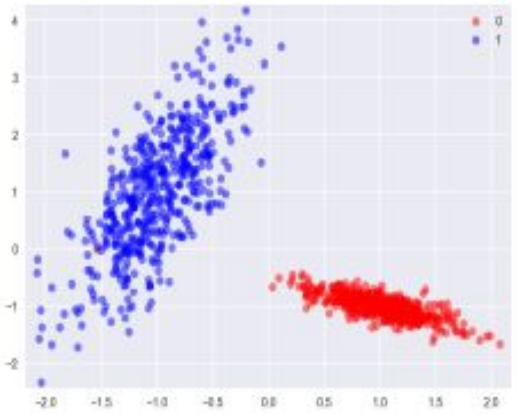
Les réseaux de neurones sont d'excellentes machines pour projeter des données brutes vers un **espace dimensionnel plus faible** dont les propriétés dépendent de la *loss* (ici **espace linéaire car la sortie du réseau est un classifieur linéaire**).



x

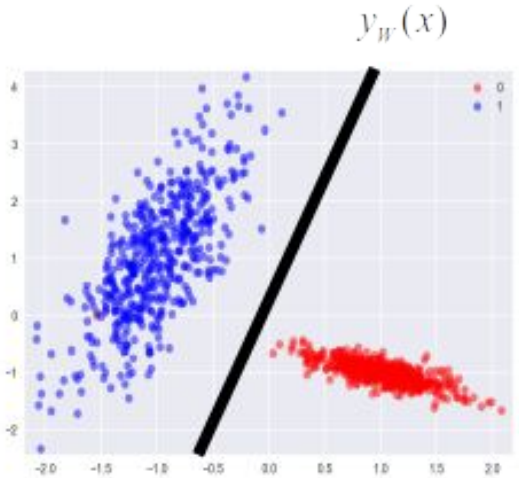


Données en entrée



Sortie de la dernière couche

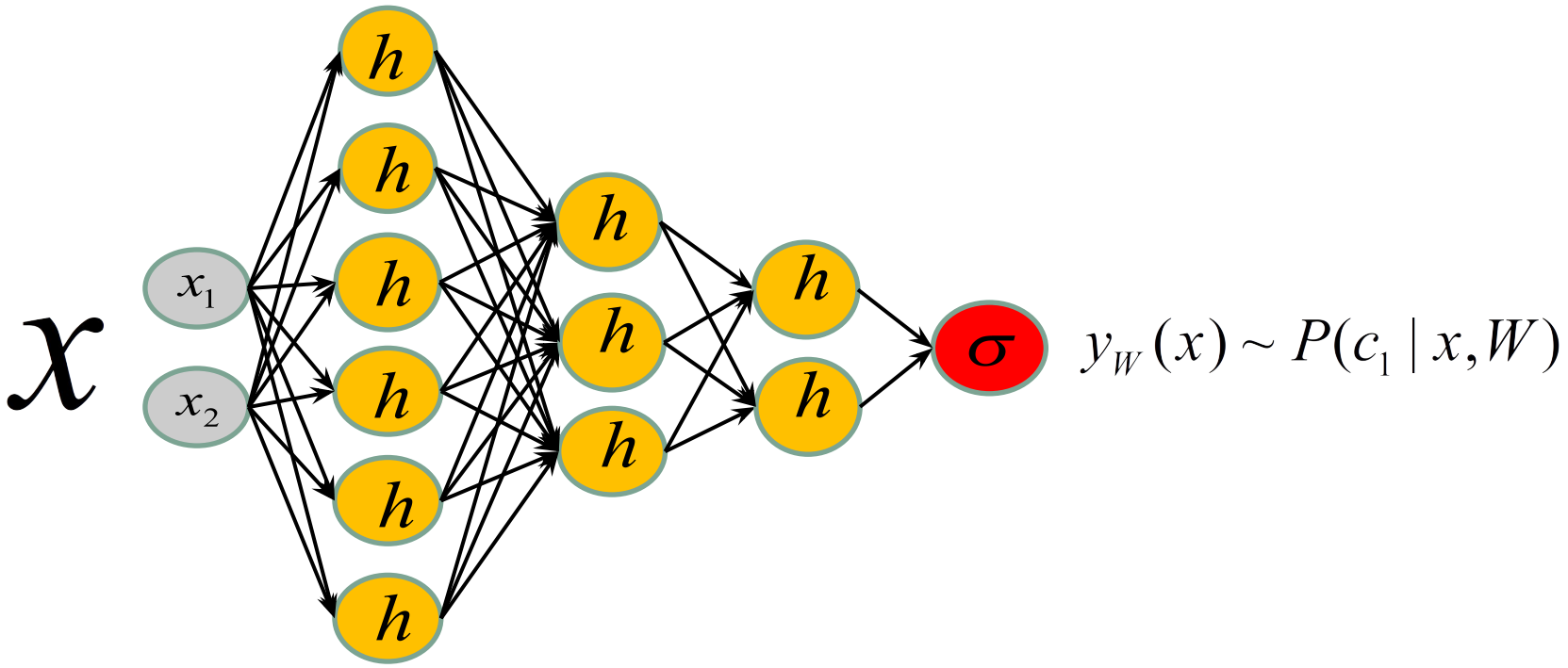
$y_W(x)$



Sortie du réseau

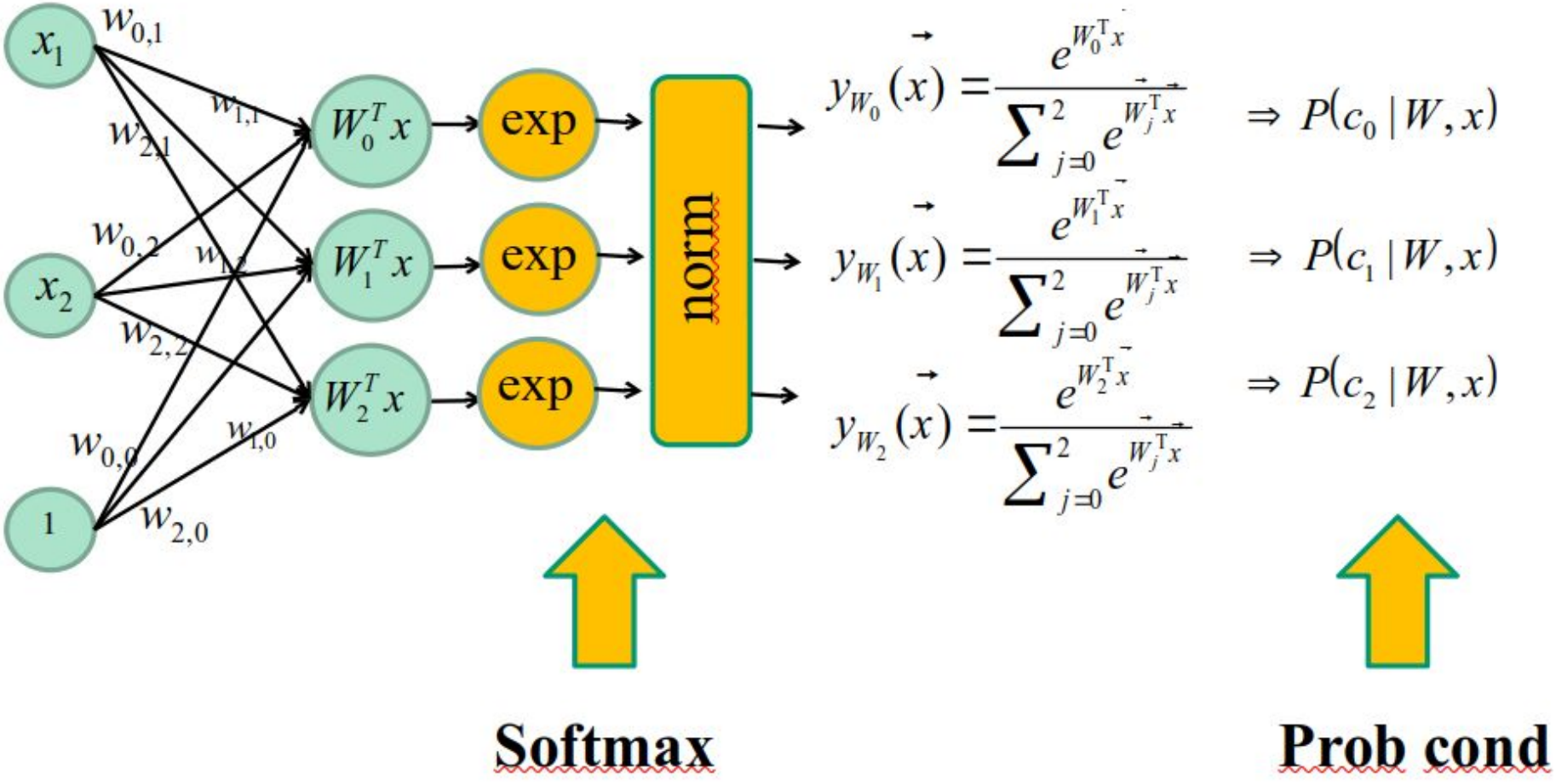
Propriété 2

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



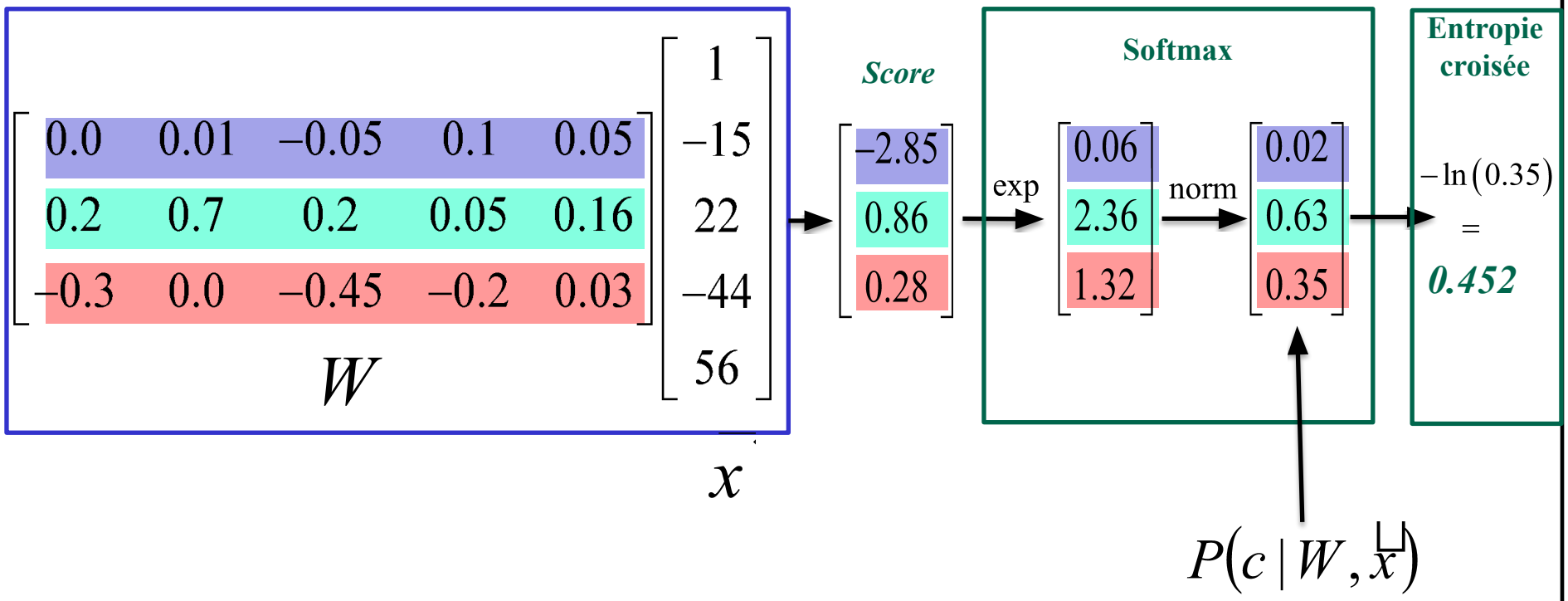
Propriété 2

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

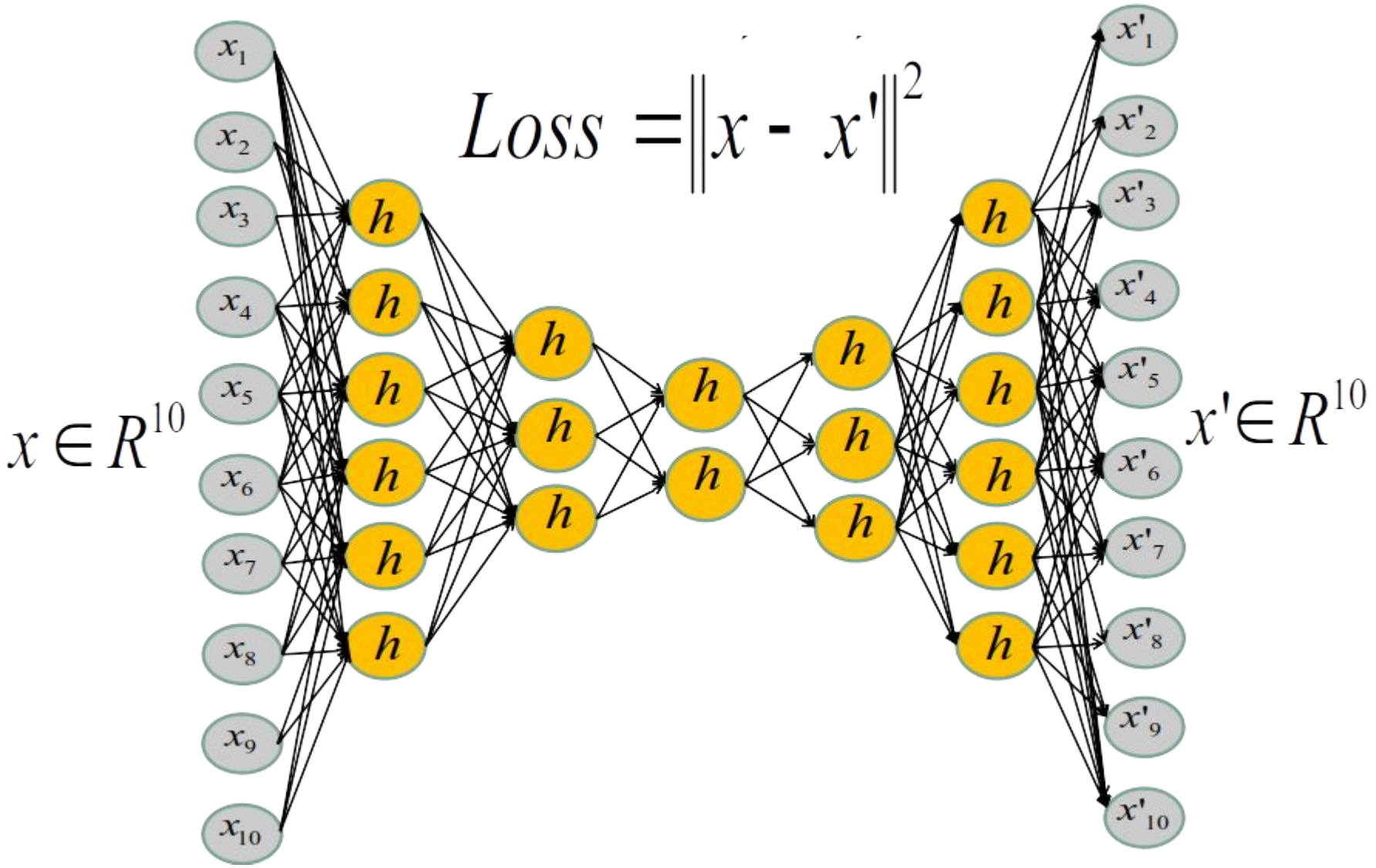


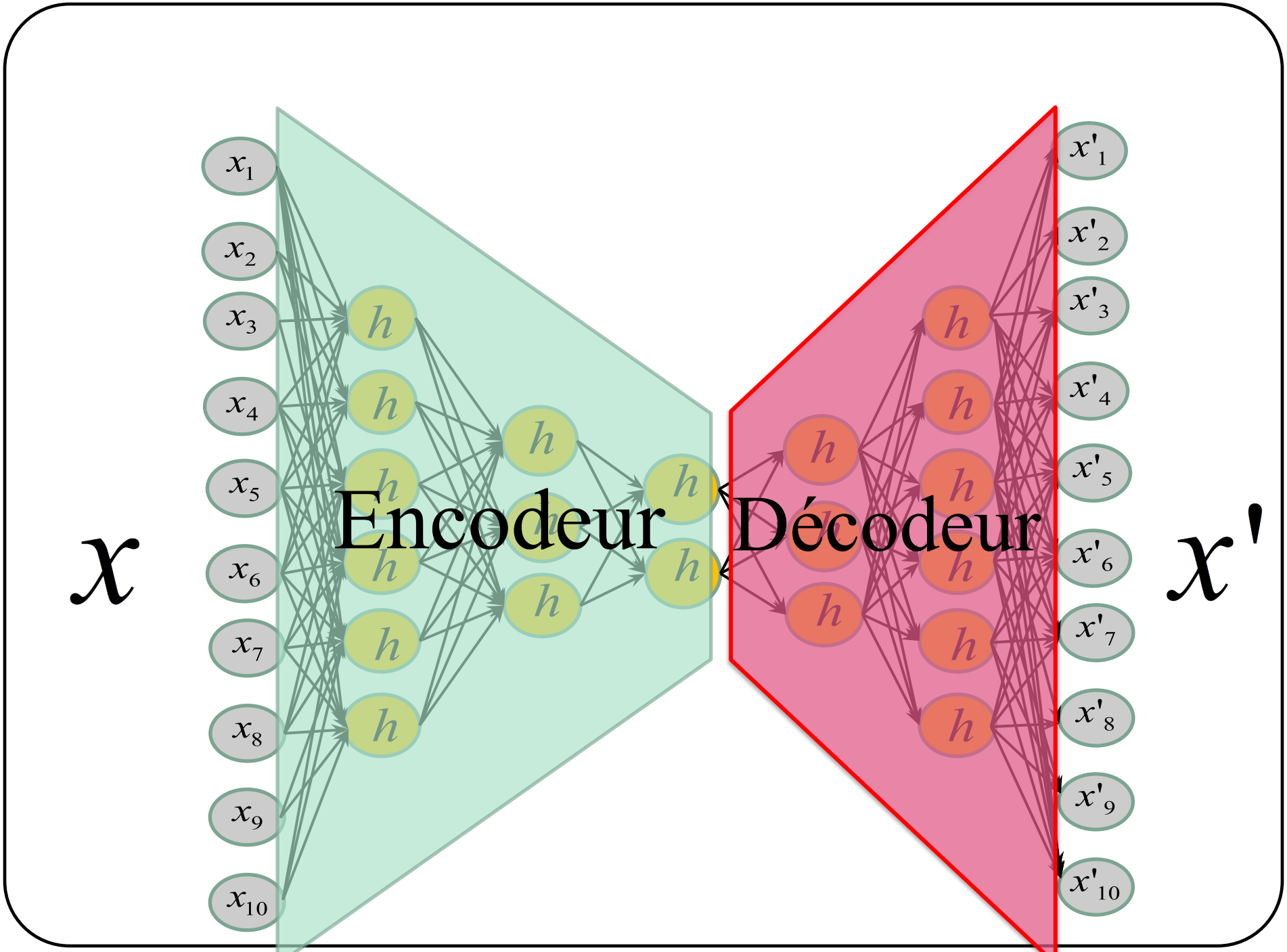
Rappel

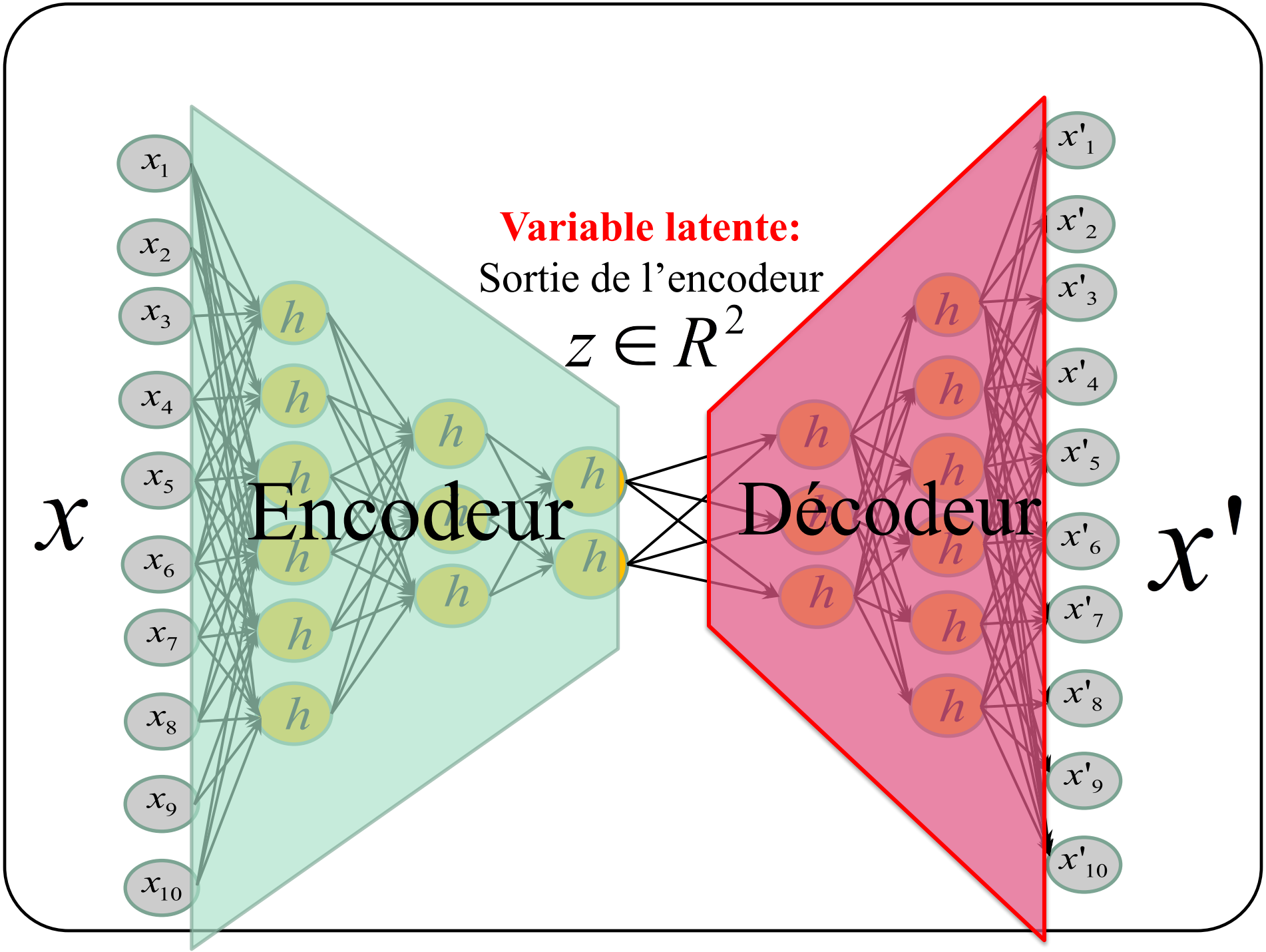
$$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 2$$

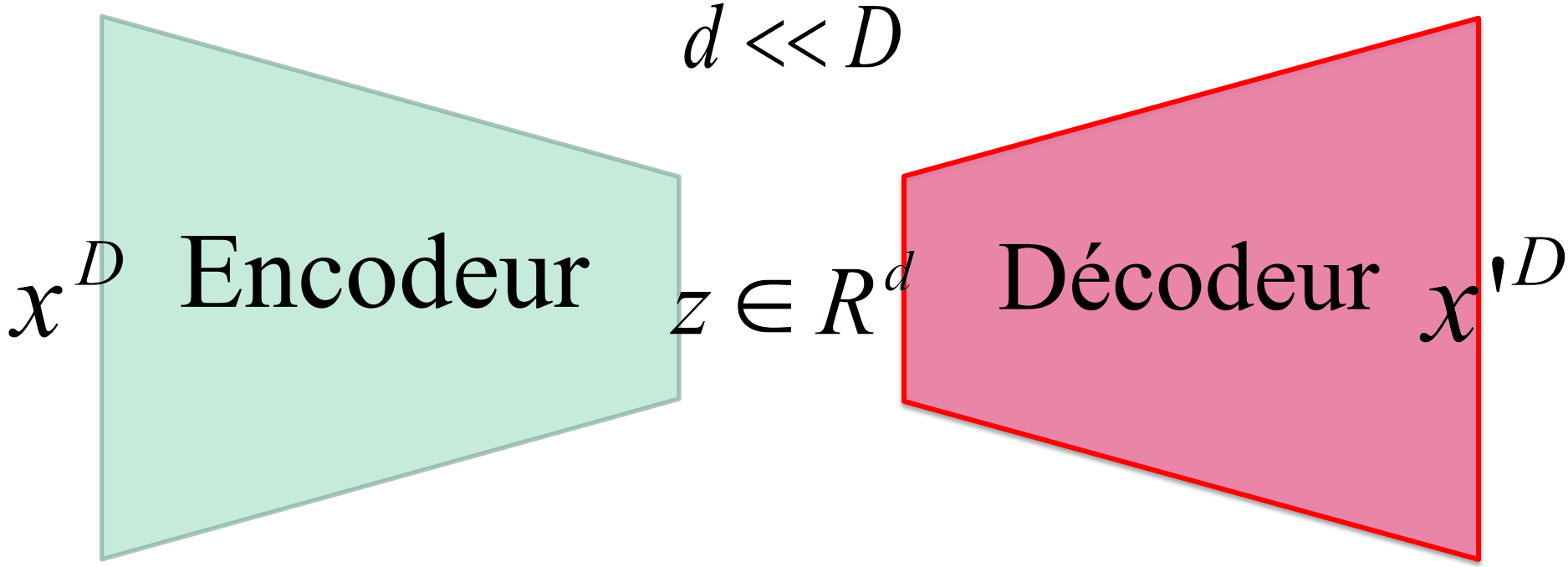


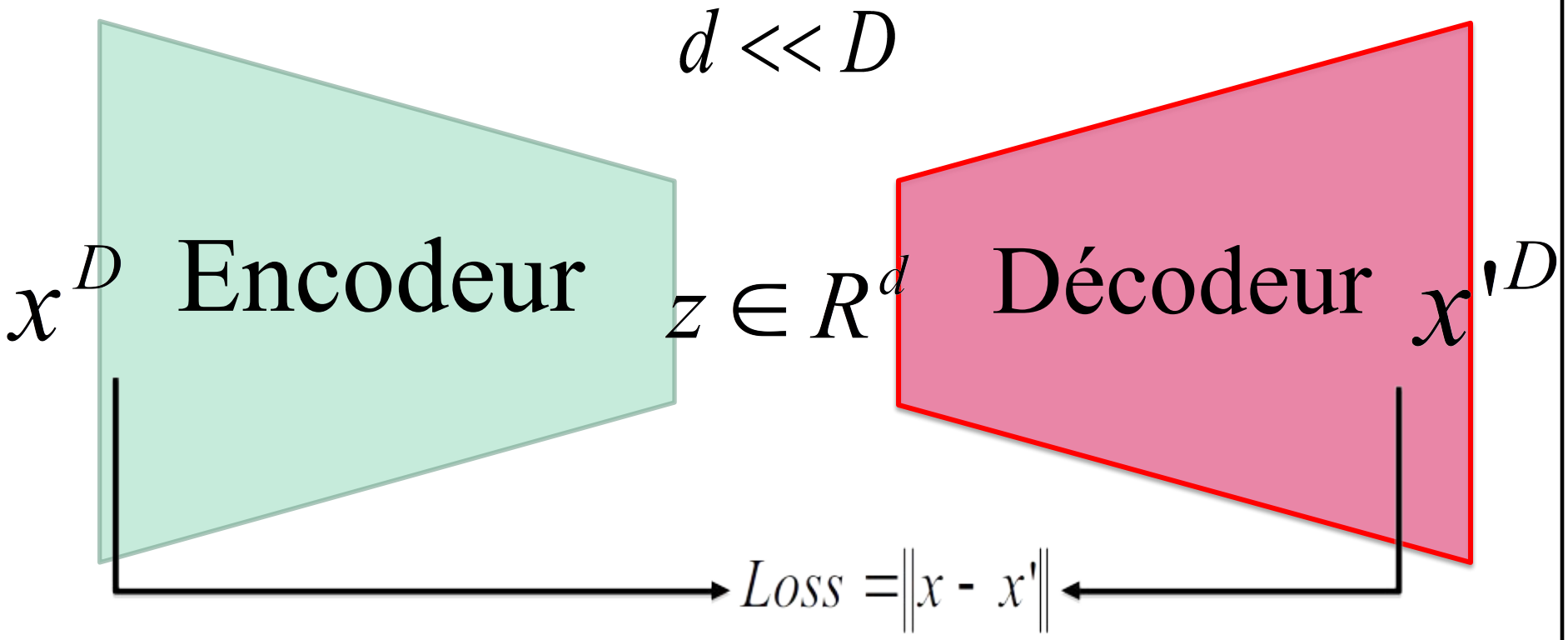
Comment utiliser un réseau de neurones pour apprendre la **configuration sous-jacente** de données non étiquettés?

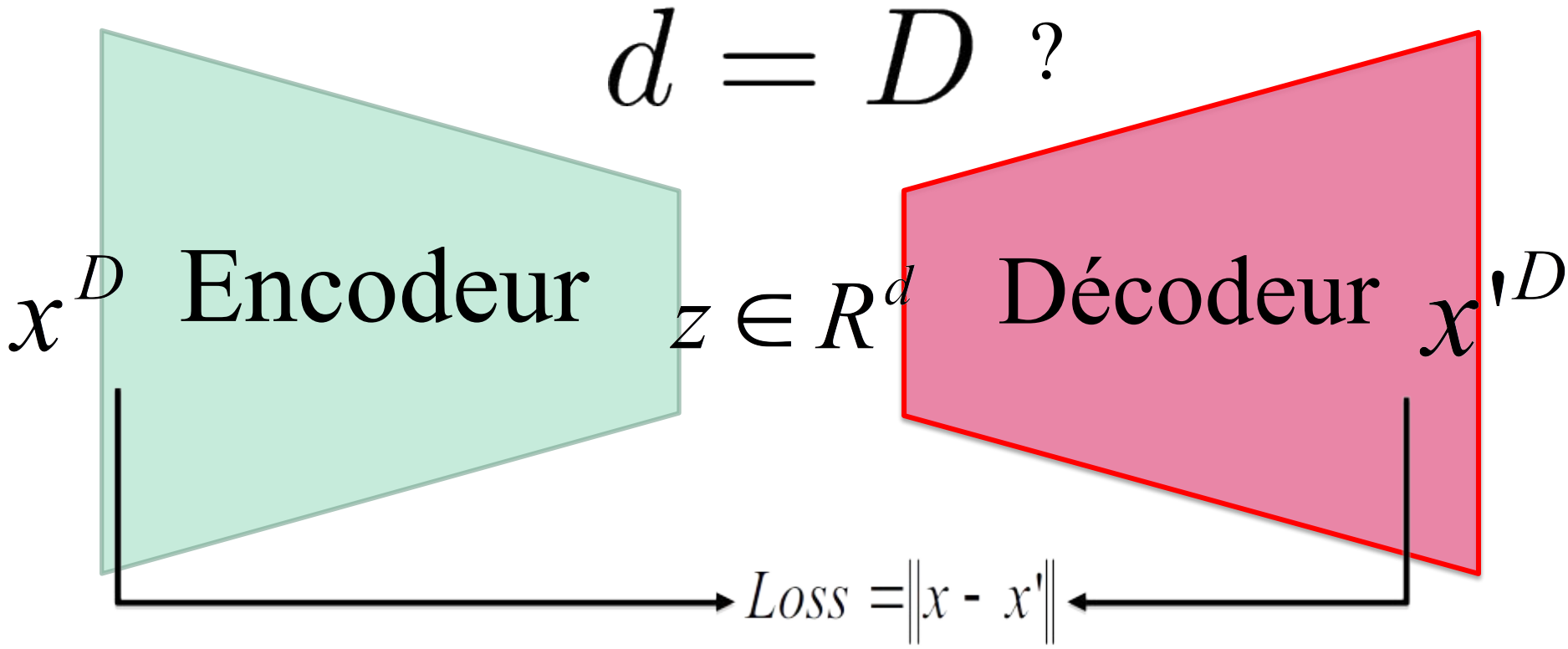

$Loss = \|x - x'\|^2$
$x \in R^{10}$
$x' \in R^{10}$
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}
x'_1 x'_2 x'_3 x'_4 x'_5 x'_6 x'_7 x'_8 x'_9 x'_{10}
h

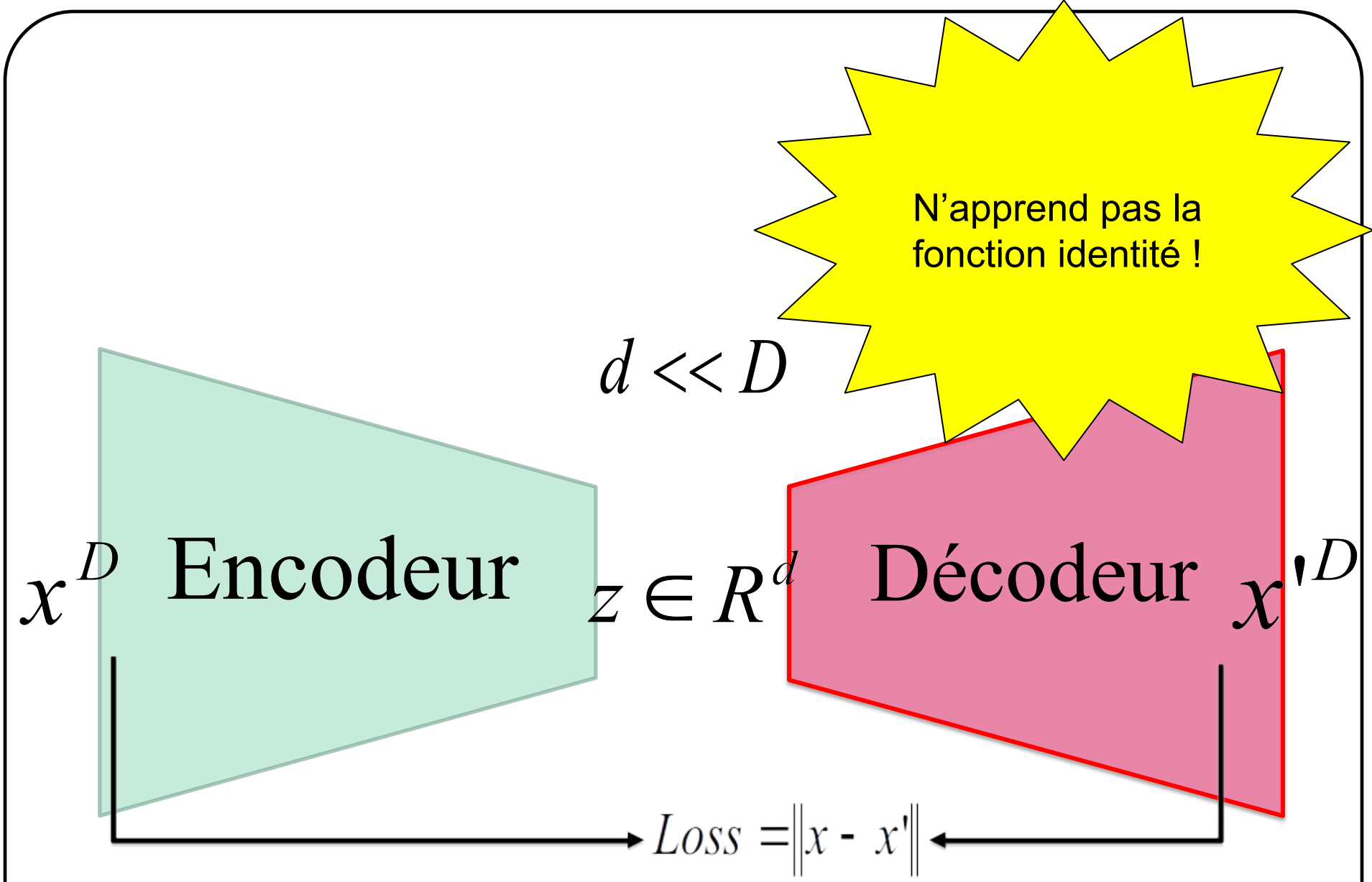

x
Encodeur
Décodeur
x'
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
h
x'_1
x'_2
x'_3
x'_4
x'_5
x'_6
x'_7
x'_8
x'_9
x'_{10}

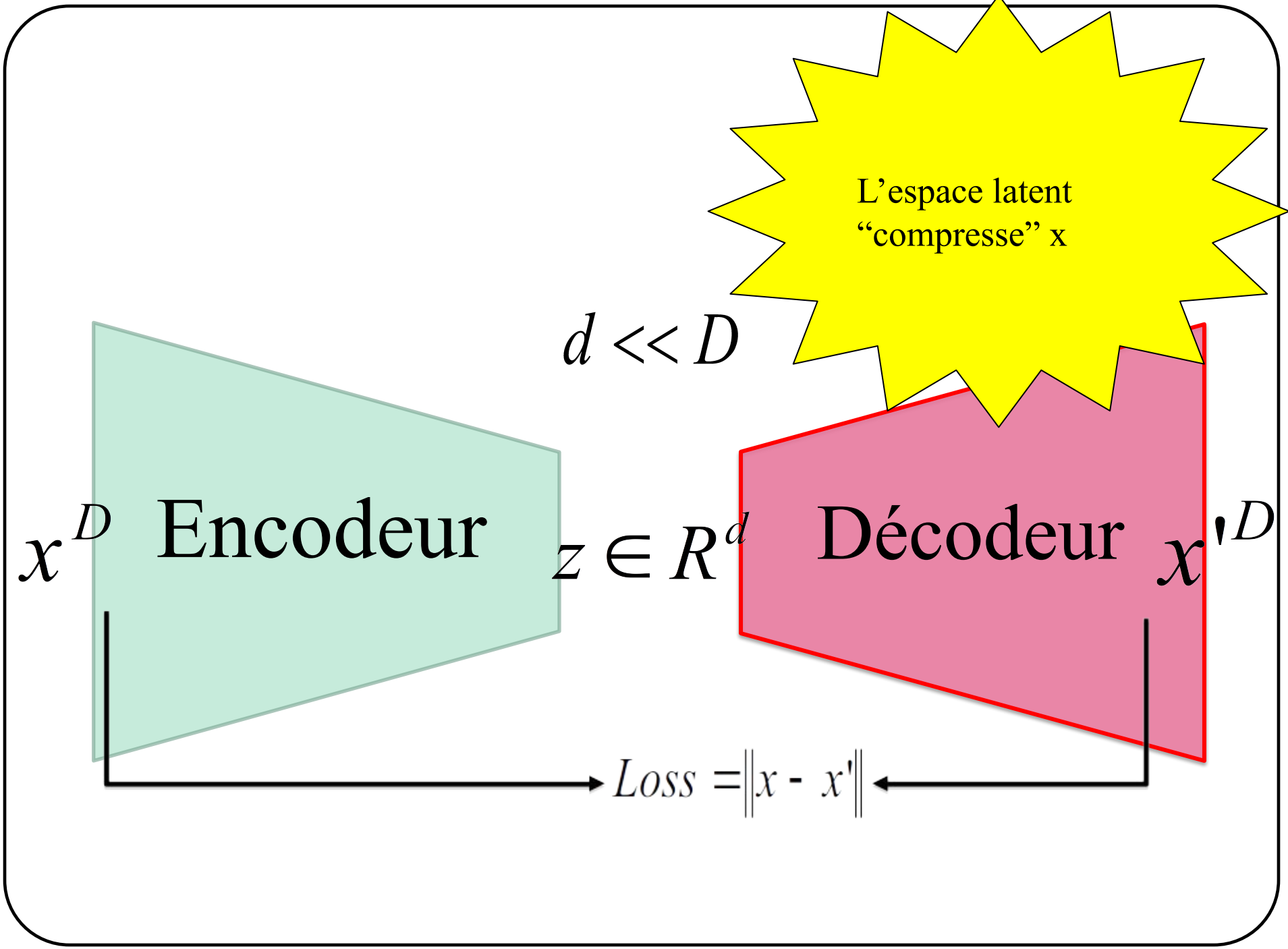

Variable latente:
Sortie de l'encodeur
$z \in R^2$
x
Encodeur
Décodeur
x'
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
x'_1
x'_2
x'_3
x'_4
x'_5
x'_6
x'_7
x'_8
x'_9
x'_{10}
h


$d << D$
x^D
Encodeur
$z \in R^d$
Décodeur
x'^D

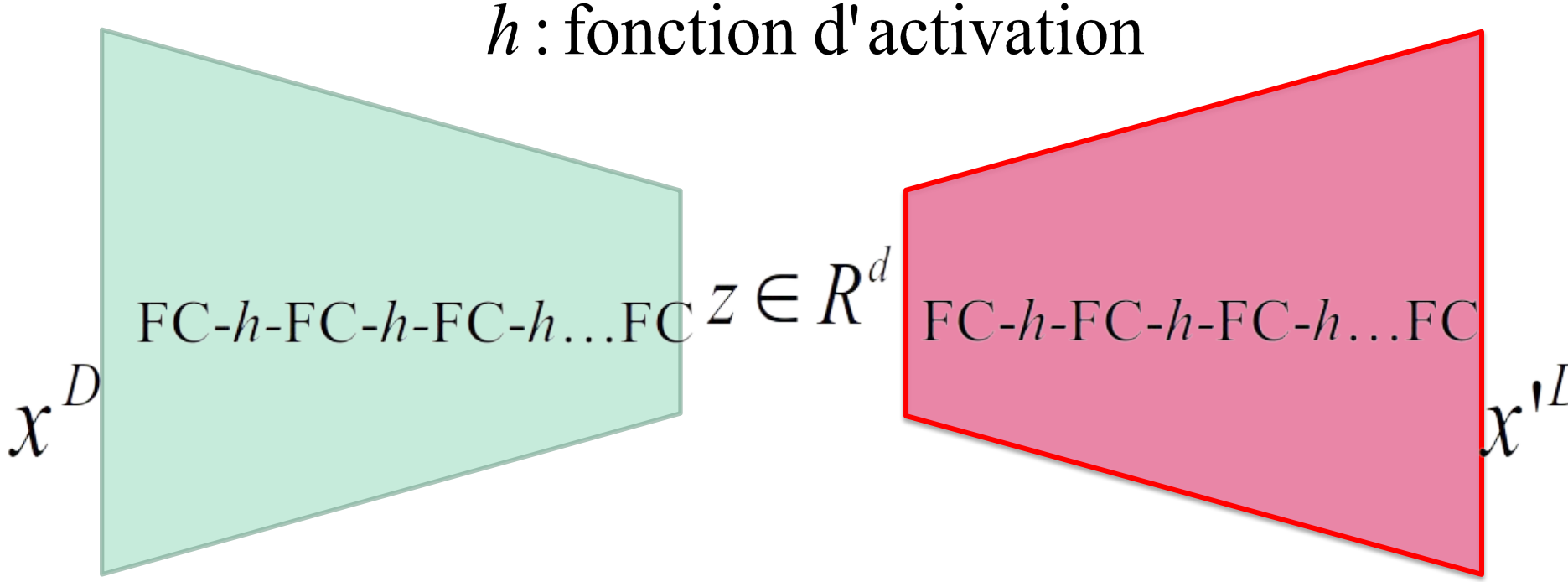

$d << D$
x^D
Encodeur
$z \in R^d$
Décodeur
x'^D
$Loss = \|x - x'\|$


$d = D$?
x^D
Encodeur
$z \in R^d$
Décodeur
x'^D
$Loss = \|x - x'\|$

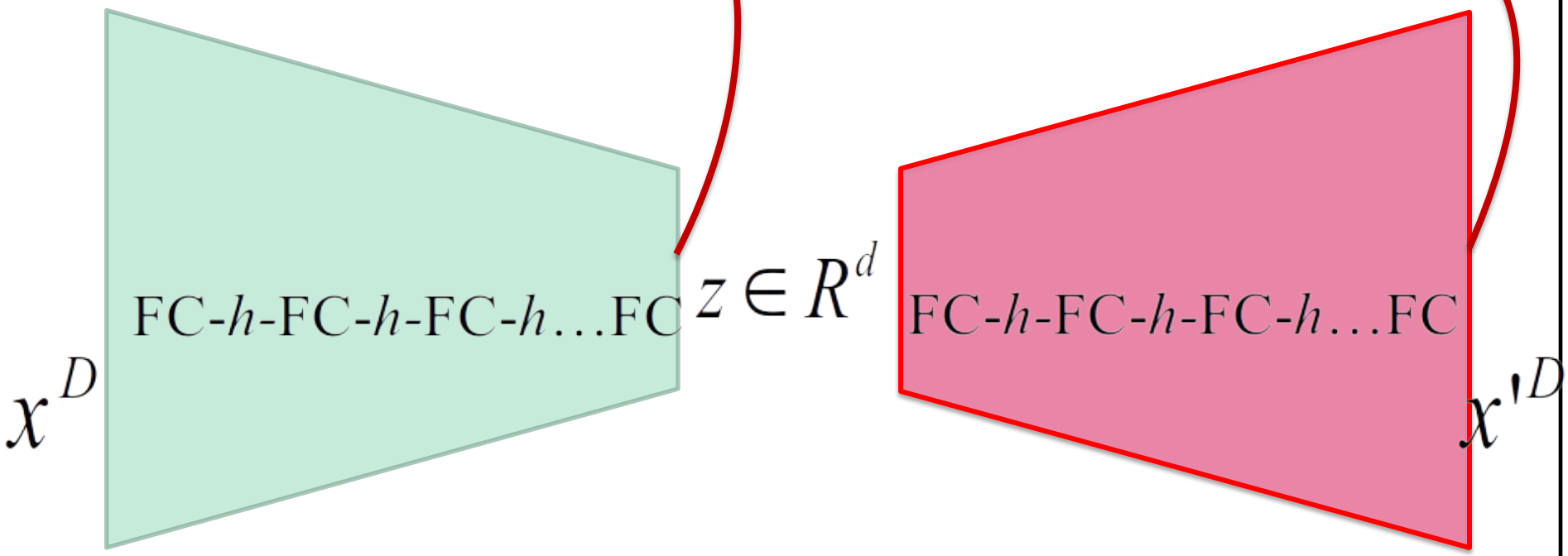

N’apprend pas la fonction identité !
$d << D$
x^D
Encodeur
$z \in R^d$
Décodeur
x'^D
$Loss = \|x - x'\|$


L’espace latent
“compresse” x
$d << D$
x^D
Encodeur
$z \in R^d$
Décodeur
x'^D
$Loss = \|x - x'\|$

Couches pleinement connectées

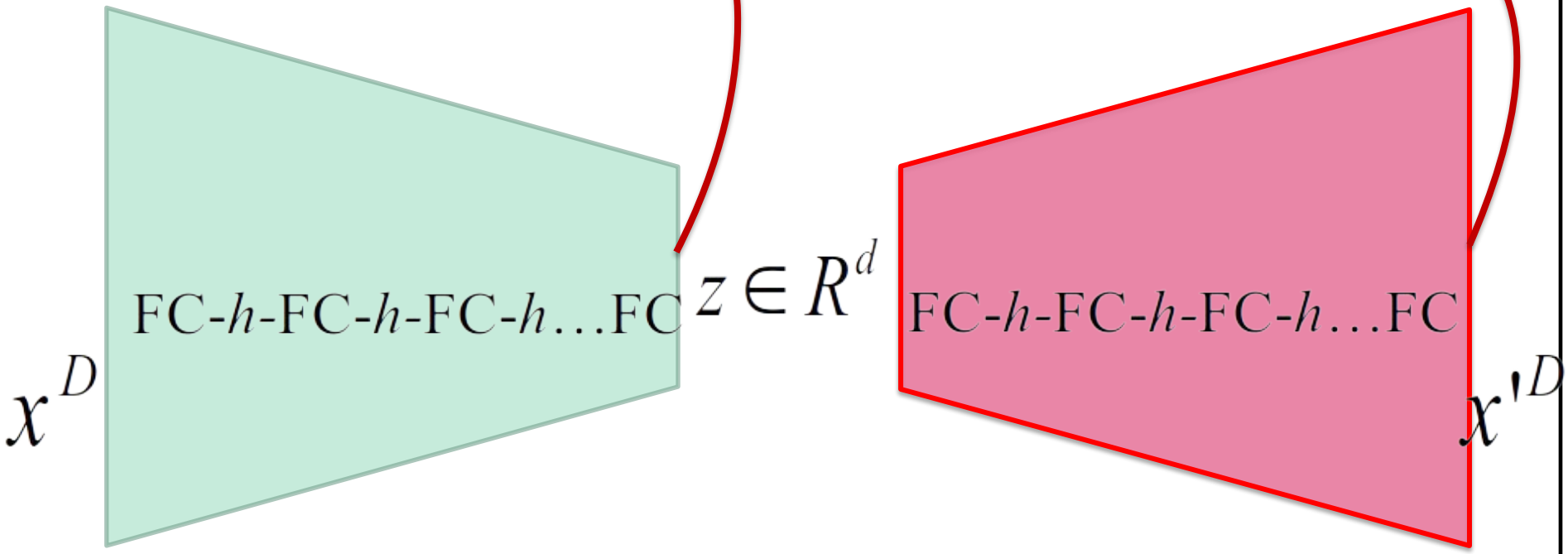


En général…
pas de fonction d'activation à la sortie
de l'encodeur et du décodeur



Selon vous, **pourquoi?**

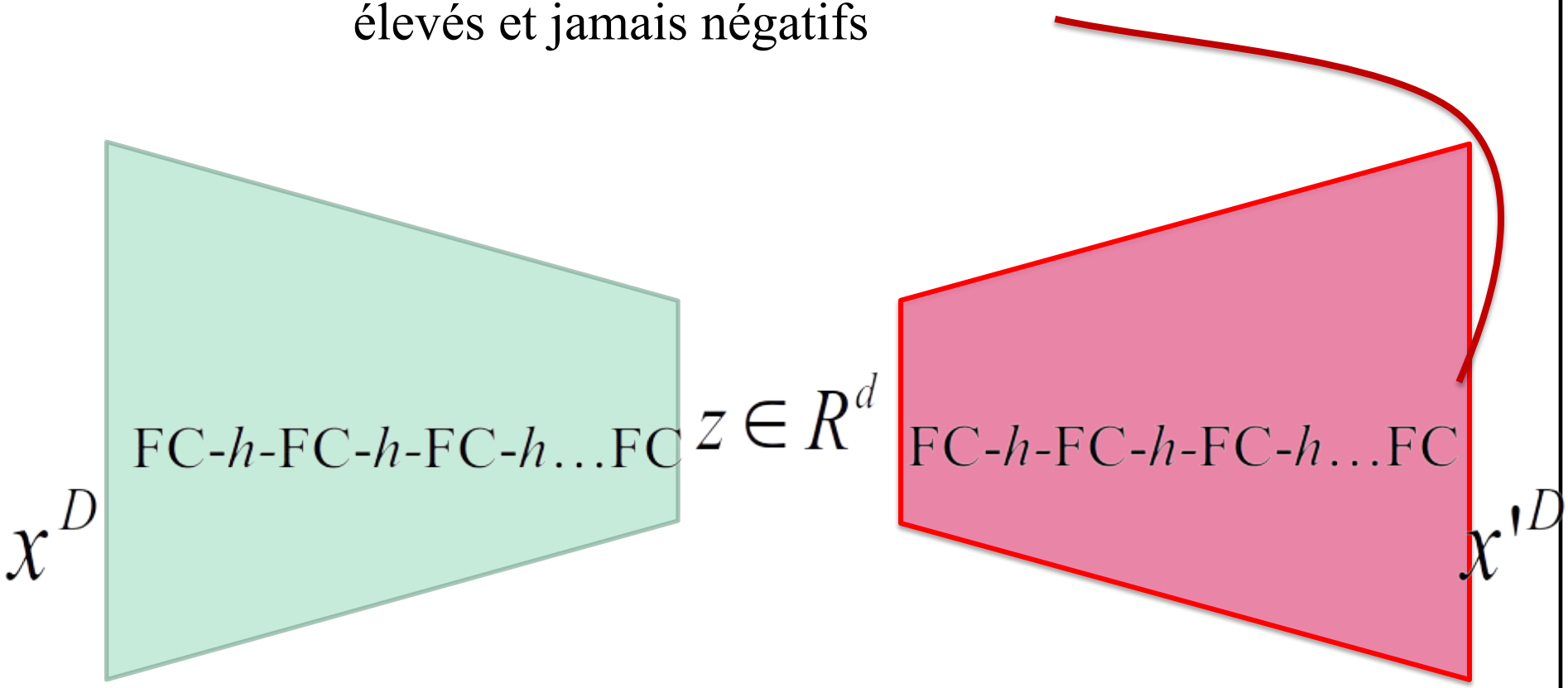
En général…
pas de fonction d'activation à la sortie
de l'encodeur et du décodeur



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 nivaux de gris peuvent être élevés et jamais négatifs



Le nombre de neurones

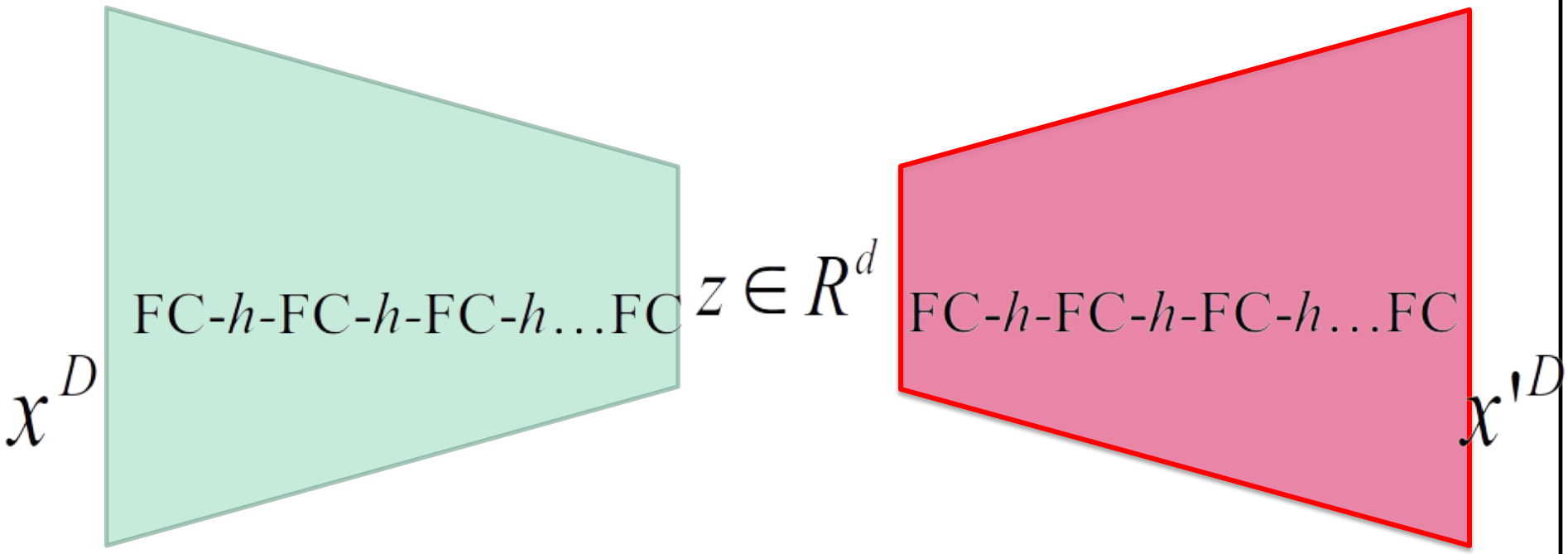
Décroit ou se maintient

d'une couche à l'autre

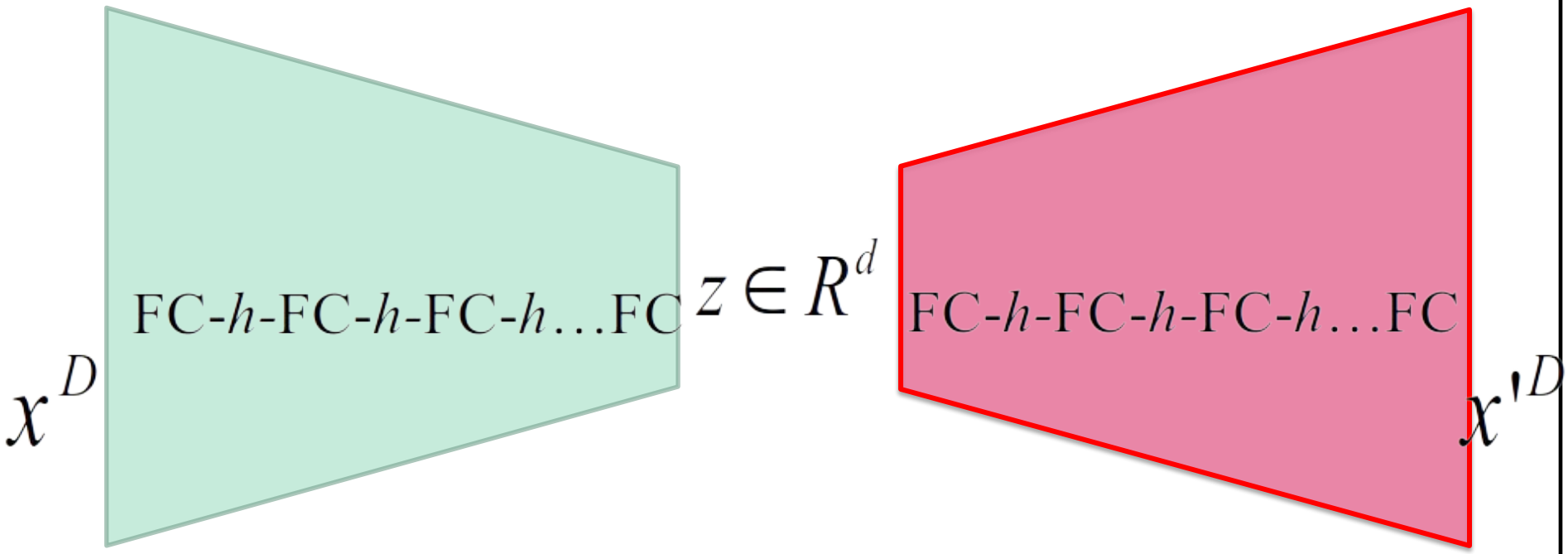
Le nombre de neurones

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

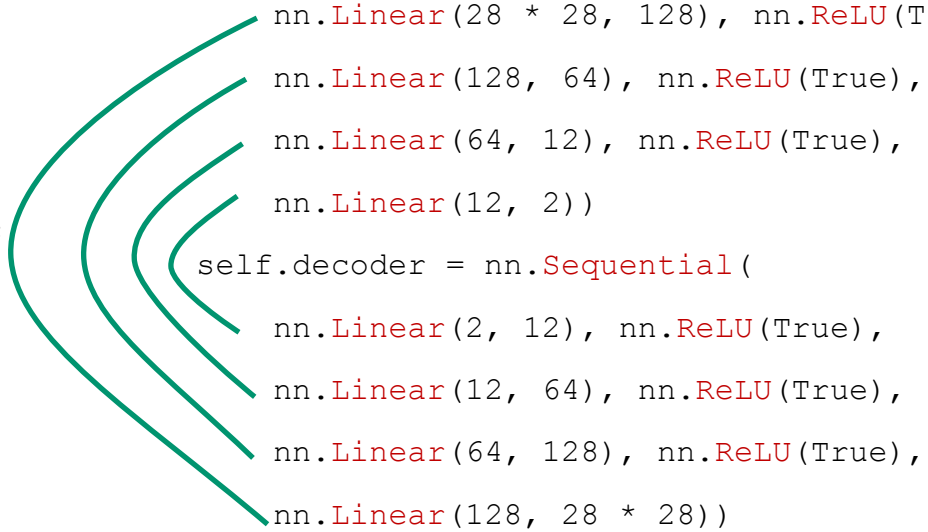
```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128), nn.ReLU(True),
            nn.Linear(128, 64), nn.ReLU(True),
            nn.Linear(64, 12), nn.ReLU(True),
            nn.Linear(12,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

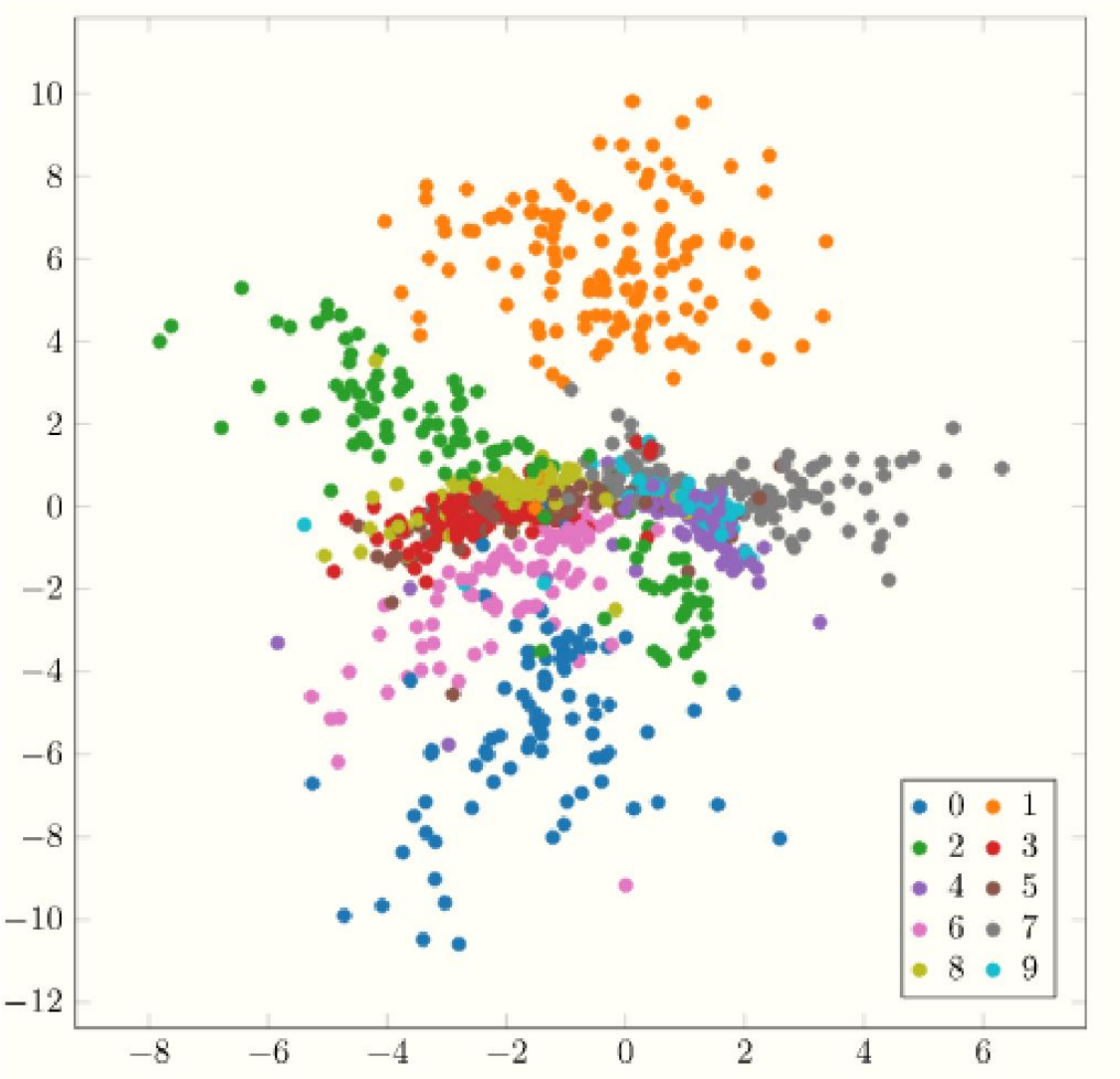
```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128), nn.ReLU(True),
            nn.Linear(128, 64), nn.ReLU(True),
            nn.Linear(64, 12), nn.ReLU(True),
            nn.Linear(12,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

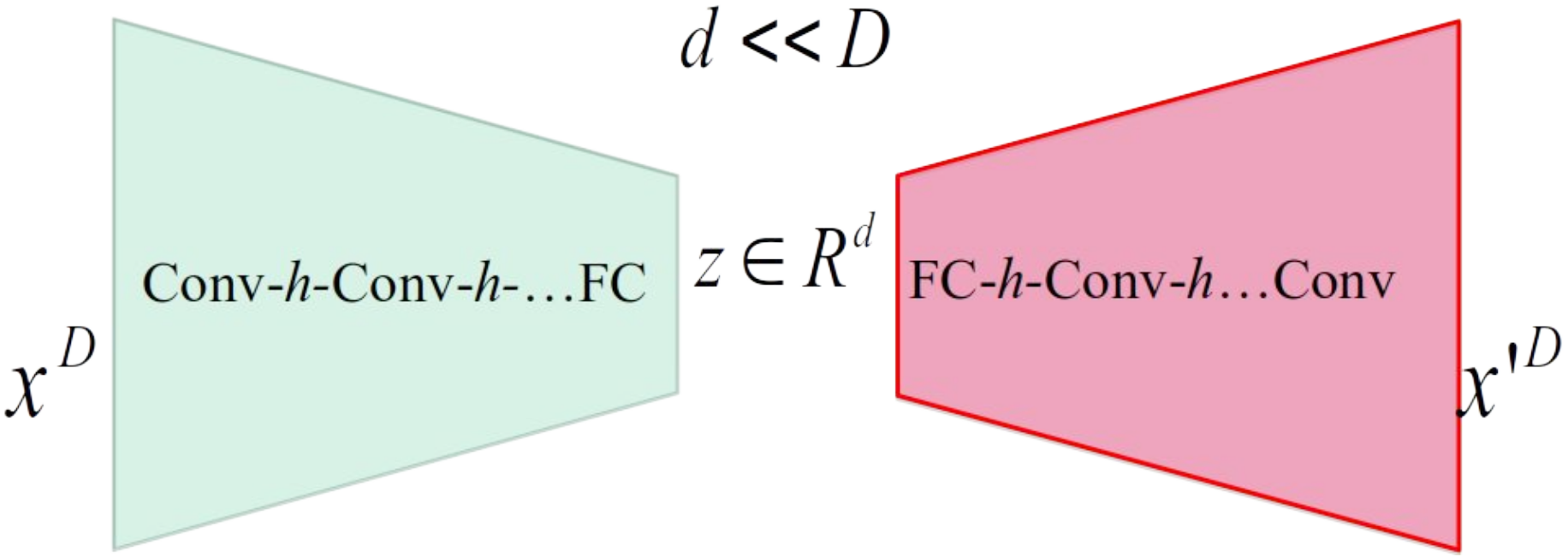


https://gertjanvandenburg.com/blog/autoencoder/

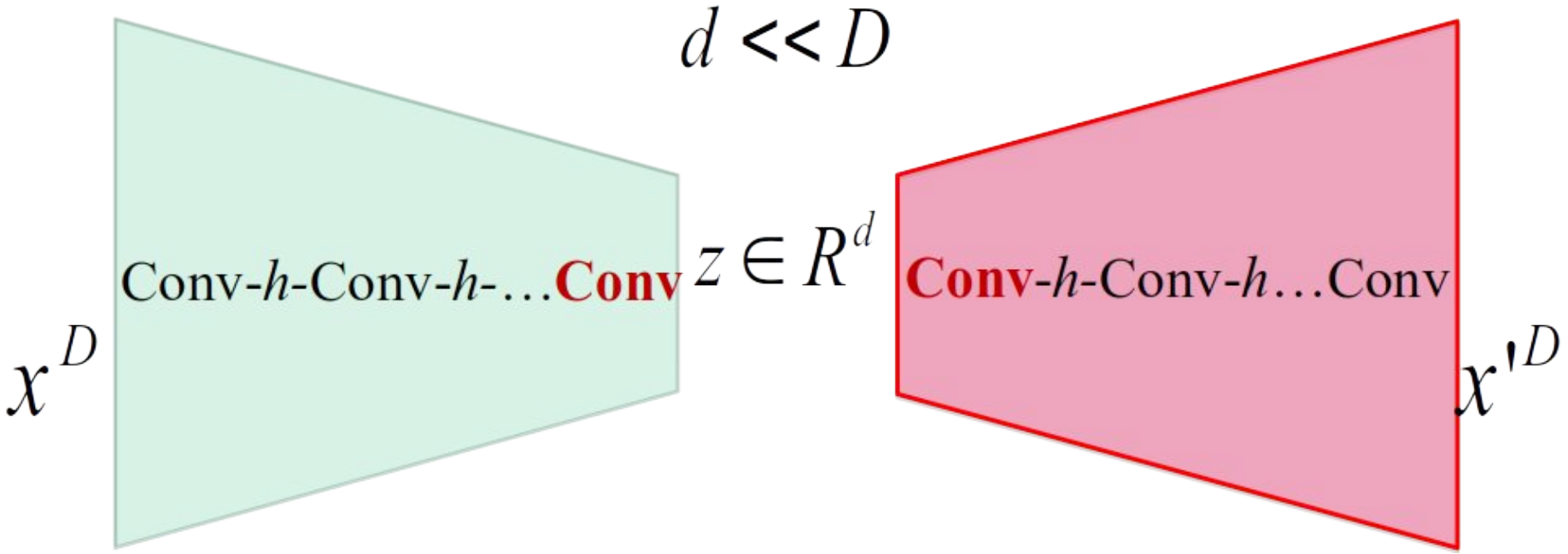
Générer des images avec le décodeur.

https://gertjanvandenburg.com/blog/autoencoder/

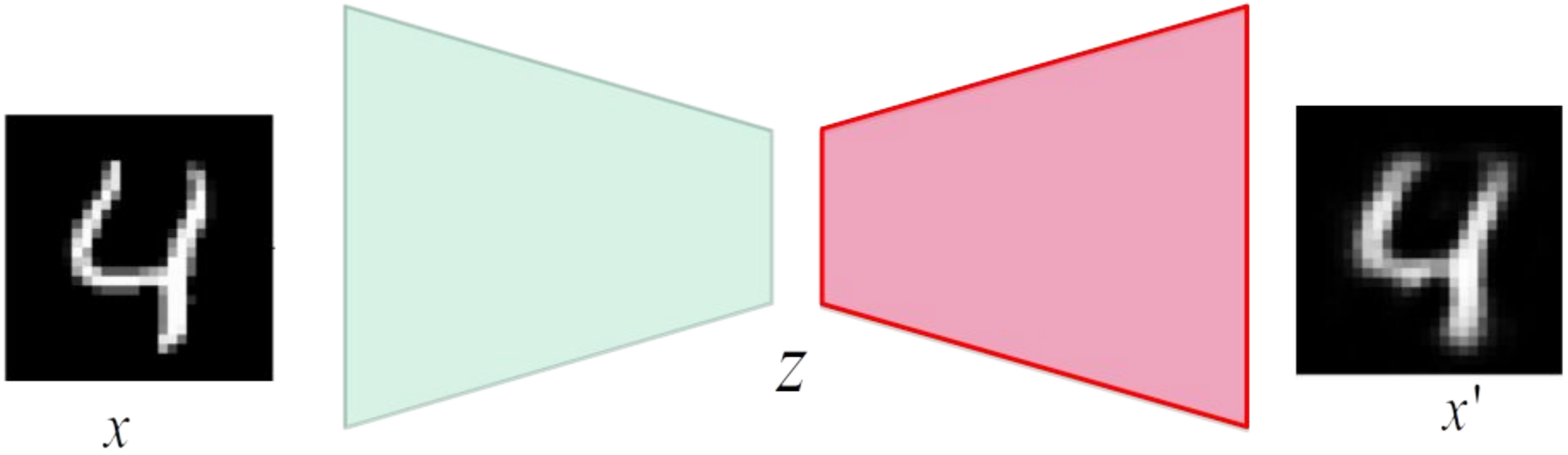
Couches convolutives



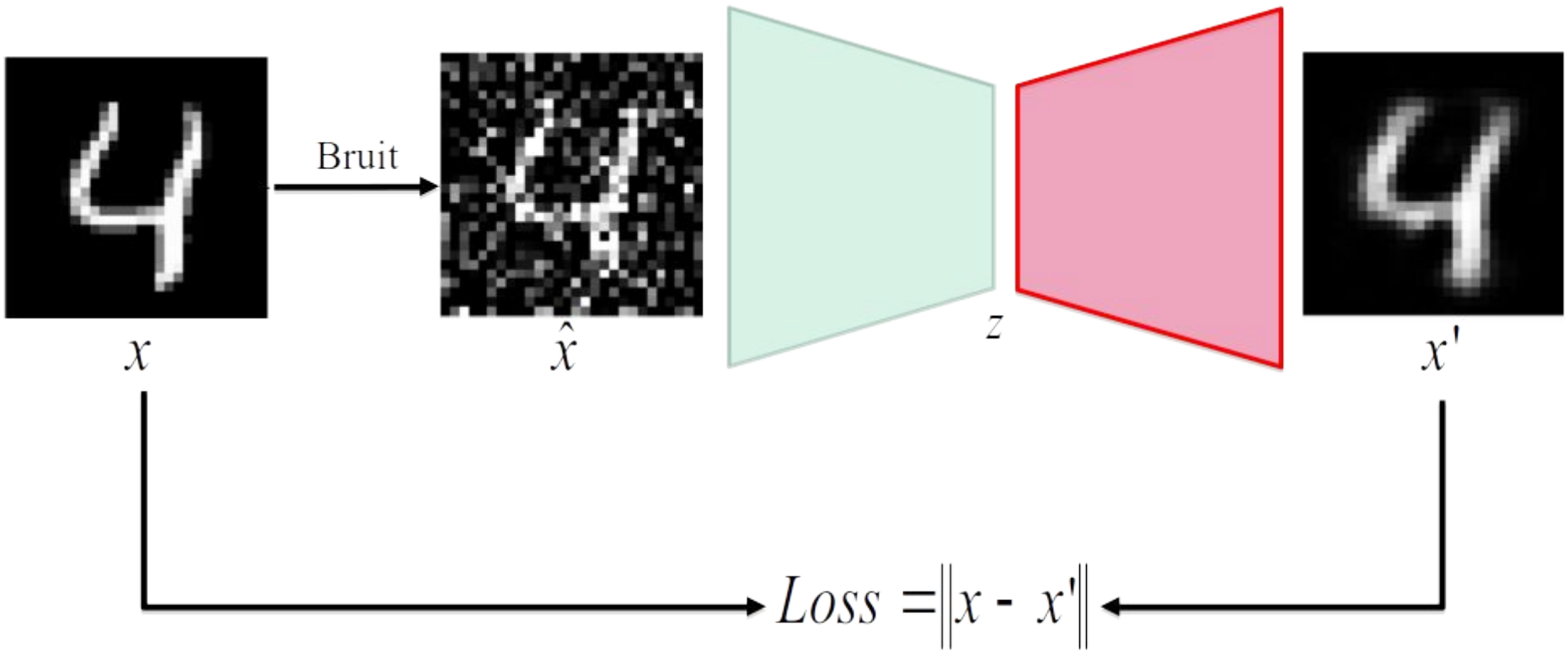
Autoencodeur pleinement convolutif



Autoencodeur de base

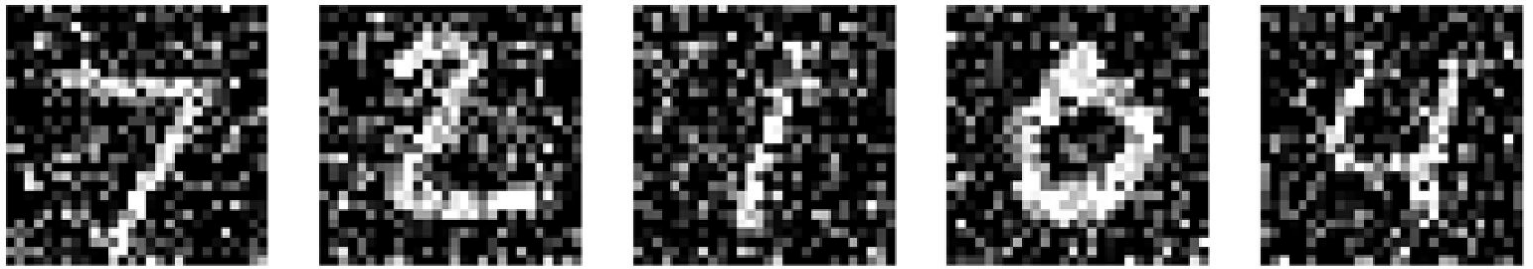


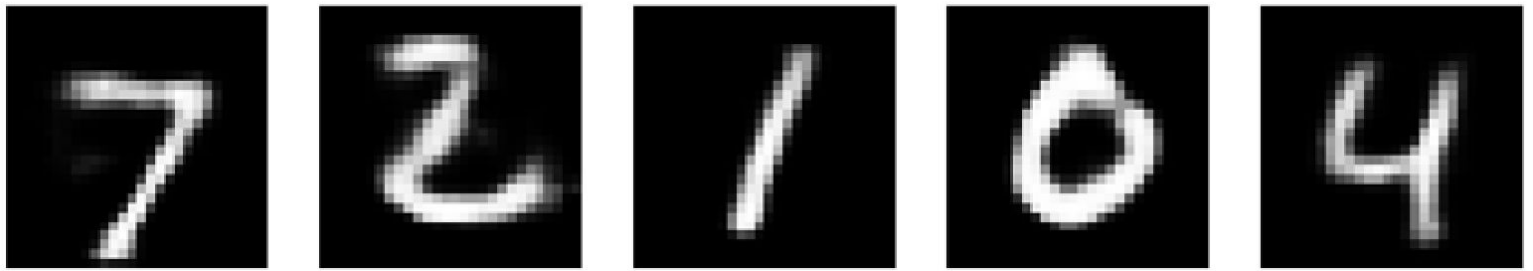
Autoencodeur pour débruitage



Une fois entraîné…

Signal bruité en entrée

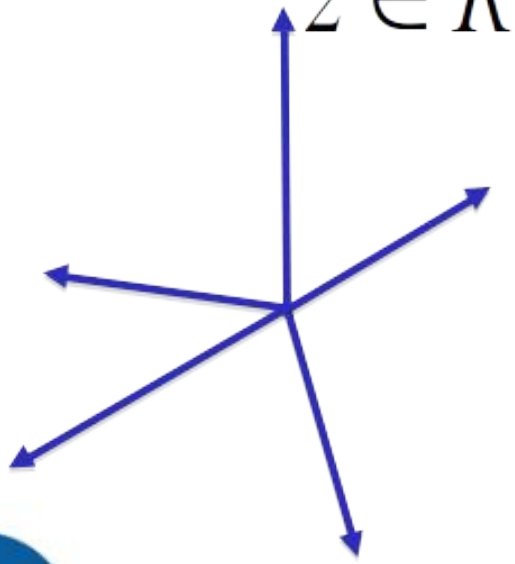


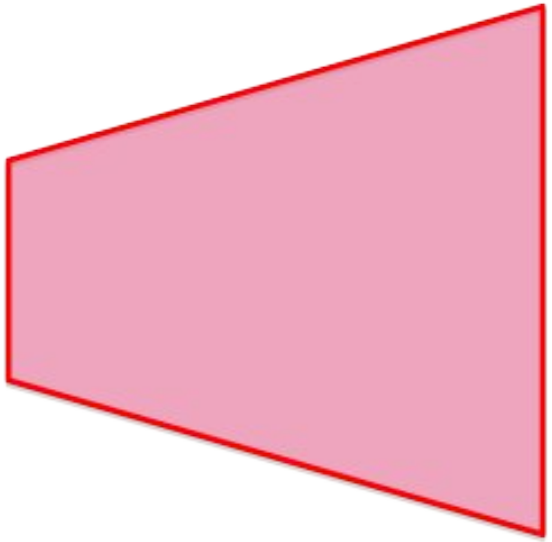


Signal reconstruit et débruité

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

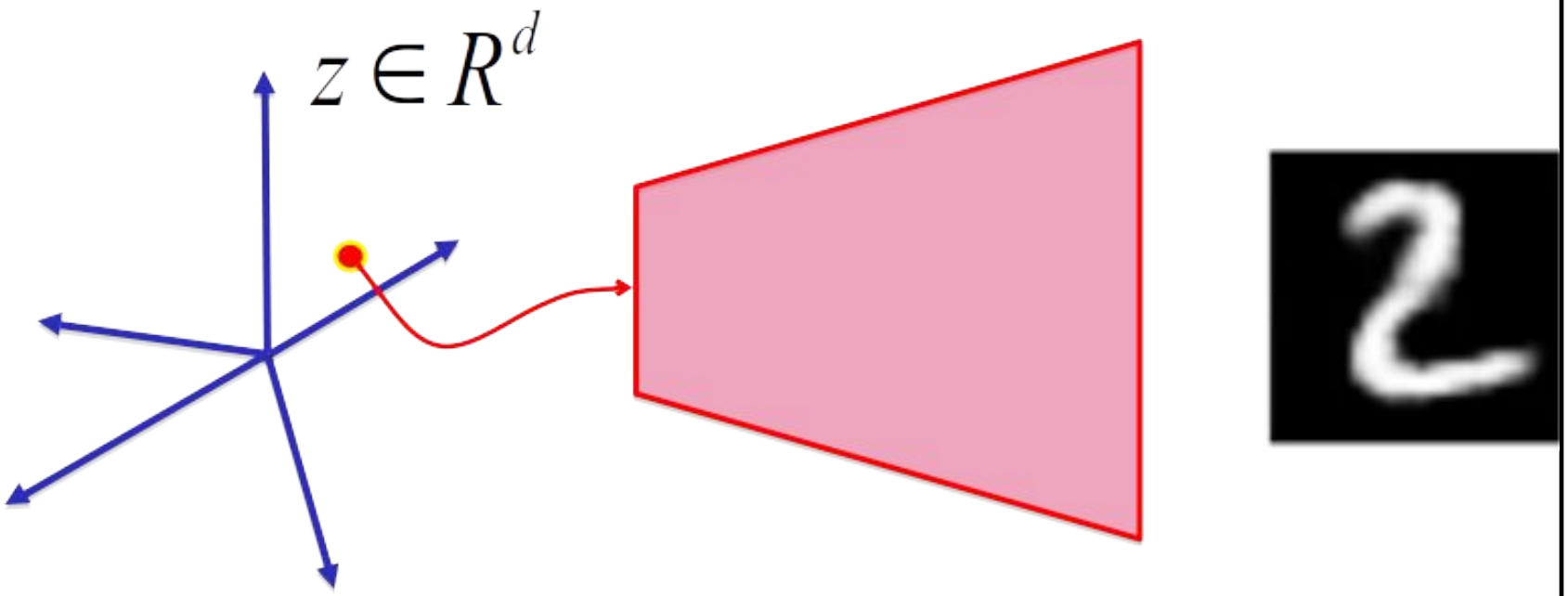
$z \in R^d$



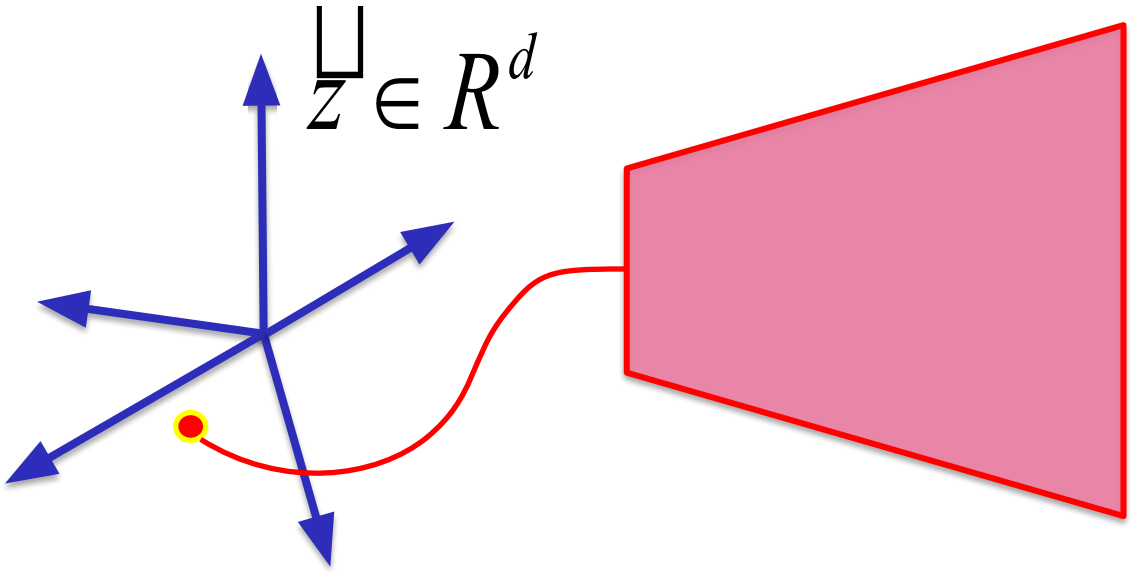


Comment, dans cet espace infini, trouver un point z qui donnera une image valable un fois décodée?

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

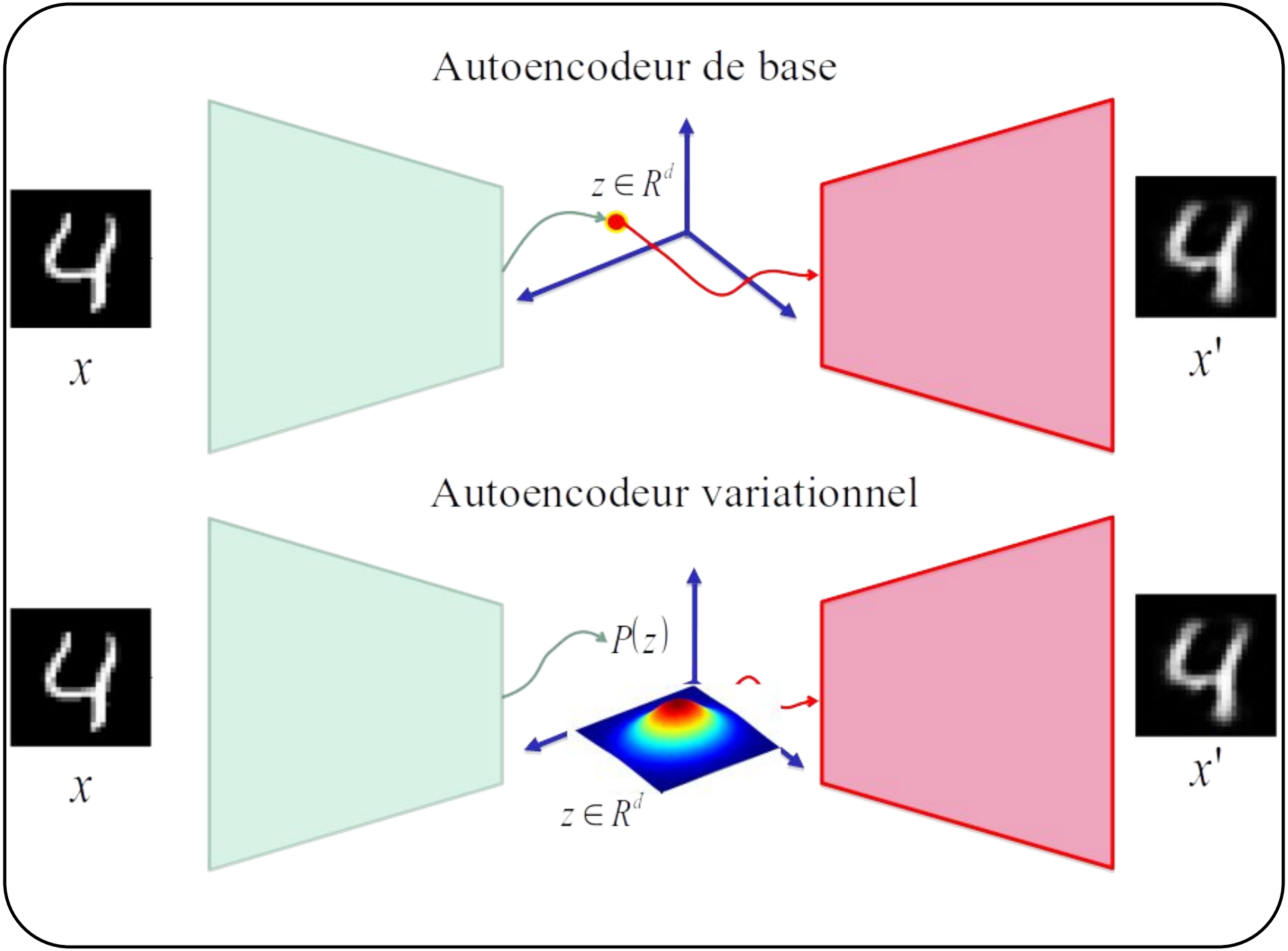


Malheureusement, la vaste majorité du temps, on reproduira du bruit

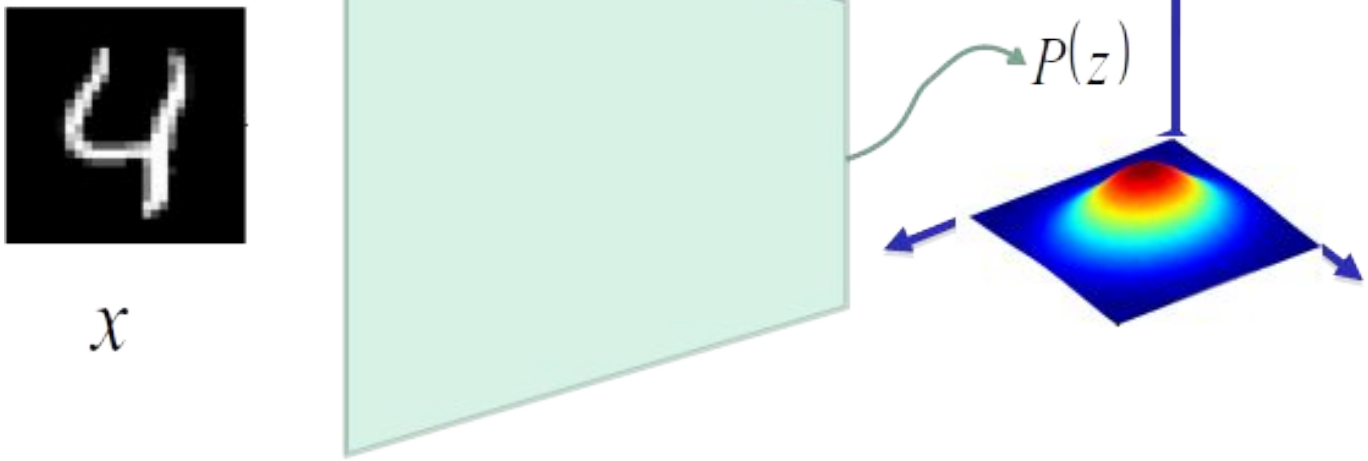


Au lieu d'apprendre à **reproduire un signal d'entrée**…

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


Autoencodeur de base
$z \in R^d$
x
x'
Autoencodeur variationnel
$P(z)$
$z \in R^d$
x
x'

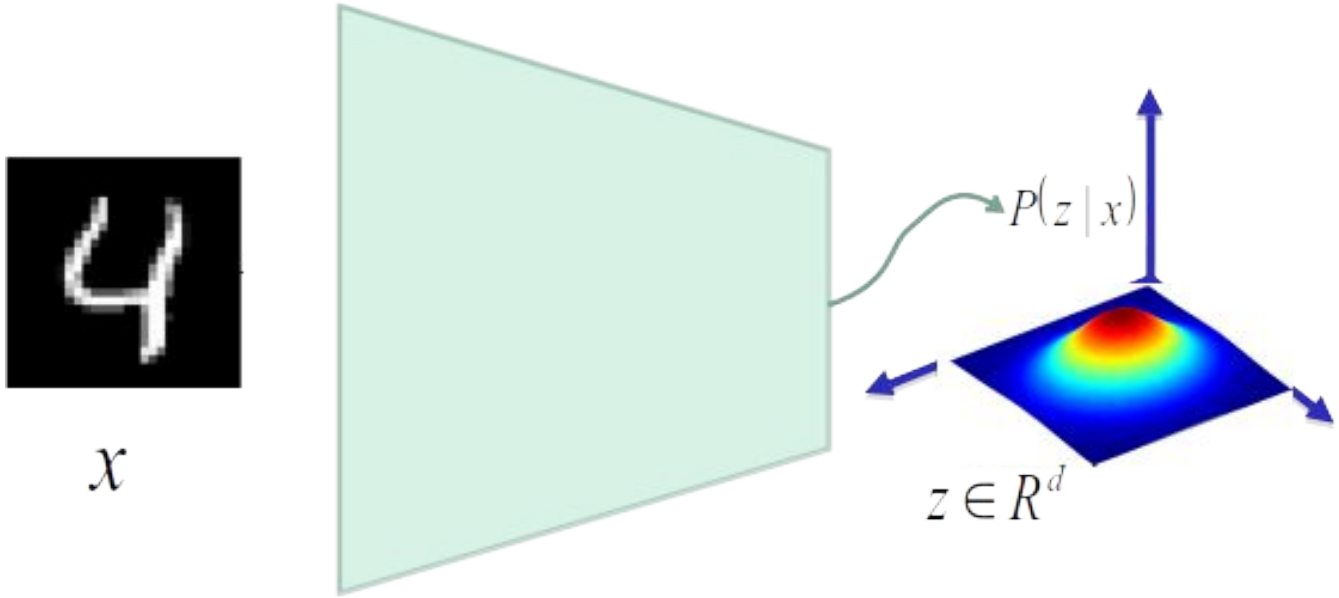
Autoencodeur variationnel



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

Autoencodeur variationnel

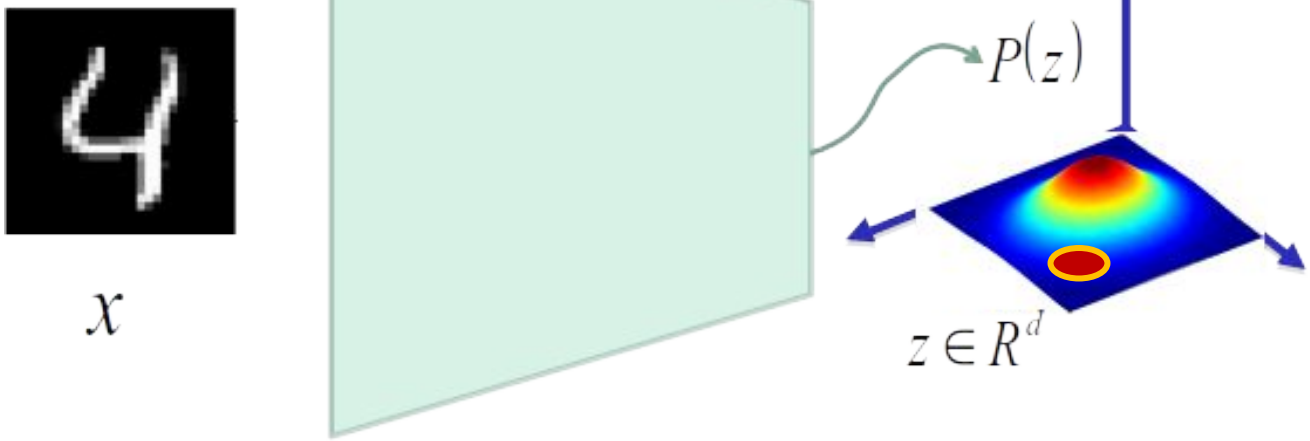
Remarque 1



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

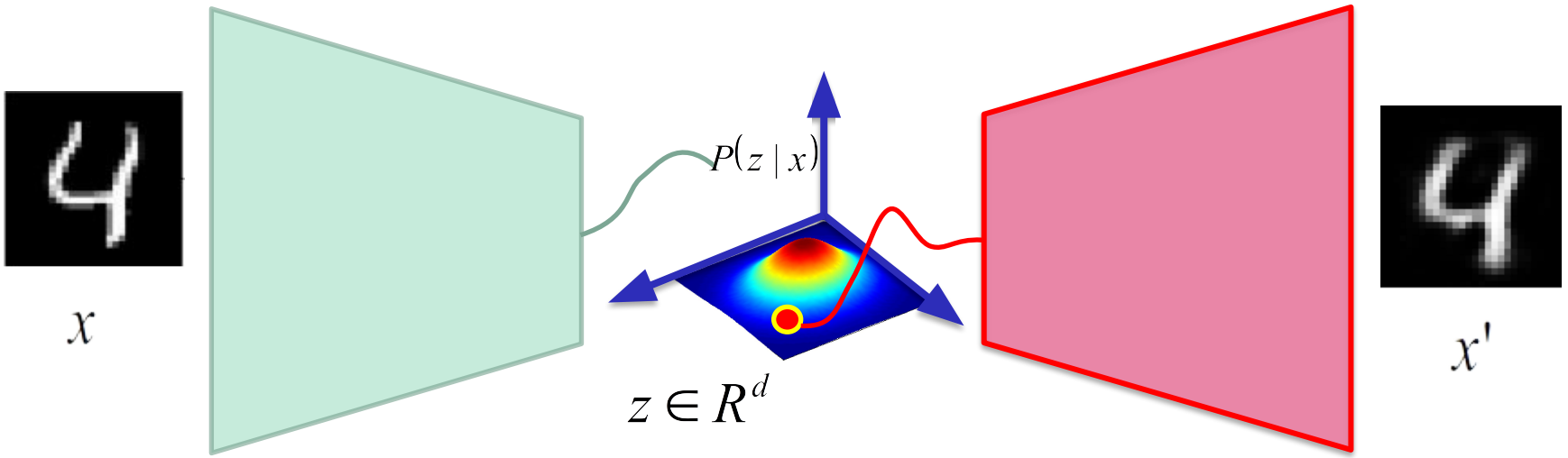
$$P(z \mid x)$$

Autoencodeur variationnel



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

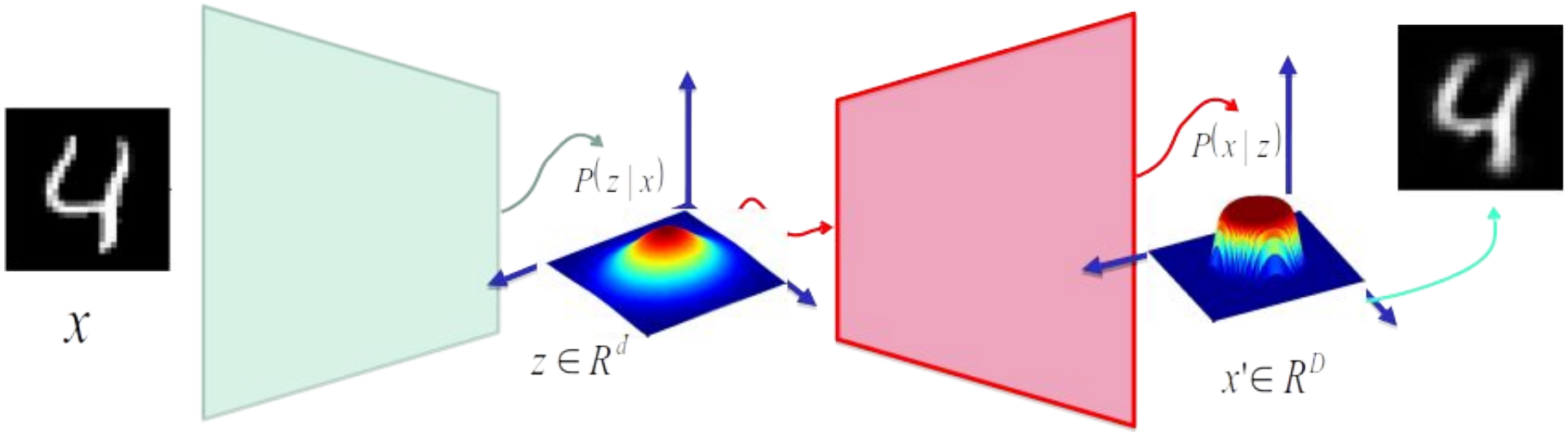
Autoencodeur variationnel



On reconstruit x'

Autoencodeur variationnel

Remarque 2



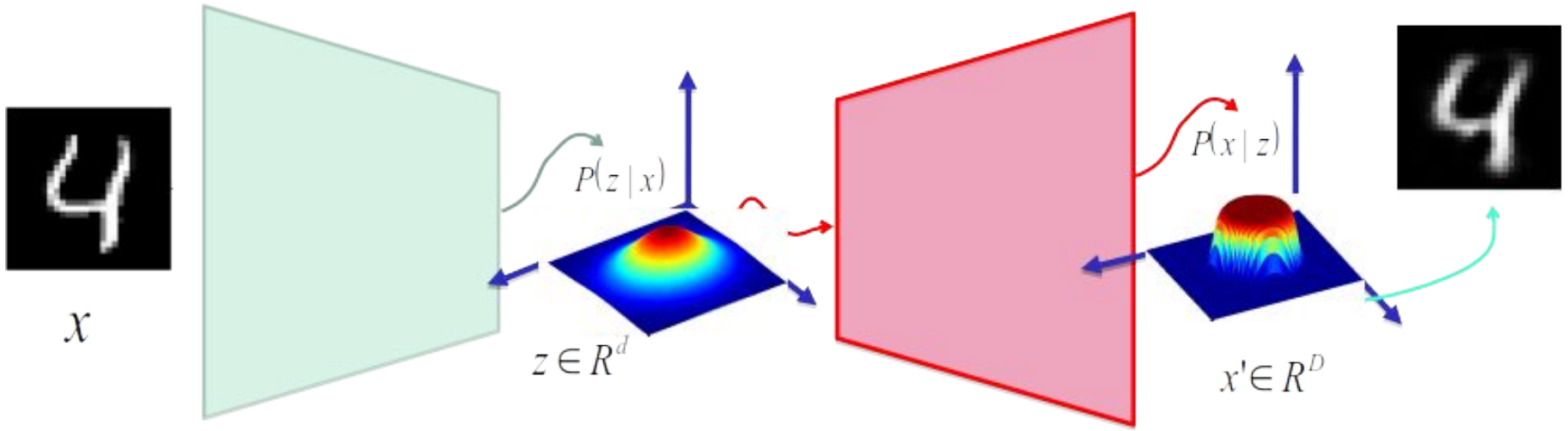
Le **décodeur** peut également produire une distribution de probabilités

$$P(x \mid z)$$

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

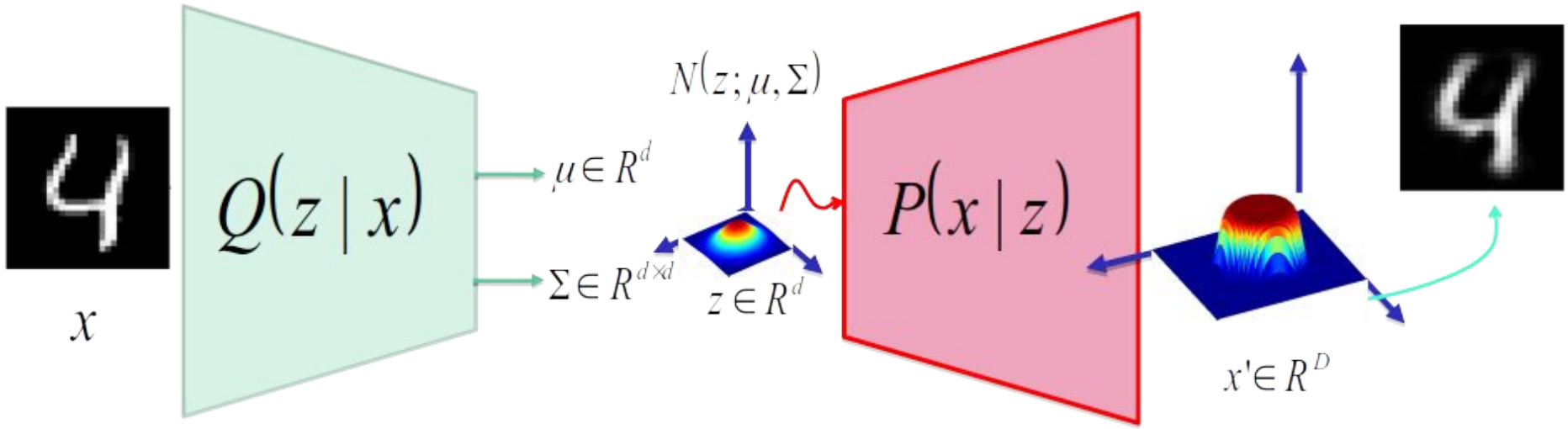
Autoencodeur variationnel

Remarque 3



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

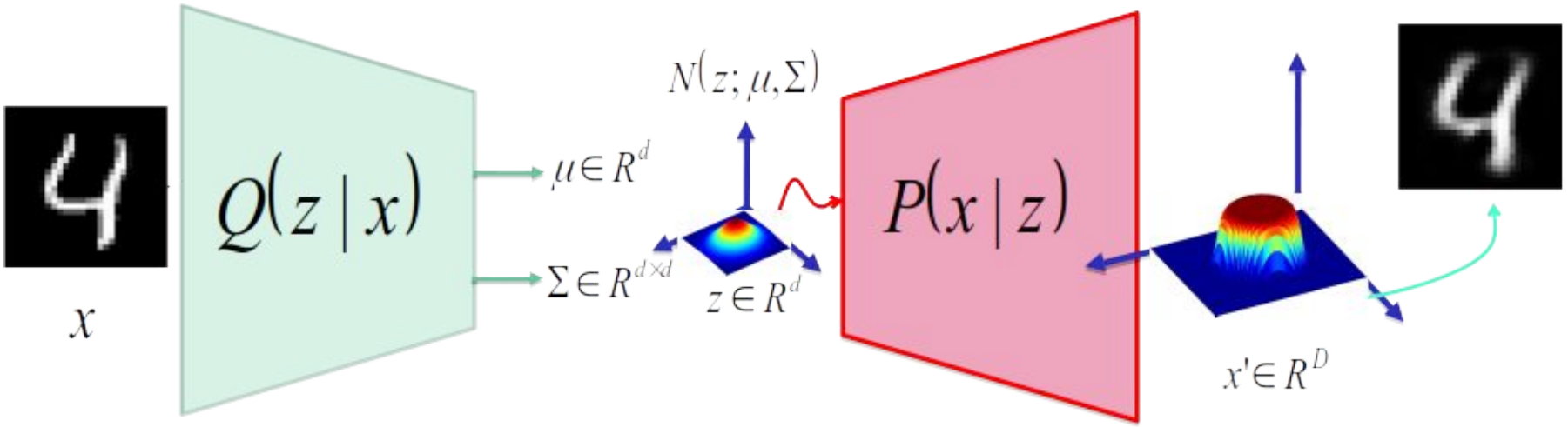
$$Q(z \mid x) \approx P(z \mid x)$$



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

Autoencodeur variationnel

Remarque 4



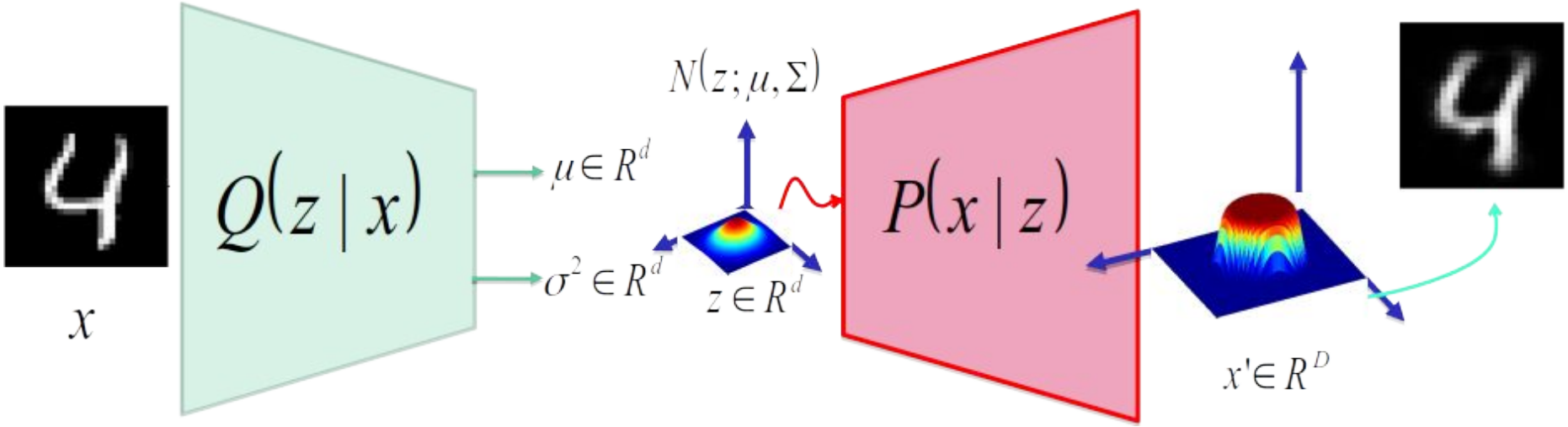
Pour simplifier les calculs, on va supposer que Σ est diagonale

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

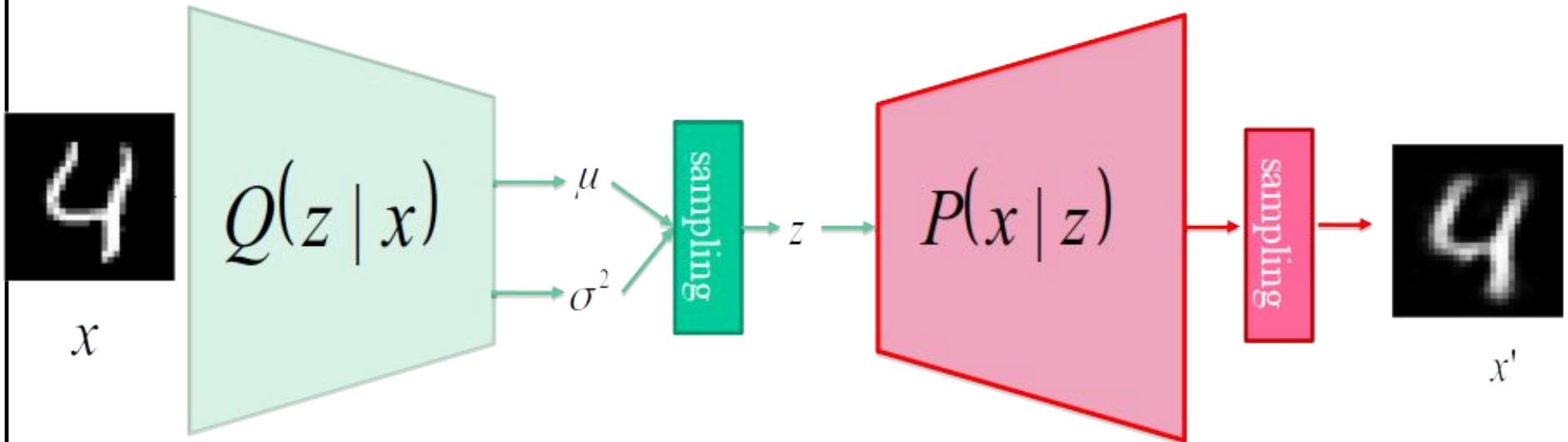
On va donc **prédire un vecteur de variances et non une matrice**

Présume que les pixels sont dépendants de z, mais indépendants entre eux

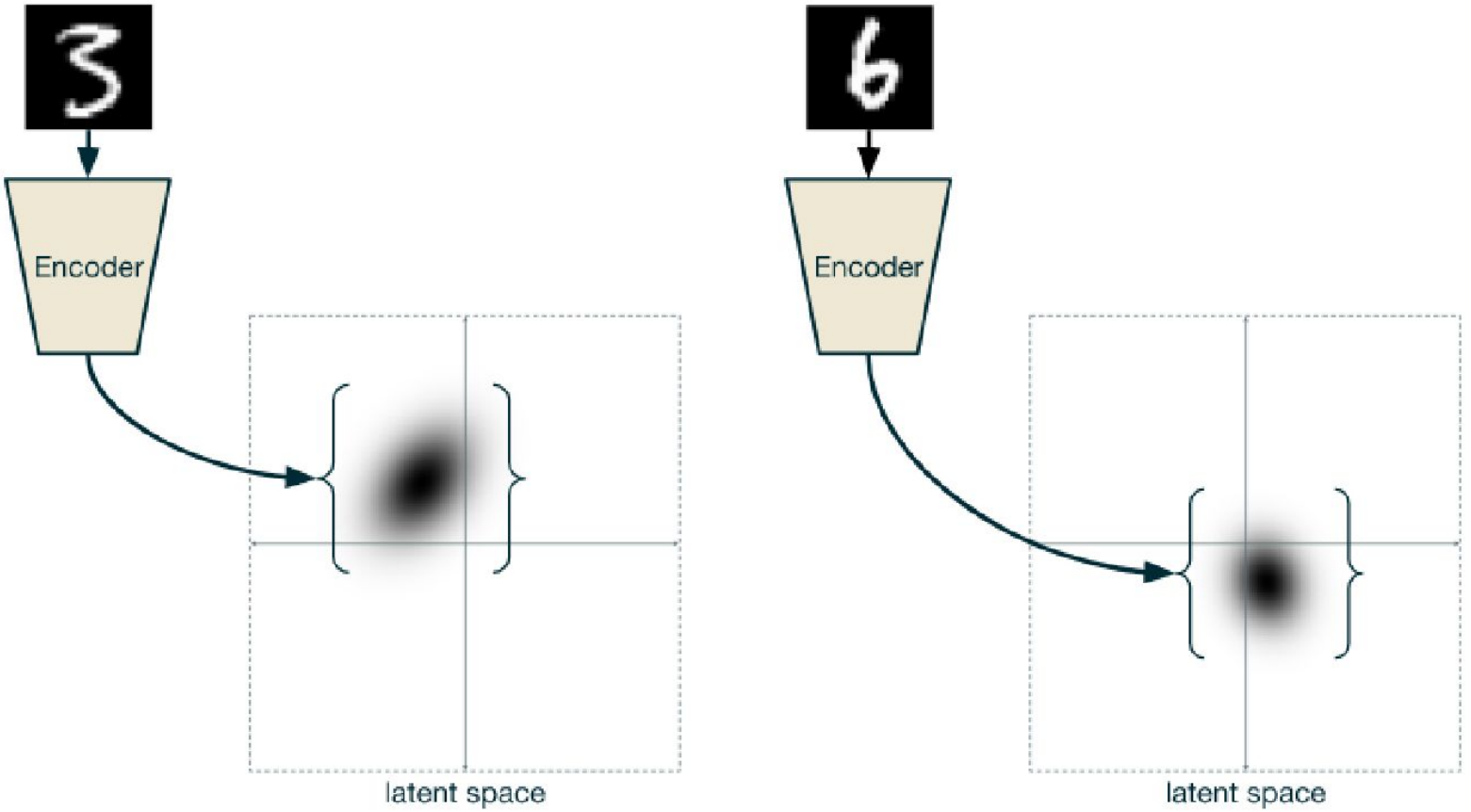
Autoencodeur variationnel



Autoencodeur variationnel

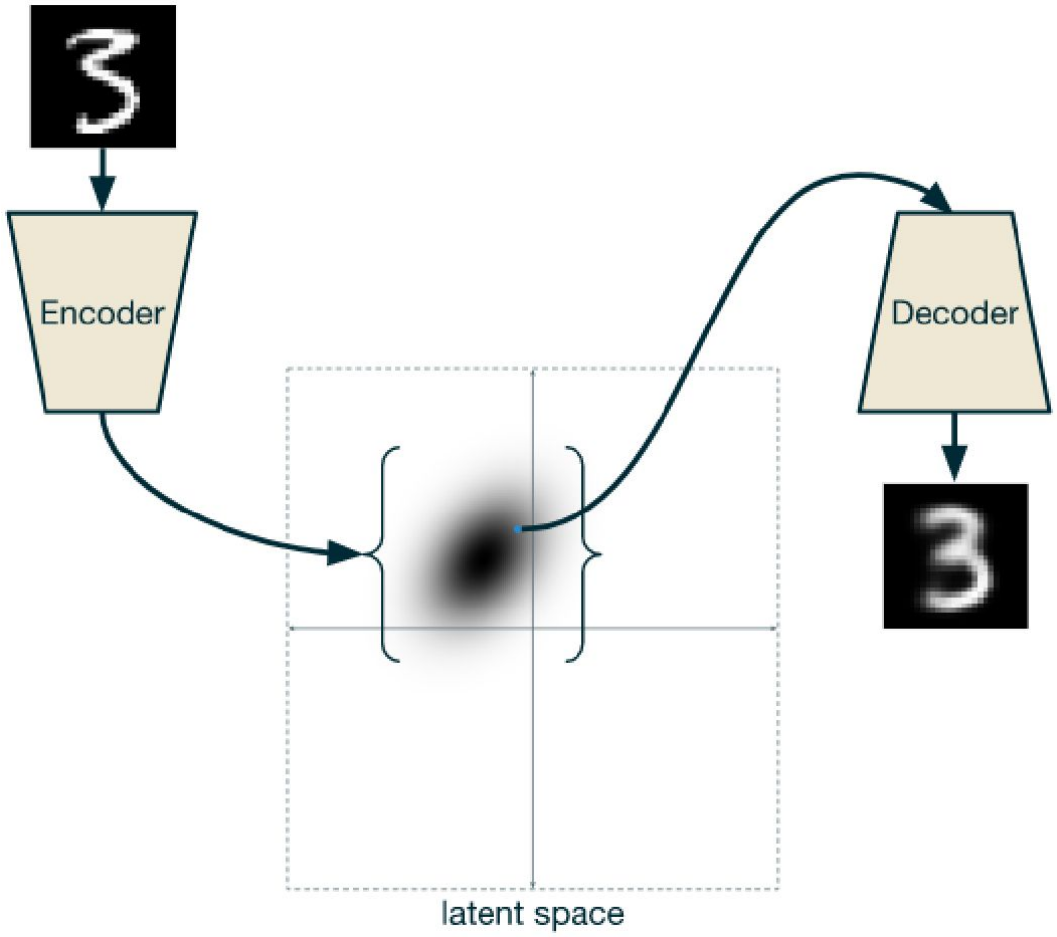


Autre façon de voir les choses…



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

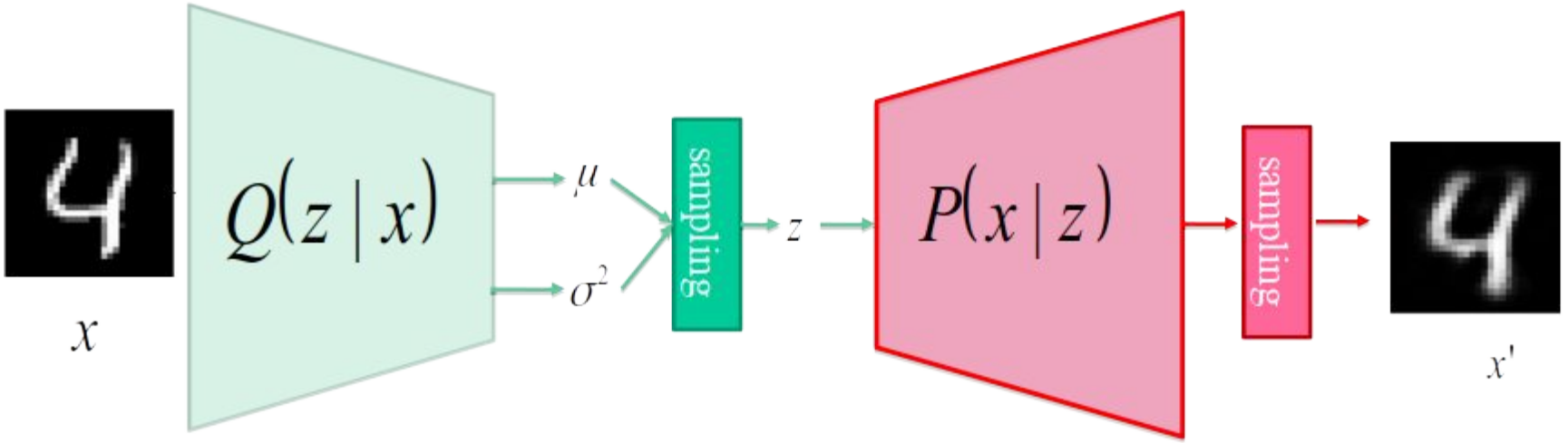
Autre façon de voir les choses…



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

Autoencodeur variationnel

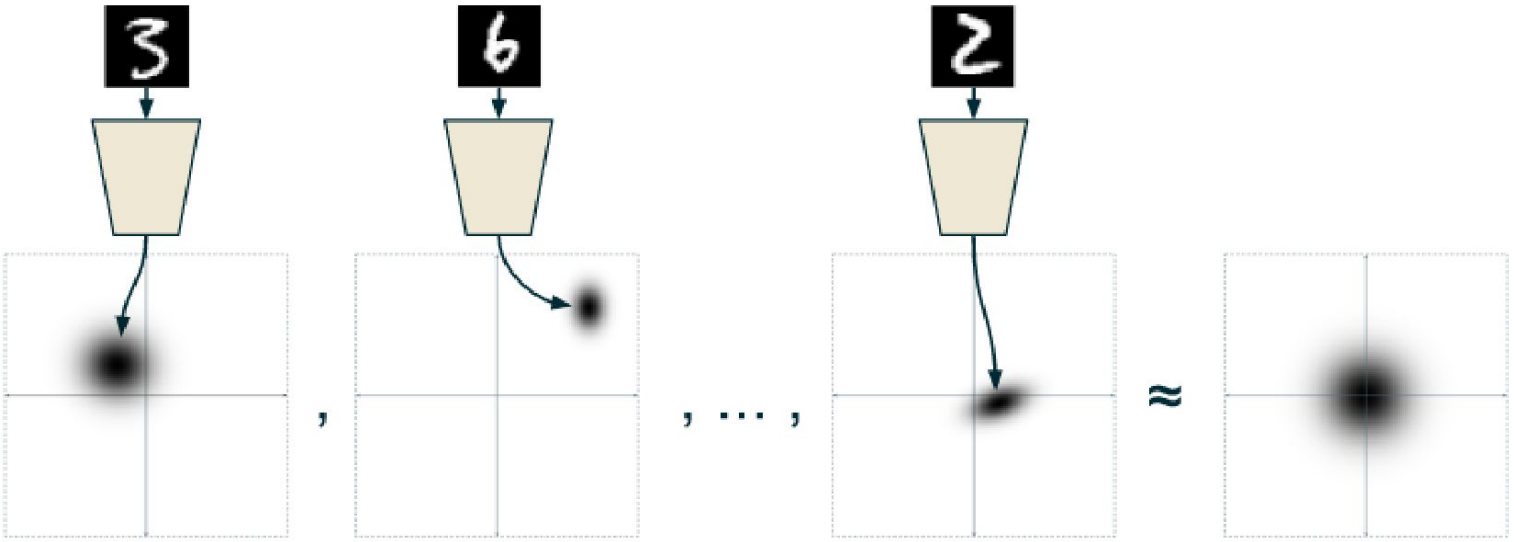
Remarque 5



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

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

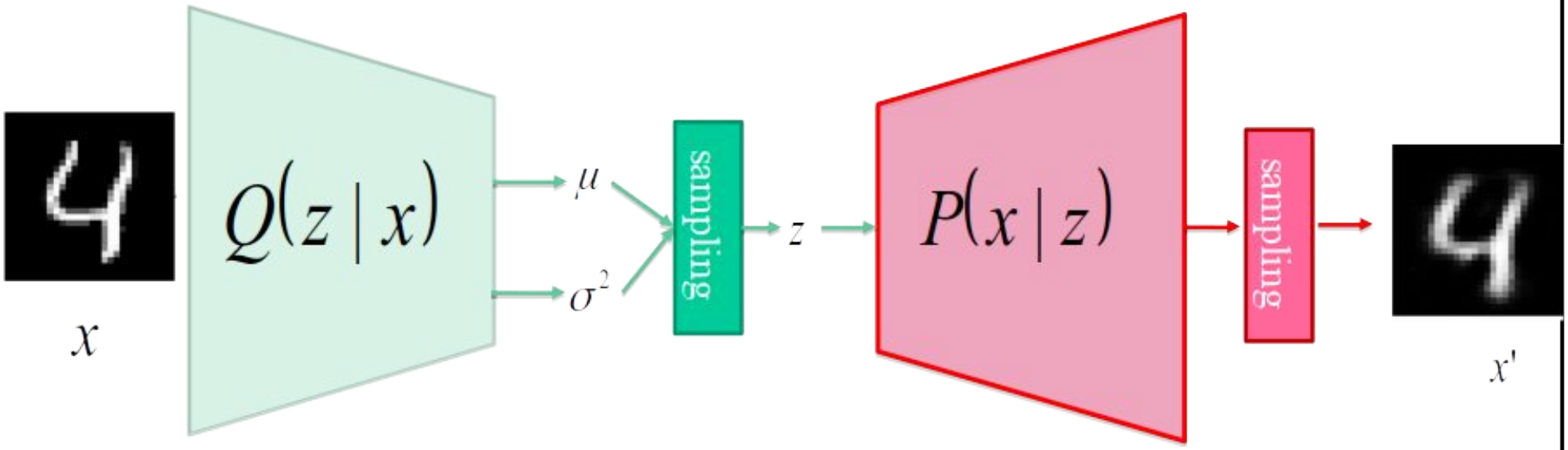
Autre façon de voir les choses…



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

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

Autoencodeur variationnel



ELBO loss : Evidence Lower Bound

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

Autoencodeur variationnel

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

ELBO loss : Evidence Lower Bound

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

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

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

Autoencodeur variationnel

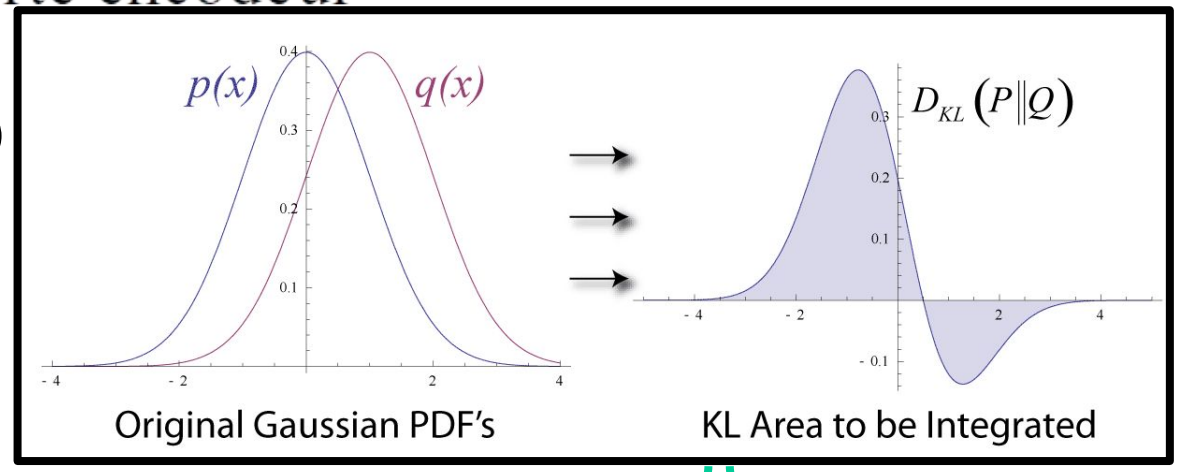
D.Kingma, M.Welling, **Auto-Encoding Variational Bayes,** arXiv:1312.6114v10 (***Annexe B***)

ELBO loss : Evidence Lower Bound

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

Si on suppose que P(x|z)

$$Loss = \underbrace{\frac{1}{2}\sum_{i=1}^{d}(1 + \quad 2}_{\text{Perte encodeur}} \qquad \underbrace{\quad}_{\text{Perte décodeur}}$$



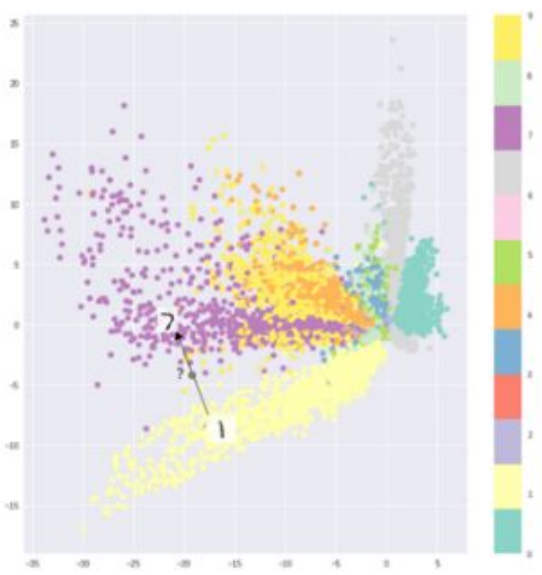
Autoencodeur variationnel

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

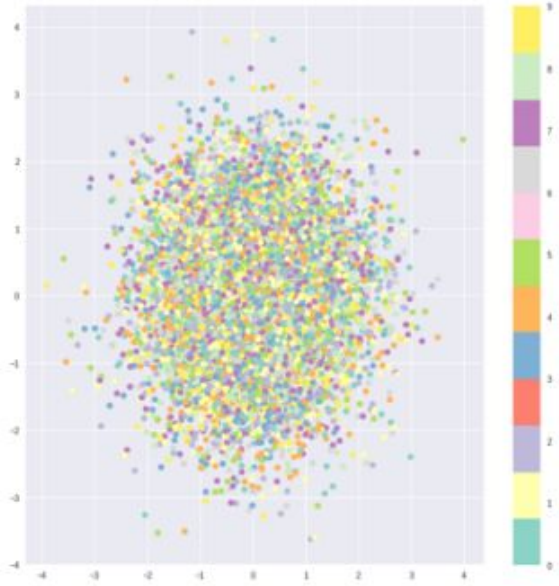
Evidence Lower Bound

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

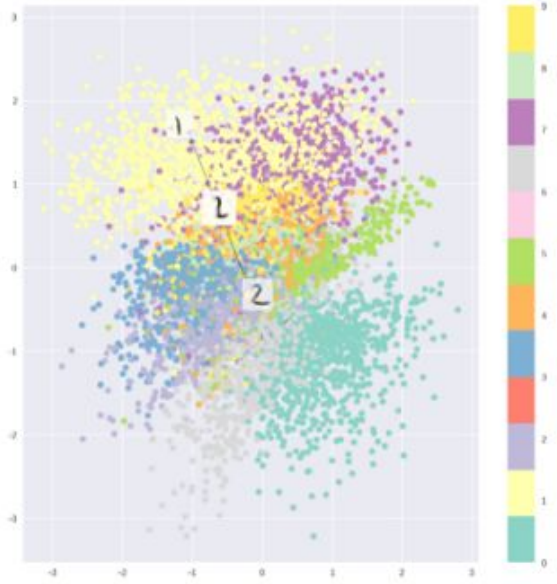
Only reconstruction loss



Only KL divergence



Combination



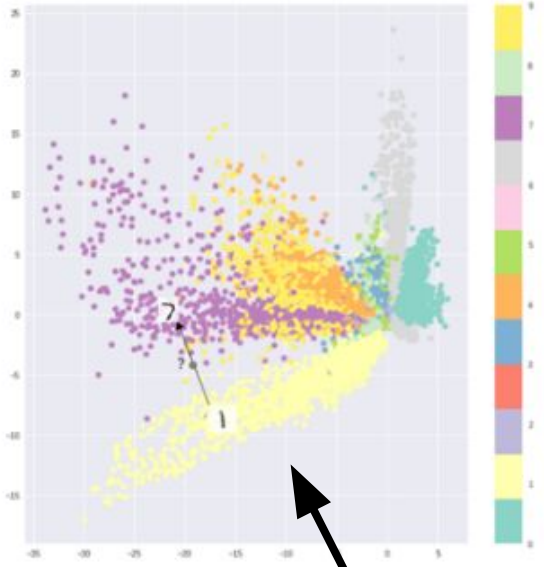
https://www.jeremyjordan.me/variational-autoencoders/

Autoencodeur variationnel

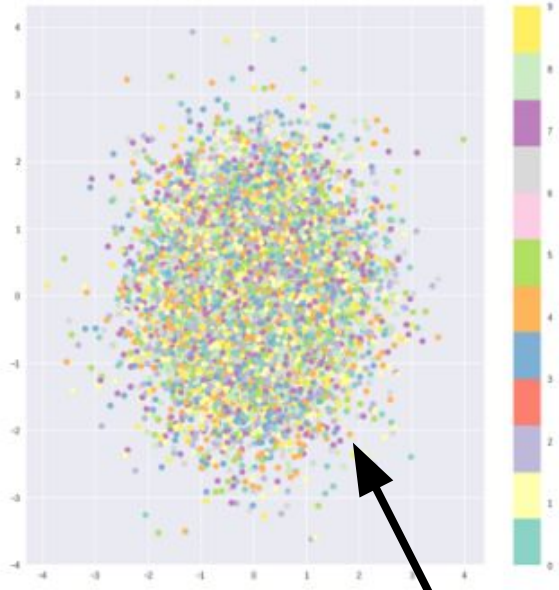
D.Kingma, M.Welling, **Auto-Encoding Variational Bayes,** arXiv:1312.6114v10 (***Annexe B***)

ELBO loss : Evidence Lower Bound

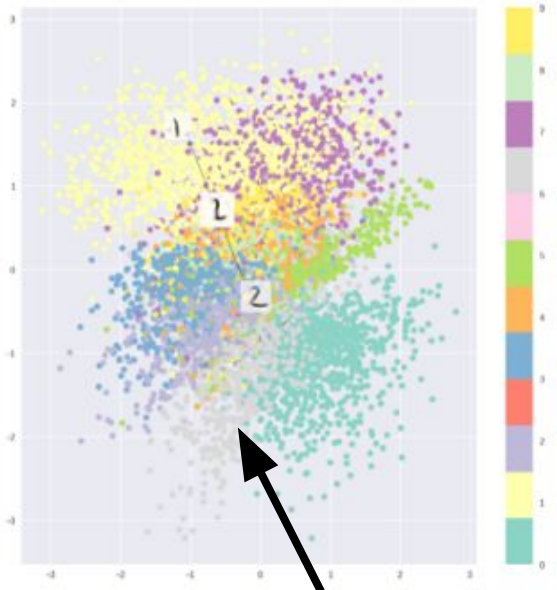
Only reconstruction loss



Only KL divergence



Combination



https://www.jeremyjordan.me/variational-autoencoders/

Pas gaussien, difficile à échantillonner

Trop gaussien, tout est indistinguable

Ensemble de distributions séparées qui forment une gaussienne

Autoencodeur variationnel

D.Kingma, M.Welling, **Auto-Encoding Variational Bayes,** ar

ELBO loss : Evidence Lower Bound

À voir plus tard

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

Perte encodeur — Perte décodeur

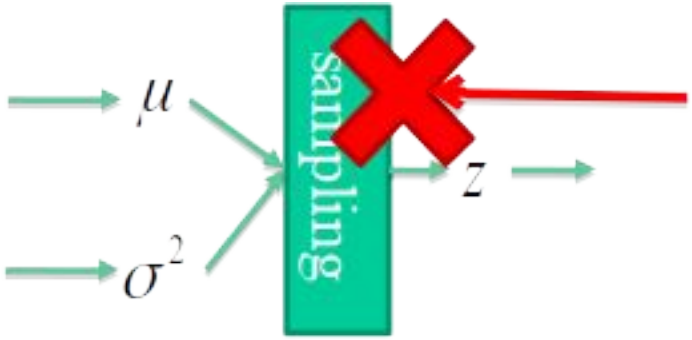
Si on suppose que P(x|z) est gaussien

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

Perte encodeur — Perte décodeur

Autoencodeur variationnel

Remarque 6

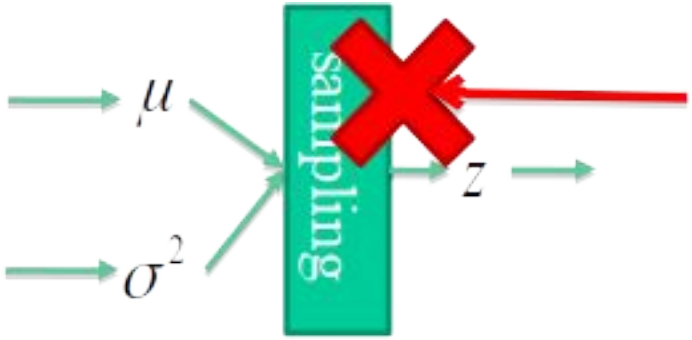


Pas de rétro-propagation à travers un processus d'échantionnage

$$z \sim N(z; \mu, \Sigma)$$

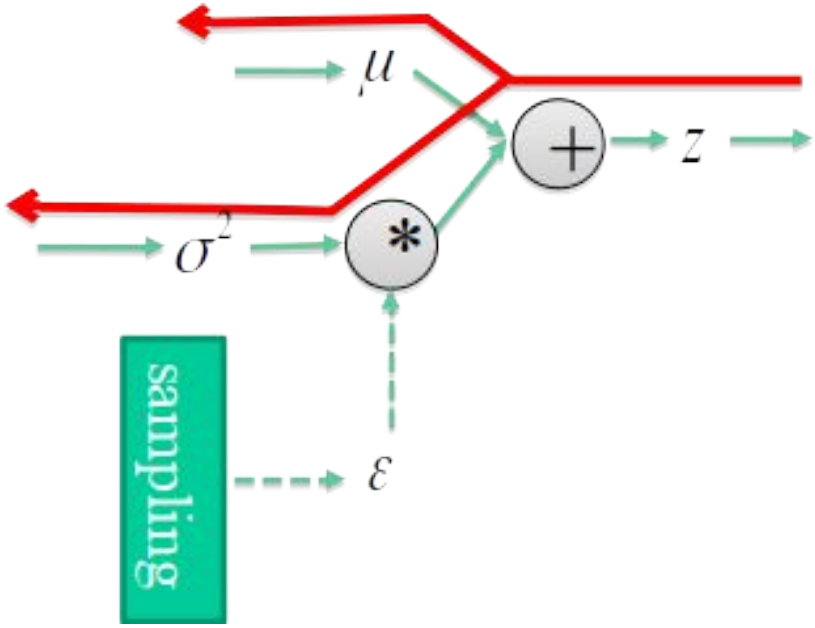
Autoencodeur variationnel

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 = enc_x[:, 32:]
        z = self.reparameterize(mu, logvar)
        return self.decoder(z), mu, logvar
```

Reparameterization trick

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) # Espace latent de 32 dimensions
        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 = enc_x[:, 32:]
        z = self.reparameterize(mu, logvar)
        return self.decoder(z), mu, logvar
```

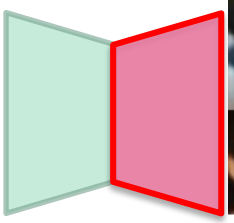
Reparameterization trick

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

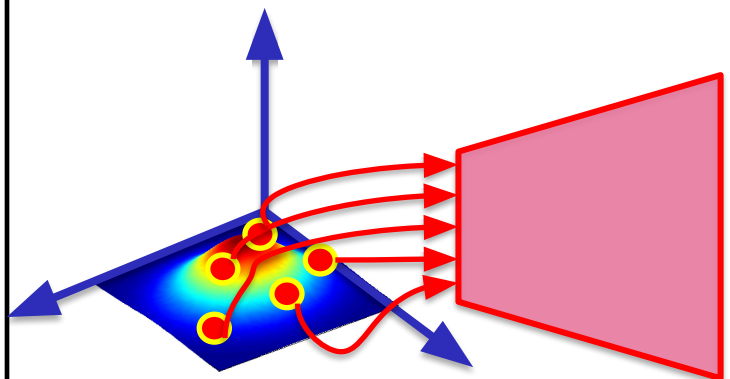


x'

https://github.com/yzwxx/vae-celebA

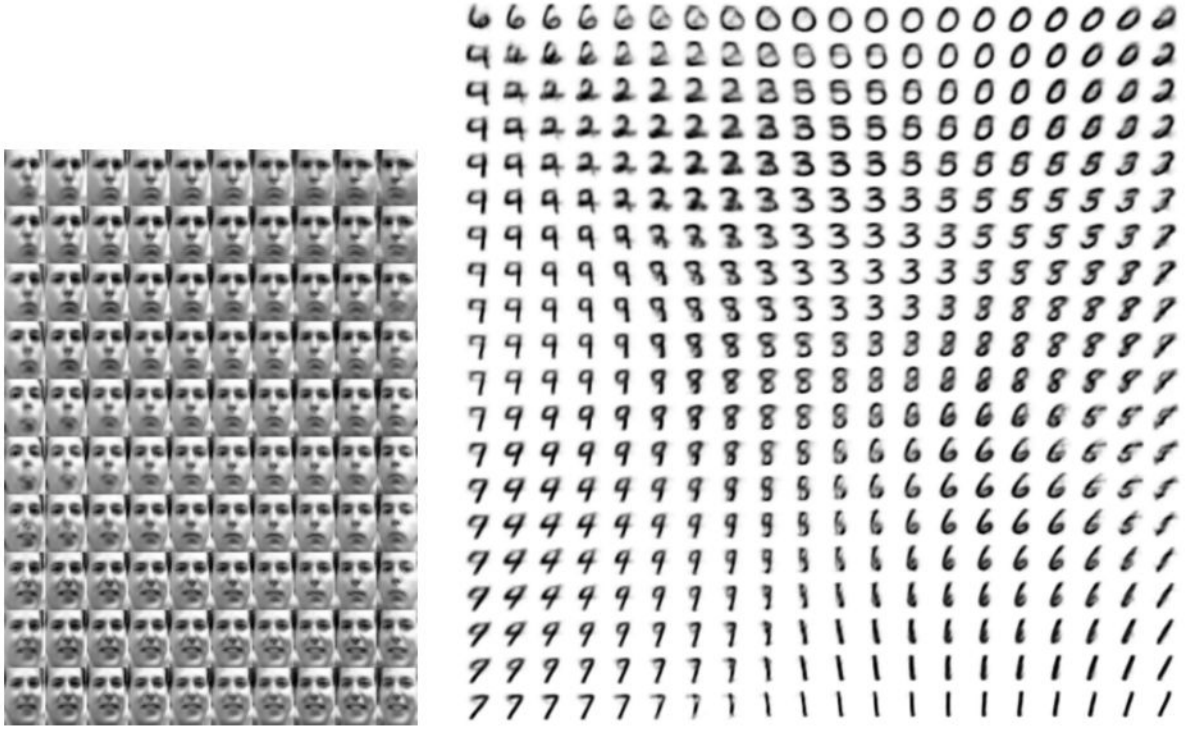
Ex.: base de données *CelebA*

Décodage d'échantillons aléatoires z



https://github.com/yzwxx/vae-celebA

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 $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ with the learned parameters $\boldsymbol{\theta}$.

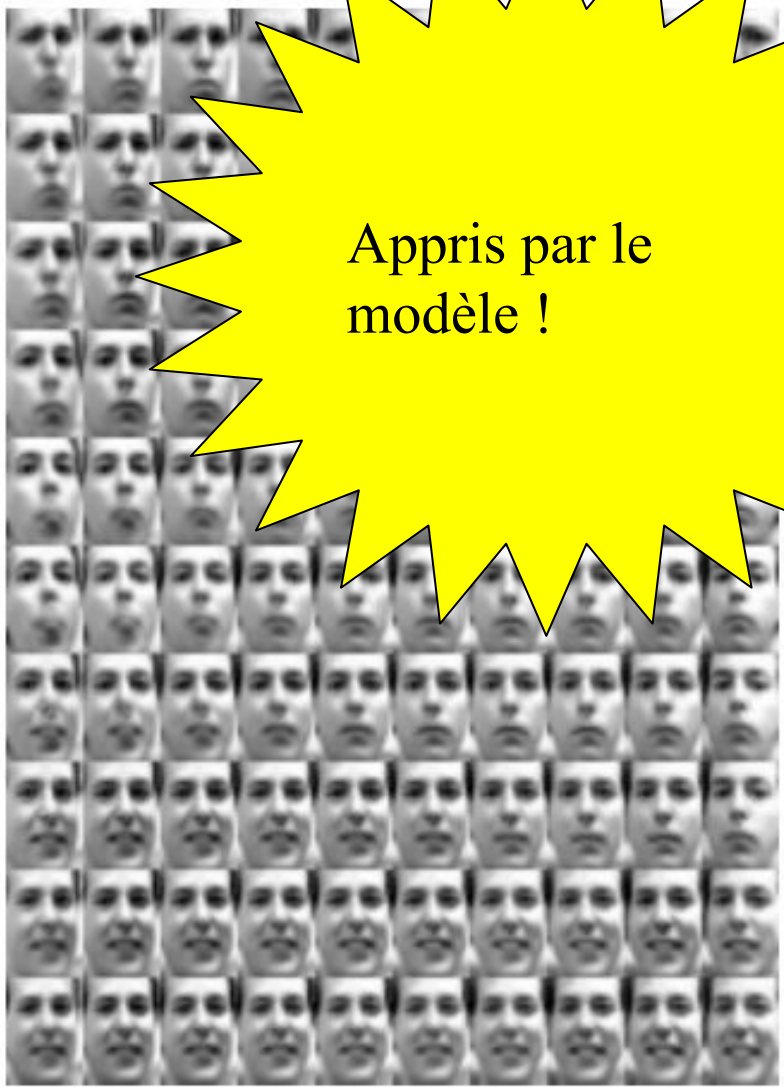
Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Ex.: VAE

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

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

Ex.: VAE



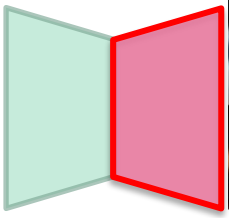
Dimension du “sourire”
Dimension de l’orientation de la tête

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

[illegible] données *CelebA*

Images floues.
Pourquoi ?

x'



https://github.com/yzwxx/vae-celebA

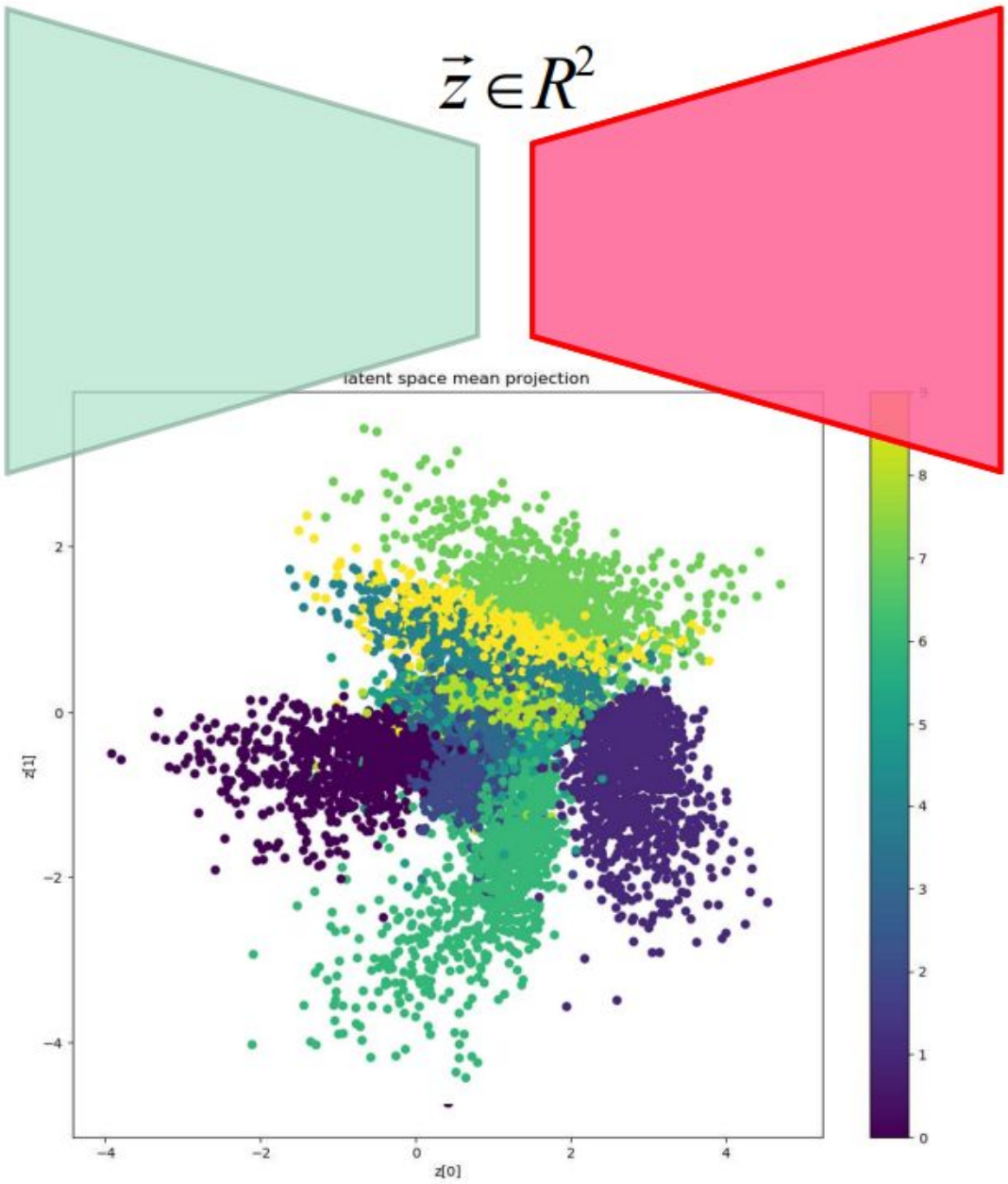
NOTE: on peut également conditionner un espace latent en y « **injectant les gradients d'un 2e réseau de neurones** »

Autoencodeur de base

$\vec{z} \in R^2$

$\vec{x}$

$\vec{x'}$

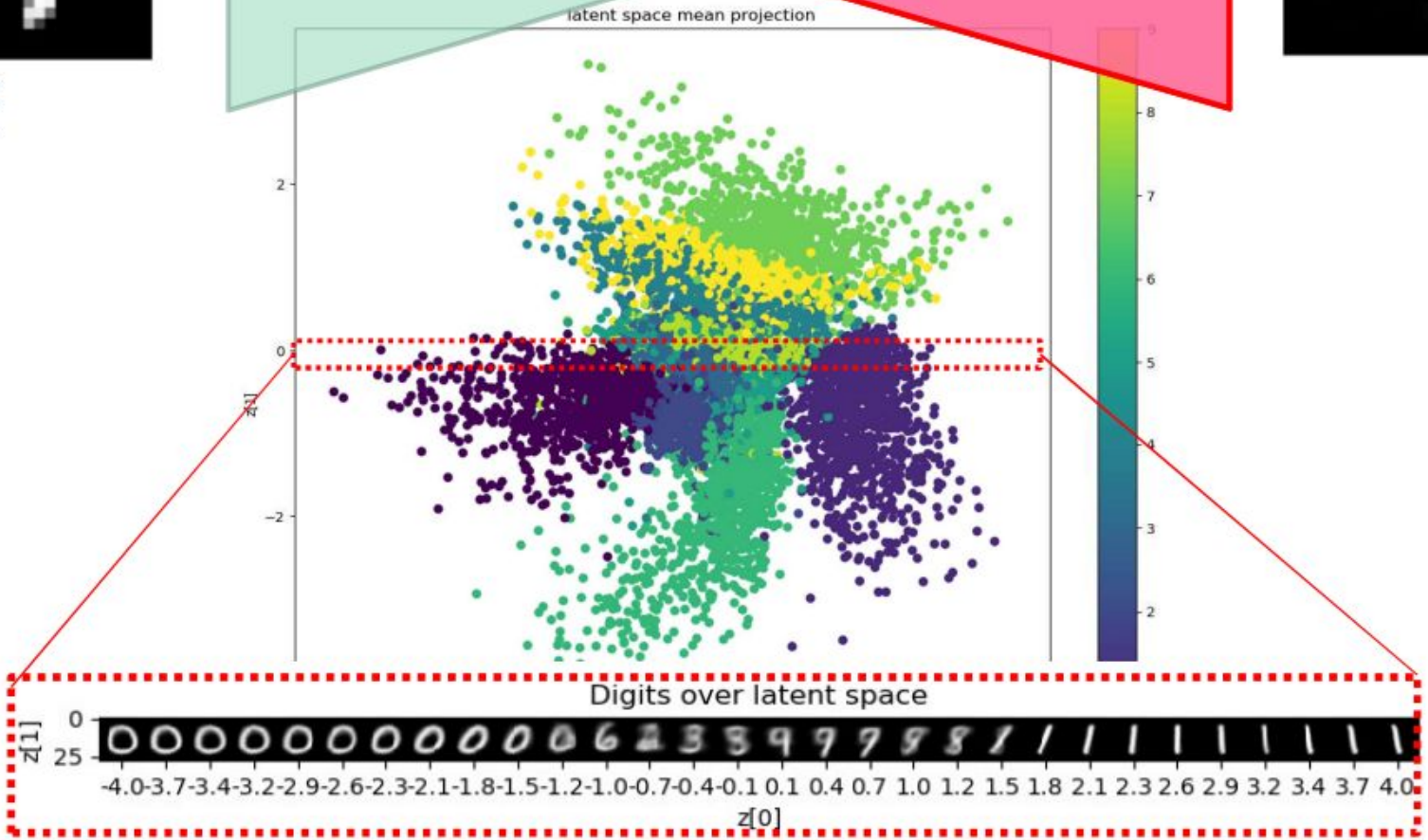


Autoencodeur de base

$\vec{z} \in R^2$

$\vec{x}$

$\vec{x'}$

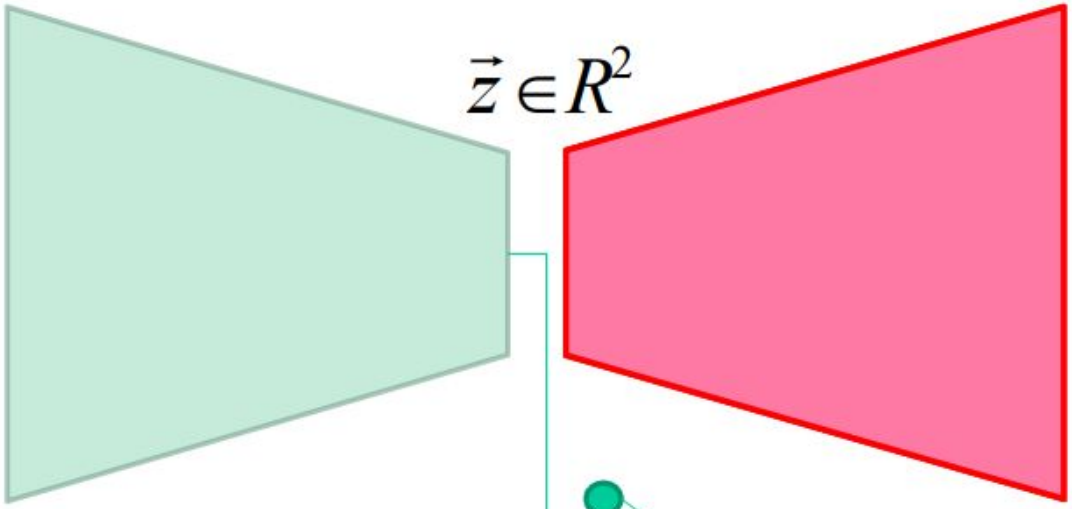


Autoencodeur contraint

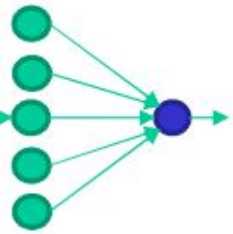
$\vec{x}$

$t = 4$

$\vec{z} \in R^2$



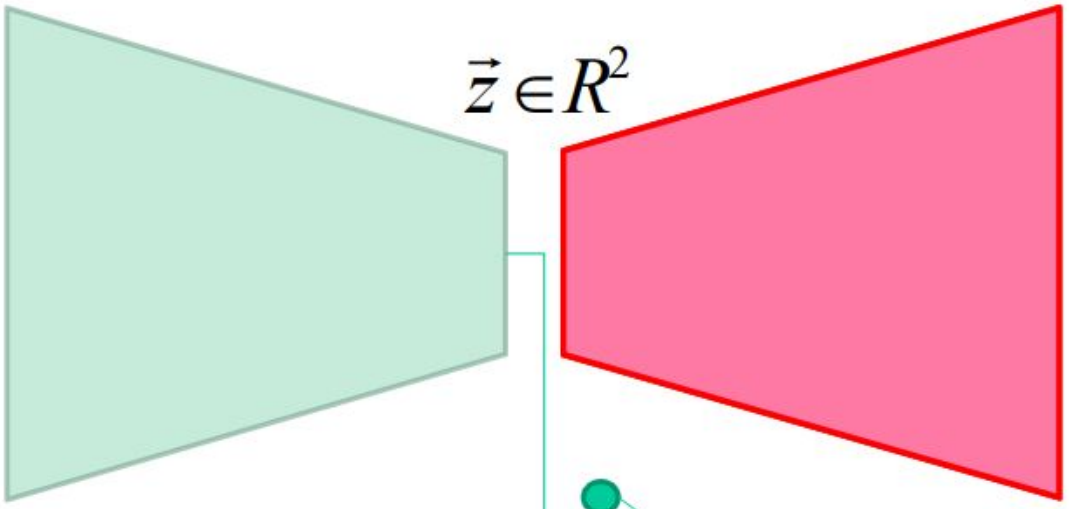
$\vec{x}'$



t

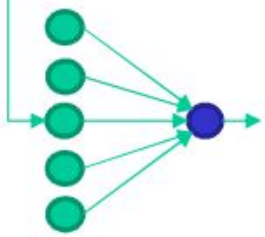
Régression linéaire
(perceptron sans couche cachée)

Autoencodeur contraint



$\vec{x}$

$t = 4$



$\vec{x}'$

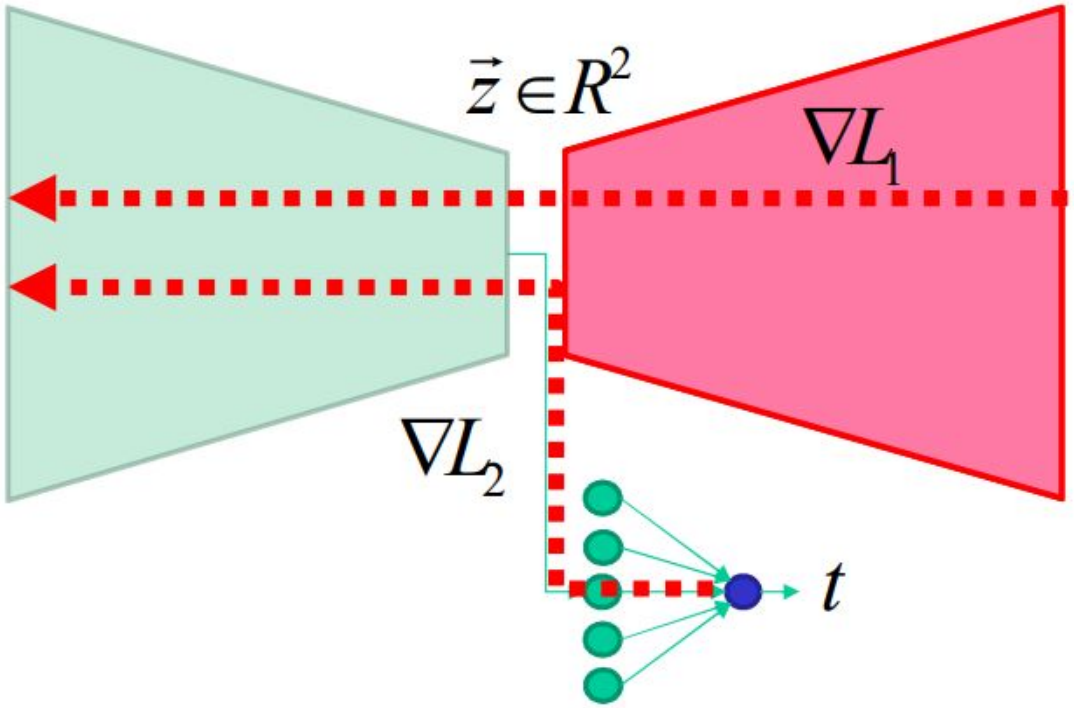
$$L_1 = \left\| \vec{x} - \vec{x}' \right\|^2$$

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

Autoencodeur contraint

$\vec{x}$

$t = 4$



$\vec{x}'$

$$L_{décodeur} = L_1$$

$$L_{reg} = L_2$$

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

Autoencodeur contraint

$\vec{z} \in R^2$

$\vec{x}$

$t = 4$

$\vec{x}'$

OBJECTIF

Que les images dans l'espace latent soient linéairement distribuées en fonction de leur classe

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

Autoencodeur contraint

$\vec{z} \in R^2$

$\vec{x}$

$t = 4$

$\vec{x'}$

OBJECTIF

La **loss L1** force l'encodeur et le décodeur à reproduire l'image d'entrée.

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

Autoencodeur contraint

$\vec{z} \in R^2$

$\vec{x}$

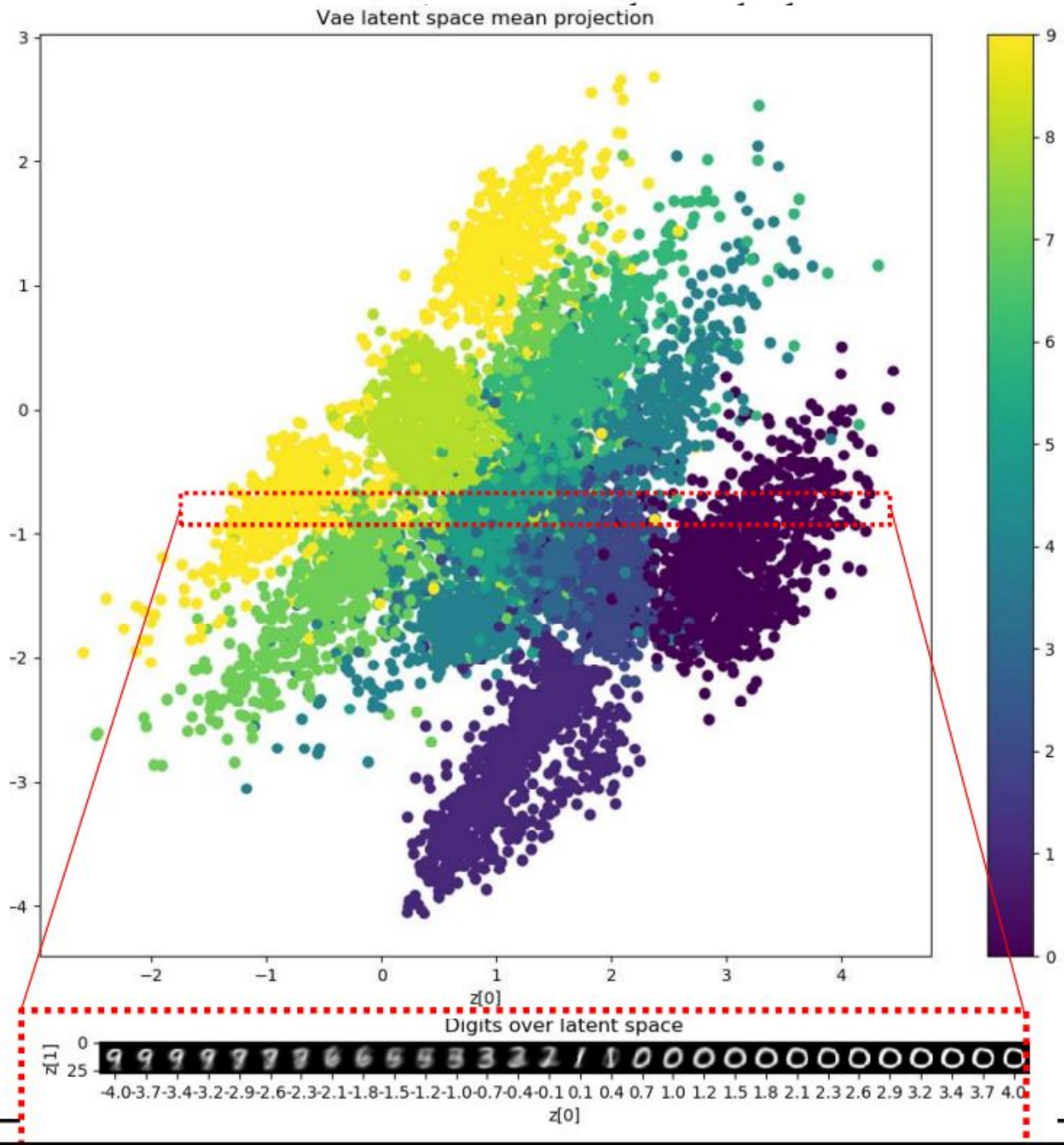
$t = 4$

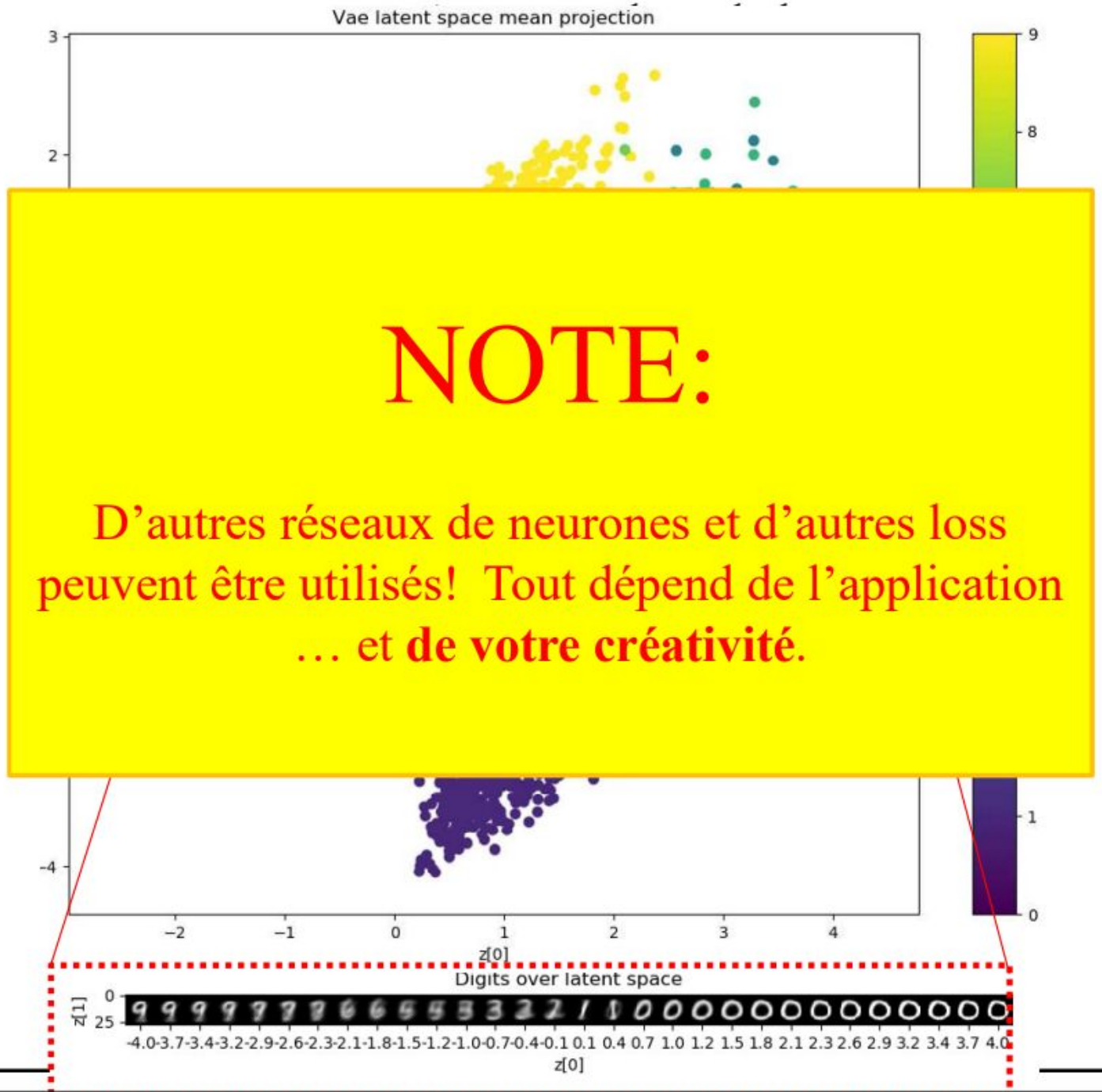
$\vec{x}'$

OBJECTIF

La **loss L2** force l'encodeur à projeter les images d'entrée vers un **espace latent linéaire**.

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


Vae latent space mean projection
z[0]
z[1]
Digits over latent space
z[0]



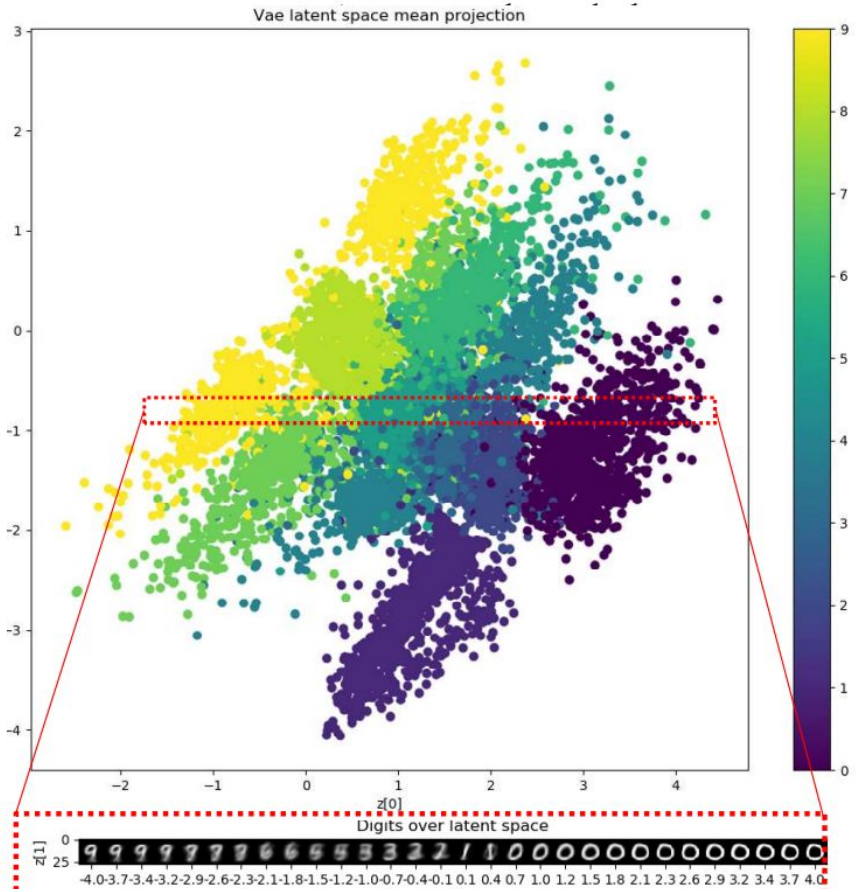
NOTE:

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

VAE conditionnels

Le problème :

Un autoencodeur (variationnel, ou pas) ne permet pas de générer une image d'une classe en particulier



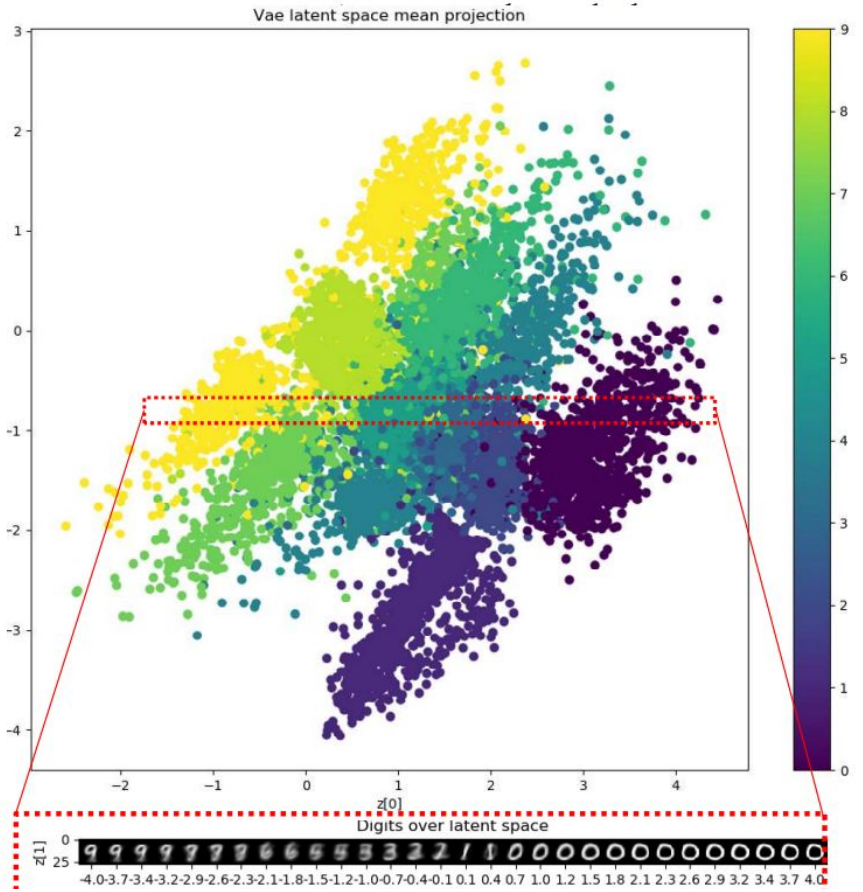
VAE conditionnels

Le problème :

Un autoencodeur (variationnel, ou pas) ne permet pas de générer une image d'une classe en particulier

La solution:

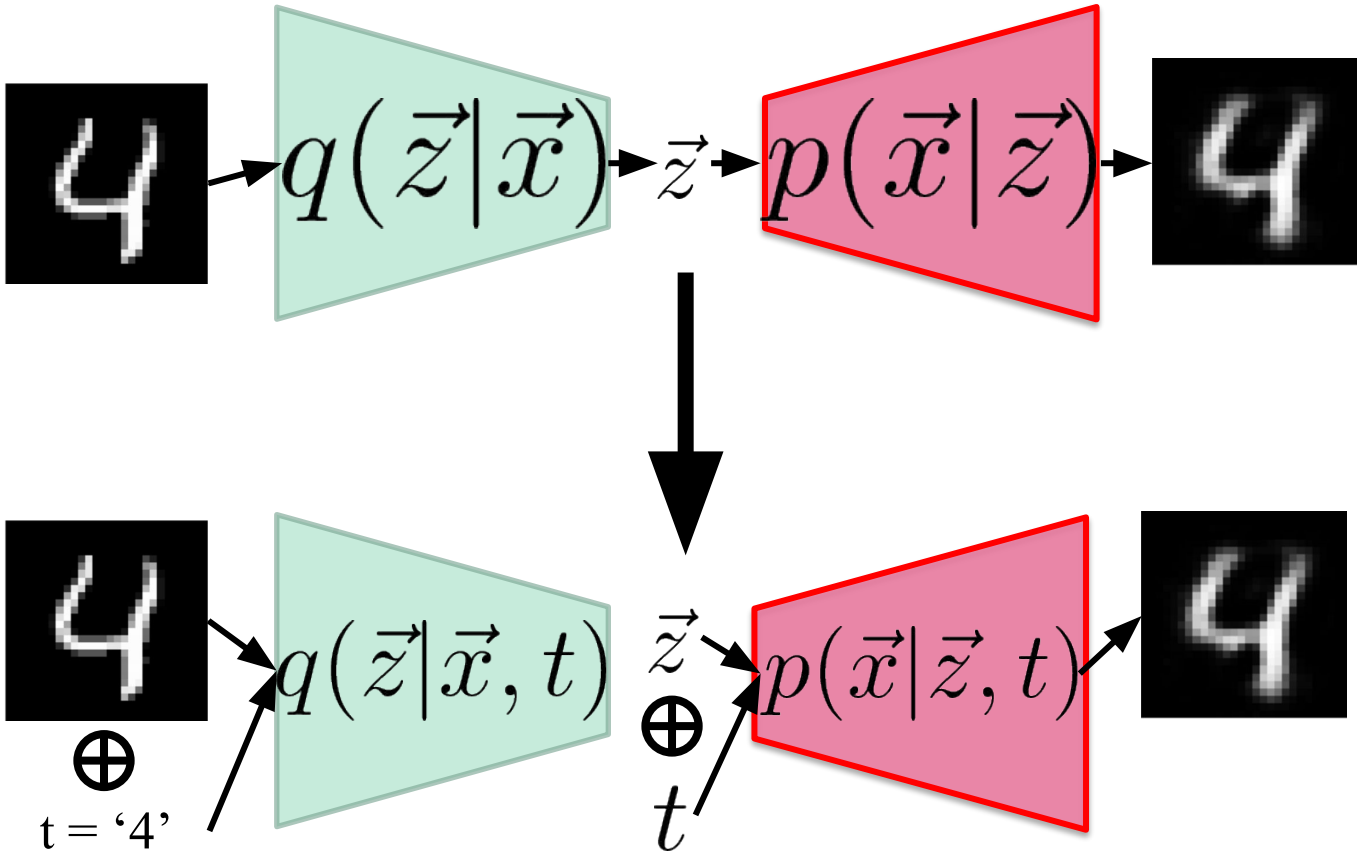
Injecter l'information de la classe dans notre encodeur et décodeur !



VAE conditionnels

La solution:

Injecter l'information de la classe dans notre encodeur et décodeur !



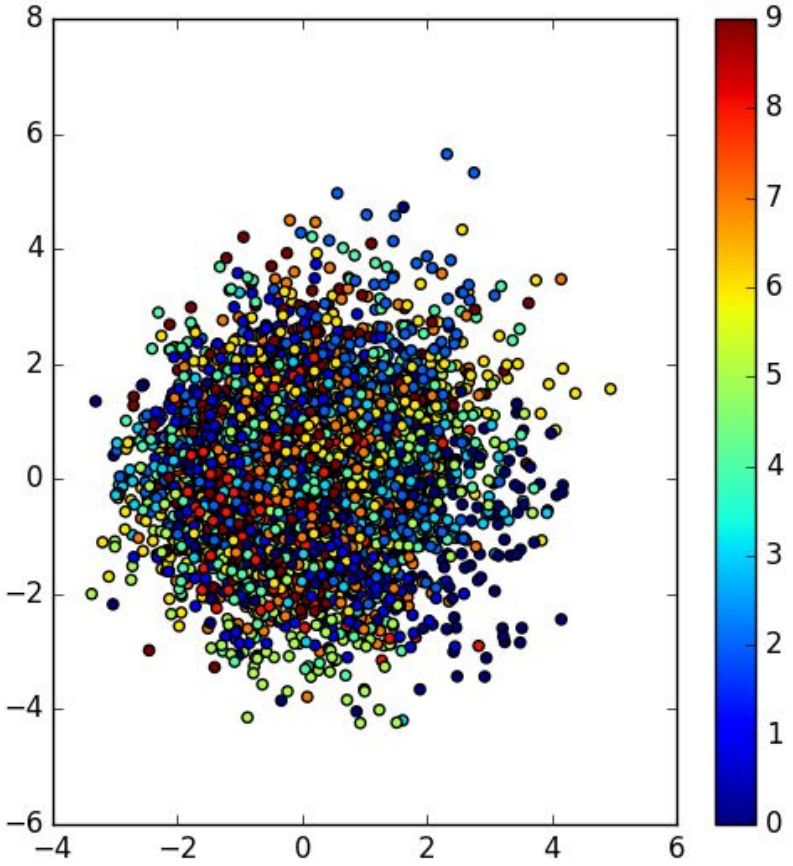
https://agustinus.kristia.de/techblog/2016/12/17/conditional-vae/

VAE conditionnels

La solution:

Injecter l'information de la classe dans notre encodeur et décodeur !

$q(\vec{z}|\vec{x},t)$



$p(\vec{x}|\vec{z},t)$

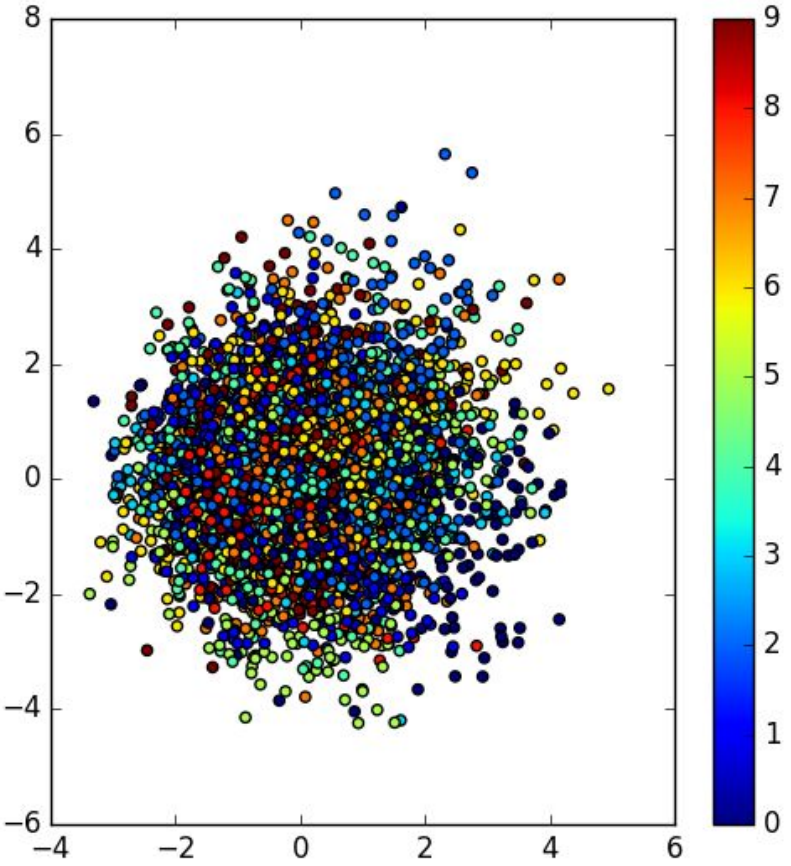
https://agustinus.kristia.de/techblog/2016/12/17/conditional-vae/

VAE conditionnels

La solution: Injecter l'information de la classe dans notre encodeur et décodeur !

Semble chaotique, mais … strictement gaussien N(0, 1) pour toutes les classes !

$q(\vec{z}|\vec{x}, t)$



$p(\vec{x}|\vec{z}, t)$

https://agustinus.kristia.de/techblog/2016/12/17/conditional-vae/

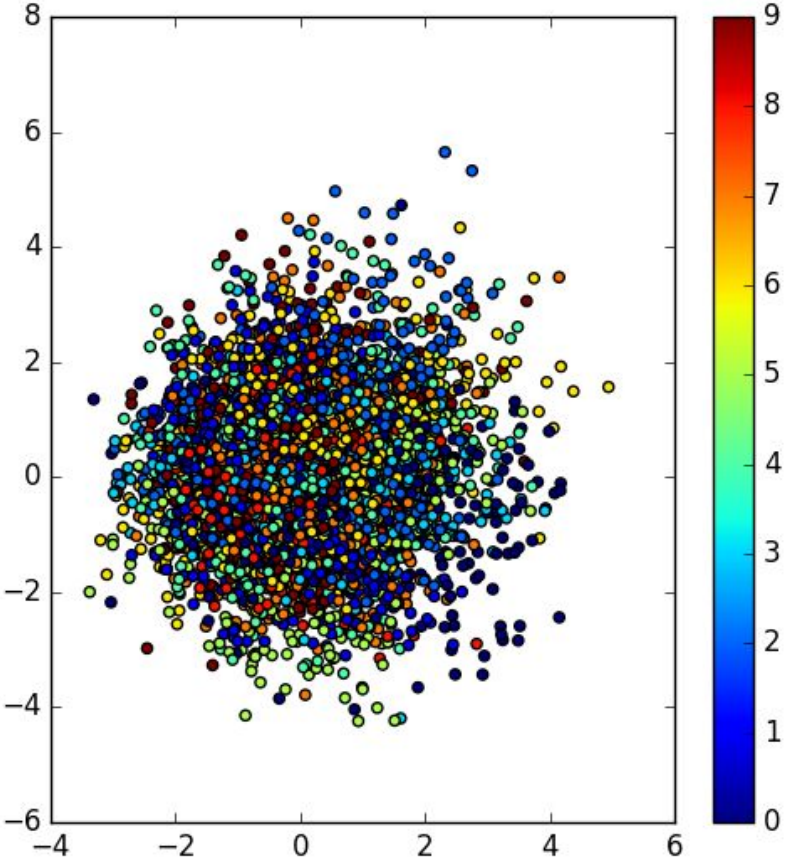
VAE conditionnels

La solution: Injecter l'information de la classe dans notre encodeur et décodeur !

Semble chaotique, mais … strictement gaussien N(0, 1) pour toutes les classes !

Le modèle n'a pas à séparer les classes

$q(\vec{z}|\vec{x},t)$



$p(\vec{x}|\vec{z},t)$

https://agustinus.kristia.de/techblog/2016/12/17/conditional-vae/

VAE conditionnels

La solution:

Injecter l'information de la classe dans notre encodeur et décodeur !

t = '5'

https://agustinus.kristia.de/techblog/2016/12/17/conditional-vae/

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

Autoencodeur variationnel

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

ELBO loss : Evidence Lower Bound

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

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

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

Autoencodeur variationnel

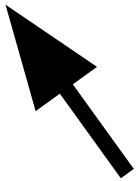
ELBO loss : Evidence Lower Bound

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.



Inconnu, impossible à estimer
On va devoir réorganiser quelques termes

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL = [q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] = -\int q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} d\vec{z}$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

Non symétrique !

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL = [q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] = -\int q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} d\vec{z}$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

Non symétrique !

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL = [q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] = -\int q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} d\vec{z}$$

$$= \int q(\vec{z}|\vec{x}) \log q(\vec{z}|\vec{x}) - \log p(\vec{z}|\vec{x}) d\vec{z}$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

Non symétrique !

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL = [q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] = -\int q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} d\vec{z}$$

$$= \int q(\vec{z}|\vec{x}) \log q(\vec{z}|\vec{x}) - \log p(\vec{z}|\vec{x}) d\vec{z}$$

$$= \int q(\vec{z}|\vec{x})(\log q(\vec{z}|\vec{x}) - \log p(\vec{x}|\vec{z}) - \log p(\vec{z}) + \log p(\vec{x})) d\vec{z}$$

Bayes:

$$p(\vec{z}|\vec{x}) = \frac{p(\vec{x}|\vec{z})p(\vec{z})}{p(\vec{x})}$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

Non symétrique !

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL = [q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] = -\int q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} d\vec{z}$$

$$= \int q(\vec{z}|\vec{x}) \log q(\vec{z}|\vec{x}) - \log p(\vec{z}|\vec{x}) d\vec{z}$$

$$= \int q(\vec{z}|\vec{x})(\log q(\vec{z}|\vec{x}) - \log p(\vec{x}|\vec{z}) - \log p(\vec{z}) + \log p(\vec{x})) d\vec{z}$$

$$= \int q(\vec{z}|\vec{x}) \log \frac{q(\vec{z}|\vec{x})}{p(\vec{z})} d\vec{z} - \int q(\vec{z}|\vec{x}) \log p(\vec{x}|\vec{z}) d\vec{z} + \log p(\vec{z}) \int q(\vec{z}|\vec{x}) d\vec{z}$$

Bayes:

$$p(\vec{z}|\vec{x}) = \frac{p(\vec{x}|\vec{z})p(\vec{z})}{p(\vec{x})}$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

Non symétrique !

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL = [q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] = -\int q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} d\vec{z}$$

$$= \int q(\vec{z}|\vec{x}) \log q(\vec{z}|\vec{x}) - \log p(\vec{z}|\vec{x}) d\vec{z}$$

Bayes:

$$p(\vec{z}|\vec{x}) = \frac{p(\vec{x}|\vec{z})p(\vec{z})}{p(\vec{x})}$$

$$= \int q(\vec{z}|\vec{x})(\log q(\vec{z}|\vec{x}) - \log p(\vec{x}|\vec{z}) - \log p(\vec{z}) + \log p(\vec{x})) d\vec{z}$$

$$= \int q(\vec{z}|\vec{x}) \log \frac{q(\vec{z}|\vec{x})}{p(\vec{z})} d\vec{z} - \int q(\vec{z}|\vec{x}) \log p(\vec{x}|\vec{z}) d\vec{z} + \log p(\vec{z}) \int q(\vec{z}|\vec{x}) d\vec{z}$$

1

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

Non symétrique !

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL = [q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] = -\int q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} d\vec{z}$$

$$= \int q(\vec{z}|\vec{x}) \log q(\vec{z}|\vec{x}) - \log p(\vec{z}|\vec{x}) d\vec{z}$$

Bayes:

$$p(\vec{z}|\vec{x}) = \frac{p(\vec{x}|\vec{z})p(\vec{z})}{p(\vec{x})}$$

$$= \int q(\vec{z}|\vec{x})(\log q(\vec{z}|\vec{x}) - \log p(\vec{x}|\vec{z}) - \log p(\vec{z}) + \log p(\vec{x})) d\vec{z}$$

$$= \underbrace{\int q(\vec{z}|\vec{x}) \log \frac{q(\vec{z}|\vec{x})}{p(\vec{z})} d\vec{z}}_{KL[q(\vec{z}|\vec{x})||p(\vec{z})]} - \underbrace{\int q(\vec{z}|\vec{x}) \log p(\vec{x}|\vec{z}) d\vec{z}}_{\mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]} + \log p(\vec{z}) \int q(\vec{z}|\vec{x}) d\vec{z}$$

1

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

Non symétrique !

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL = [q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] = -\int q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} d\vec{z}$$

$$= \int q(\vec{z}|\vec{x}) \log q(\vec{z}|\vec{x}) - \log p(\vec{z}|\vec{x}) d\vec{z}$$

Bayes:

$$p(\vec{z}|\vec{x}) = \frac{p(\vec{x}|\vec{z})p(\vec{z})}{p(\vec{x})}$$

$$= \int q(\vec{z}|\vec{x})(\log q(\vec{z}|\vec{x}) - \log p(\vec{x}|\vec{z}) - \log p(\vec{z}) + \log p(\vec{x})) d\vec{z}$$

$$= \underbrace{\int q(\vec{z}|\vec{x}) \log \frac{q(\vec{z}|\vec{x})}{p(\vec{z})} d\vec{z}}_{KL[q(\vec{z}|\vec{x})||p(\vec{z})]} - \underbrace{\int q(\vec{z}|\vec{x}) \log p(\vec{x}|\vec{z}) d\vec{z}}_{\mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]} + \log p(\vec{z}) \int q(\vec{z}|\vec{x}) d\vec{z}$$

1

En réorganisant les termes:

$$KL[q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] - KL[q(\vec{z}|\vec{x})||p(\vec{z})] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})] = \log p(\vec{x})$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

Non symétrique !

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL = [q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] = -\int q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} d\vec{z}$$

$$= \int q(\vec{z}|\vec{x}) \log q(\vec{z}|\vec{x}) - \log p(\vec{z}|\vec{x}) d\vec{z}$$

Bayes:

$$p(\vec{z}|\vec{x}) = \frac{p(\vec{x}|\vec{z})p(\vec{z})}{p(\vec{x})}$$

$$= \int q(\vec{z}|\vec{x})(\log q(\vec{z}|\vec{x}) - \log p(\vec{x}|\vec{z}) - \log p(\vec{z}) + \log p(\vec{x})) d\vec{z}$$

$$= \underbrace{\int q(\vec{z}|\vec{x}) \log \frac{q(\vec{z}|\vec{x})}{p(\vec{z})} d\vec{z}}_{KL[q(\vec{z}|\vec{x})||p(\vec{z})]} - \underbrace{\int q(\vec{z}|\vec{x}) \log p(\vec{x}|\vec{z}) d\vec{z}}_{\mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]} + \log p(\vec{z}) \int q(\vec{z}|\vec{x}) d\vec{z}$$

1

En réorganisant les termes:

$$\underbrace{KL[q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] - KL[q(\vec{z}|\vec{x})||p(\vec{z})] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]}_{\textit{Evidence LOwer BOund (ELBO)}} = \underbrace{\log p(\vec{x})}_{\text{Constante par rapport à q !}}$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL[q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] - \underbrace{KL[q(\vec{z}|\vec{x})||p(\vec{z})] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]}_{\text{Evidence LOwer BOund (ELBO)}} = \underbrace{\log p(\vec{x})}_{\text{Constante par rapport à q !}}$$

KL divergence ≥ 0	ELBO	$\log p(x) \leq 0$
8	-12	-4

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL[q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] - \underbrace{KL[q(\vec{z}|\vec{x})||p(\vec{z})] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]}_{\text{Evidence LOwer BOund (ELBO)}} = \underbrace{\log p(\vec{x})}_{\text{Constante par rapport à q !}}$$

KL divergence ≥ 0	ELBO	$\log p(x) \leq 0$
8	-12	-4
3	-7	-4

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL[q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] - \underbrace{KL[q(\vec{z}|\vec{x})||p(\vec{z})] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]}_{\text{Evidence LOwer BOund (ELBO)}} = \underbrace{\log p(\vec{x})}_{\text{Constante par rapport à q !}}$$

KL divergence ≥ 0	ELBO	$\log p(x) \leq 0$
8	-12	-4
3	-7	-4
1	-5	-4

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On souhaite obtenir $q(\vec{z}|\vec{x}) \approx p(\vec{z}|\vec{x})$. Pour ce faire, on minimisera la KL-divergence entre les deux.

$$KL[q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})] - \underbrace{KL[q(\vec{z}|\vec{x})||p(\vec{z})] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]}_{\text{Evidence LOwer BOund (ELBO)}} = \underbrace{\log p(\vec{x})}_{\text{Constante par rapport à q !}}$$

KL divergence ≥ 0	ELBO	log p(x) ≤ 0
8	-12	-4
3	-7	-4
1	-5	-4
0	-4	-4

On atteint notre objectif en minimisant la KL divergence ou en maximisant la ELBO

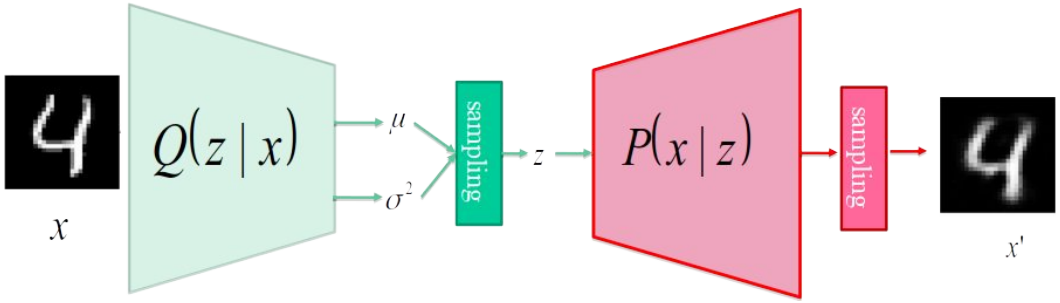
Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On atteint notre objectif en minimisant la KL divergence ou en **maximisant la ELBO**

$$\underbrace{-KL[q(\vec{z}|\vec{x})||p(\vec{z})] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]}$$

Evidence LOwer BOund (ELBO)



On présume que $p(\vec{z}) = \mathcal{N}(\vec{z}; 0, 1)$

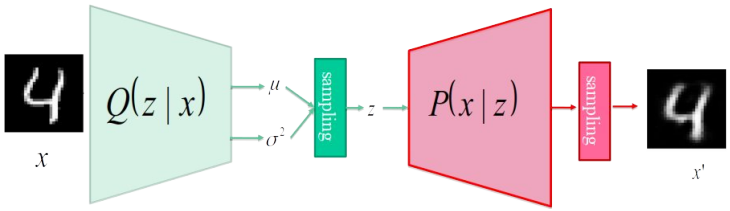
$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z}; 0, 1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

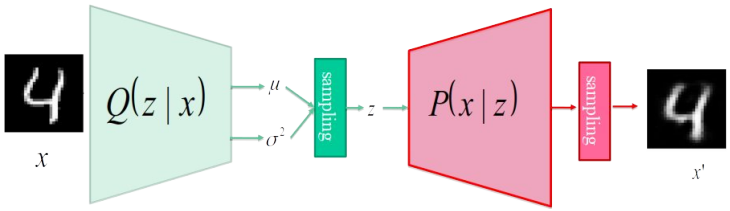
On souhaite minimiser la loss

$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$



Autoencodeur variationnel

ELBO loss : Evidence Lower Bound



On souhaite minimiser la loss

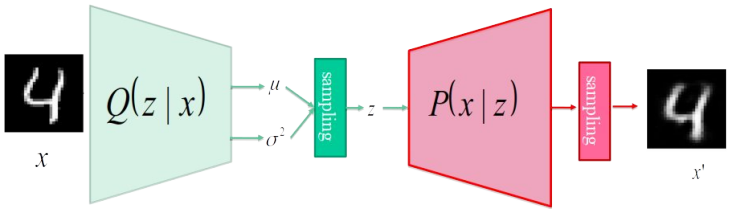
$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

On formule q par l'encodeur

$$-KL[\mathcal{N}(\vec{z};\mu,\sigma^2)||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound



On souhaite minimiser la loss

$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

On formule q par l'encodeur

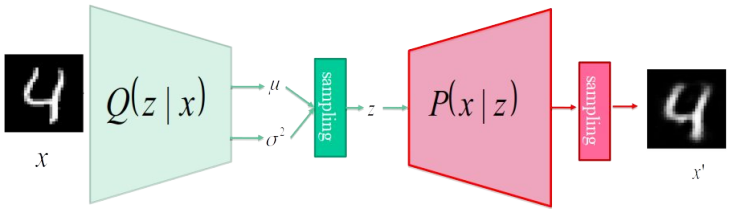
$$-KL[\mathcal{N}(\vec{z};\mu,\sigma^2)||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

Définition de la KL-divergence pour deux gaussiennes:

$$KL[\mathcal{N}(\mu_1,\sigma_1^2)||\mathcal{N}(\mu_2,\sigma_2^2)] = \frac{(\mu_1-\mu_2)^2}{2\sigma_2^2} + \frac{1}{2}(\frac{\sigma_1^2}{\sigma_2^2} - 1\log\frac{\sigma_1^2}{\sigma_2^2})$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound



On souhaite minimiser la loss

$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

On formule q par l'encodeur

$$-KL[\mathcal{N}(\vec{z};\mu,\sigma^2)||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

Définition de la KL-divergence pour deux gaussiennes:

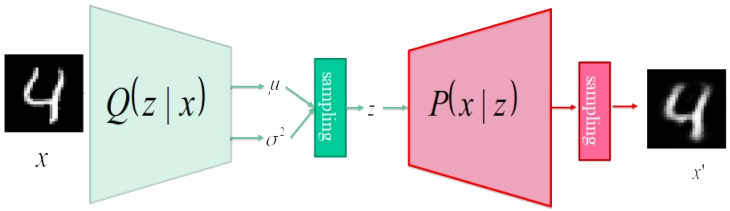
$$KL[\mathcal{N}(\mu_1,\sigma_1^2)||\mathcal{N}(\mu_2,\sigma_2^2)] = \frac{(\mu_1-\mu_2)^2}{2\sigma_2^2} + \frac{1}{2}(\frac{\sigma_1^2}{\sigma_2^2} - 1\log\frac{\sigma_1^2}{\sigma_2^2})$$

En replaçant pour nos valeurs:

$$KL[\mathcal{N}(\vec{z};\mu,\sigma^2)||\mathcal{N}(z;0,1)] = \frac{1}{2}(\mu-0)^2 + \frac{1}{2}(\frac{\sigma^2}{1^2} - 1 - \log\frac{\sigma^2}{1^2})$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound



On souhaite minimiser la loss

$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

On formule q par l'encodeur

$$-KL[\mathcal{N}(\vec{z};\mu,\sigma^2)||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

Définition de la KL-divergence pour deux gaussiennes:

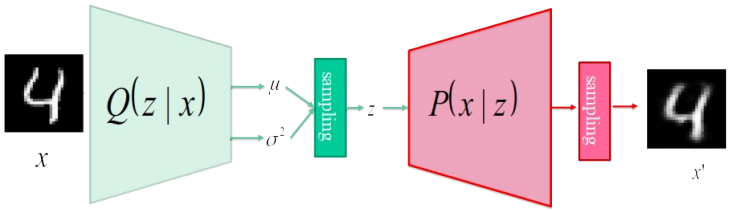
$$KL[\mathcal{N}(\mu_1,\sigma_1^2)||\mathcal{N}(\mu_2,\sigma_2^2)] = \frac{(\mu_1-\mu_2)^2}{2\sigma_2^2} + \frac{1}{2}(\frac{\sigma_1^2}{\sigma_2^2} - 1\log\frac{\sigma_1^2}{\sigma_2^2})$$

En replaçant pour nos valeurs:

$$\begin{aligned} KL[\mathcal{N}(\vec{z};\mu,\sigma^2)||\mathcal{N}(z;0,1)] &= \frac{1}{2}(\mu-0)^2 + \frac{1}{2}(\frac{\sigma^2}{1^2} - 1 - \log\frac{\sigma^2}{1^2}) \\ &= \frac{1}{2}(\mu^2+\sigma^2-1-\log\sigma^2)) \end{aligned}$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound



On souhaite minimiser la loss

$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

On formule q par l'encodeur

$$-KL[\mathcal{N}(\vec{z};\mu,\sigma^2)||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

Définition de la KL-divergence pour deux gaussiennes:

$$KL[\mathcal{N}(\mu_1,\sigma_1^2)||\mathcal{N}(\mu_2,\sigma_2^2)] = \frac{(\mu_1-\mu_2)^2}{2\sigma_2^2} + \frac{1}{2}(\frac{\sigma_1^2}{\sigma_2^2} - 1\log\frac{\sigma_1^2}{\sigma_2^2})$$

En replaçant pour nos valeurs:

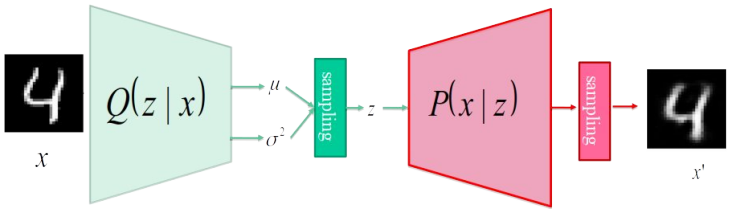
$$\begin{aligned} KL[\mathcal{N}(\vec{z};\mu,\sigma^2)||\mathcal{N}(z;0,1)] &= \frac{1}{2}(\mu-0)^2 + \frac{1}{2}(\frac{\sigma^2}{1^2} - 1 - \log\frac{\sigma^2}{1^2}) \\ &= \frac{1}{2}(\mu^2+\sigma^2-1-\log\sigma^2)) \\ &= \frac{1}{2}(-1-\log\sigma^2-\mu^2+\sigma^2) \end{aligned}$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On souhaite minimiser la loss

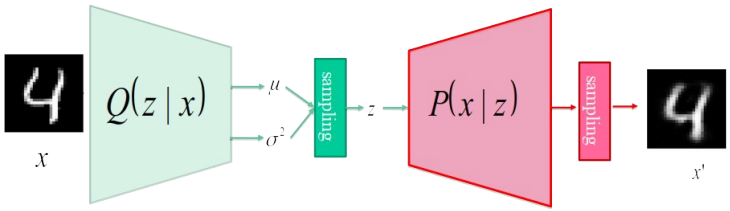
$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$



$$Loss = \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound



On souhaite minimiser la loss

$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

$$Loss = \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

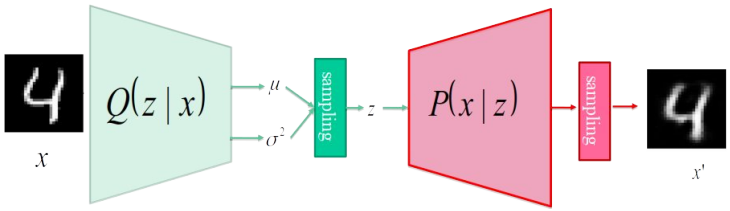
$$= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \frac{1}{K}\sum_{i=1}^{K}[\log p(\vec{x_i}|\vec{z_i})]$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On souhaite minimiser la loss

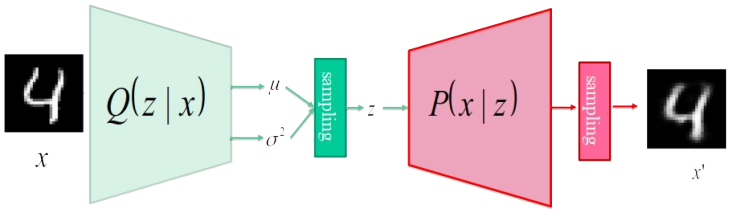
$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$



$$\begin{aligned} Loss &= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})] \\ &= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \frac{1}{K}\sum_{i=1}^{K}[\log p(\vec{x_i}|\vec{z_i})] \\ &= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \log p(\vec{x}|\vec{z}) \end{aligned}$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound



On souhaite minimiser la loss

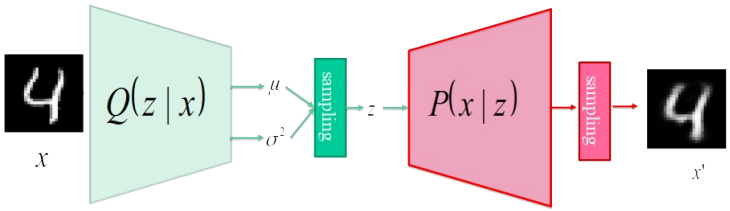
$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

$$\begin{aligned} Loss &= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})] \\ &= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \frac{1}{K}\sum_{i=1}^{K}[\log p(\vec{x_i}|\vec{z_i})] \\ &= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \log p(\vec{x}|\vec{z}) \end{aligned}$$

$$p(\vec{x}|\vec{z}) = p(\vec{x}|\mathcal{N}(\vec{x};\vec{x}',\sigma))$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound



On souhaite minimiser la loss

$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

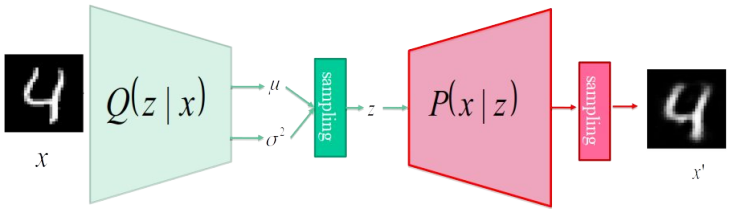
$$\begin{aligned} Loss &= \frac{1}{2}\left(1+\log\sigma^2+\mu^2-\sigma^2\right) + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})] \\ &= \frac{1}{2}\left(1+\log\sigma^2+\mu^2-\sigma^2\right) + \frac{1}{K}\sum_{i=1}^{K}[\log p(\vec{x_i}|\vec{z_i})] \\ &= \frac{1}{2}\left(1+\log\sigma^2+\mu^2-\sigma^2\right) + \log p(\vec{x}|\vec{z}) \end{aligned}$$

$$p(\vec{x}|\vec{z}) = p(\vec{x}|\mathcal{N}(\vec{x};\vec{x}',\sigma))$$

$$\log p(\vec{x}|\mathcal{N}(\vec{x};\mu,\sigma)) = -\log\sigma - \frac{1}{2}\log(2\pi) - \frac{1}{2\pi\sigma^2}(\vec{x}-\mu)^2$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound



On souhaite minimiser la loss

$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

$$\begin{aligned} Loss &= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})] \\ &= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \frac{1}{K}\sum_{i=1}^{K}[\log p(\vec{x_i}|\vec{z_i})] \\ &= \frac{1}{2}\left(1 + \log\sigma^2 + \mu^2 - \sigma^2\right) + \log p(\vec{x}|\vec{z}) \end{aligned}$$

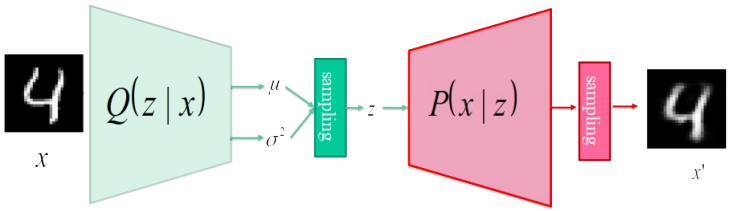
$$p(\vec{x}|\vec{z}) = p(\vec{x}|\mathcal{N}(\vec{x};\vec{x}',\sigma))$$

$$\log p(\vec{x}|\mathcal{N}(\vec{x};\mu,\sigma)) = -\cancel{\log\sigma} - \cancel{\frac{1}{2}\log(2\pi)} - \cancel{\frac{1}{2\pi\sigma^2}}(\vec{x}-\mu)^2$$

Autoencodeur variationnel

ELBO loss : Evidence Lower Bound

On souhaite minimiser la loss



$$-KL[q(\vec{z}|\vec{x})||\mathcal{N}(\vec{z};0,1)] + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})]$$

$$\begin{aligned}
Loss &= \frac{1}{2}\left(1+\log\sigma^2+\mu^2-\sigma^2\right) + \mathbb{E}_{z\sim q}[\log p(\vec{x}|\vec{z})] \\
&= \frac{1}{2}\left(1+\log\sigma^2+\mu^2-\sigma^2\right) + \frac{1}{K}\sum_{i=1}^{K}[\log p(\vec{x_i}|\vec{z_i})] \\
&= \frac{1}{2}\left(1+\log\sigma^2+\mu^2-\sigma^2\right) + \log p(\vec{x}|\vec{z}) \\
&= \frac{1}{2}\left(1+\log\sigma^2+\mu^2-\sigma^2\right) - \lambda(\vec{x}-\vec{x}')^2
\end{aligned}$$

$$p(\vec{x}|\vec{z}) = p(\vec{x}|\mathcal{N}(\vec{x};\vec{x}',\sigma))$$

$$\log p(\vec{x}|\mathcal{N}(\vec{x};\mu,\sigma)) = -\cancel{\log\sigma} - \cancel{\frac{1}{2}\log(2\pi)} - \cancel{\frac{1}{2\pi\sigma^2}}(\vec{x}-\mu)^2$$

$$\log p(\vec{x}|\mathcal{N}(\vec{x};x',\sigma)) = -(\vec{x}-\vec{x}')^2$$

Autoencodeur variationnel

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

ELBO loss : Evidence Lower Bound

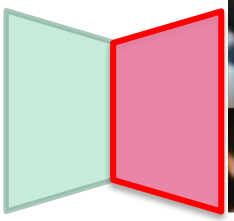
$$Loss = \underbrace{KL(N(z;0,1), N(z;\mu,\Sigma))}_{\text{Perte encodeur}} - \underbrace{\log(P(x \mid z))}_{\text{Perte décodeur}}$$

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

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

Ex.: base de données *CelebA*

x



x'

https://github.com/yzwxx/vae-celebA

GAN

Generative Adversarial Nets

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

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

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*

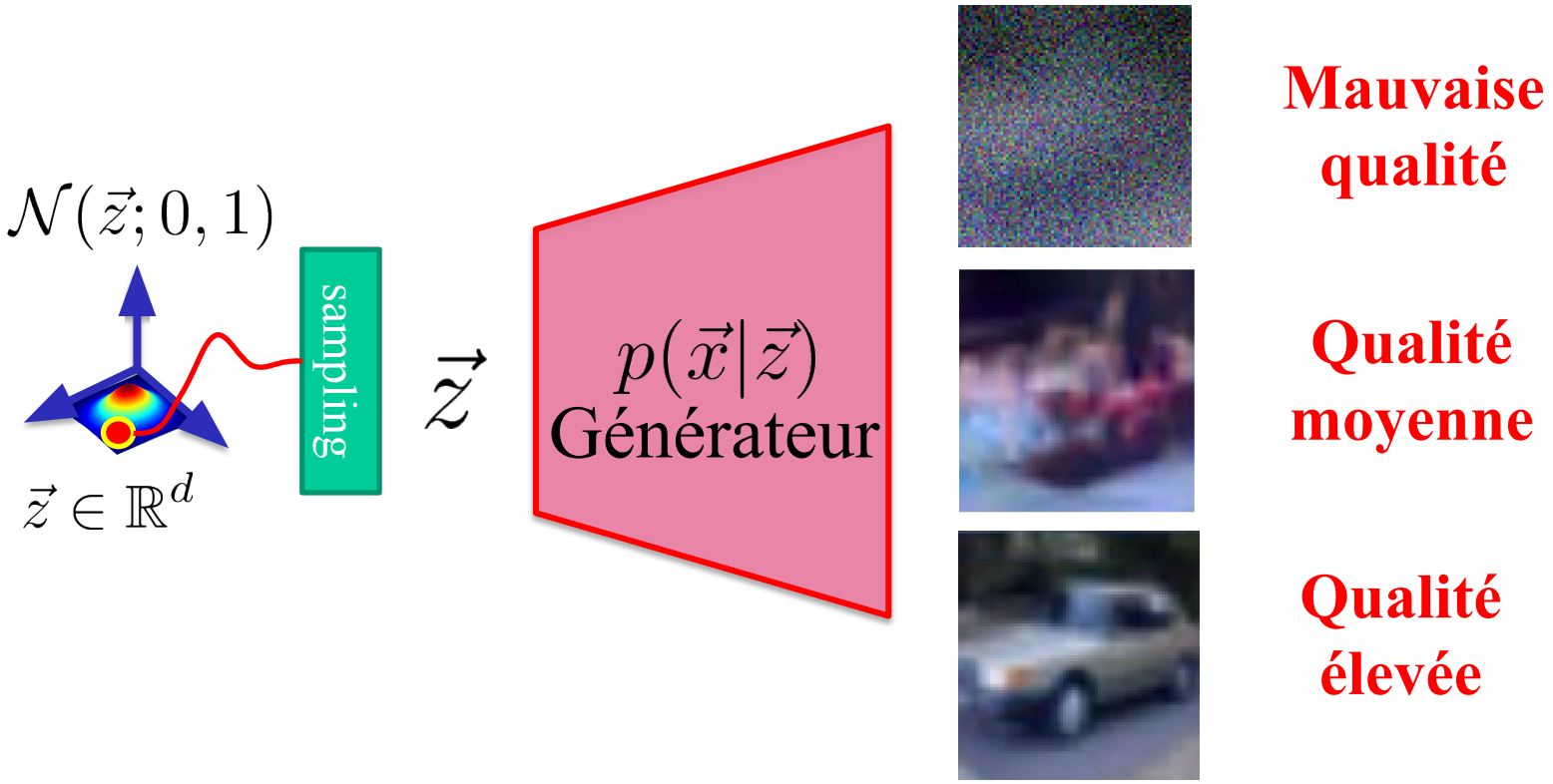
Comme pour les VAE, on utilisera une **distribution *a priori*** facile à échantillonner : une **gaussienne**!

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

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**

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

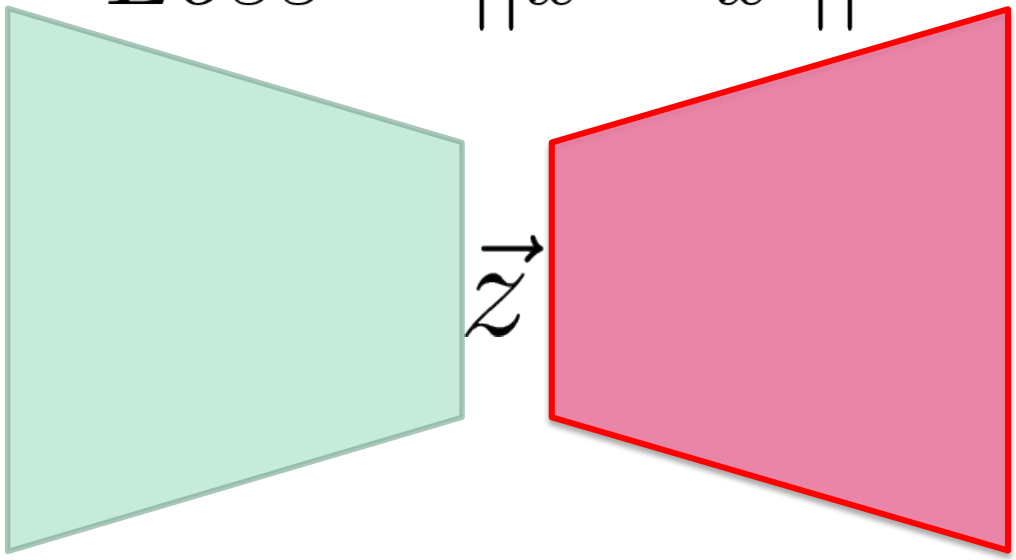


Pour entraîner un décodeur, il faut une **fonction de perte** (*loss*) qui **mesure la qualité (degré de réalisme) des images produites**

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

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

$\vec{x}$

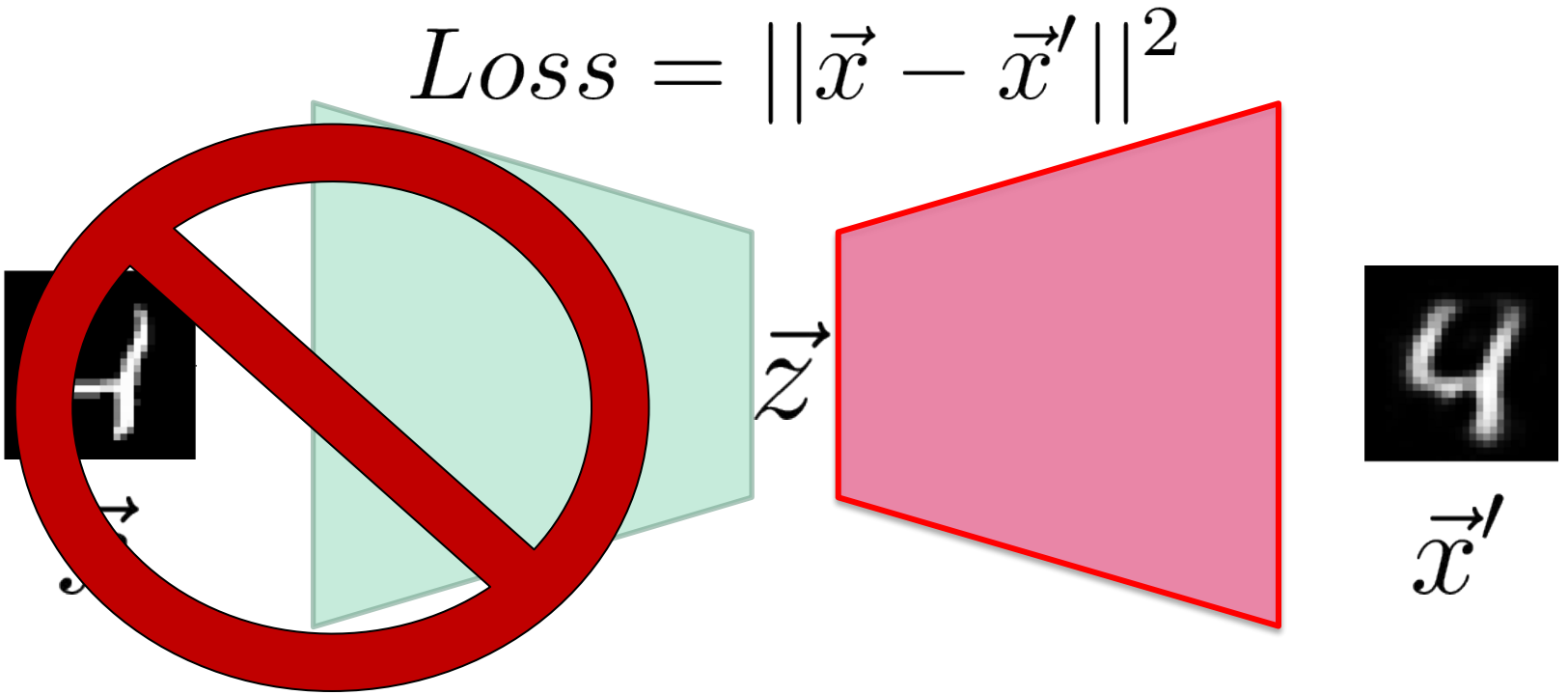


$\vec{z}$

$\vec{x'}$

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**?



$\vec{z} \sim p(\vec{z})$

$G(\vec{z})$

$\vec{x}$

Loss sans cible de référence?

$\vec{z} \sim p(\vec{z}$ éférence?

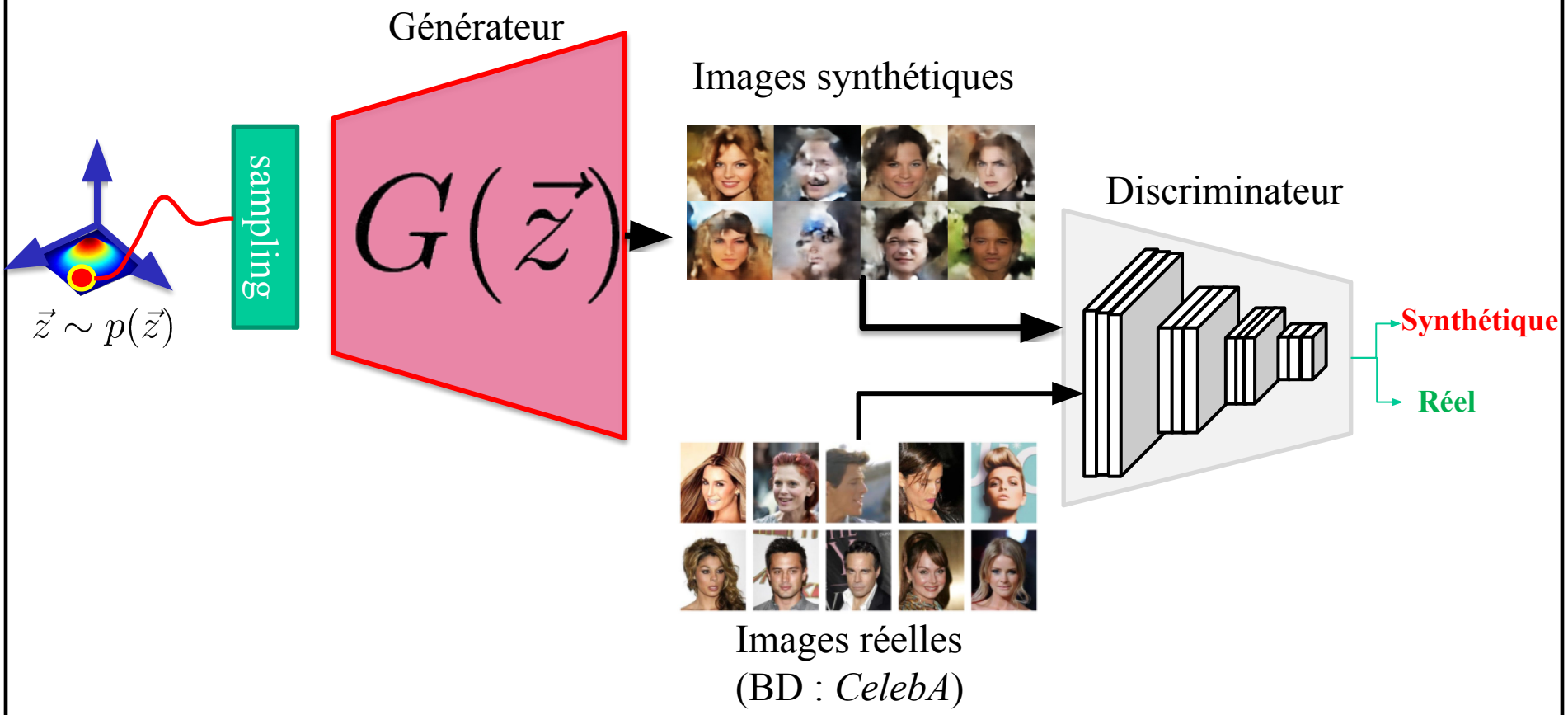
Approximer la perte à l'aide d'un

2e réseau de neurones

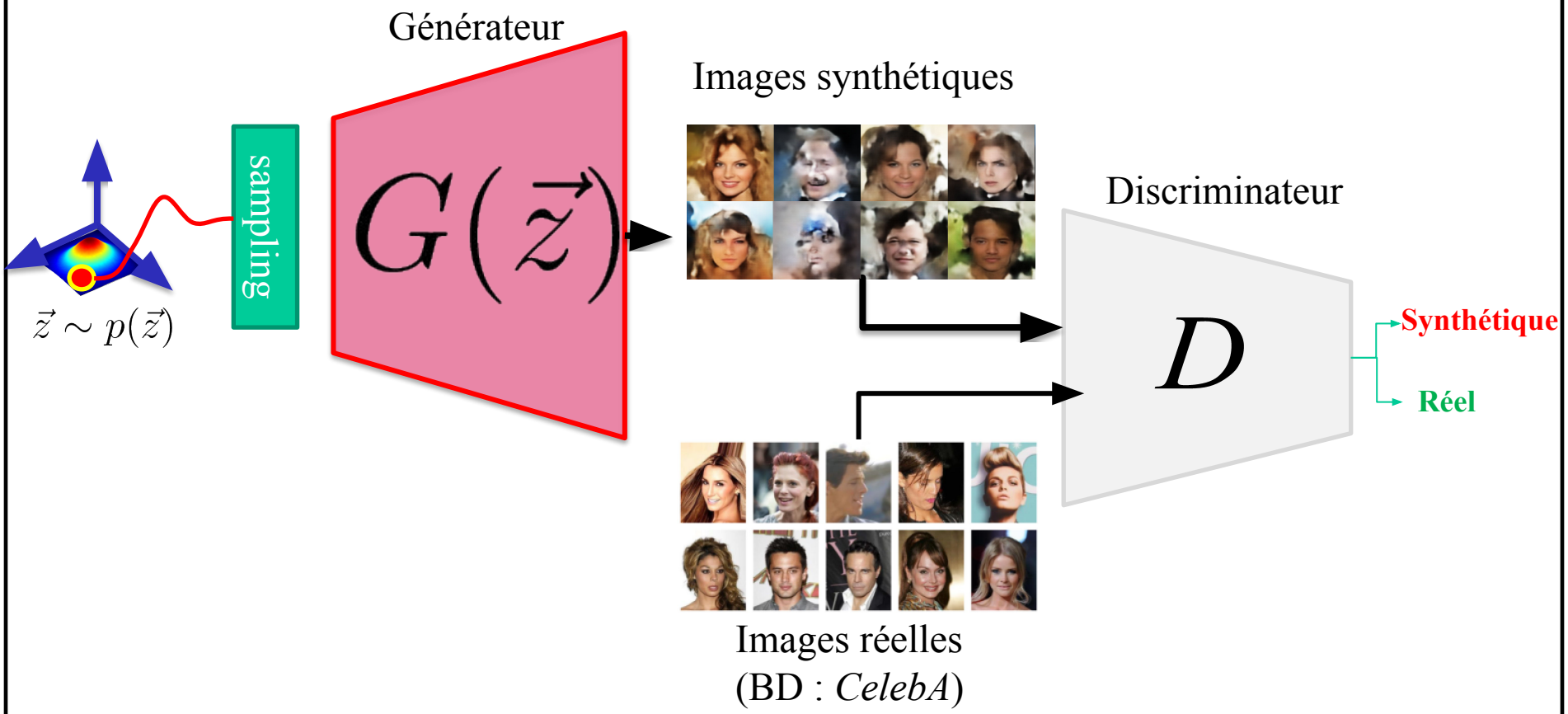
et d'une

Base de données de référence

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



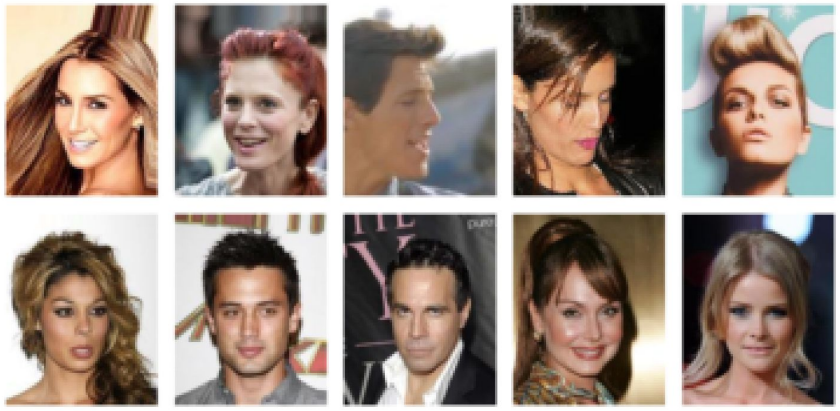
Pour entraîner un décodeur, il faut une **fonction de perte** (loss) qui **mesure la qualité des images produites**



Données étiquetées

Produites par le générateur

$t = 0$ ('*synthétique*')

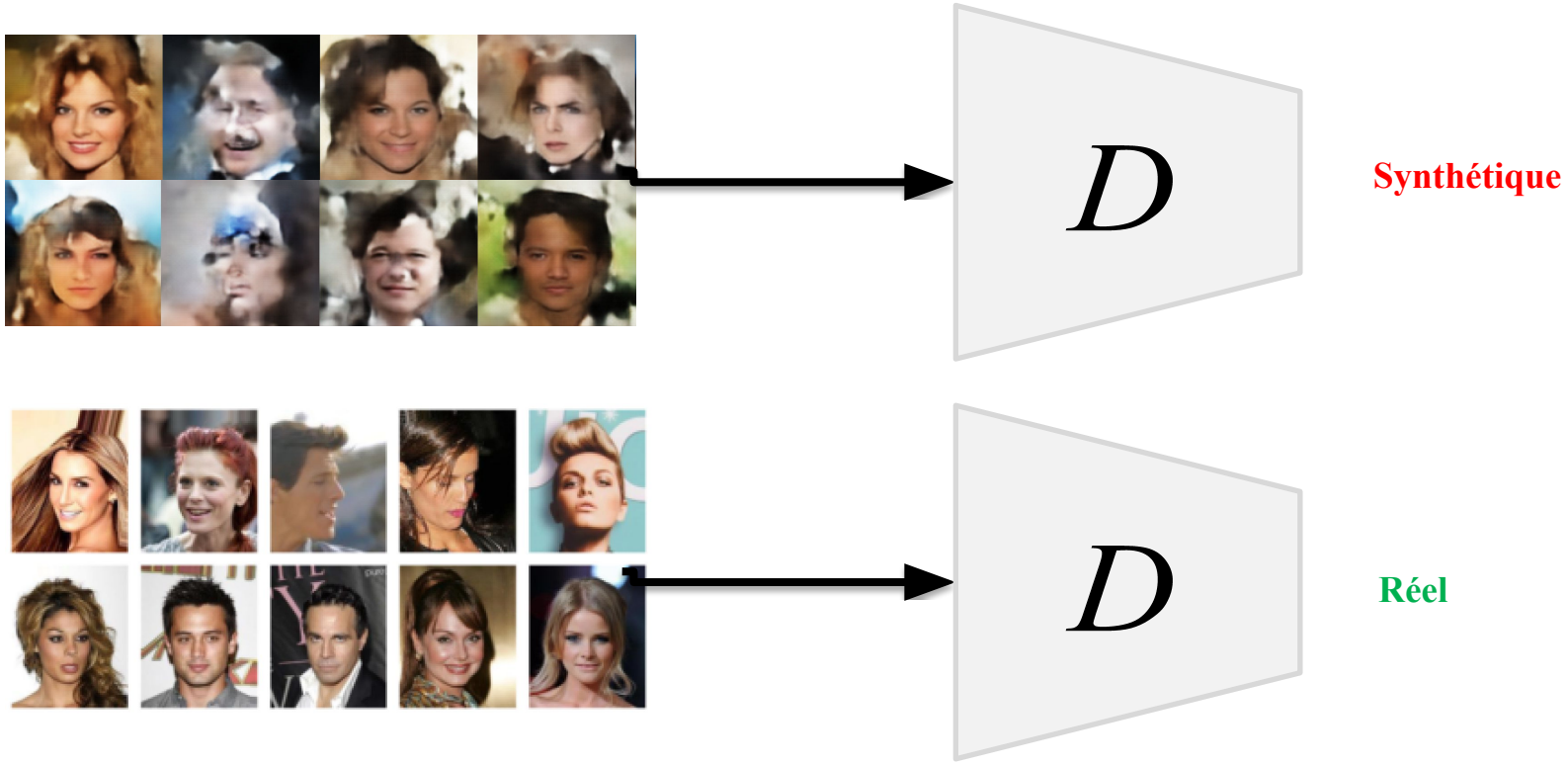


$t = 1$ ('*réel*')

Issues d'une vraie BD

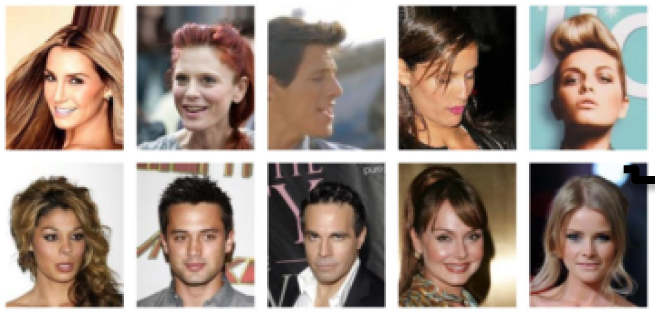
Deux réseaux aux **objectifs différents** :

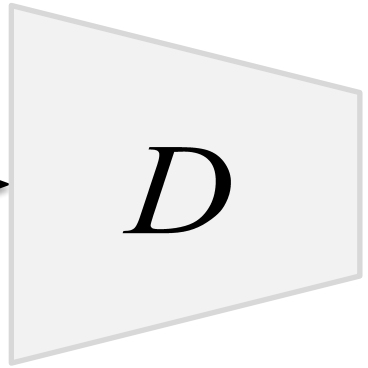
Discriminateur : différencie les images synthétiques des images réelles

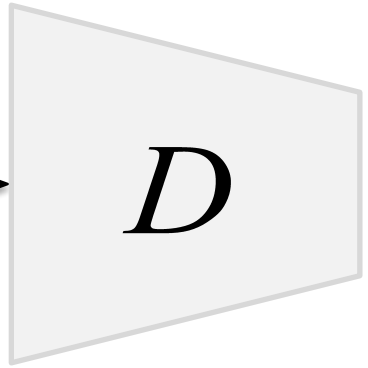


Discriminateur : classifieur binaire (régression logistique)

=> Perte **l'entropie croisée**





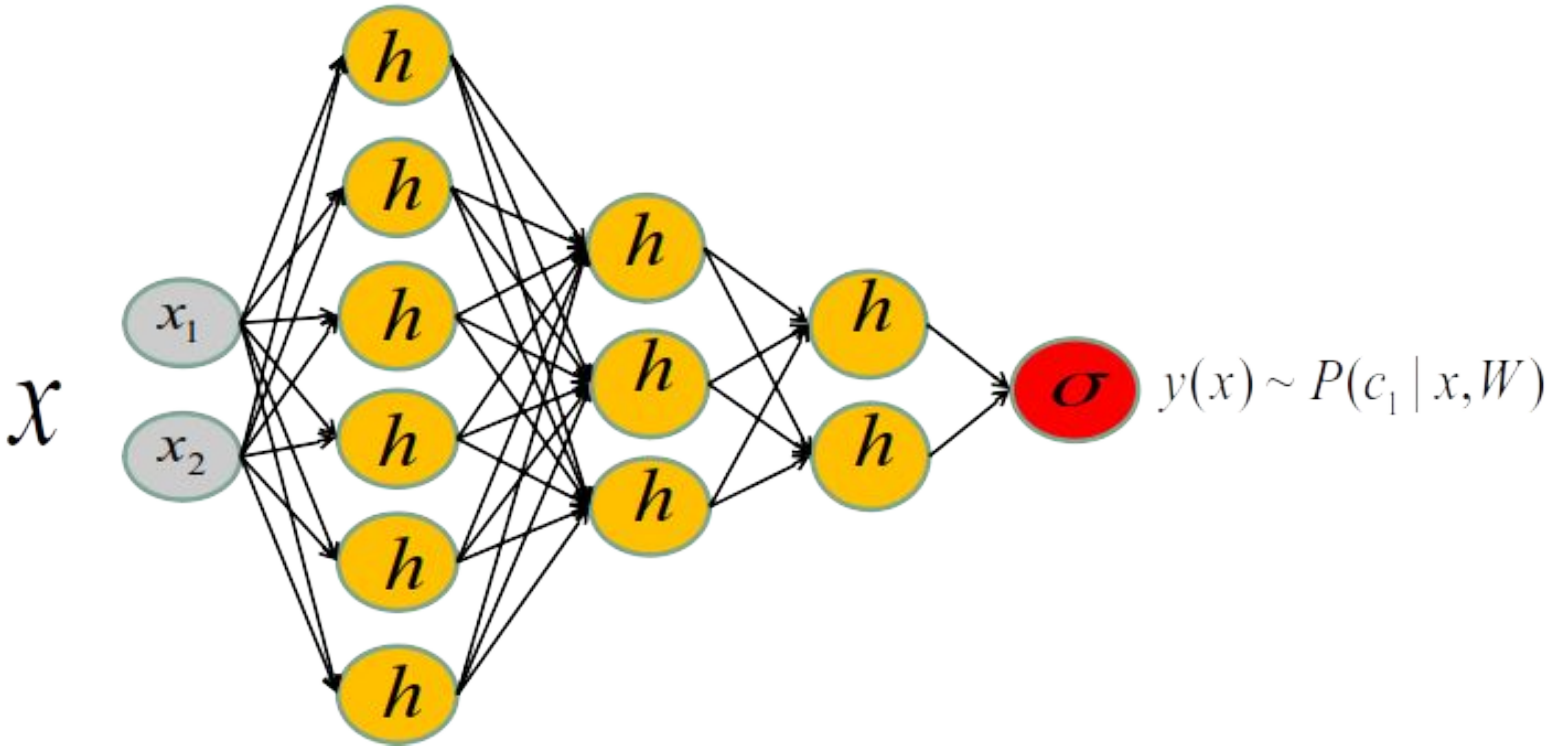


Synthétique

Réel

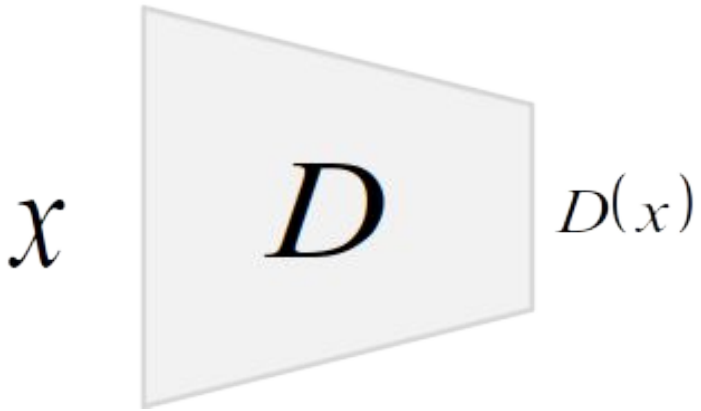
Rappel, entropie croisée pour une régression logistique binaire:

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



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

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



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

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

$G(z)$

x

D

$D(G(z))$

$D(x)$

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

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

$G(z)$

x

D

$-\ln(1 - D(G(z)))$

$-\ln(D(x))$

Rappel: Espérance mathématique et approximation Monte Carlo

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

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

Rappel: Espérance mathématique et approximation Monte Carlo

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

$$\approx \frac{1}{N} \sum_{i=1}^{N} x_i \qquad \text{où } x_i \sim p(x)$$

approximation Monte Carlo

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

$$\approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) \qquad \text{où } x_i \sim p(x)$$

Rappel: Espérance mathématique et estimateur Monte Carlo

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

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

Rappel: Espérance mathématique et estimateur Monte Carlo

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

Approximation Monte-Carlo

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

Objectif du discriminateur

Paramètres du discriminateur

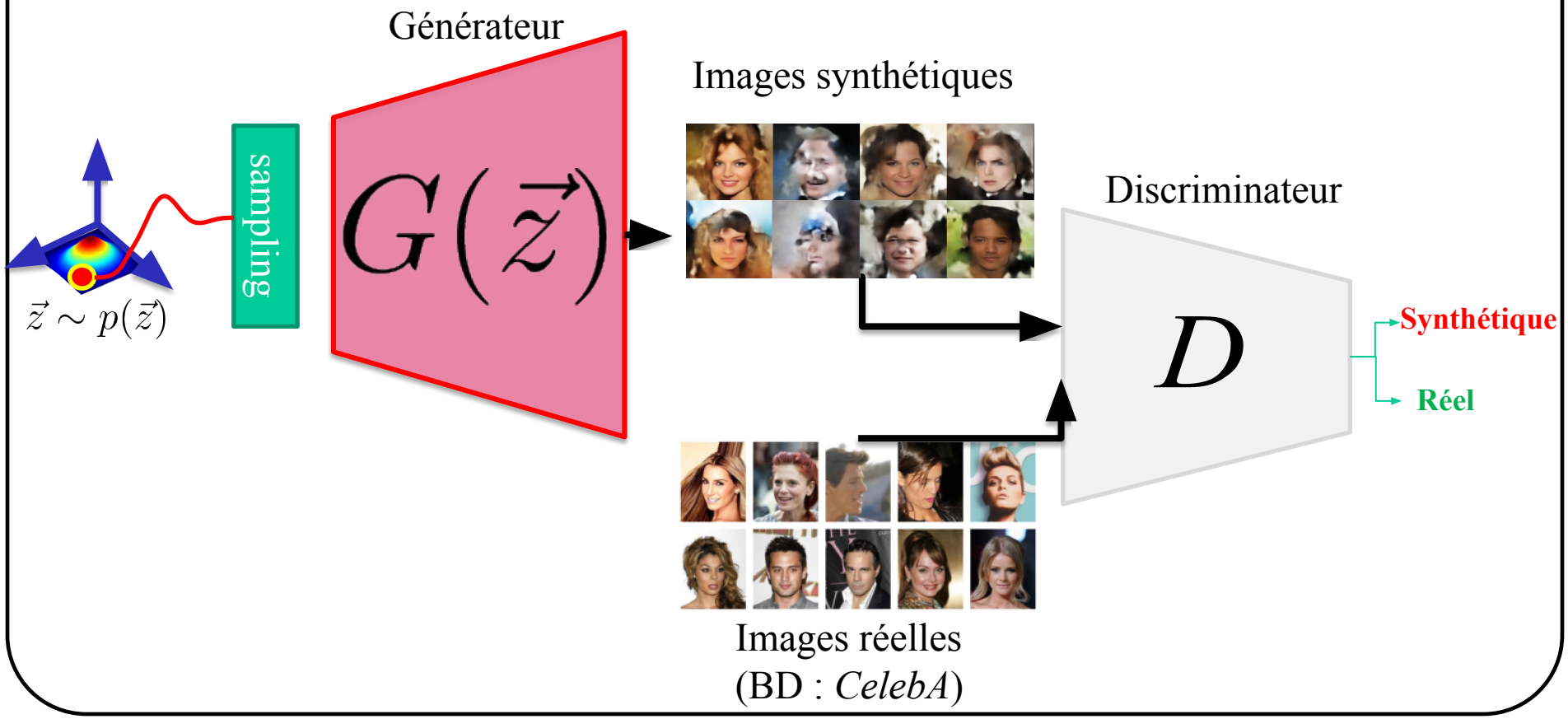
$$W_D = \arg\min_{W_D} - IE_{x \sim P_{reel}}^{\rightarrow} \left[\ln(D(x)) \right] - IE_{z \sim P_z}^{\rightarrow} \left[\ln(1 - D(G(z))) \right]$$

Ou encore, de façon équivalente

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

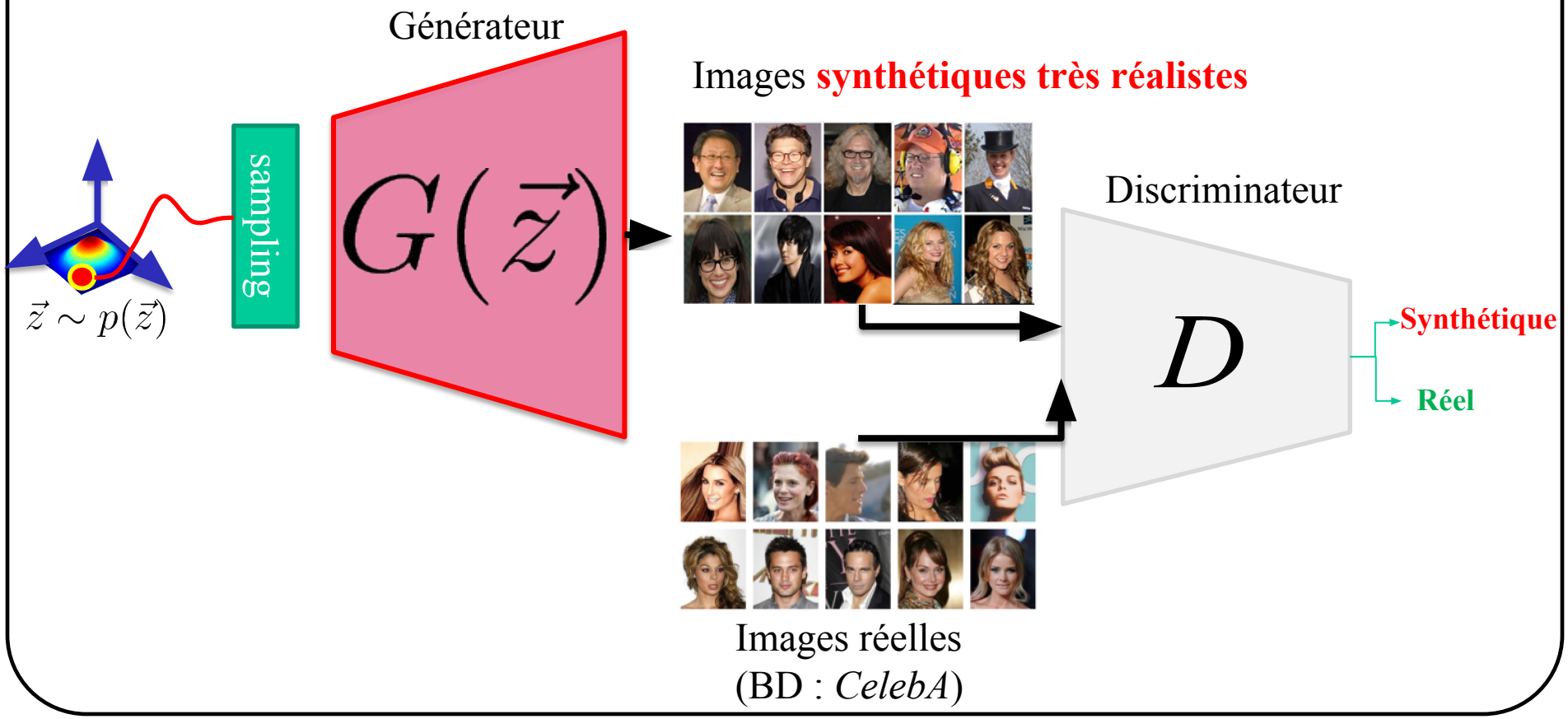
Objectif du générateur

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



Objectif du générateur

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



Objectif du discriminateur :

bien distinguer les images réelles des images synthétiques

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

Objectif du générateur :

produire des images synthétiques indistinguables des images réelles

$$W_G = \arg\min_{W_G} IE_{z \sim P_z} \left[\ln(1 - D(G(z))) \right]$$

« Two player » mini-max game

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

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

« Two player » mini-max game

Discriminateur veux D(x) = 1 pour les vrais données

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

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

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

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

NOTE

dans les faits, on ne minimise pas cette loss

$$W_G = \arg\min_{W_G} IE_{z\sim P_z}\left[\ln(1 - D(G(z)))\right]$$

on maximise plutôt celle-ci

$$W_G = \arg\max_{W_G} IE_{z\sim P_z}\left[\ln(D(G(z)))\right]$$

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

for number of training iterations **do**

for k steps **do**

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

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

end for

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

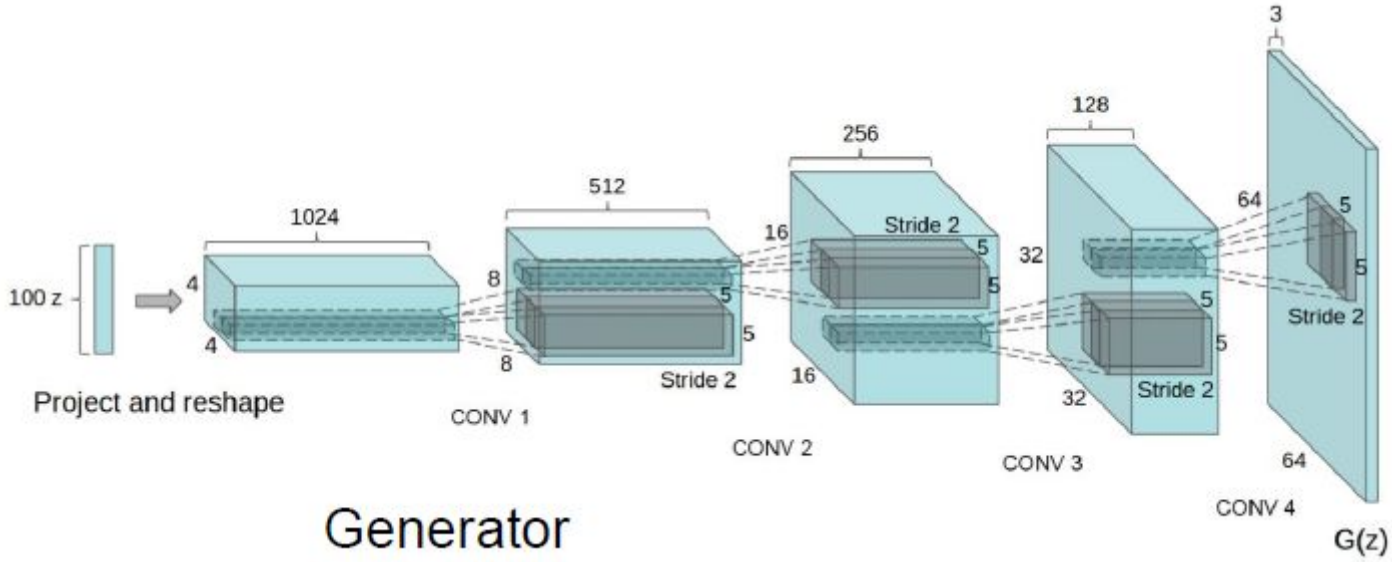
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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)

Deep Convolution Generative Adversarial Net (DCGAN)



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

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

Deep Convolution Generative Adversarial Net (DCGAN)

Recommandations discriminateur

- C
- F

https://github.com/soumith/ganhacks

Rec

- C
- L

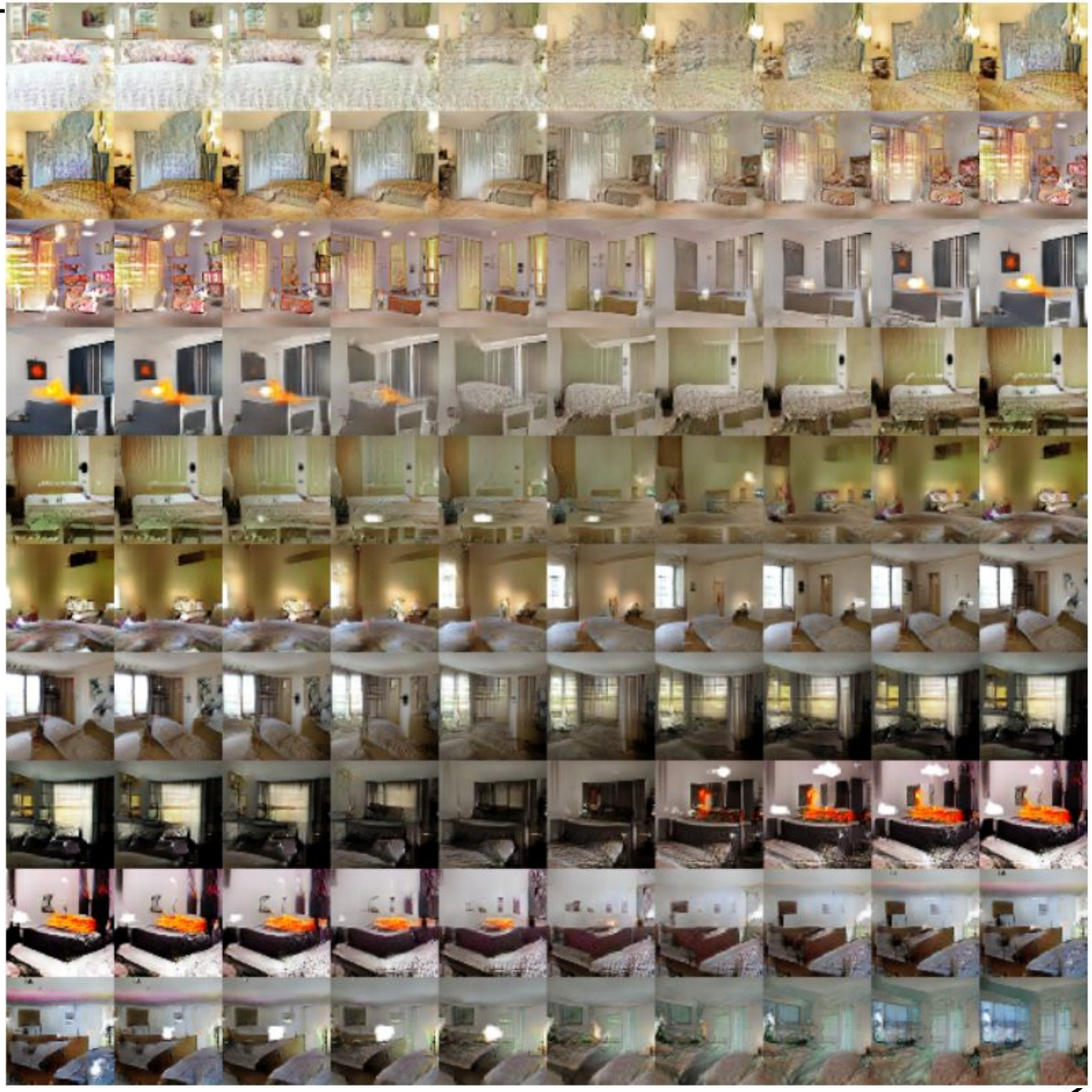
Autre recommandations

- BatchNorm partout
- Pas de FC, juste des conv

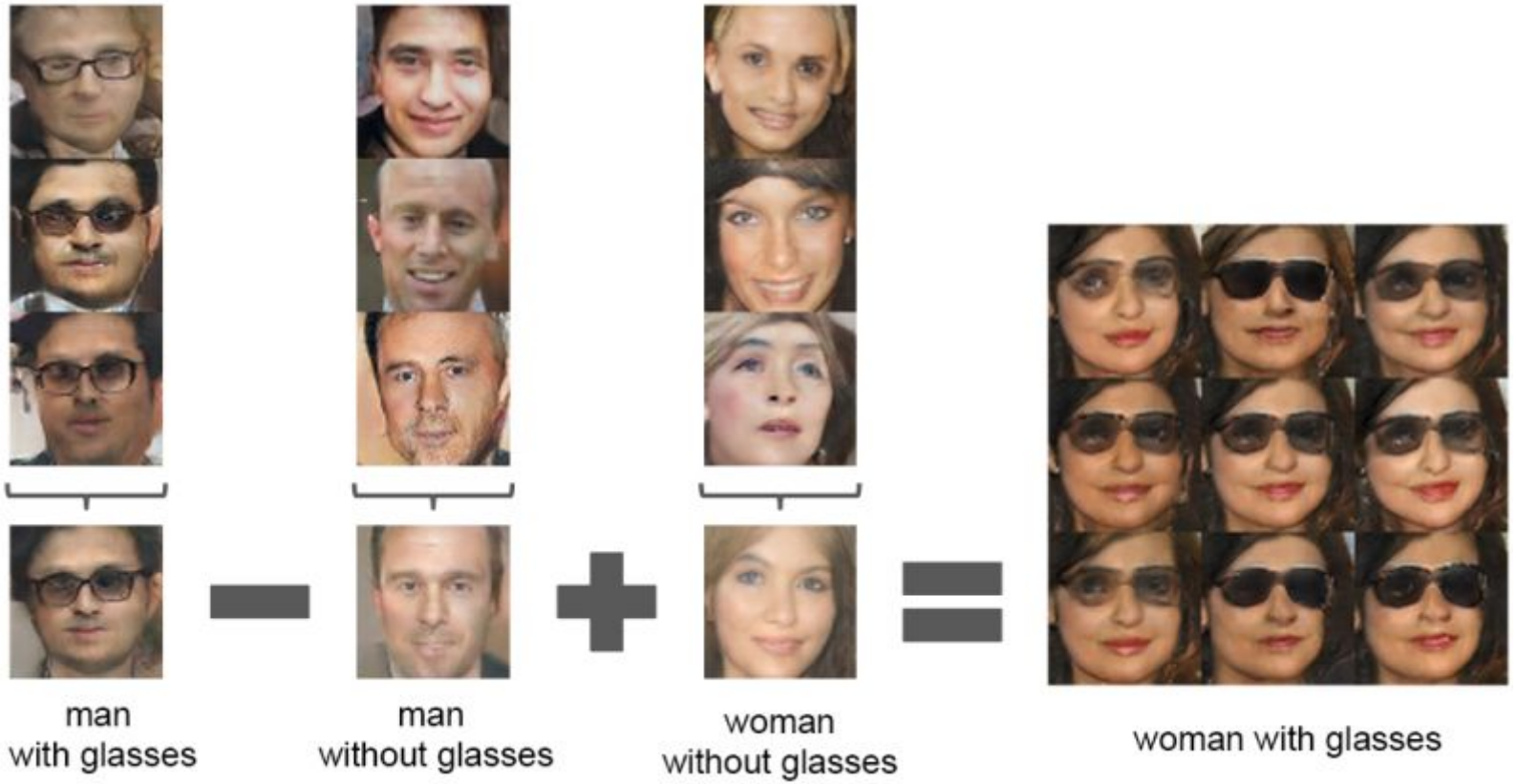
Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

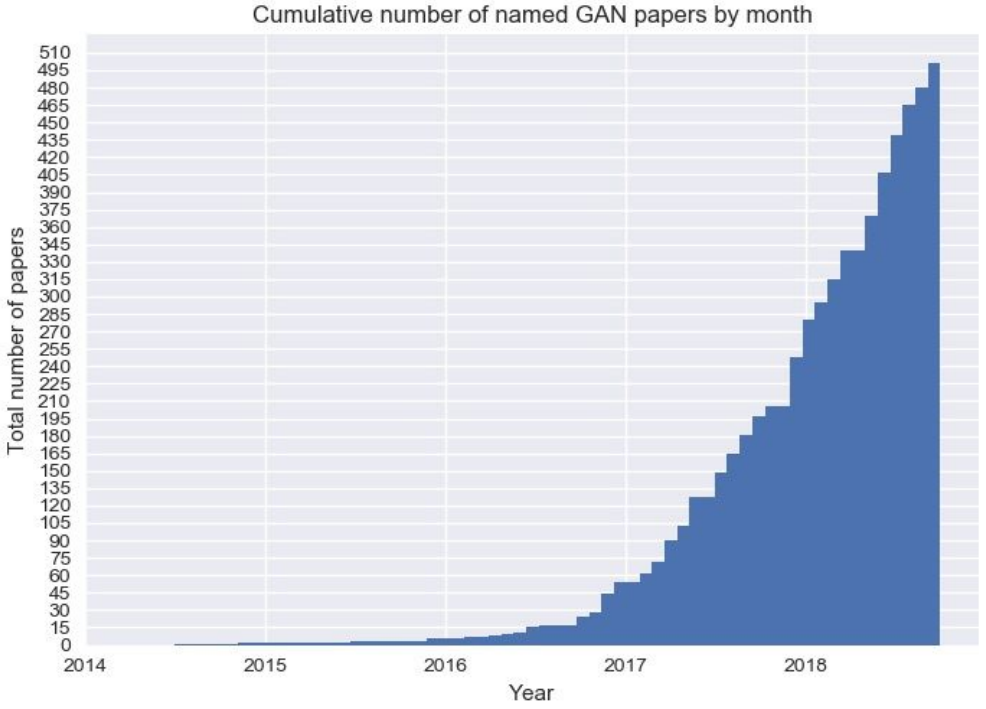
Deep Convolution Generative Adversarial Net (DCGAN)

Interpolation entre 9 vecteurs latents aléatoires



"Vector arithmetic for visual concepts"





https://github.com/hindupuravinash/the-gan-zoo

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*
  - ….

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

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 générer tout le temps la même image qui bat le discriminateur

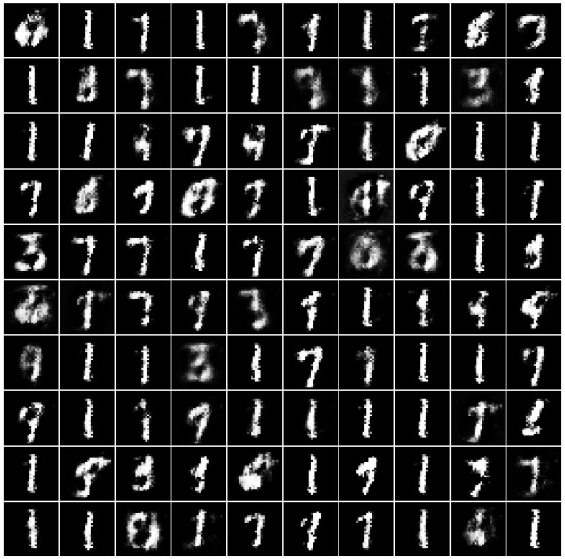
Problèmes d'instabilité

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

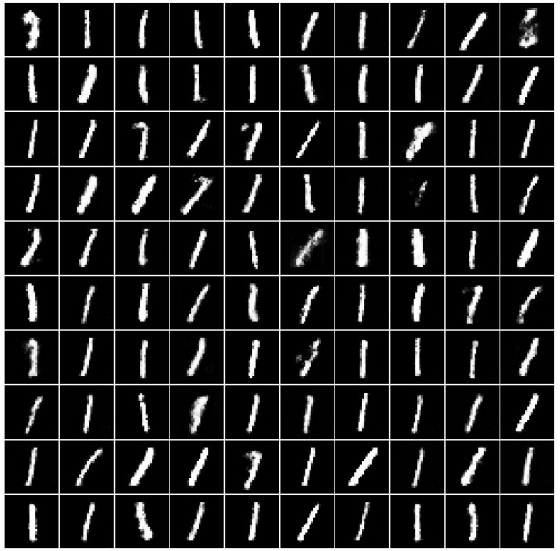
Le gén
le disc

bat

Epoch 7



Epoch 21



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

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**

Beaucoup d’artéfacts

LS GAN

Problème des GANs de base

"**sigmoïde** de sortie" **oublie** les exemples correctement classifiés et loin du plan de séparation

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

$$\min_G V(G) = \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \textbf{ or } \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})} - [\log D(G(\boldsymbol{z}))]$$

"Least Squares GAN" Mao et al. ICCV'17

LS GAN

Problème des GANs de base

"sigm séparation

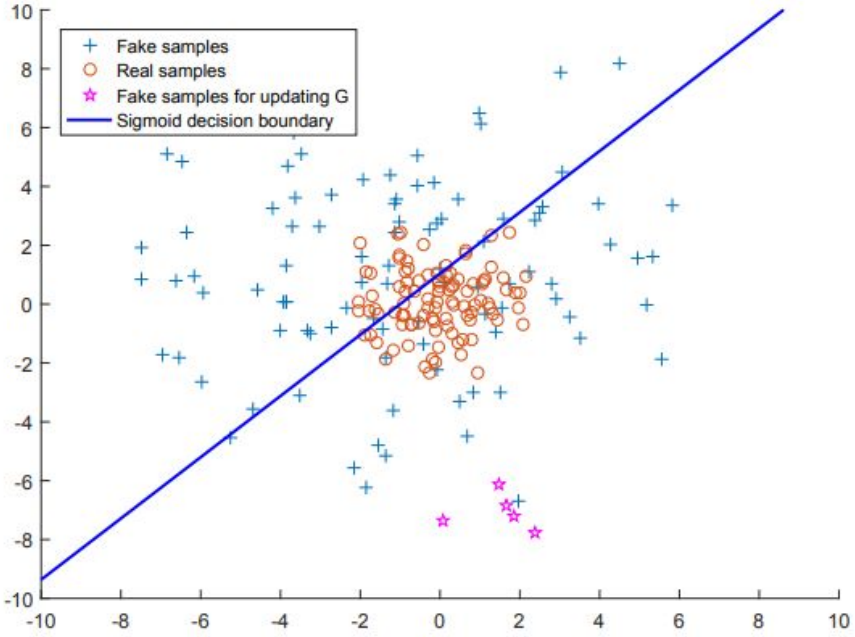
Le discriminateur ne s'entraîne plus lorsque les images synthétiques sont très différentes des images réelles

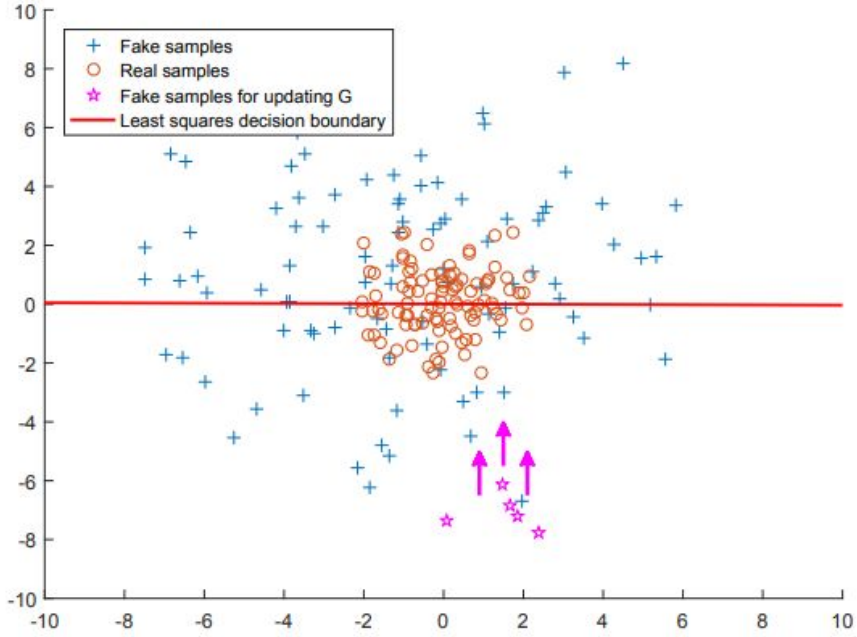
$$\min_G V(G) = \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \textbf{ or } \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})} - [\log D(G(\boldsymbol{z}))]$$

"Least Squares GAN" Mao et al. ICCV'17

LS GAN

Problème des GANs de base





"Least Squares GAN" Mao et al. ICCV'17

LS GAN

Quand on utilise **l'erreur quadratique**, même les exemples « trop biens classifiés » contribuent aux gradients du générateur. Le but est de rapprocher les images synthétiques des images réelles

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

$$\min_G V(G) = \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \text{ or } \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})} - [\log D(G(\boldsymbol{z}))]$$

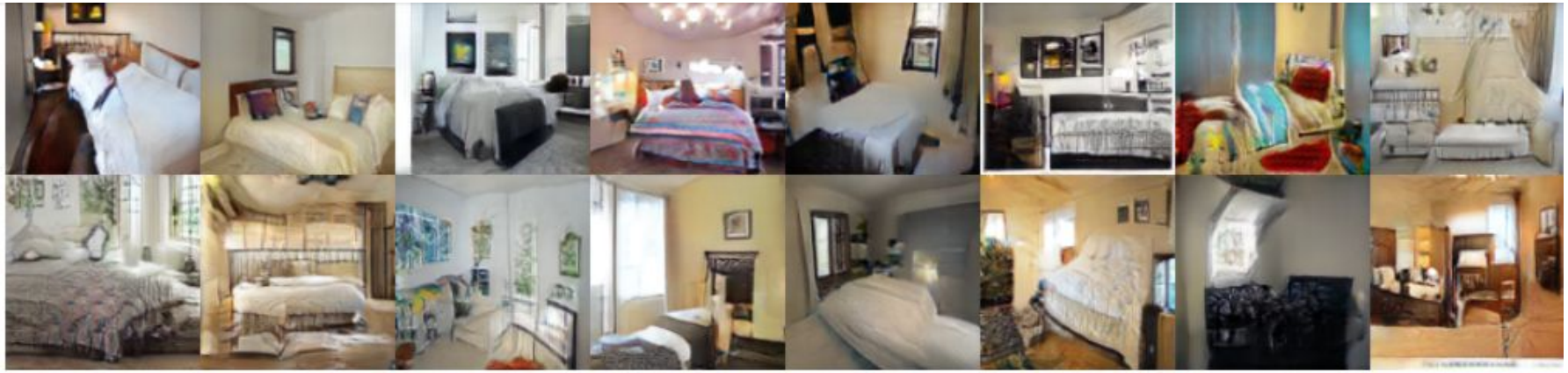
Pour LS GAN, la sortie du réseau n'est plus une sigmoïde

$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x}) - 1)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z})))^2\right]$$

$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z})) - 1)^2\right],$$

"Least Squares GAN" Mao et al. ICCV'17

LS GAN



(a) Generated by LSGANs.

(b) Generated by DCGANs (Reported in [11]).

"Least Squares GAN" Mao et al. ICCV'17

LS GAN

(a) LSGANs.

(b) Regular GANs.

(c) LSGANs.

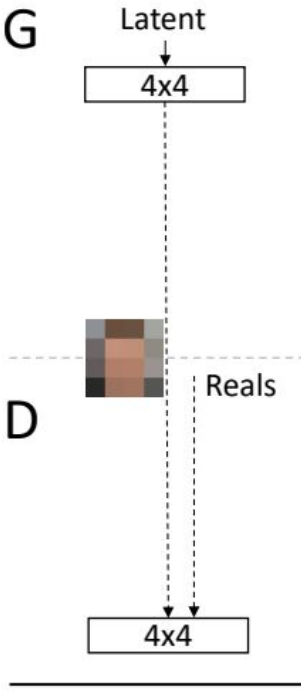
(d) Regular GANs.

"Least Squares GAN" Mao et al. ICCV'17

progressive GAN

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

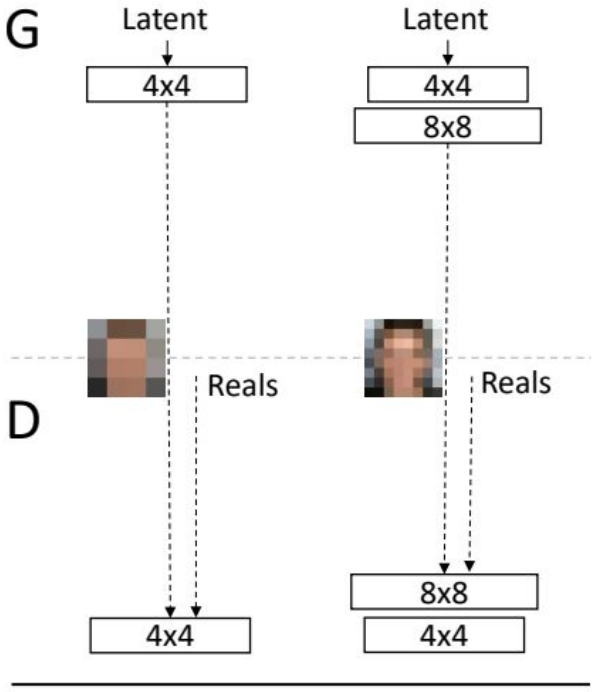
On commence avec des images de **faible résolution : 4x4 pixels**



"Progessive GAN" Karras et al. ICLR'18

progressive GAN

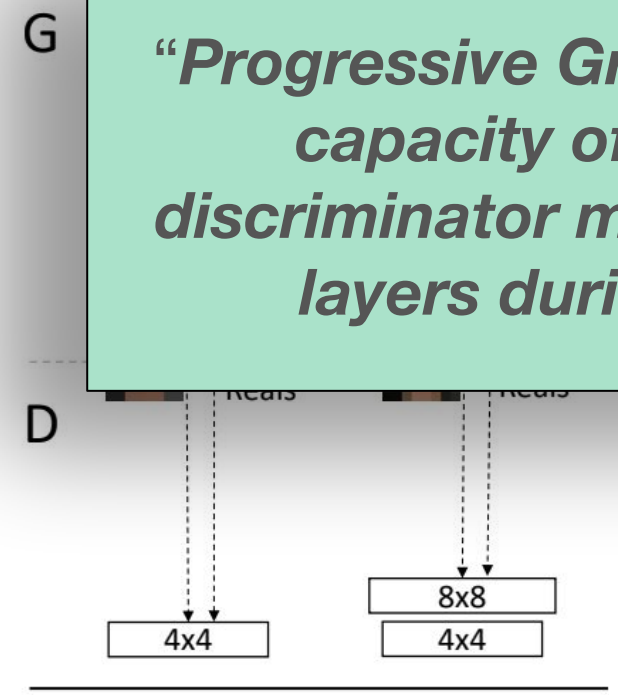
Et progressivement, on augmente la résolution de l'image



"Progessive GAN" Karras et al. ICLR'18

progressive GAN

Et progressivement, on augmente la résolution de l'image

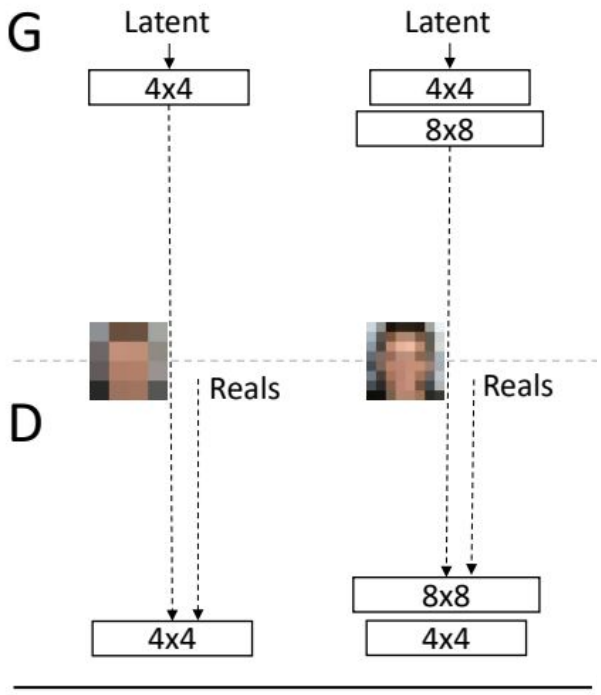


"Progressive Growing GAN requires that the capacity of both the generator and discriminator model be expanded by adding layers during the training process"

"Progessive GAN" Karras et al. ICLR'18

progressive GAN

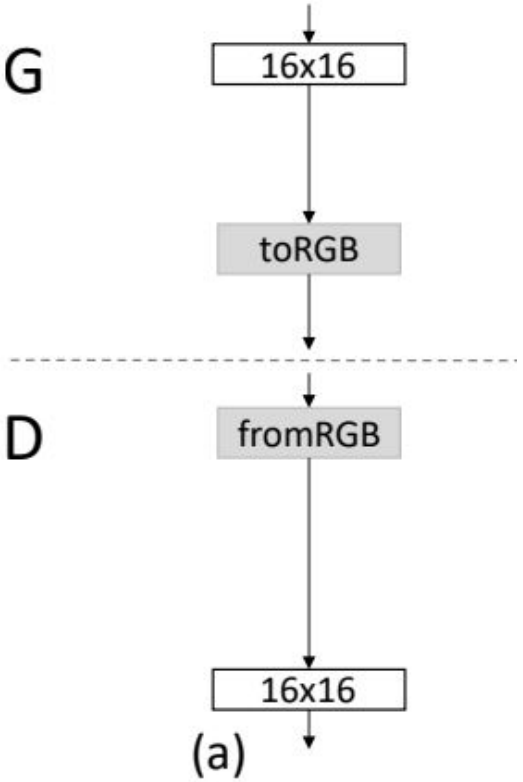
Et progressivement, chaque couche qu'on ajoute vient bonifier la couche précédente : cela se fait à l'aide d'une **opération « résiduelle ».**



"Progessive GAN" Karras et al. ICLR'18

Ajout de couches

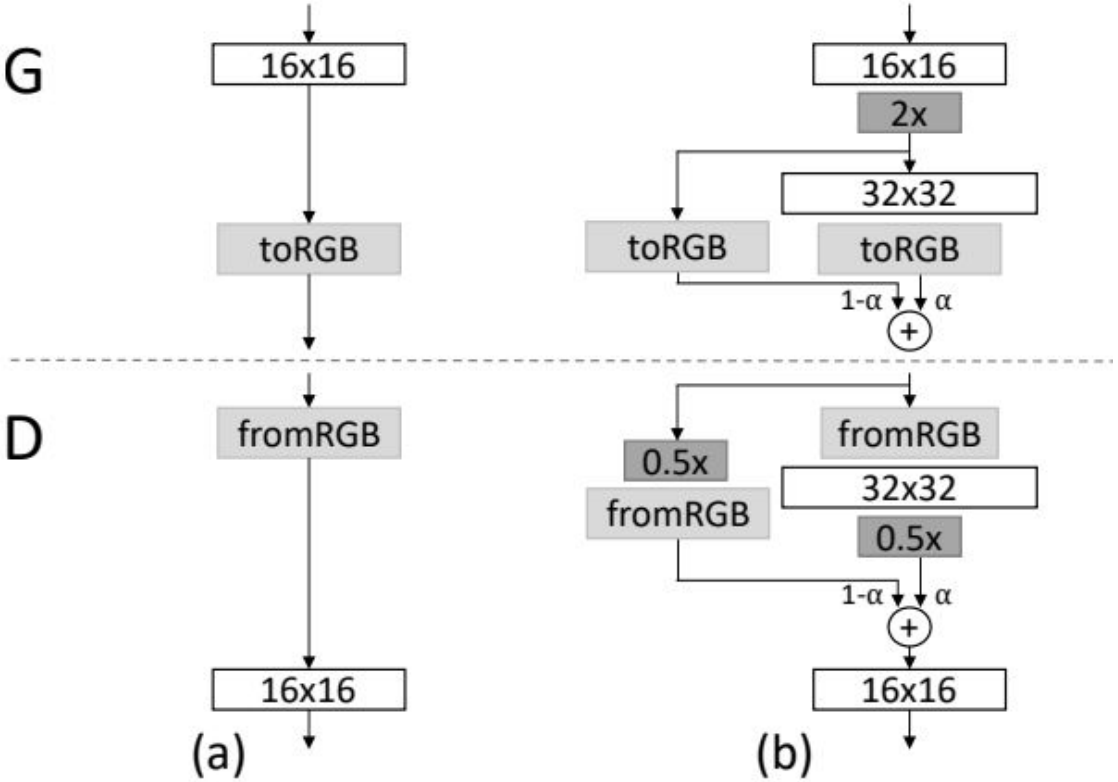
Lorsque **l'entraînement** d'une couche de résolution RxR (ici 16x16) est **terminé**…



"Progessive GAN" Karras et al. ICLR'18

Ajout de couches

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

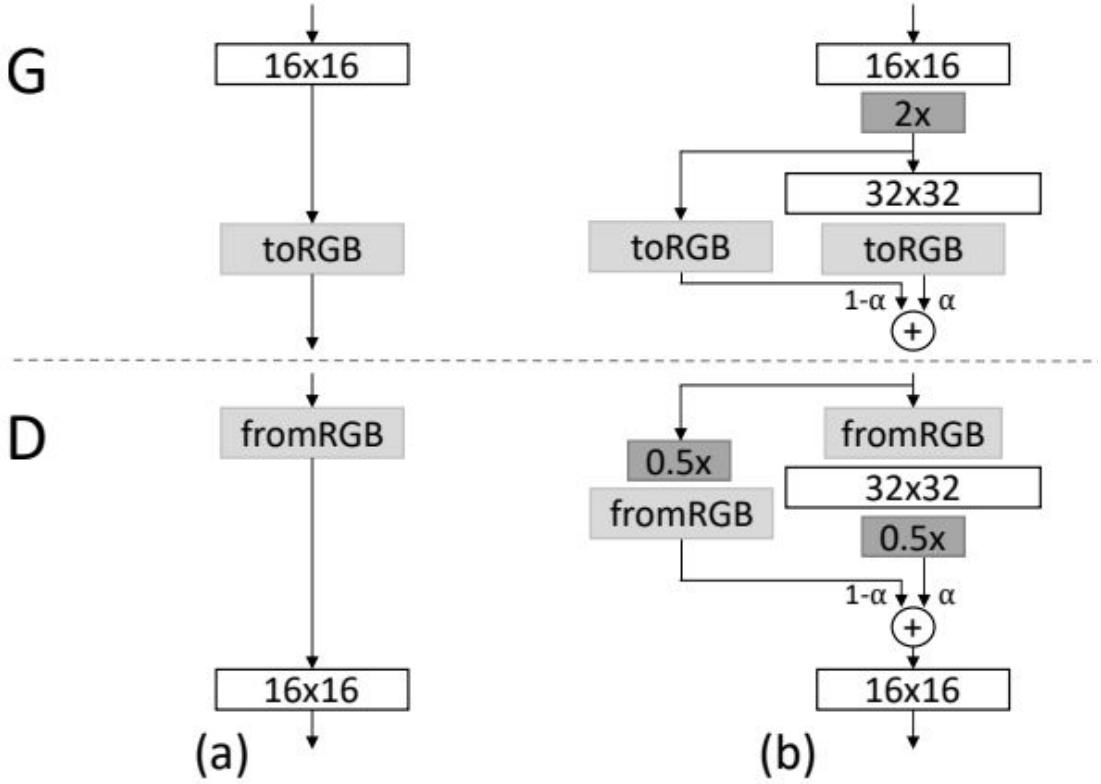


"Progessive GAN" Karras et al. ICLR'18

Ajout de couches

… mais pour éviter un choc, on ajoute une **composante résiduelle** comprenant un facteur α

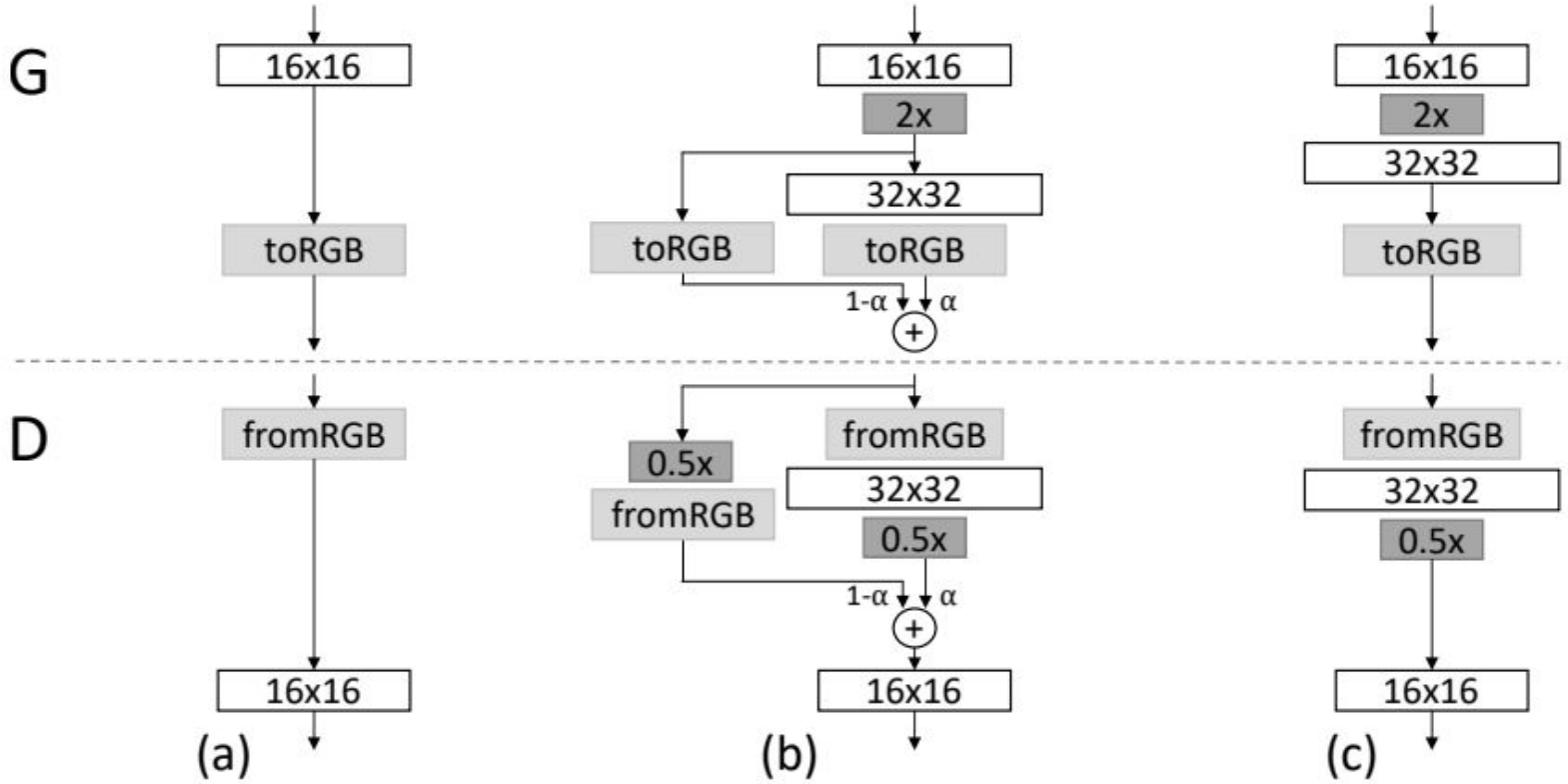
Au début de l'entrainement, $\alpha=0$ et progressivemnt α augmente pour atteindre $\alpha=1$ à la fin



"Progessive GAN" Karras et al. ICLR'18

Ajout de couches

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



"Progessive GAN" Karras et al. ICLR'18

"Progessive 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			**23.1M**

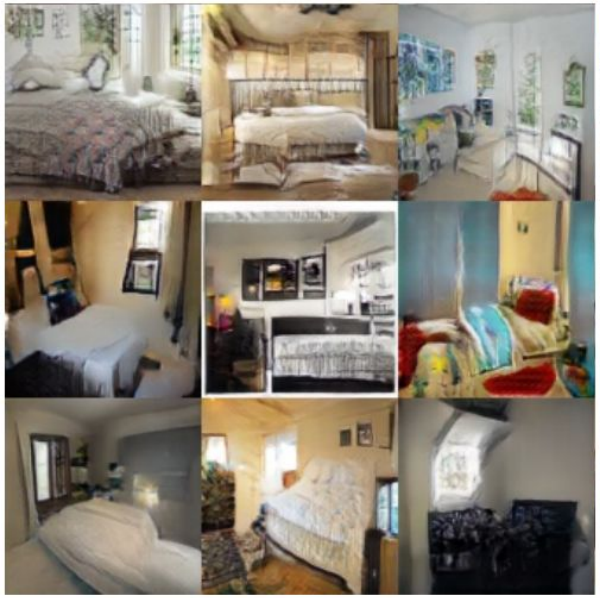
Discriminator	Act.	Output shape	Params
Input image	–	3 × 1024 × 1024	–
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	–	32 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	–	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			**23.1M**

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

"Progessive GAN" Karras et al. ICLR'18

https://youtu.be/XOxxPcy5Gr4

"Progessive GAN" Karras et al. ICLR'18



Mao et al. (2016b) (128 × 128)

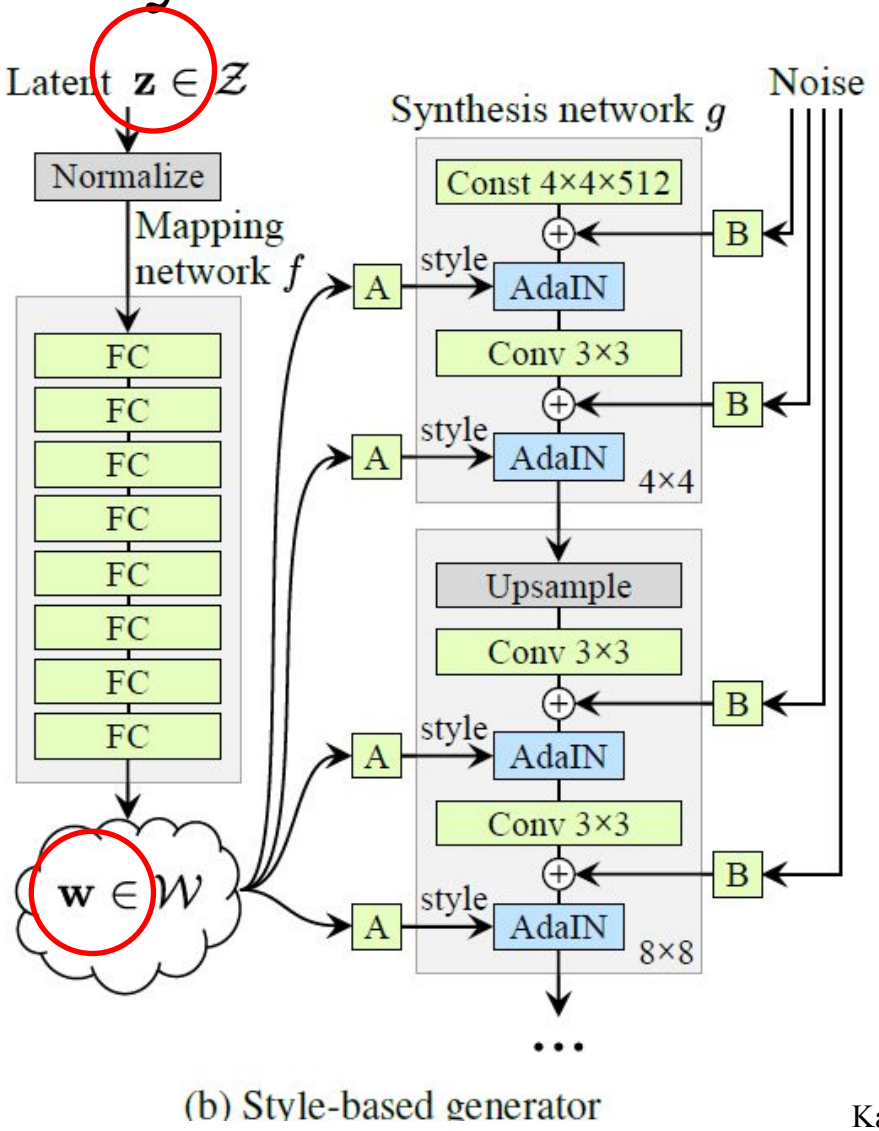
Gulrajani et al. (2017) (128 × 128)

Our (256 × 256)

LSGAN

PGAN

Style GAN

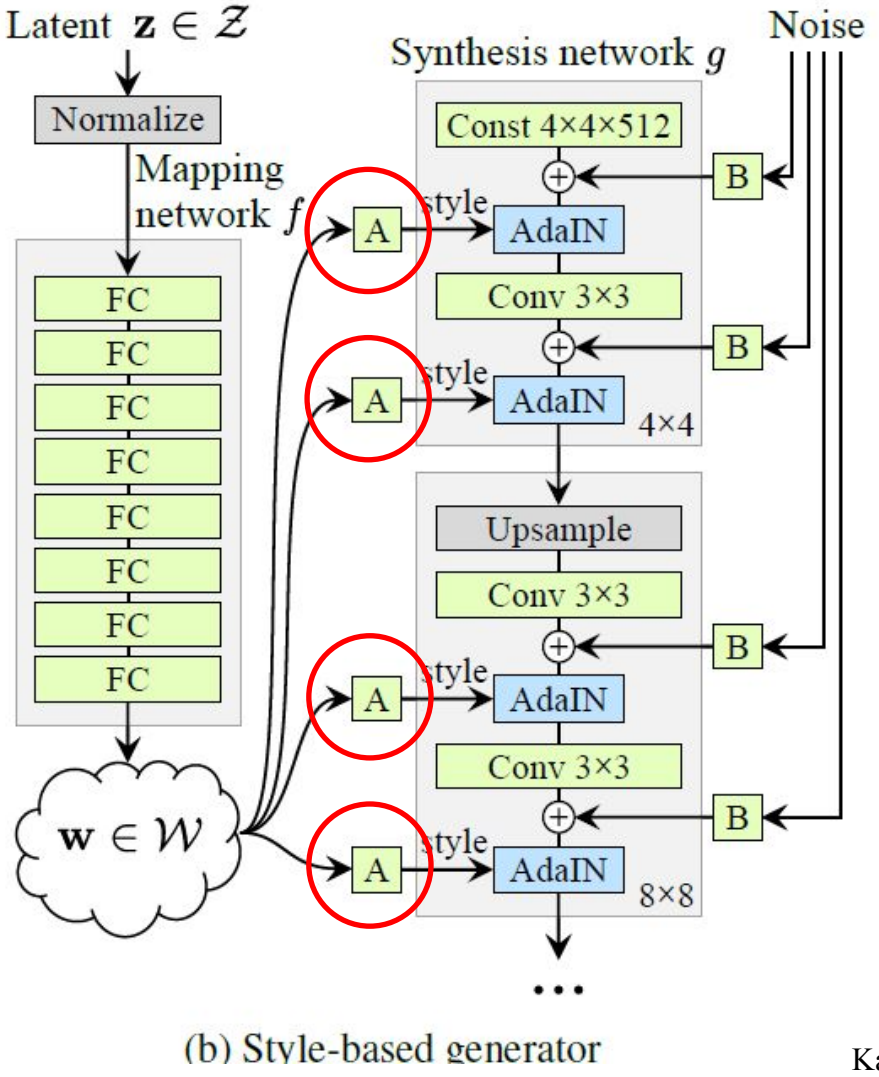


(b) Style-based generator

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

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

Style GAN



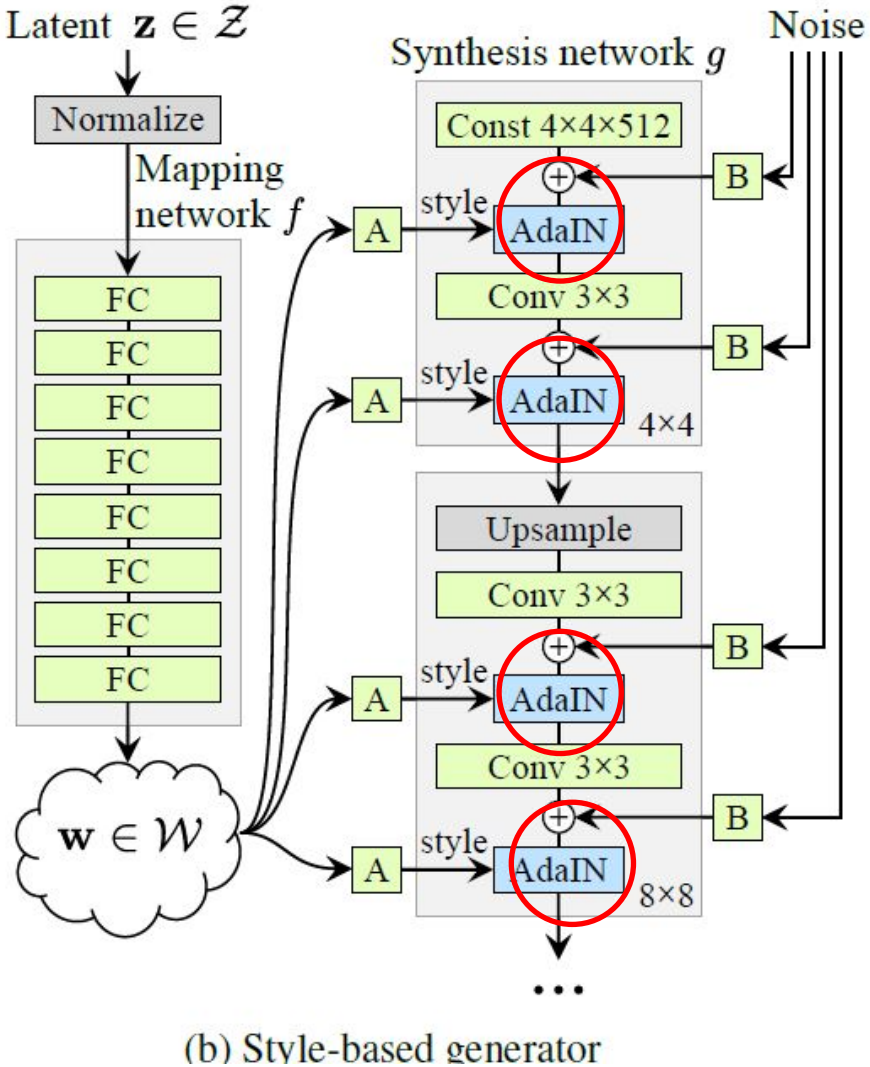
(b) Style-based generator

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

Et ce, à **chaque couche** du réseau

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

Style GAN



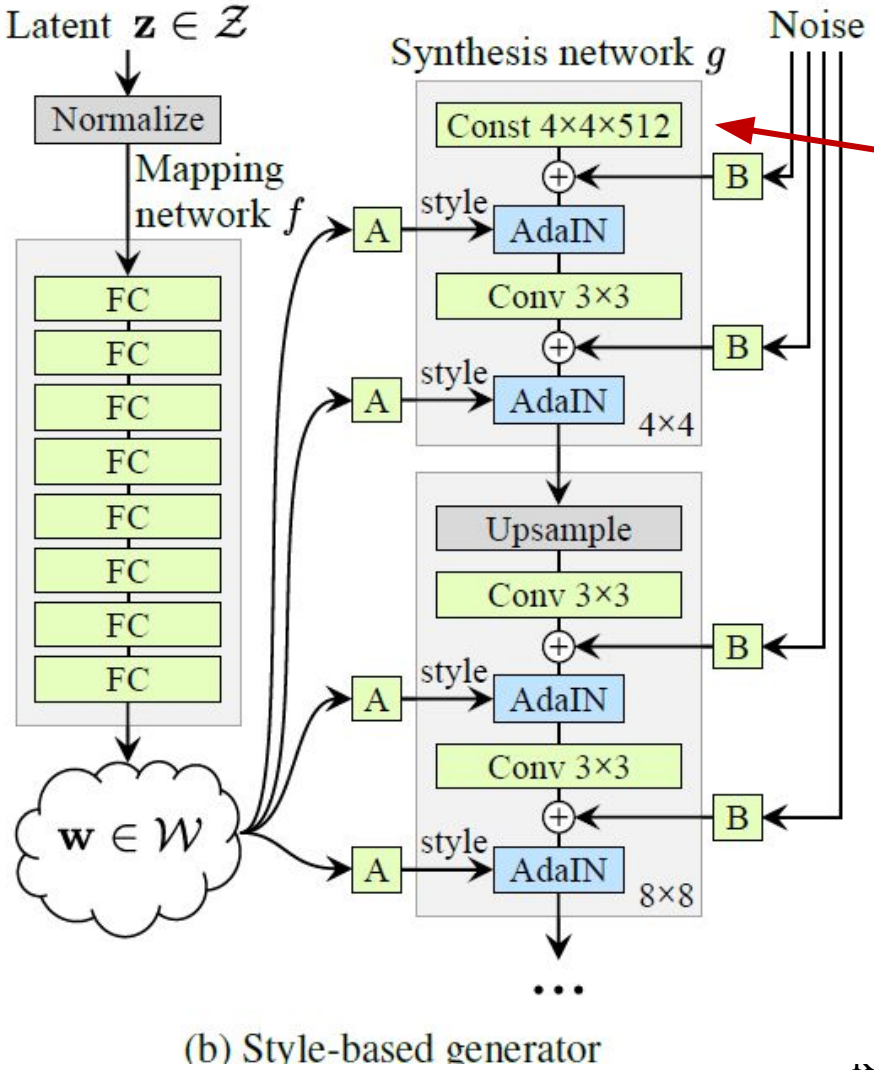
(b) Style-based generator

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 affines sont fournis par **w**

Style GAN

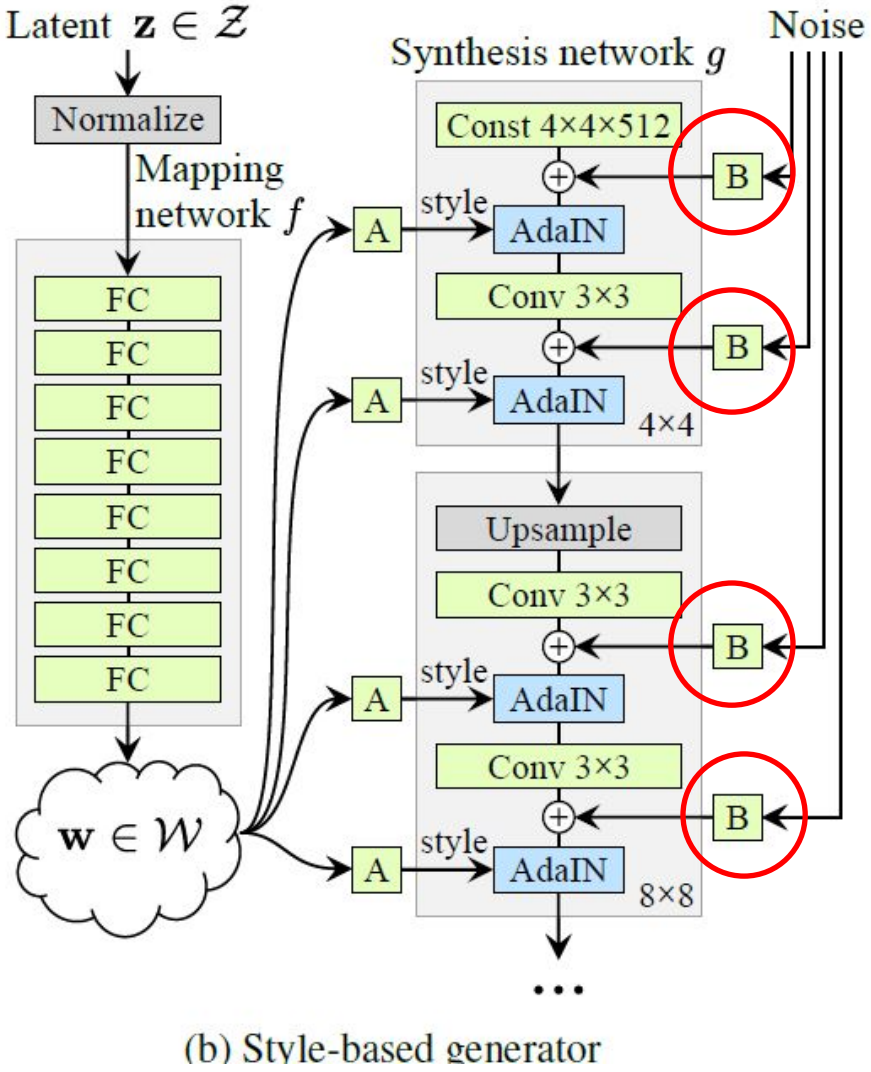


(b) Style-based generator

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

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

Style GAN



(b) Style-based generator

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

Effet du bruit

(a) Generated image (b) Stochastic variation (c) Standard deviation

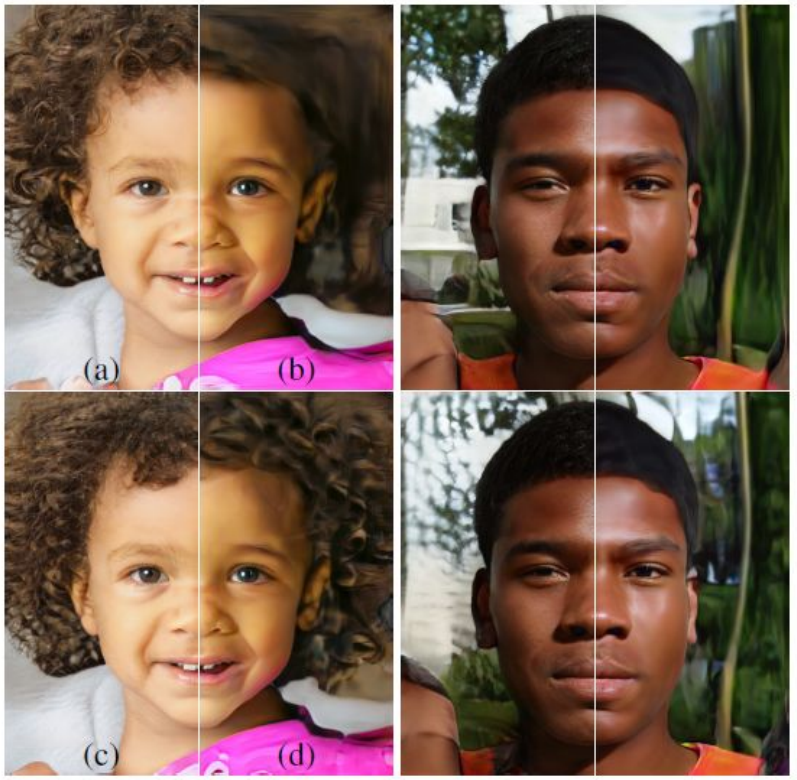
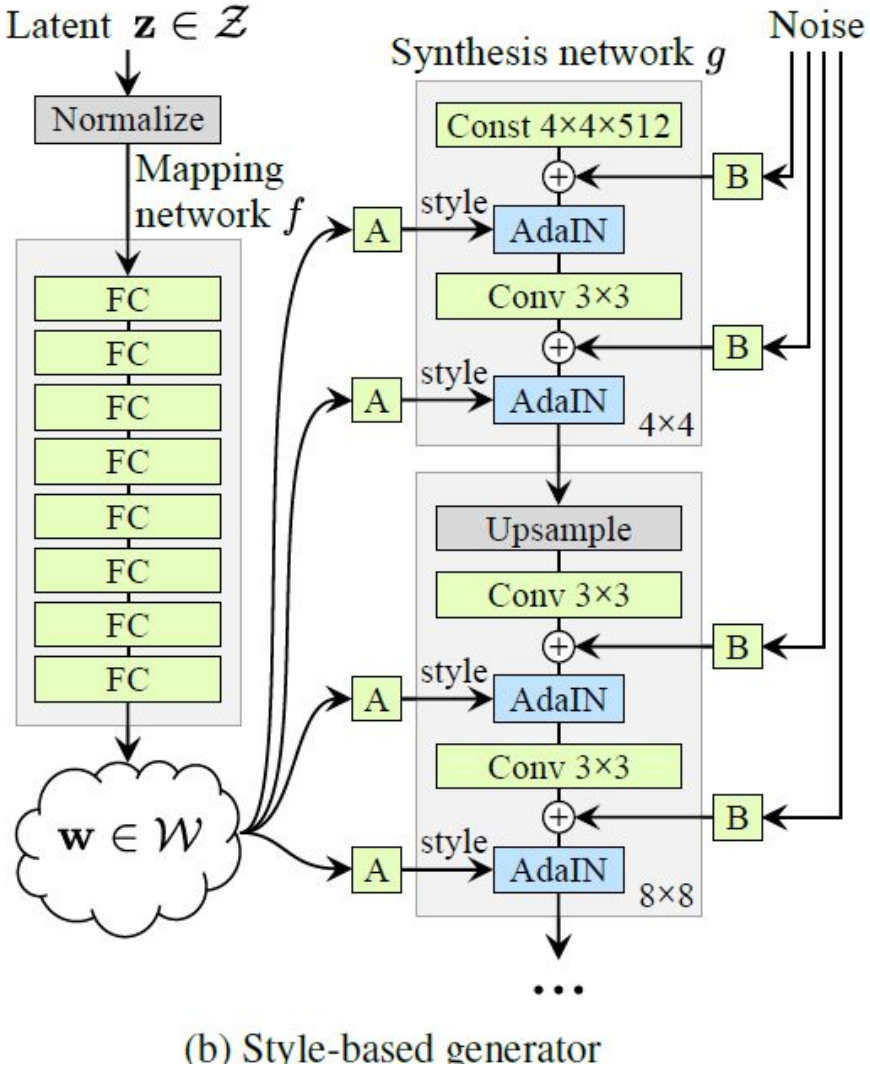


Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only (64^2 – 1024^2). (d) Noise in coarse layers only (4^2 – 32^2). We can see that the artificial omission of noise leads to featureless "painterly" look. 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

Style GAN



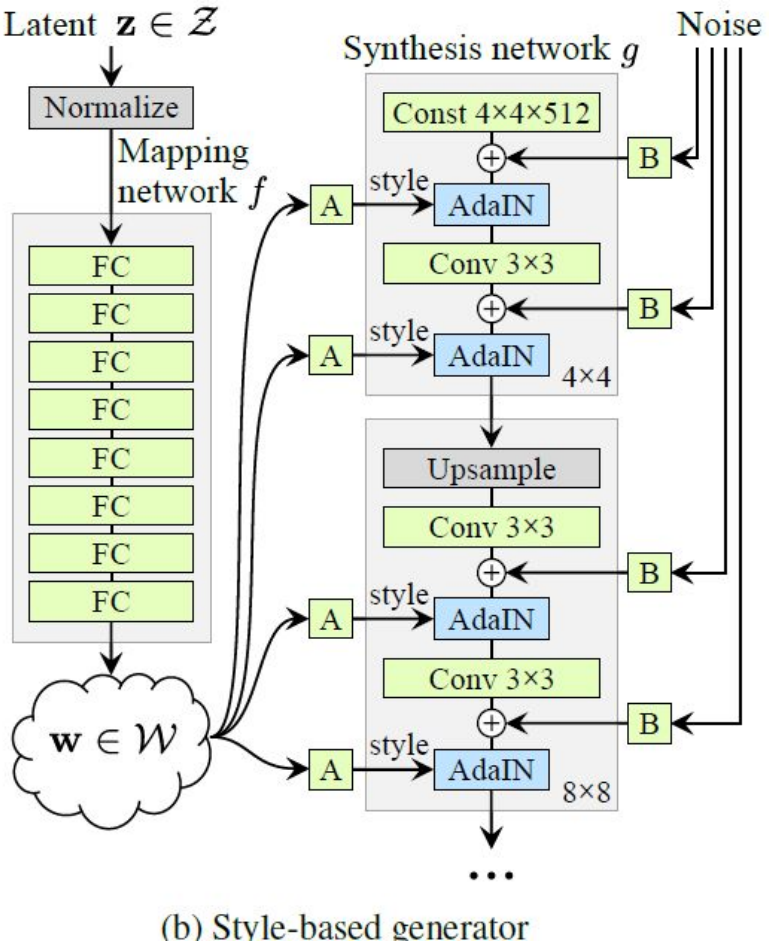
(b) Style-based generator

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

Entraînement progressif comme pour **progressive GAN**

Style GAN

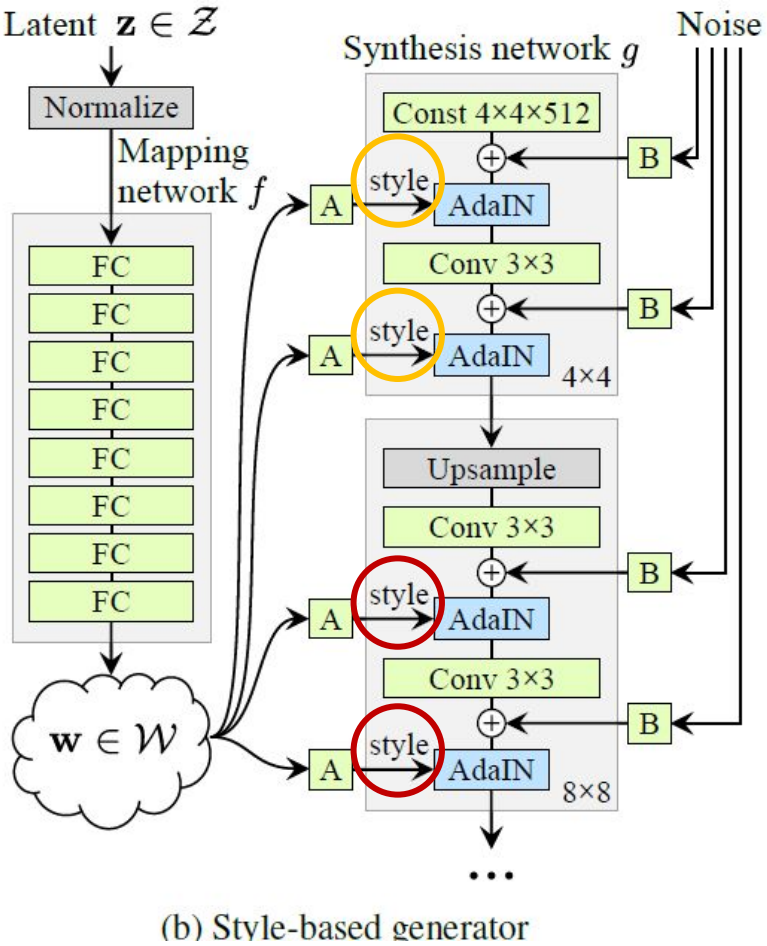
Style GAN



(b) Style-based generator

Qu'est-ce qui arrive si on fait varier le z entre les couches ?

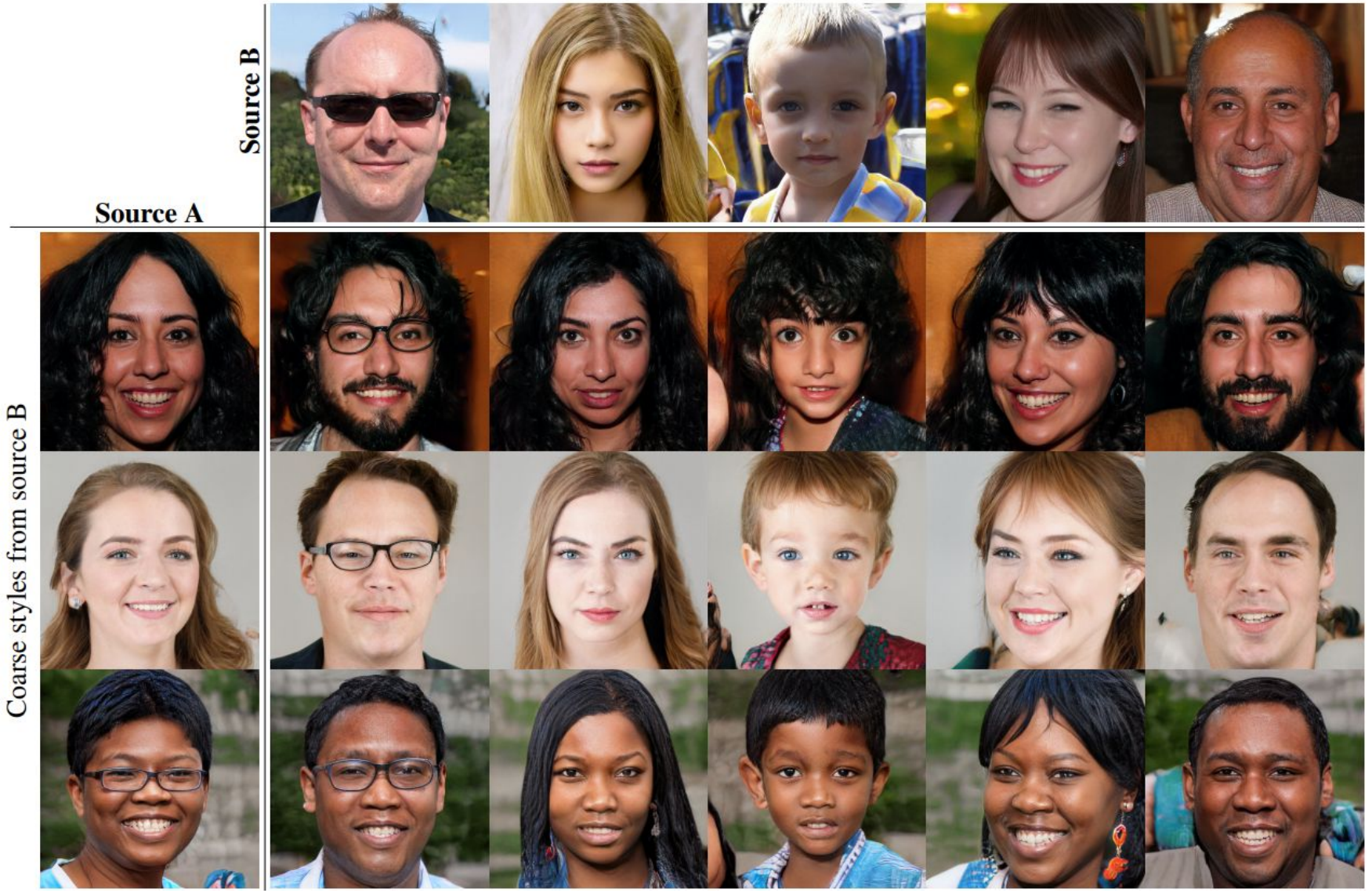
Style GAN



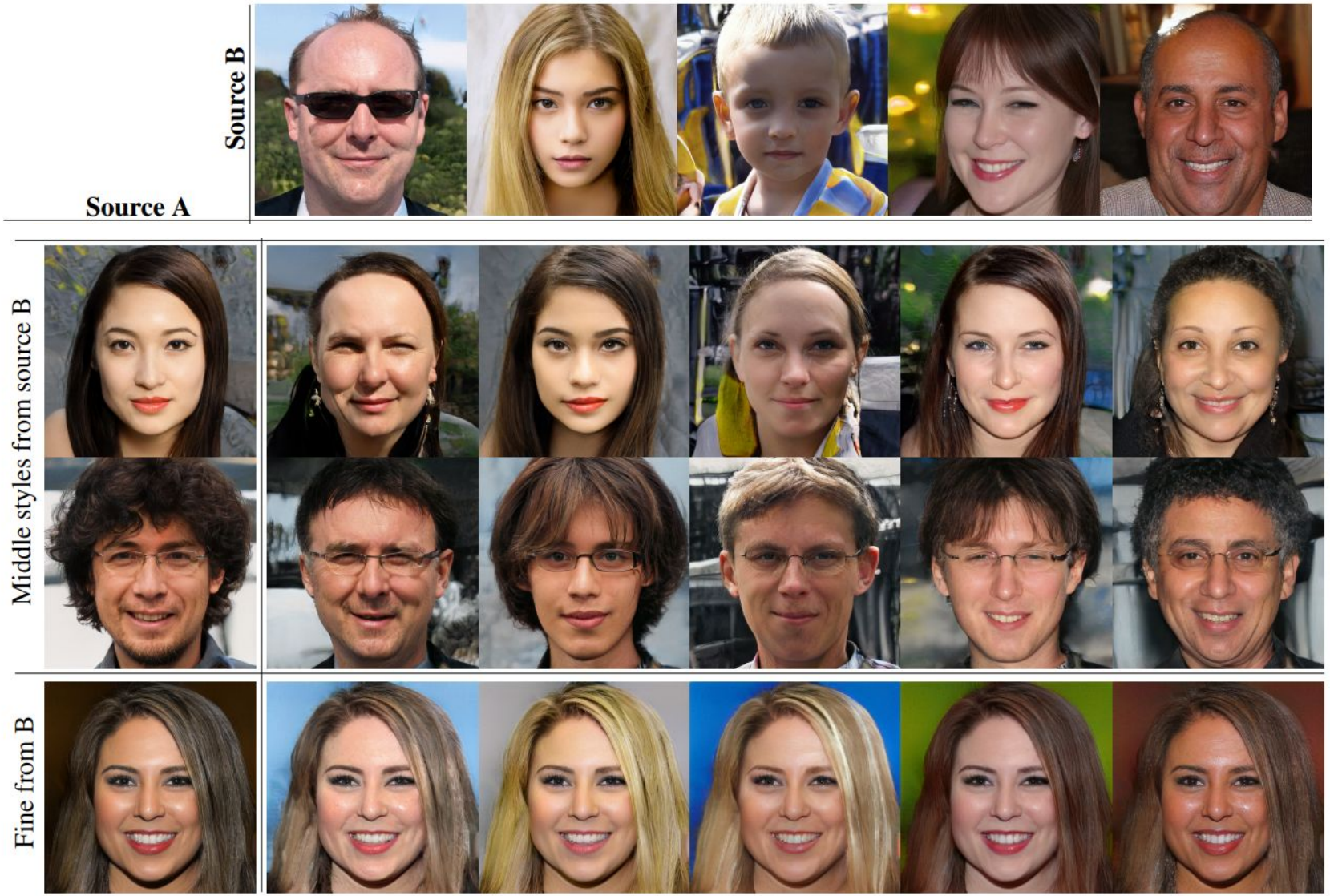
(b) Style-based generator

Qu'est-ce qui arrive si on fait varier le z entre les couches ?

Style GAN

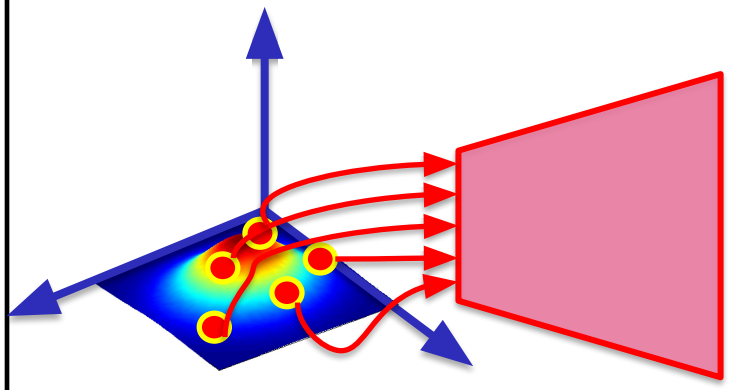


Style GAN



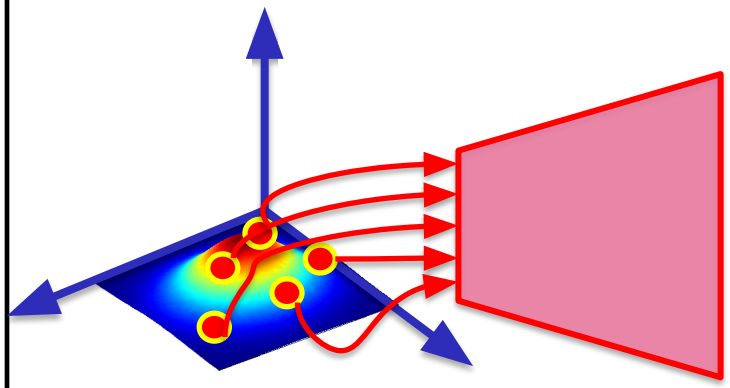
Défi avec les GAN

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



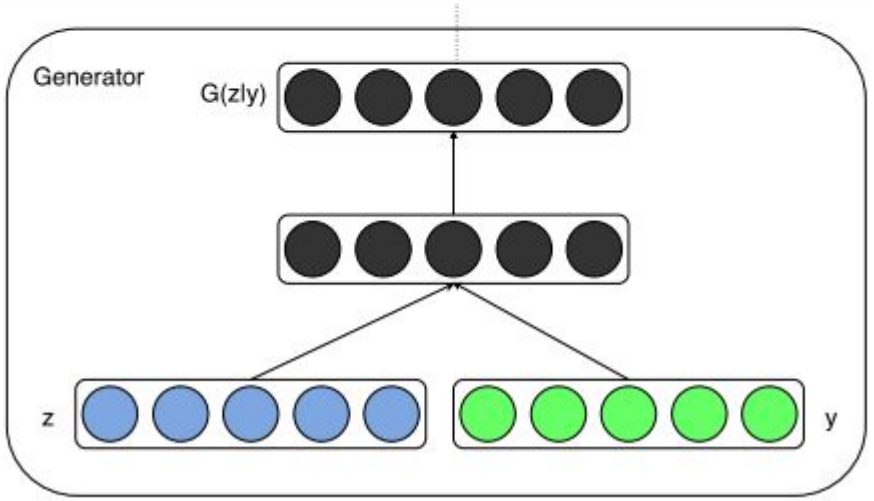
Défi avec les GAN

Question: comment générer des images de catégories prédéterminées? Ex. comment sélectionner 10 vecteurs latent afin de produire la séquence de caractères : 0,1,2,3,4,5,6,7,8,9?



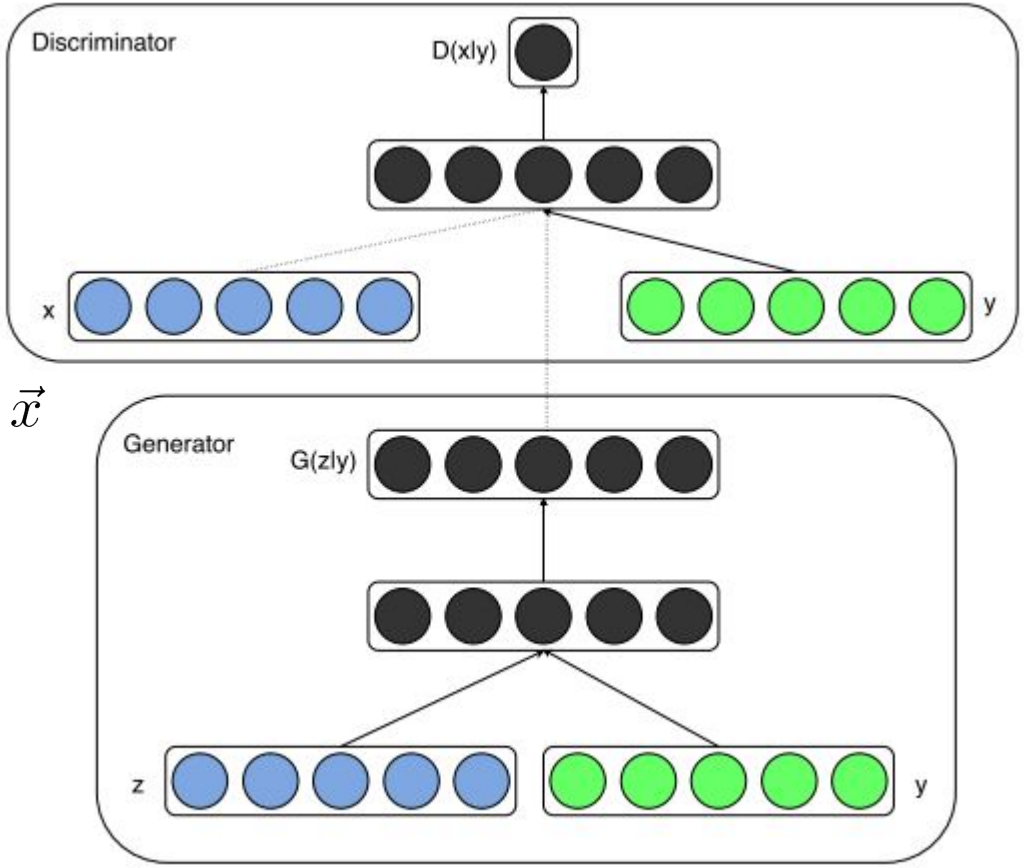
Gan conditionnel

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

Gan conditionnel

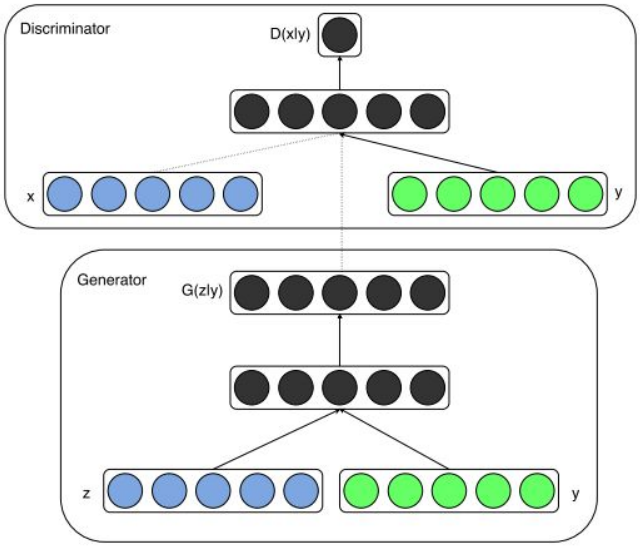


Et de discriminer une image $\vec{x}$

avec **le même « one-hot »** $\vec{y}$

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

Gan conditionnel



GAN de base

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

GAN conditionnel

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

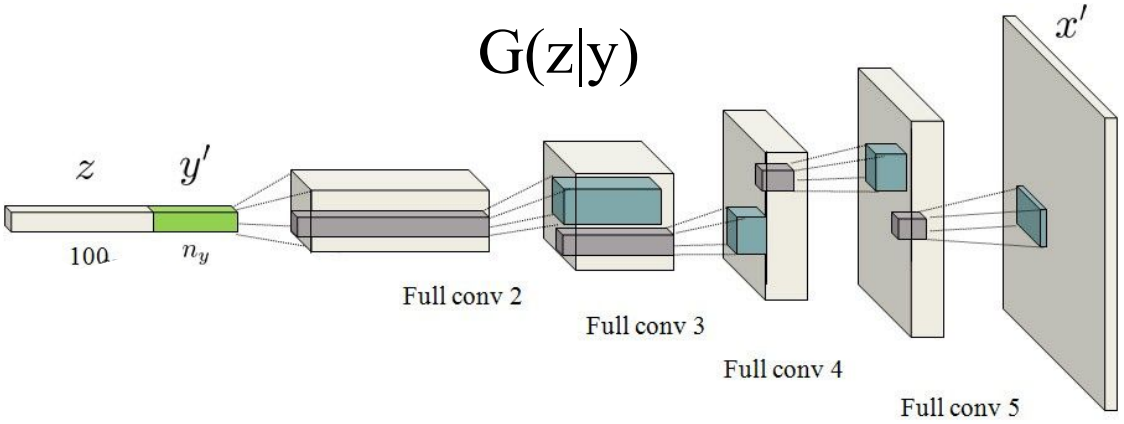
Mirza, Mehdi & Osindero, Simon. (2014). Conditional Generative Adversarial Nets. arXiv:1411.1784v1

“Fashion MNIST”

“*t-shirt*“, ‘*pants*‘, ‘*pullover*‘, ‘*dress*‘, ‘*coat*‘, ‘*sandal*‘, ‘*shirt*‘, ‘*sneaker*‘, ‘*bag*‘, ‘*ankle boot*‘.

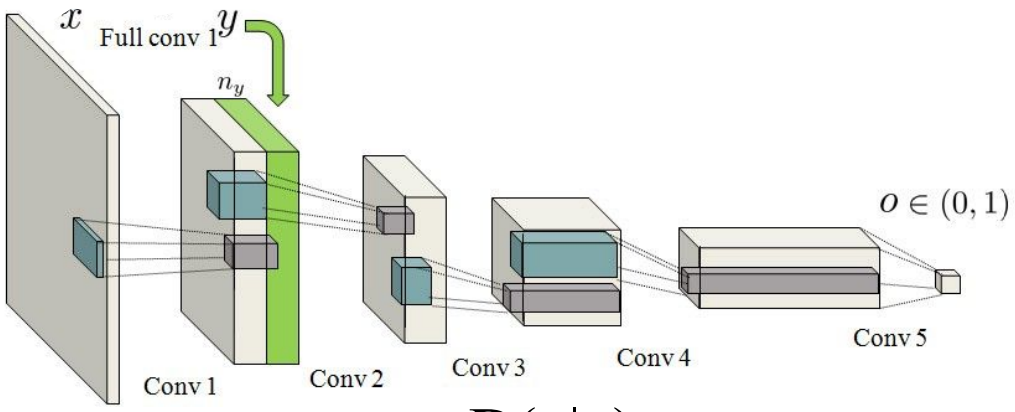
Mirza, Mehdi & Osindero, Simon. (2014). Conditional Generative Adversarial Nets. arXiv:1411.1784v1

Version convolutionnelle



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

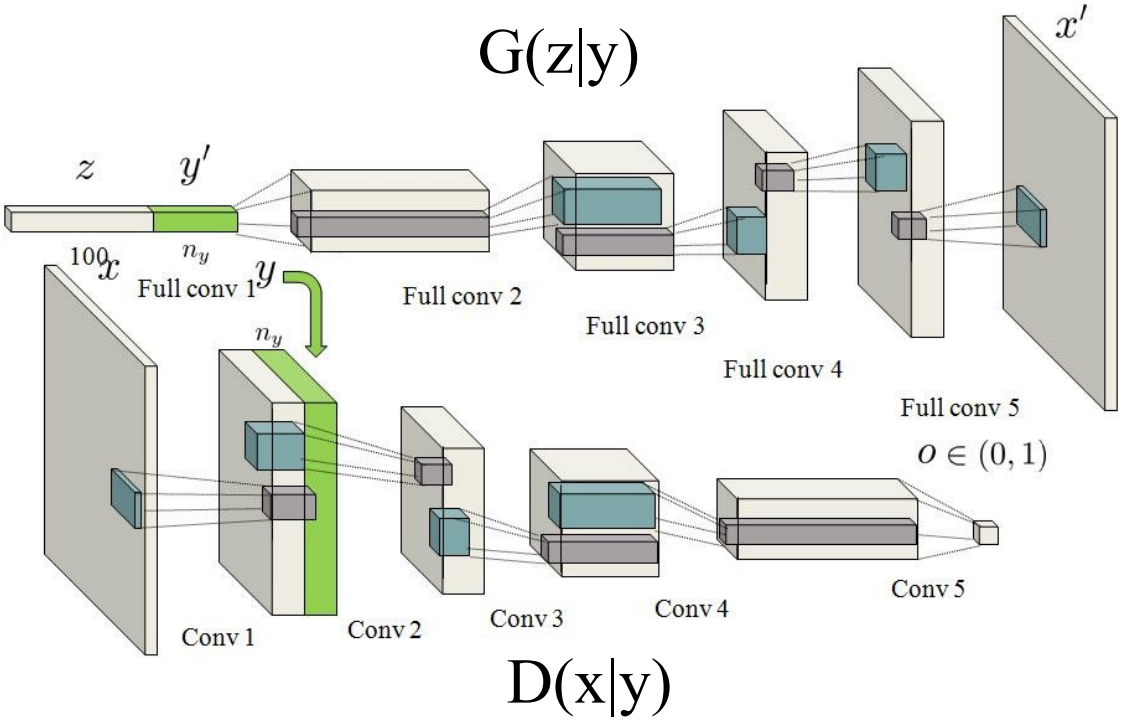
Version convolutionnelle



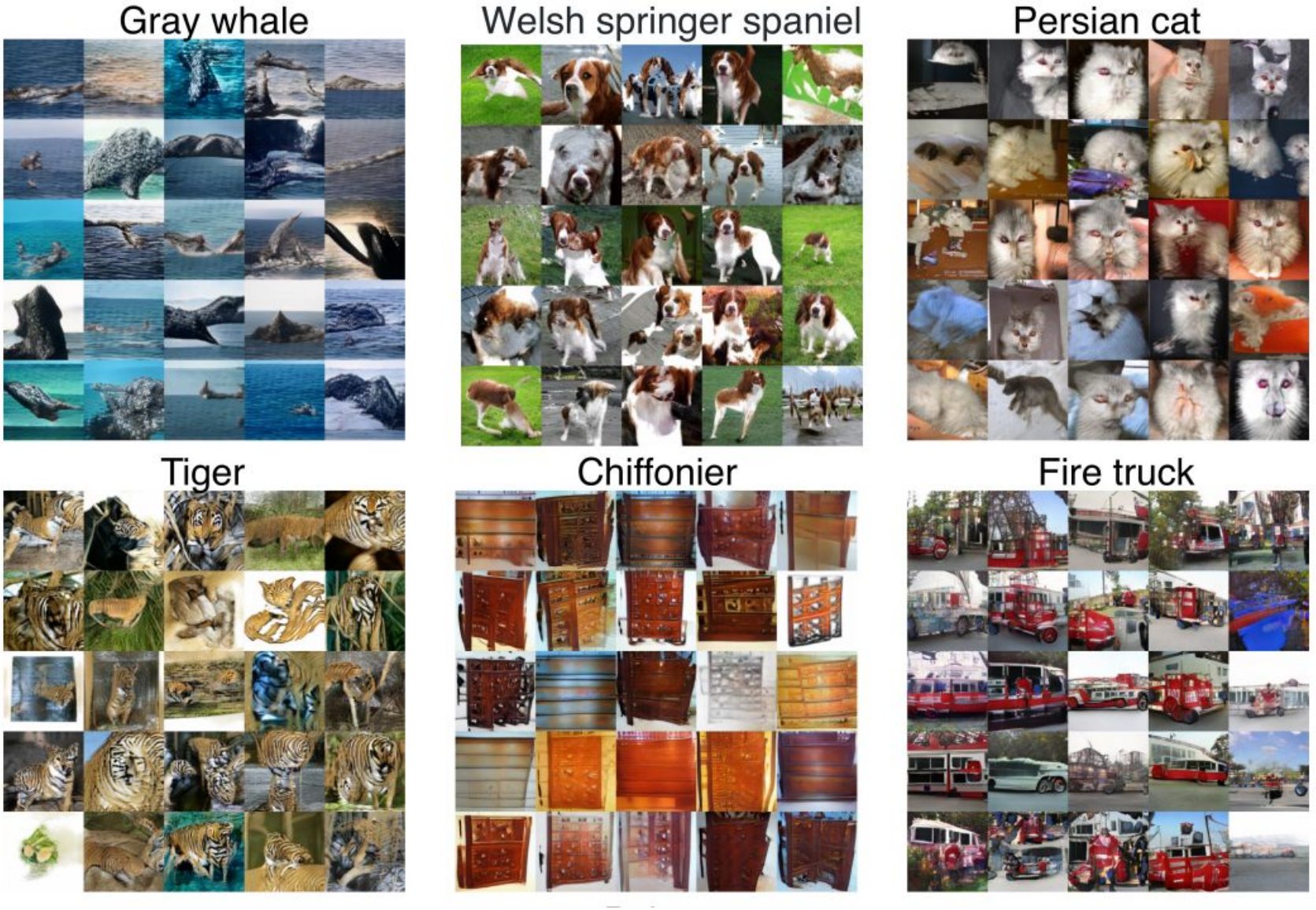
D(x|y)

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

Version convolutionnelle



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



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

Self-attention GAN

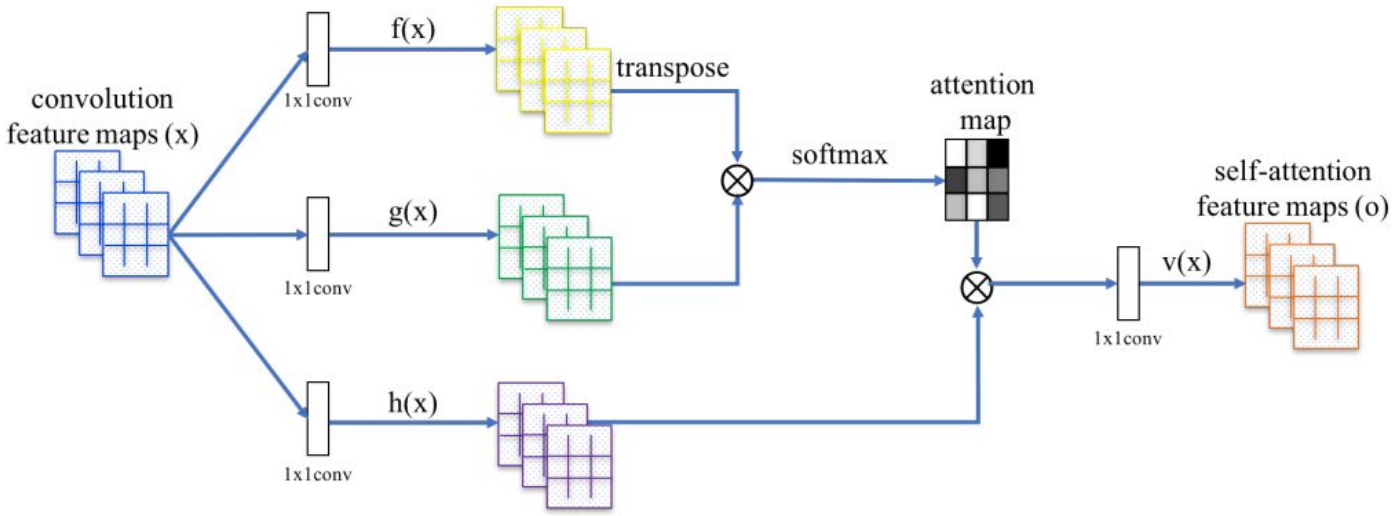
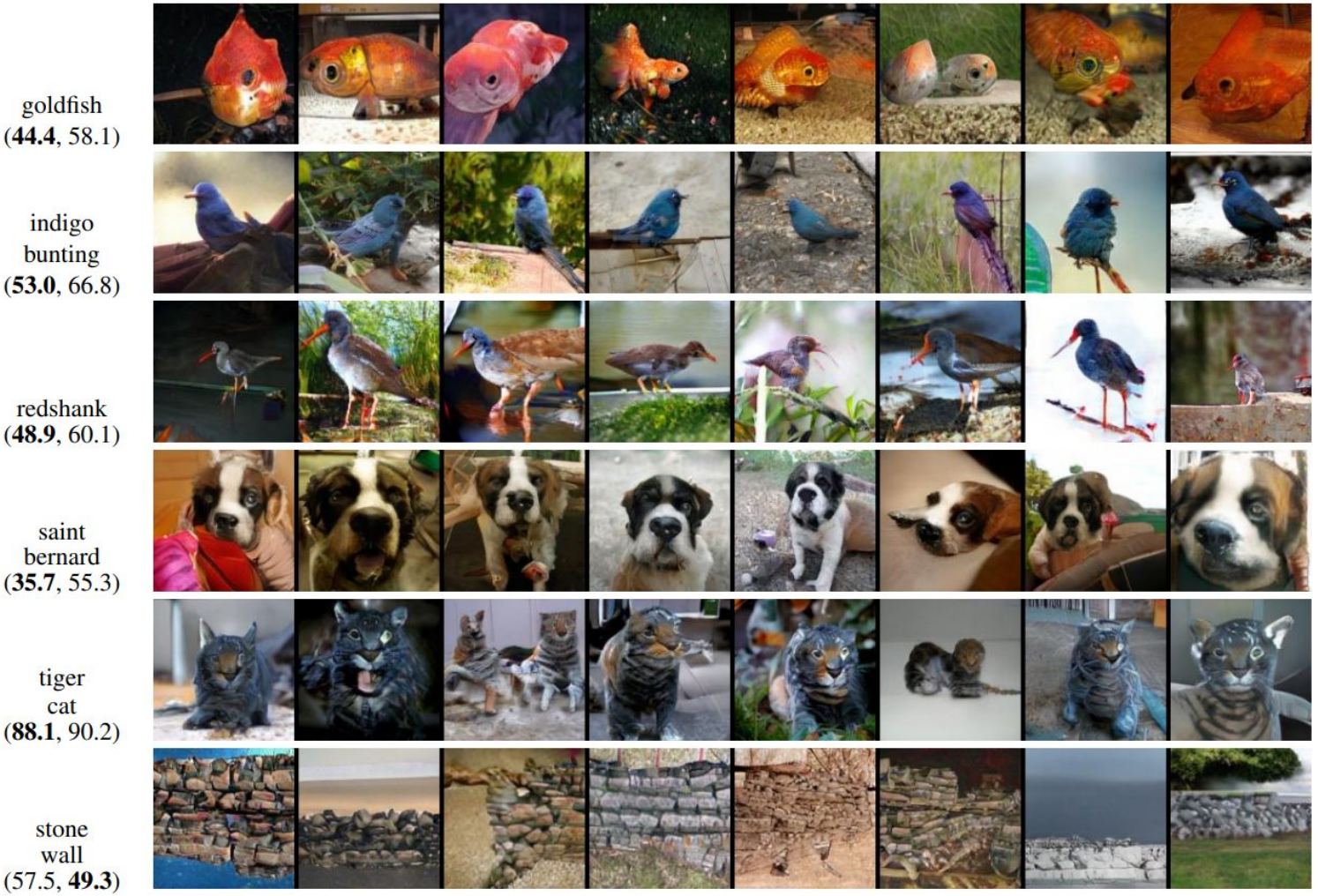


Figure 2. The proposed self-attention module for the SAGAN. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

GAN standard + *intra-attention* + d'autres détails moins intéressants

Zhang, H., Goodfellow, I., Metaxas, D., & Odena, A. (2019, May). Self-attention generative adversarial networks. In *International conference on machine learning* (pp. 7354-7363). PMLR.

Self-attention GAN



Zhang, H., Goodfellow, I., Metaxas, D., & Odena, A. (2019, May). Self-attention generative adversarial networks. In *International conference on machine learning* (pp. 7354-7363). PMLR.

BigGAN

Un plus gros self-attention GAN

Figure 1: Class-conditional samples generated by our model.

Brock, A., Donahue, J., & Simonyan, K. (2018). Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

BigGAN

Un plus gros self-attention GAN

Figure 6: Samples generated by our BigGAN model at 512×512 resolution.

Brock, A., Donahue, J., & Simonyan, K. (2018). Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

BigGAN

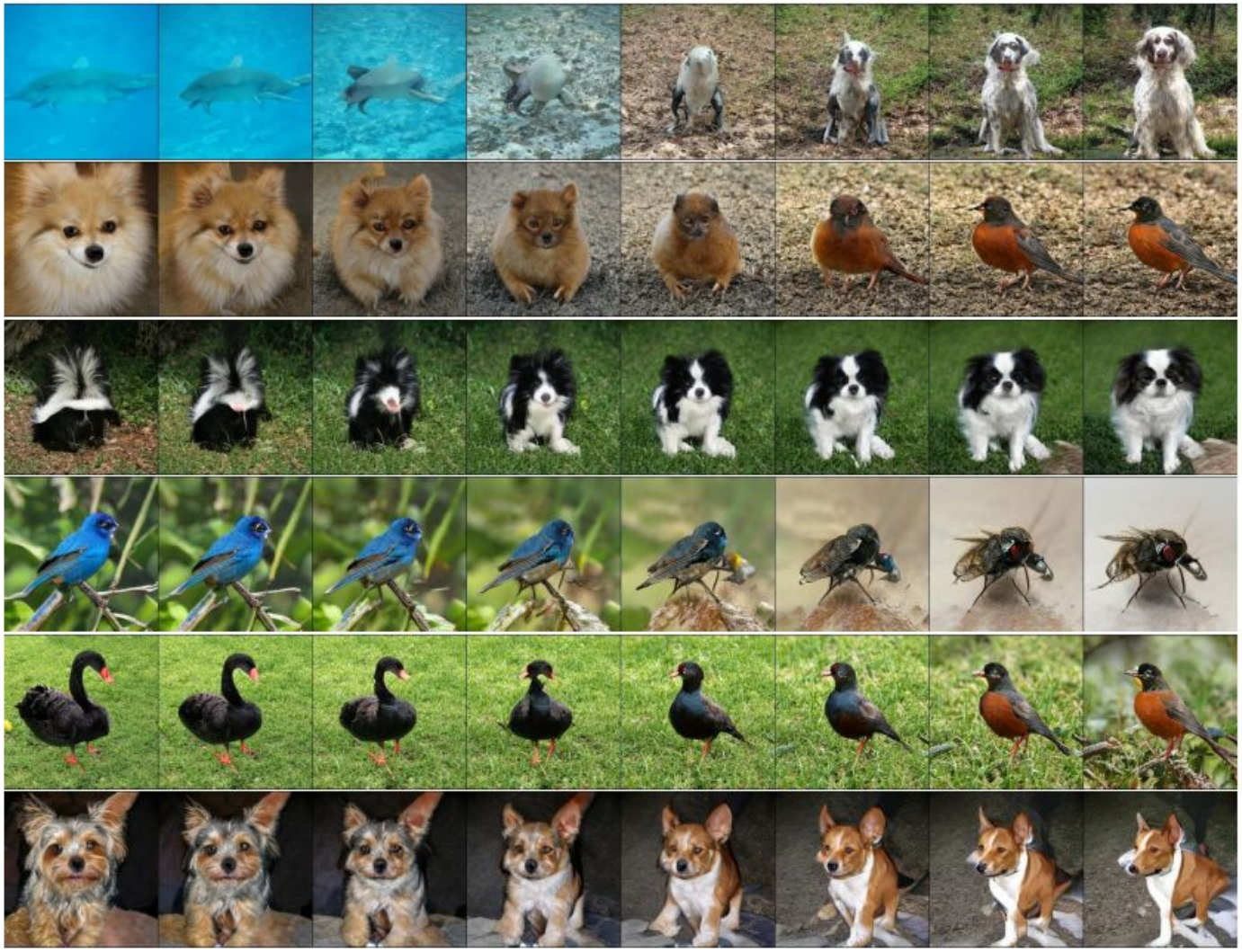
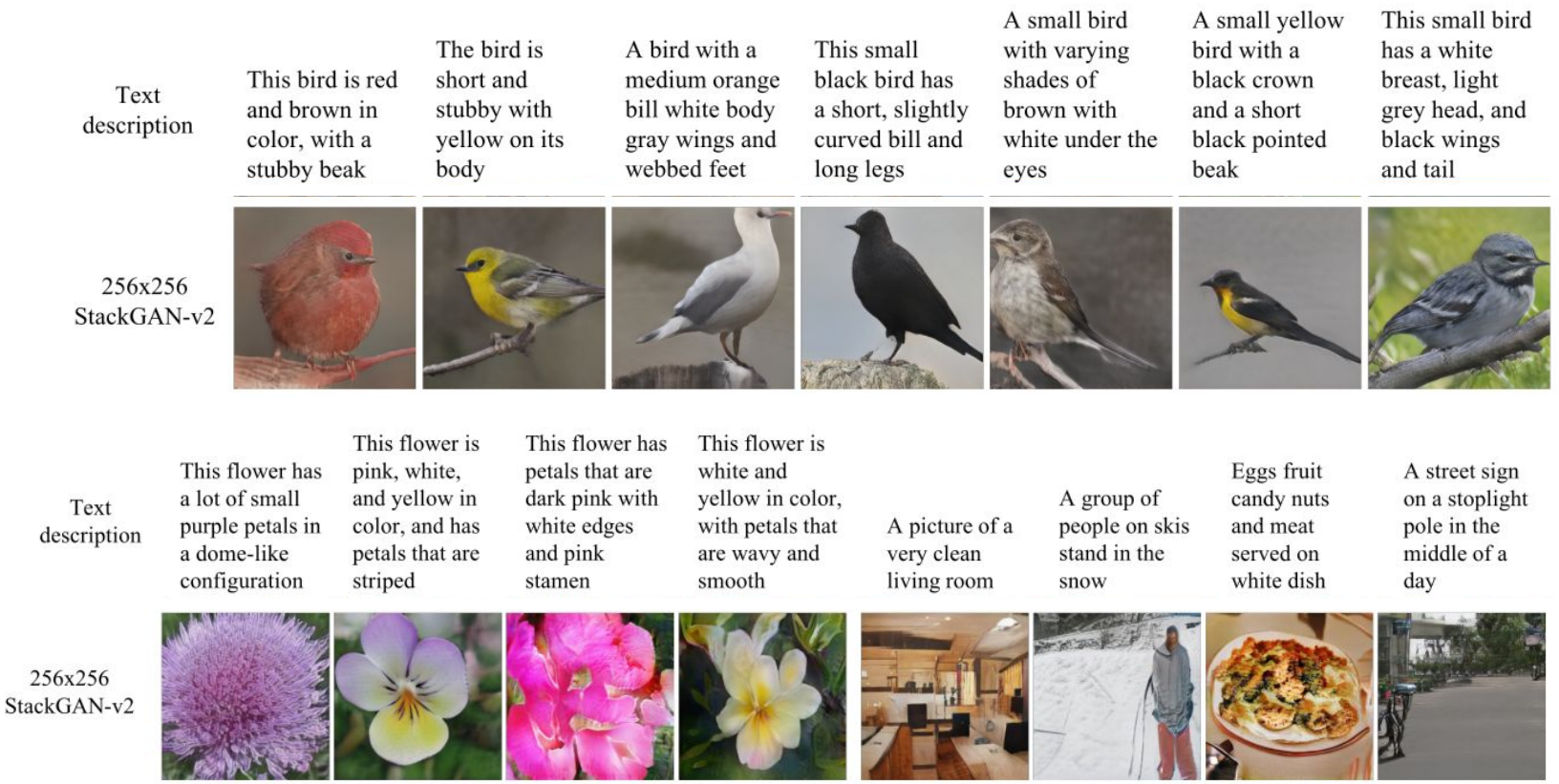


Figure 8: Interpolations between z, c pairs.

Est-ce qu’on doit conditionner seulement sur des classes ?

Texte -> Images



Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., & Metaxas, D. N. (2018). Stackgan++: Realistic image synthesis with stacked generative adversarial networks. IEEE transactions on pattern analysis and machine intelligence, 41(8), 1947-1962.

Pix2Pix

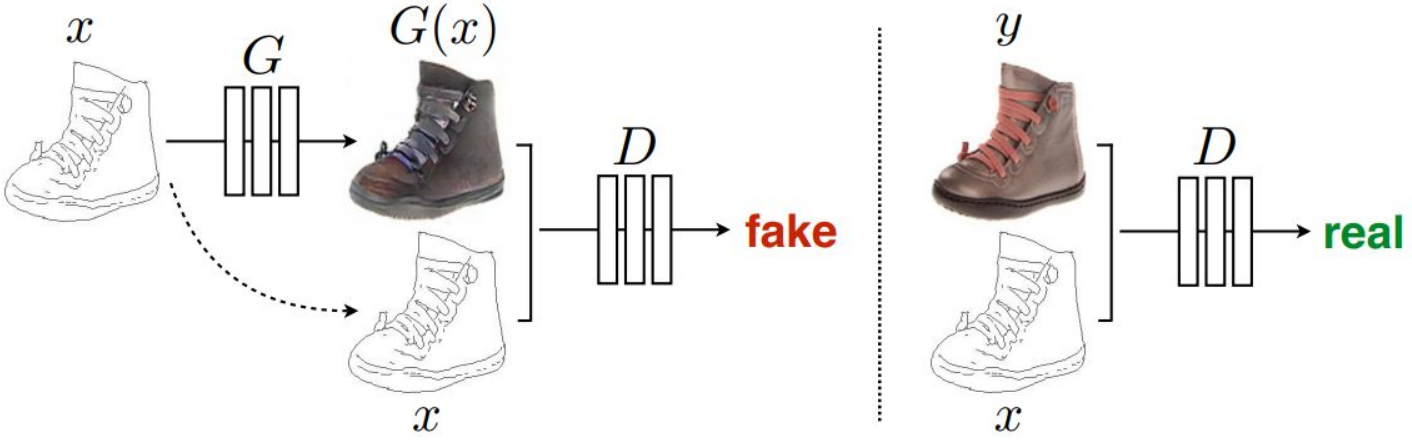


Figure 2: Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1125-1134).

Pix2Pix

Figure 16: Example results of our method on automatically detected edges→handbags, compared to ground truth.

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1125-1134).

Pix2Pix

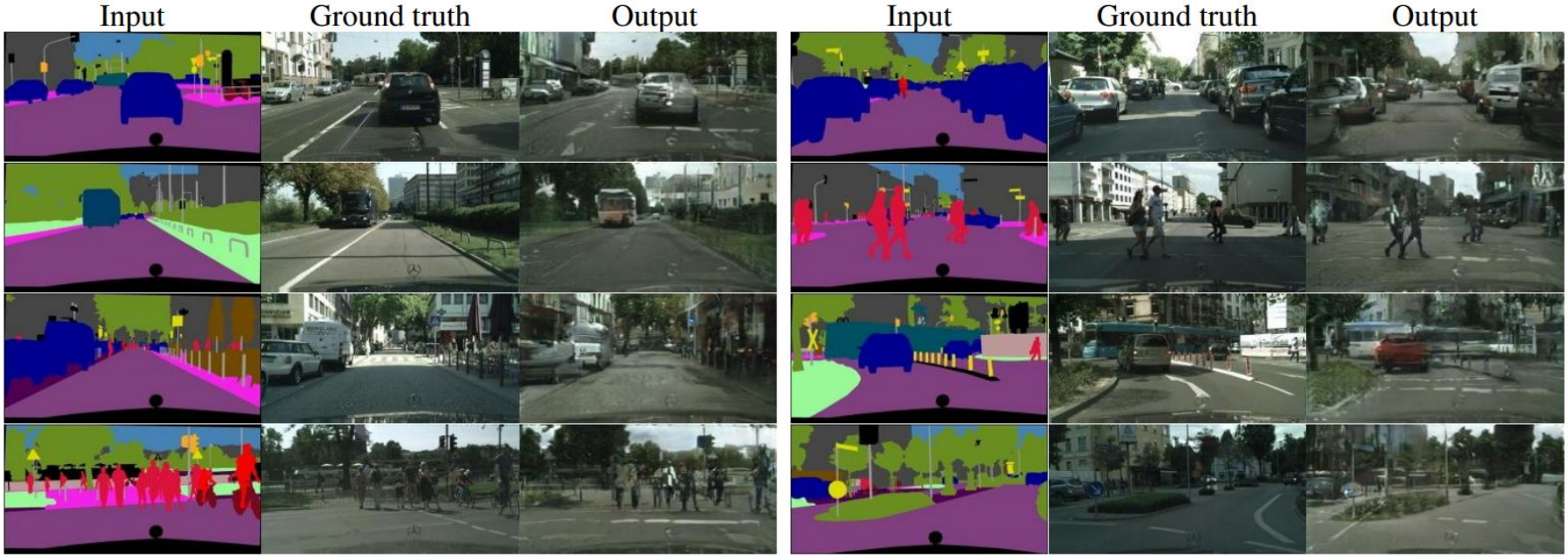


Figure 13: Example results of our method on Cityscapes labels→photo, compared to ground truth.

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1125-1134).

Pix2Pix

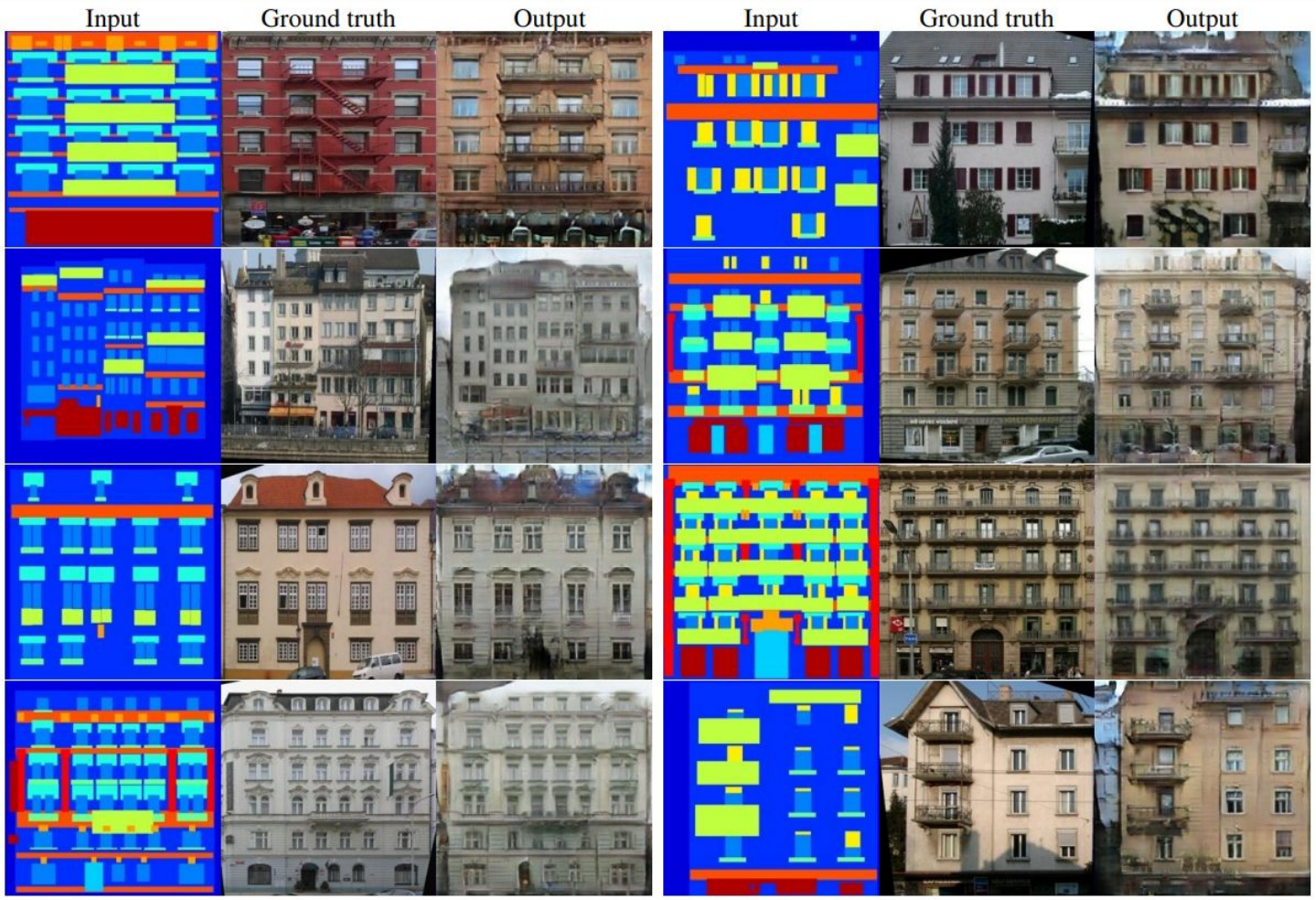


Figure 14: Example results of our method on facades labels→photo, compared to ground truth.

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1125-1134).

Pix2Pix

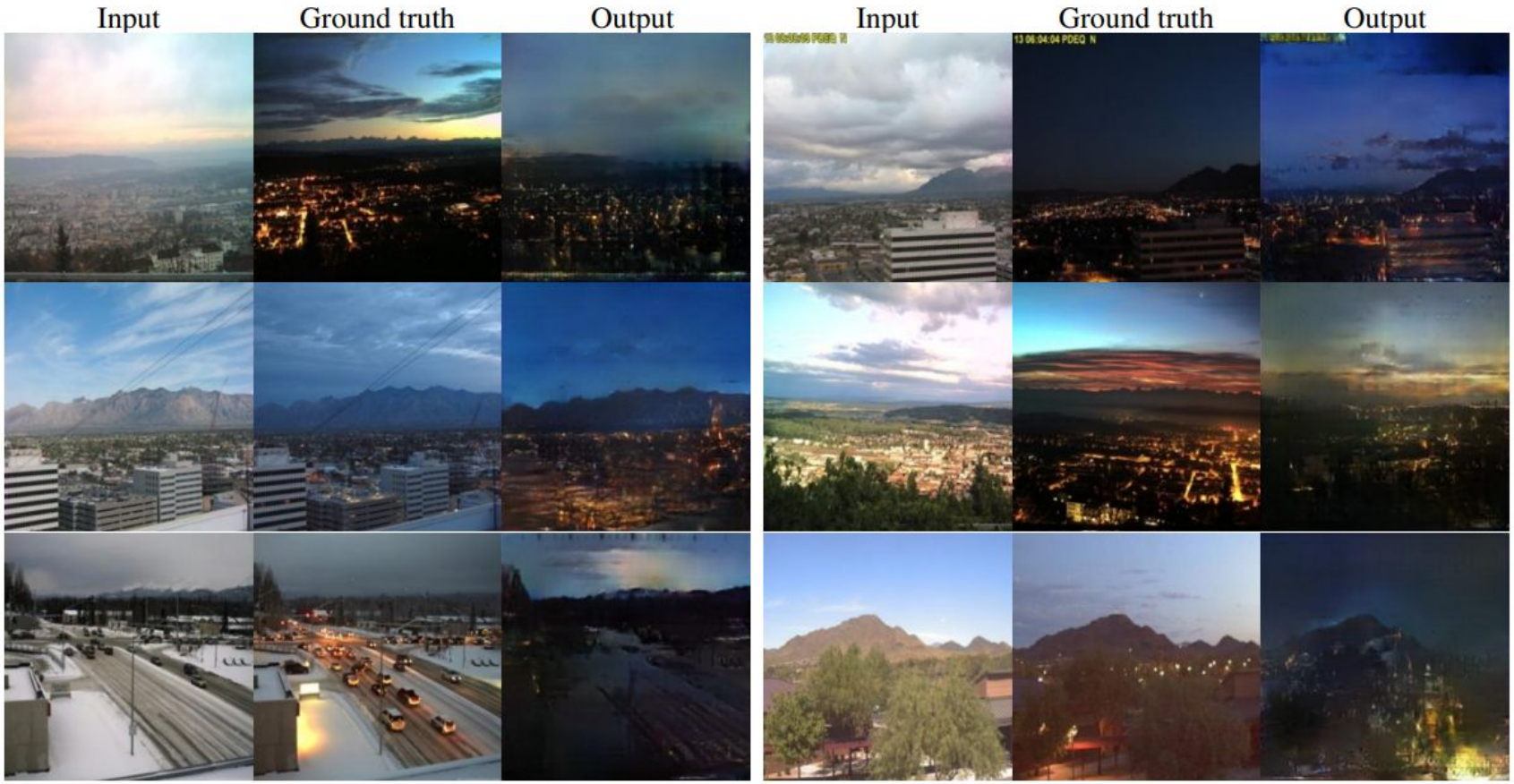
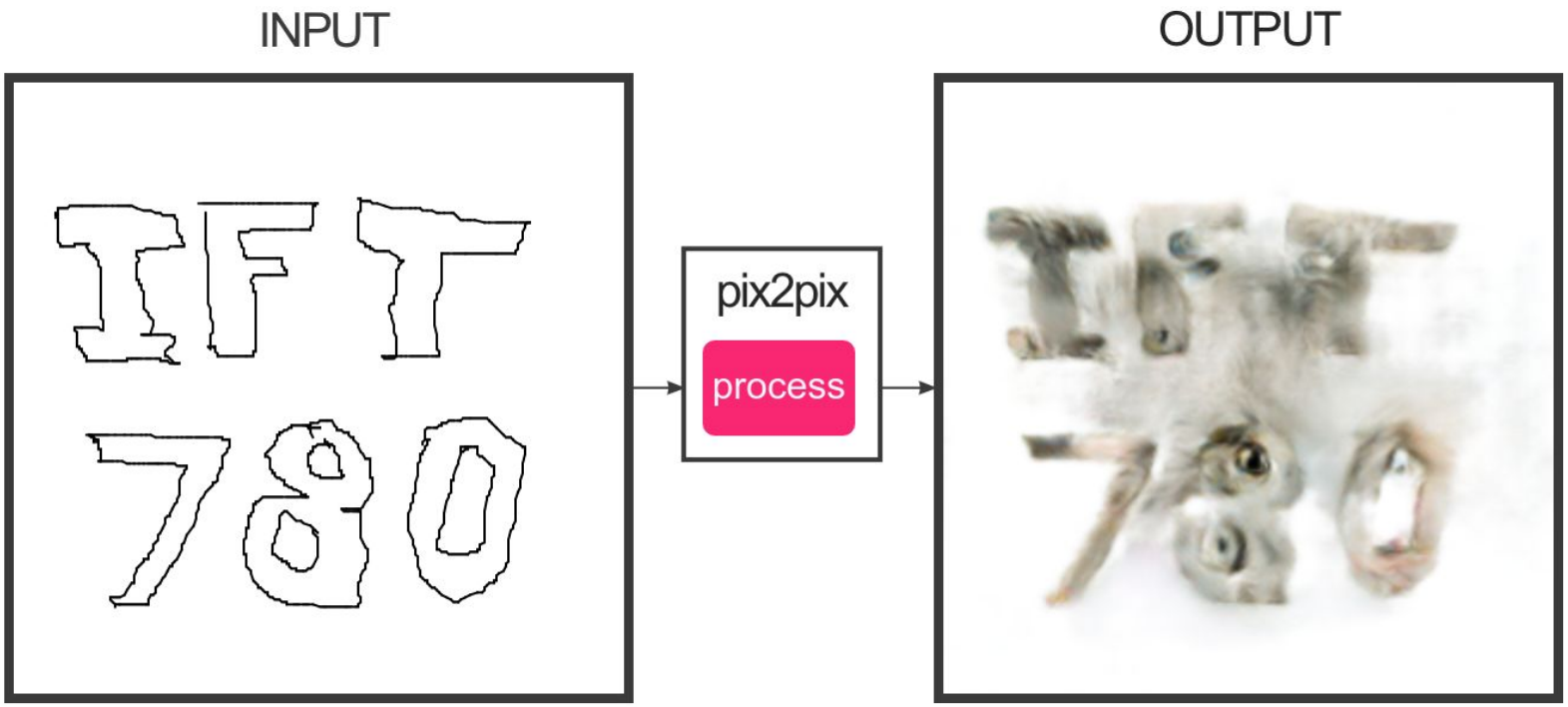


Figure 15: Example results of our method on day→night, compared to ground truth.

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1125-1134).

Pix2Pix



https://affinelayer.com/pixsrv/

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1125-1134).

Pix2Pix

https://twitter.com/brannondorsey/status/808461108881268736

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1125-1134).

Semantic image synthesis

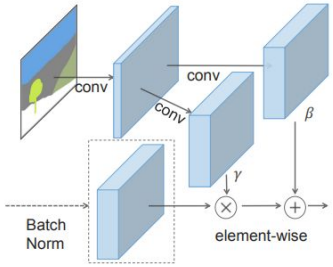


Figure 2: In the SPADE, the mask is first projected onto an embedding space and then convolved to produce the modulation parameters γ and β. Unlike prior conditional normalization methods, γ and β are not vectors, but tensors with spatial dimensions. The produced γ and β are multiplied and added to the normalized activation element-wise.

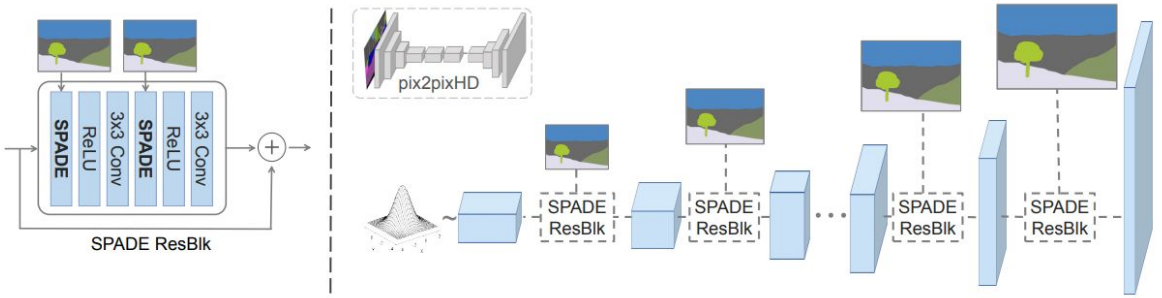


Figure 4: In the SPADE generator, each normalization layer uses the segmentation mask to modulate the layer activations. *(left)* Structure of one residual block with the SPADE. *(right)* The generator contains a series of the SPADE residual blocks with upsampling layers. Our architecture achieves better performance with a smaller number of parameters by removing the downsampling layers of leading image-to-image translation networks such as the pix2pixHD model [48].

Park, T., Liu, M. Y., Wang, T. C., & Zhu, J. Y. (2019). Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2337-2346).

Semantic image synthesis

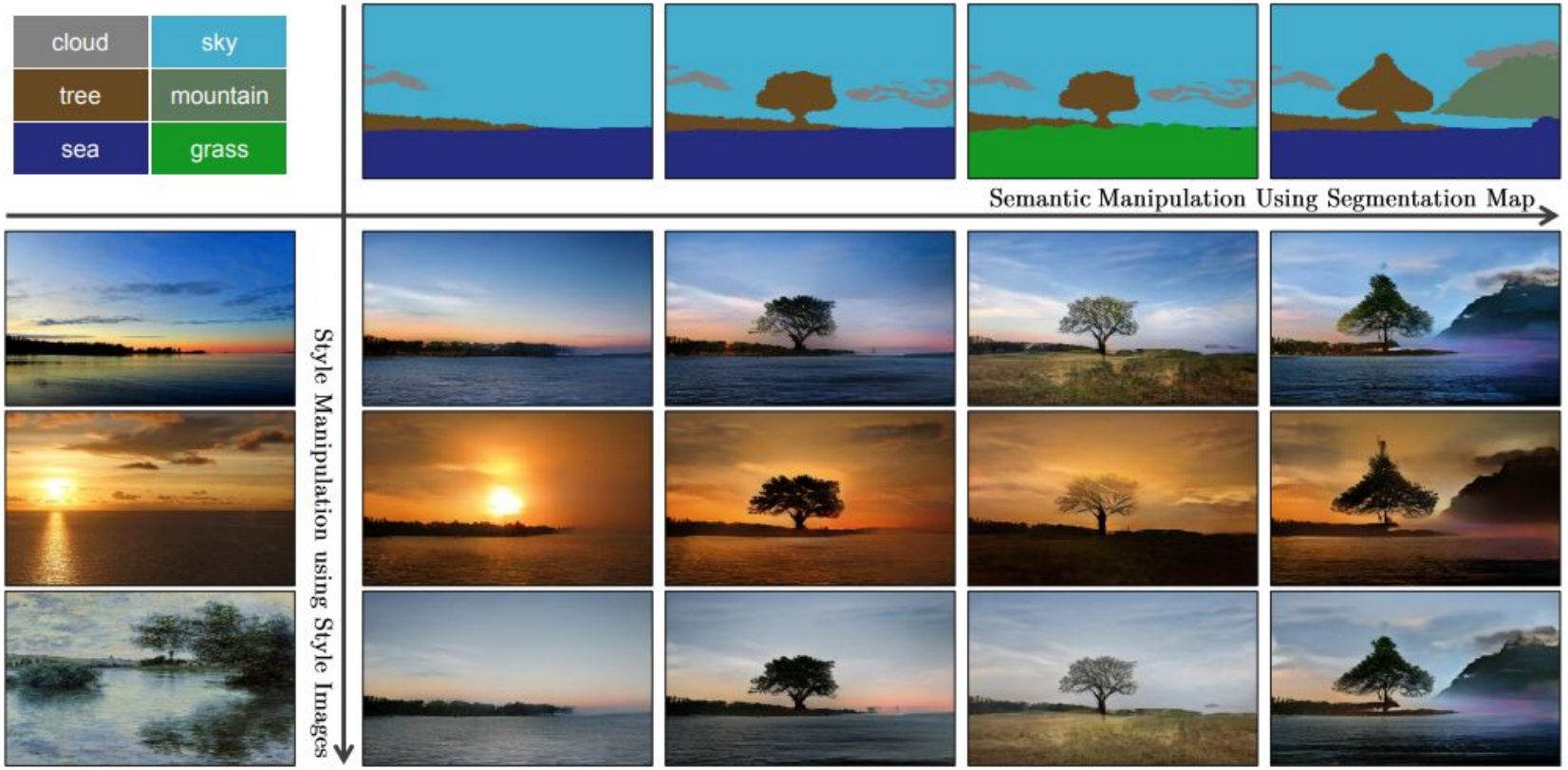


Figure 1: Our model allows user control over both semantic and style as synthesizing an image. The semantic (e.g., the existence of a tree) is controlled via a label map (the top row), while the style is controlled via the reference style image (the leftmost column). Please visit our website for interactive image synthesis demos.

Park, T., Liu, M. Y., Wang, T. C., & Zhu, J. Y. (2019). Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2337-2346).

Semantic image synthesis

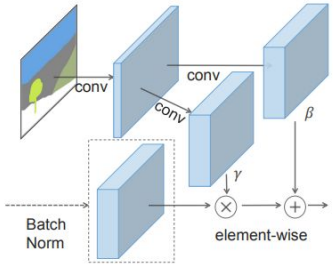


Figure 2: In the SPADE, the mask is first projected onto an embedding space and then convolved to produce the modulation parameters γ and β. Unlike prior conditional normalization methods, γ and β are not vectors, but tensors with spatial dimensions. The produced γ and β are multiplied and added to the normalized activation element-wise.

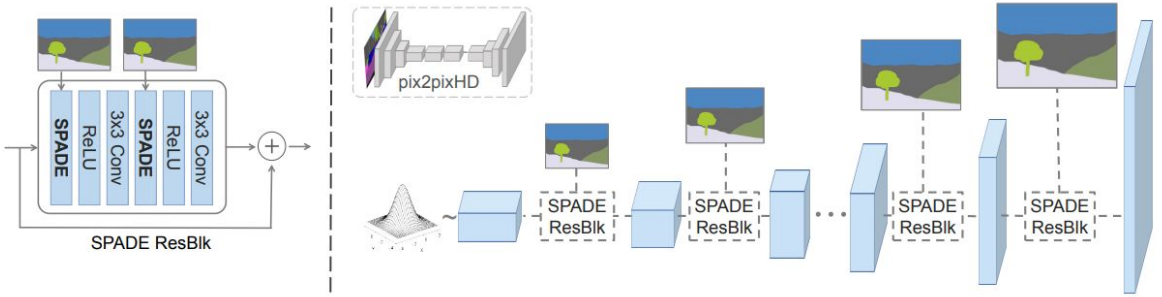
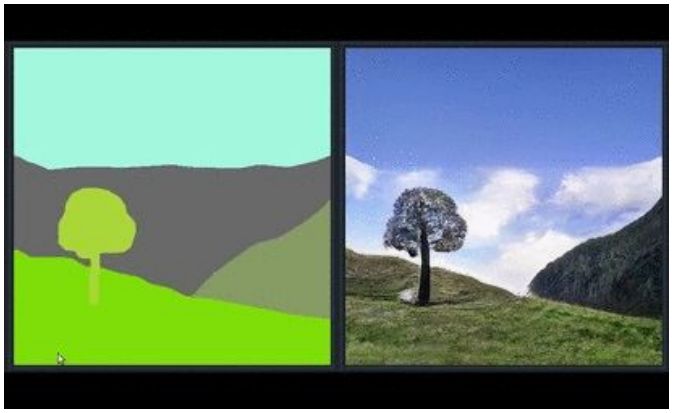
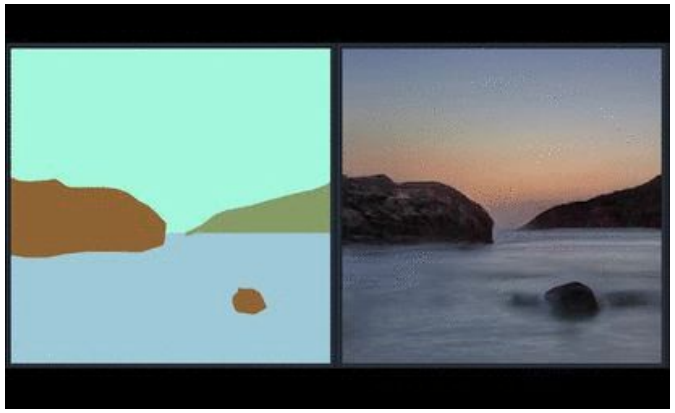


Figure 4: In the SPADE generator, each normalization layer uses the segmentation mask to modulate the layer activations. *(left)* Structure of one residual block with the SPADE. *(right)* The generator contains a series of the SPADE residual blocks with upsampling layers. Our architecture achieves better performance with a smaller number of parameters by removing the downsampling layers of leading image-to-image translation networks such as the pix2pixHD model [48].





https://www.youtube.com/watch?v=MXWm6w4E5q0

Park, T., Liu, M. Y., Wang, T. C., & Zhu, J. Y. (2019). Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2337-2346).

Et si on a pas de paires d’images ?

CycleGAN

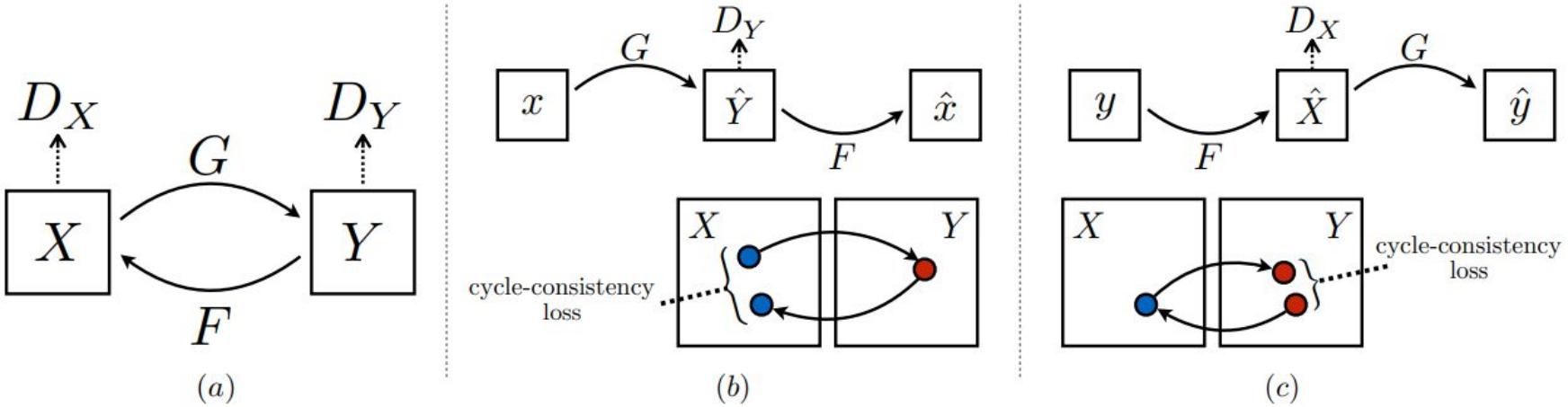
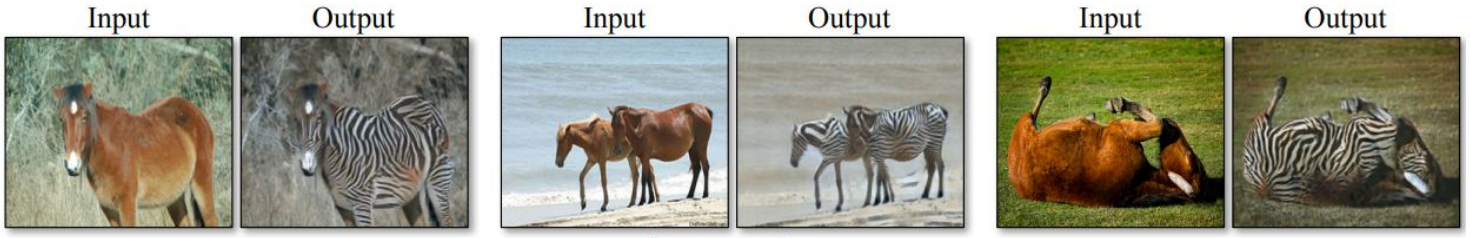


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

CycleGAN



horse → zebra

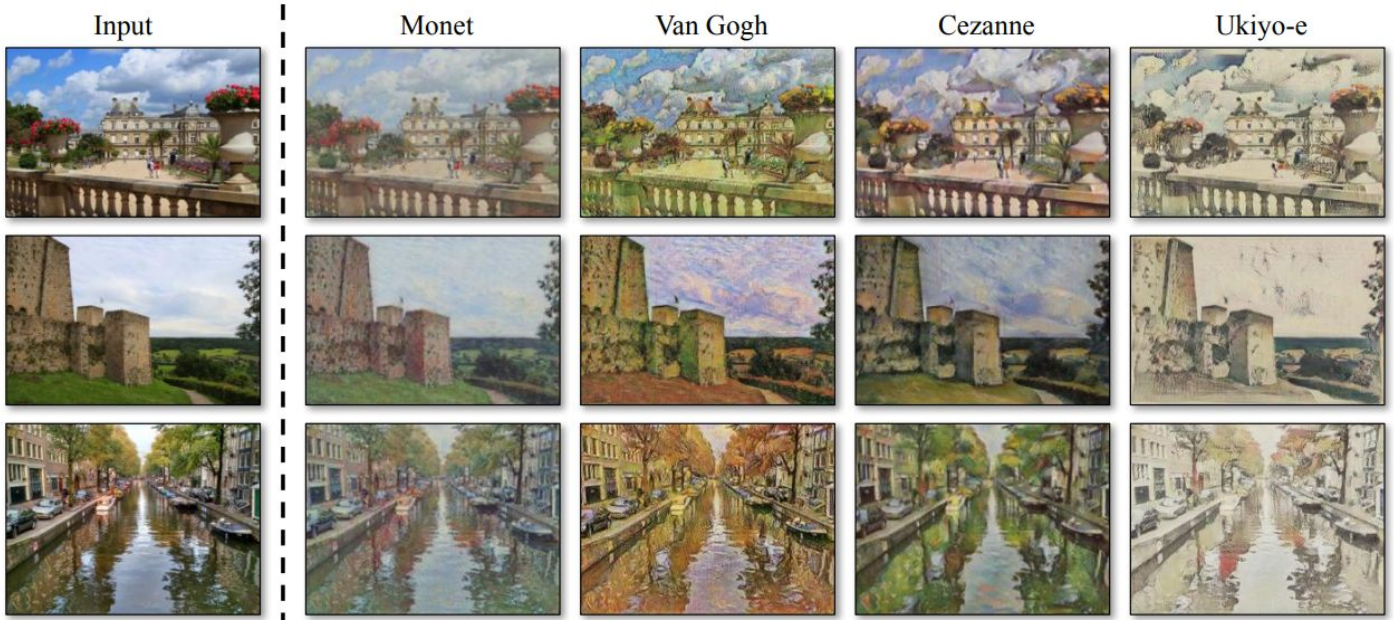
zebra → horse

winter Yosemite → summer Yosemite

summer Yosemite → winter Yosemite

CycleGAN


apple → orange
orange → apple


Input
Monet
Van Gogh
Cezanne
Ukiyo-e

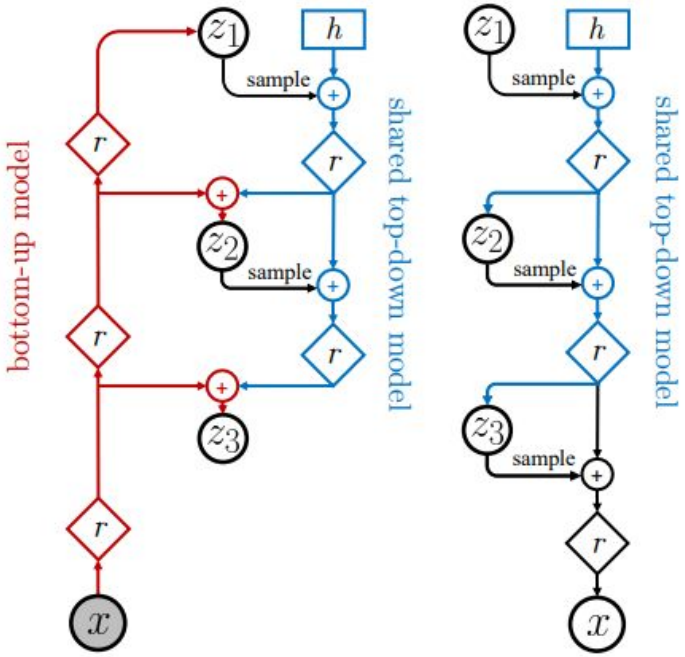
Code pytorch pour plus de 30 modèles de GANs

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

Mais les VAE ne sont pas obsolètes !

Nouveau VAE (NVAE)

Vahdat, A., & Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, *33*, 19667-19679.



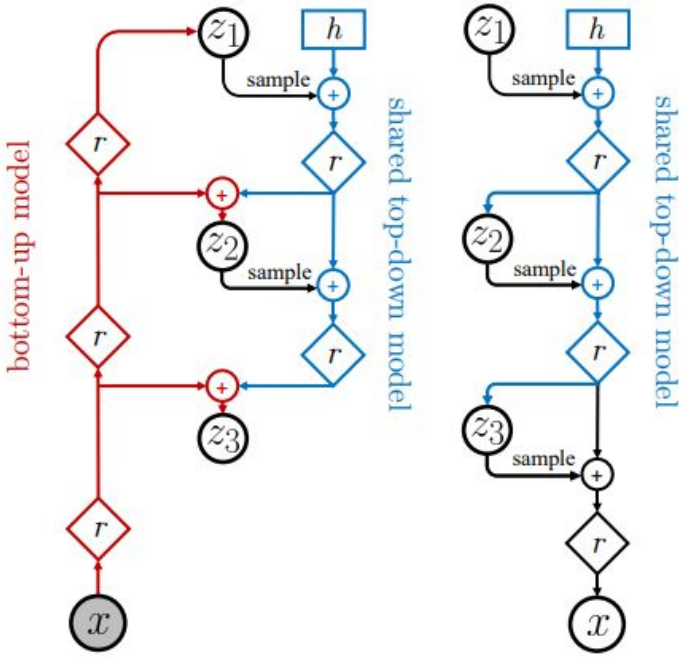
(a) Bidirectional Encoder (b) Generative Model

Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. ◇ denotes residual neural networks, ⊕ denotes feature combination (e.g., concatenation), and $\boxed{\text{h}}$ is a trainable parameter.

- VAE hiérarchique

Nouveau VAE (NVAE)

Vahdat, A., & Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, *33*, 19667-19679.



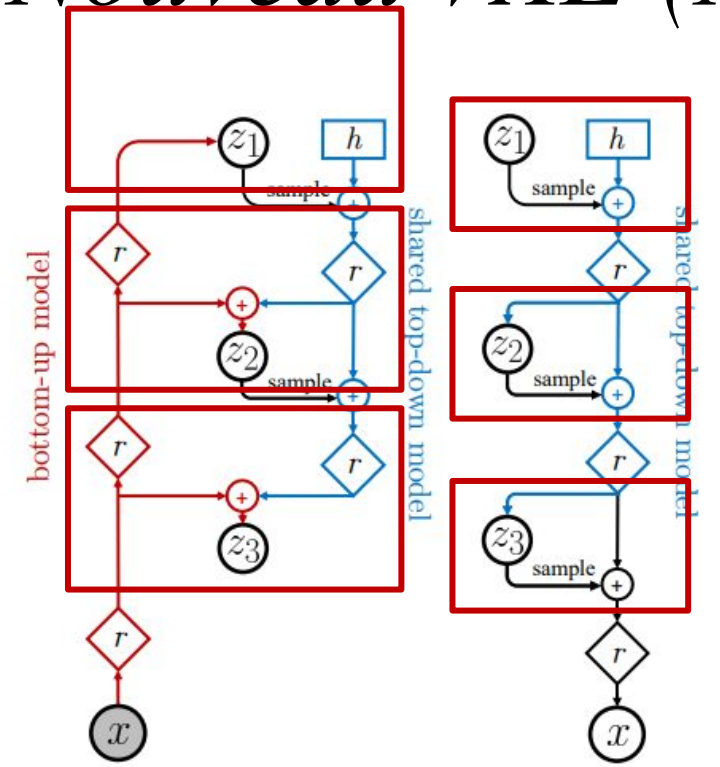
(a) Bidirectional Encoder (b) Generative Model

Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. ◇ denotes residual neural networks, ⊕ denotes feature combination (e.g., concatenation), and $\boxed{h}$ is a trainable parameter.

- VAE hiérarchique
- Les différents niveaux correspondent à des résolutions différentes

Nouveau VAE (NVAE)

Vahdat, A., & Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, *33*, 19667-19679.



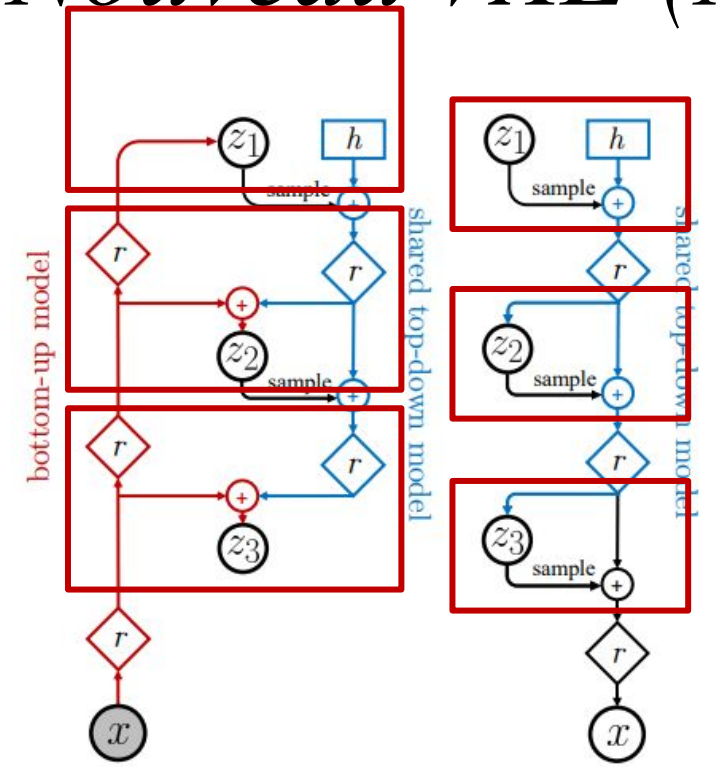
(a) Bidirectional Encoder (b) Generative Model

Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. ◇ denotes residual neural networks, ⊕ denotes feature combination (e.g., concatenation), and $\boxed{\text{h}}$ is a trainable parameter.

- VAE hiérarchique
- Les différents niveaux correspondent à des résolutions différentes
- Des z sont produits à chaque niveaux

Nouveau VAE (NVAE)

Vahdat, A., & Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, *33*, 19667-19679.



(a) Bidirectional Encoder (b) Generative Model

Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. ◇ denotes residual neural networks, ⊕ denotes feature combination (e.g., concatenation), and $\boxed{h}$ is a trainable parameter.

- VAE hiérarchique
- Les différents niveaux correspondent à des résolutions différentes
- Des z sont produits à chaque niveaux
- Les différents niveaux de bruits correspondent à différents niveaux de détails

Nouveau VAE (NVAE)

Vahdat, A., & Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, *33*, 19667-19679.

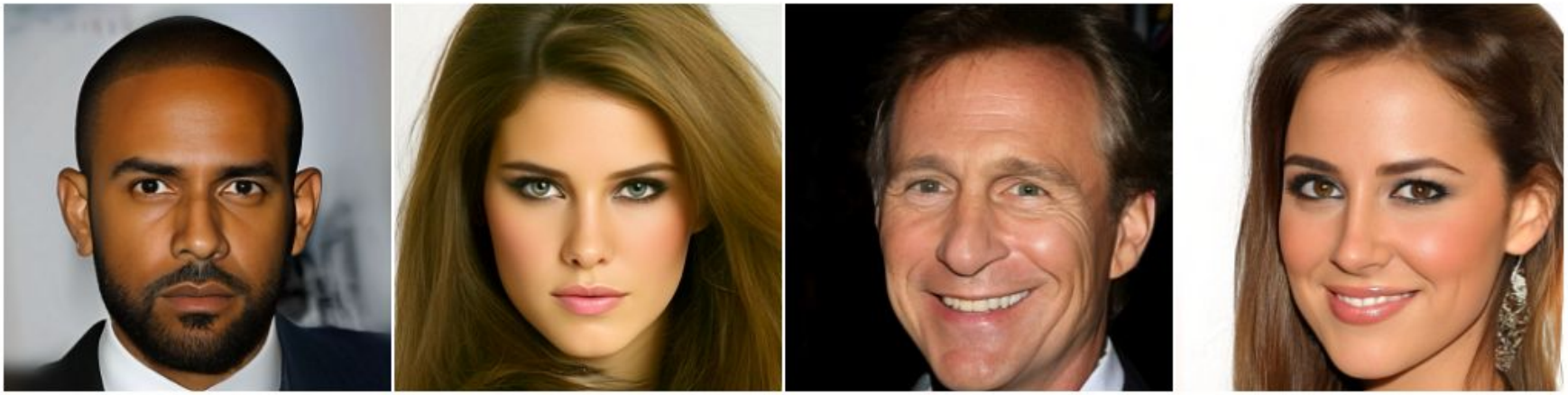


Figure 1: 256×256-pixel samples generated by NVAE, trained on CelebA HQ [28].

Nouveau VAE (NVAE)

Vahdat, A., & Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, *33*, 19667-19679.

(a) MNIST ($t = 1.0$) (b) CIFAR-10 ($t = 0.7$) (c) CelebA 64 ($t = 0.6$)

(d) CelebA HQ ($t = 0.6$) (e) FFHQ ($t = 0.5$)

VQ-VAE

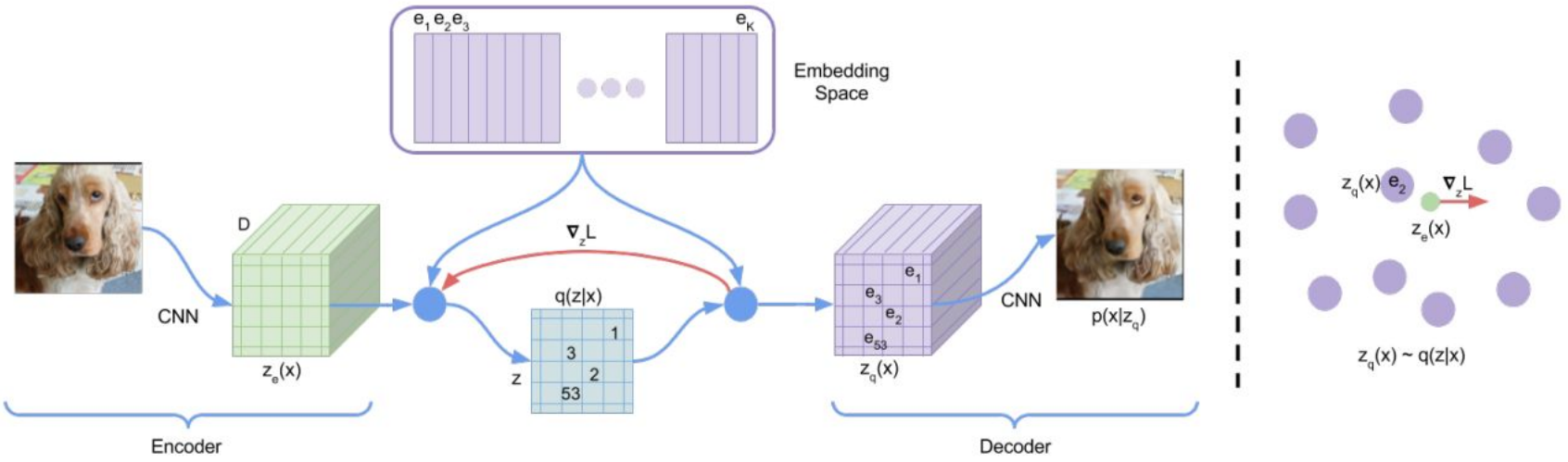


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

- Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.
- Razavi, A., Van den Oord, A., & Vinyals, O. (2019). Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, *32*.

VQ-VAE

Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.

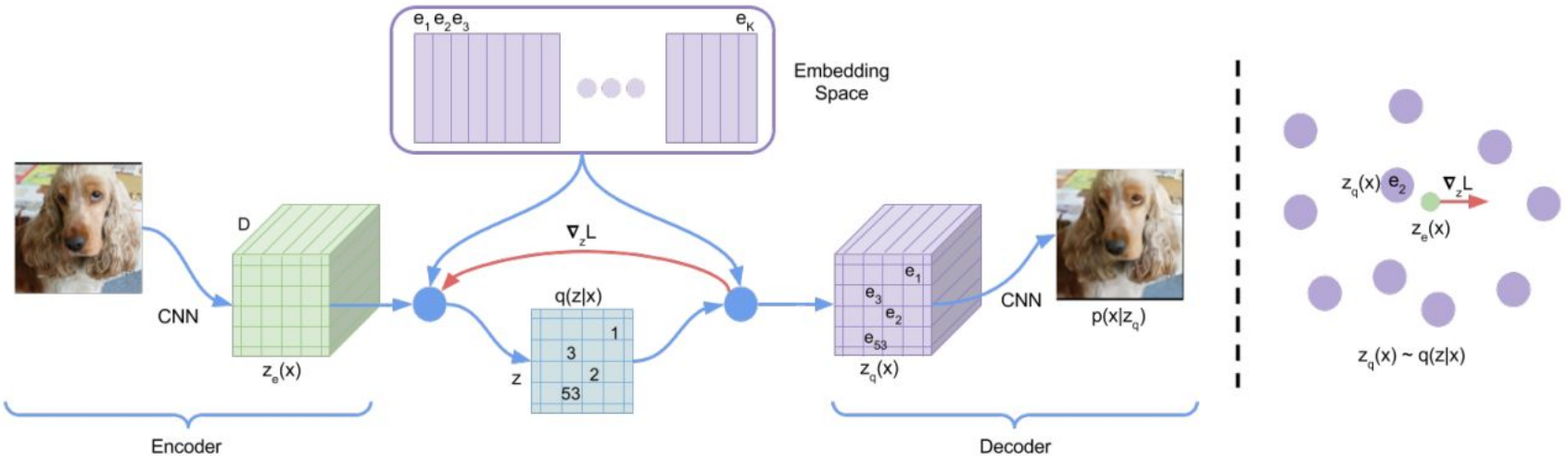


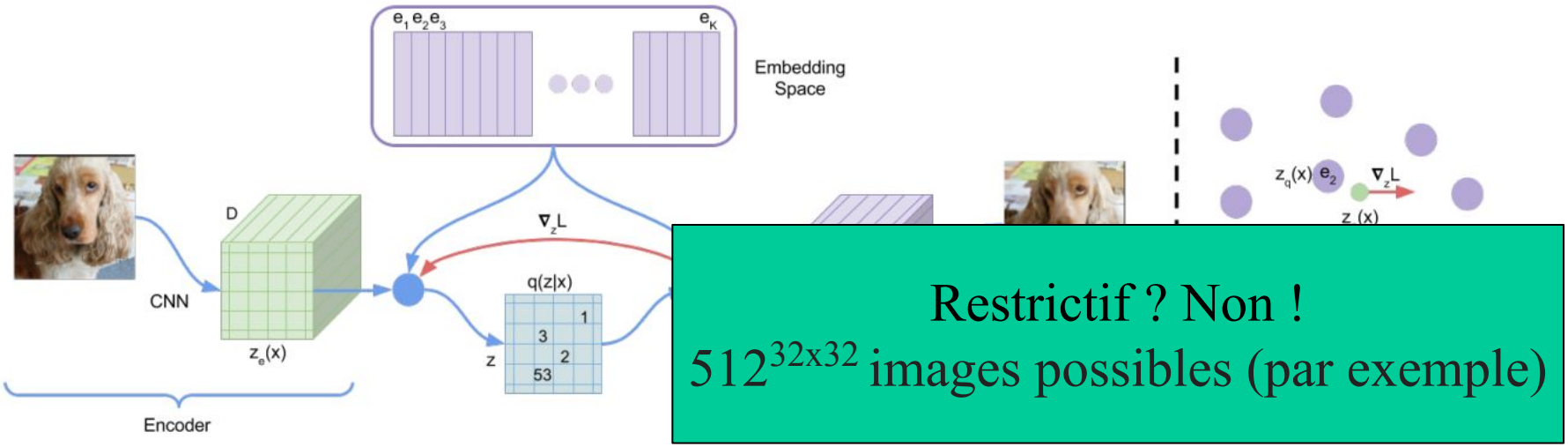
Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

- L'espace latent est *discret* plutôt que continu
- L'encodeur apprend à projeter les images vers des centroïdes
- Chaque composante de z est un centroïde

- Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.
- Razavi, A., Van den Oord, A., & Vinyals, O. (2019). Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, *32*.

VQ-VAE

Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.



Restrictif ? Non !

512^{32x32} images possibles (par exemple)

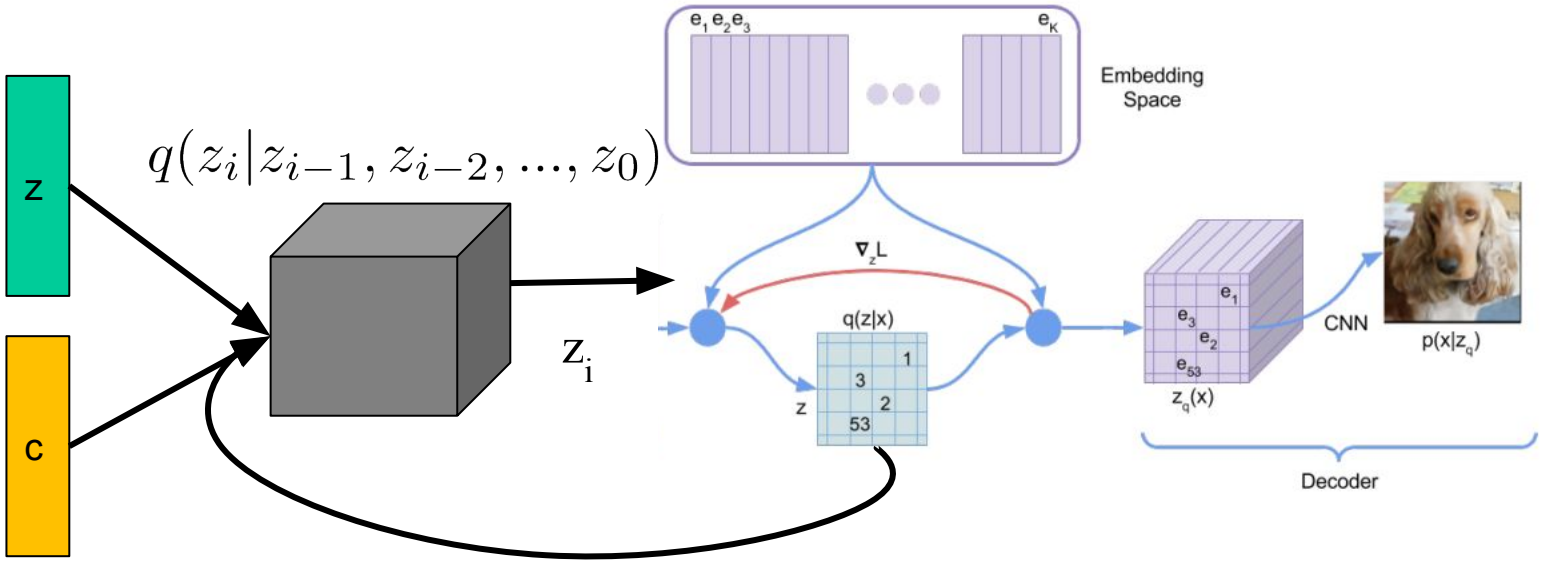
Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

- L'espace latent est *discret* plutôt que continu
- L'encodeur apprend à projeter les images vers des centroïdes
- Chaque composante de z est un centroïde
- Les centroïdes sont appris en étant rapprochés de la sortie de l'encodeur

- Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.
- Razavi, A., Van den Oord, A., & Vinyals, O. (2019). Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, *32*.

VQ-VAE

Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.



- En génération, un autre modèle apprend à choisir les e

- Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.
- Razavi, A., Van den Oord, A., & Vinyals, O. (2019). Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, *32*.

VQ-VAE

Figure 1: Class-conditional 256x256 image samples from a two-level model trained on ImageNet.

- Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.
- Razavi, A., Van den Oord, A., & Vinyals, O. (2019). Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, *32*.

VQ-VAE

VQ-VAE (Proposed)

BigGAN deep

- Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.
- Razavi, A., Van den Oord, A., & Vinyals, O. (2019). Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, *32*.

DALL-E

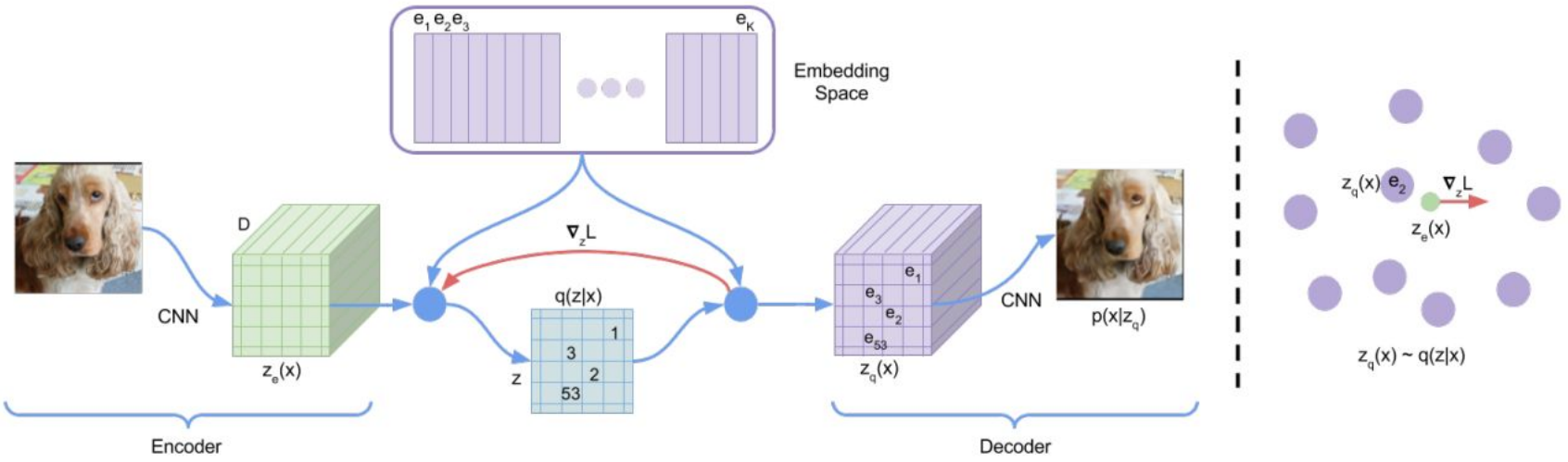


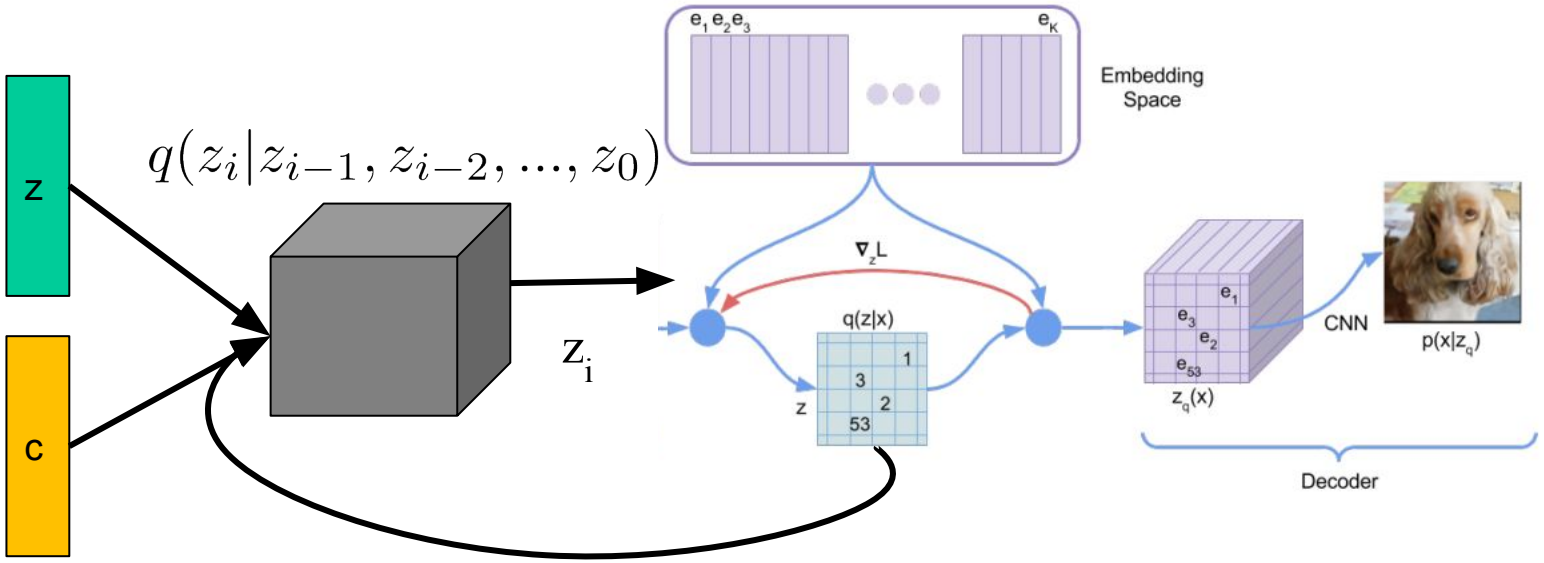
Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

- Comme VQ-VAE, mais plus gros et avec quelques twists
- Voir le code: https://github.com/openai/DALL-E

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021, July). Zero-shot text-to-image generation. In *International Conference on Machine Learning* (pp. 8821-8831). PMLR.

DALL-E

Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.

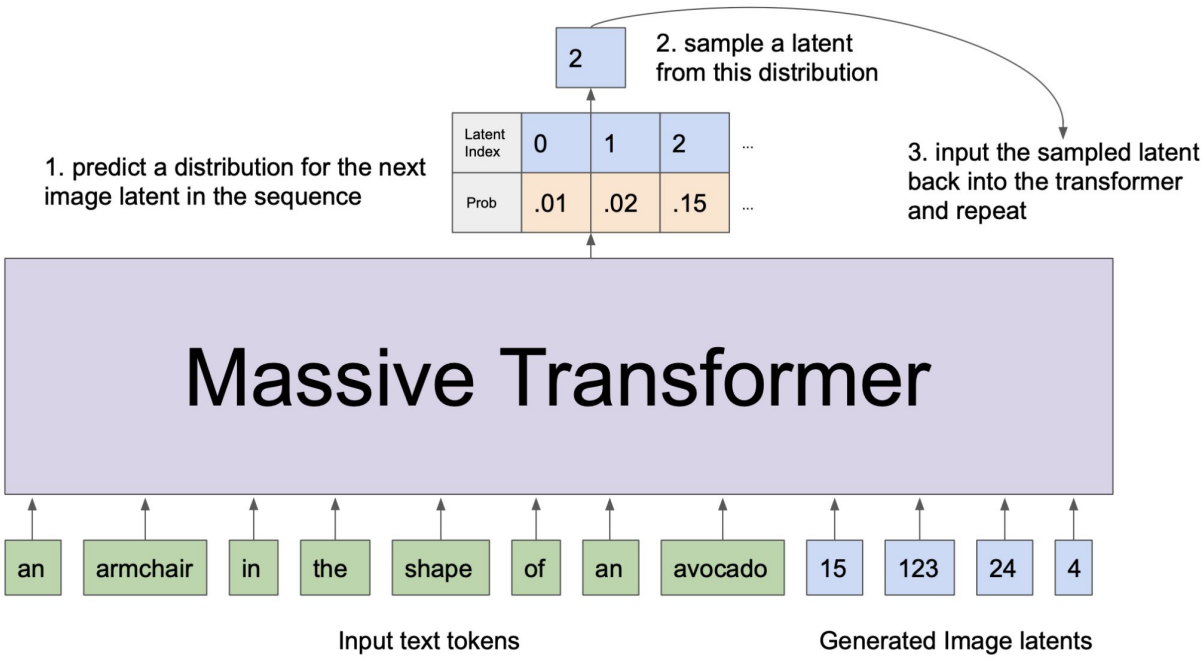


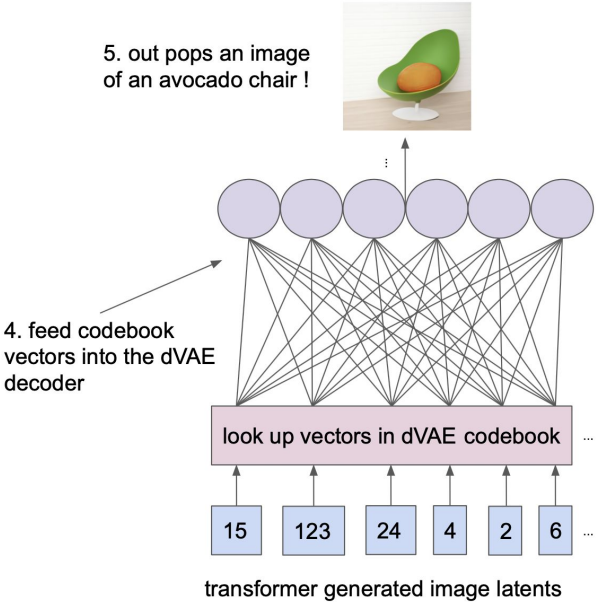
- Le “générateur” est un transformer (GPT-3 avec 12 millions de paramètres)

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021, July). Zero-shot text-to-image generation. In *International Conference on Machine Learning* (pp. 8821-8831). PMLR.

DALL-E

Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.





source: https://ml.berkeley.edu/blog/posts/dalle2/

- Le “générateur” est un transformer (GPT-3 avec 12 millions de paramètres)

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021, July). Zero-shot text-to-image generation. In *International Conference on Machine Learning* (pp. 8821-8831). PMLR.

DALL-E

Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.

TEXT PROMPT an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES

TEXT PROMPT an armchair in the shape of an avocado. . . .

AI-GENERATED IMAGES

https://openai.com/blog/dall-e/

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021, July). Zero-shot text-to-image generation. In *International Conference on Machine Learning* (pp. 8821-8831). PMLR.

DALL-E

Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, *30*.

TEXT PROMPT a store front that has the word 'openai' written on it. . . .

AI-GENERATED IMAGES

TEXT & IMAGE PROMPT the exact same cat on the top as a sketch on the bottom

AI-GENERATED IMAGES

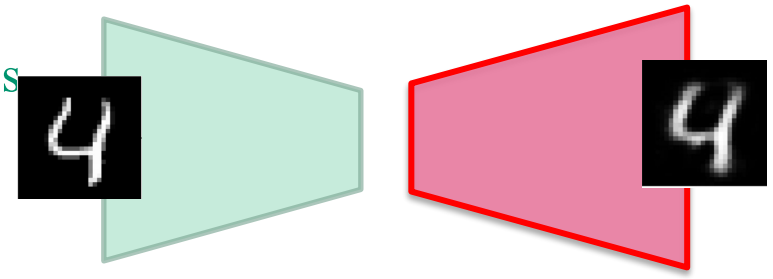
https://openai.com/blog/dall-e/

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021, July). Zero-shot text-to-image generation. In *International Conference on Machine Learning* (pp. 8821-8831). PMLR.

Conclusion

Autoencodeurs:

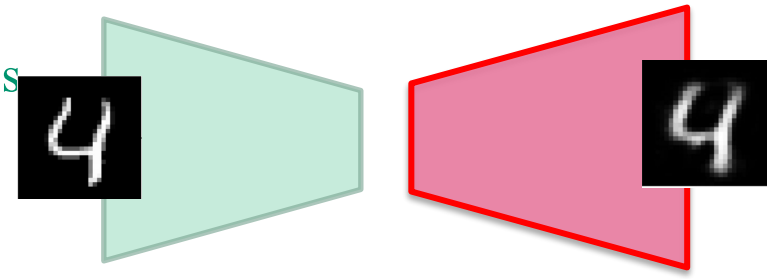
- Permettent d'apprendre des caractéristiques des données sans étiquettes de classe
- Peuvent être utilisés par des modèles supervisés
- Difficiles à échantillonner



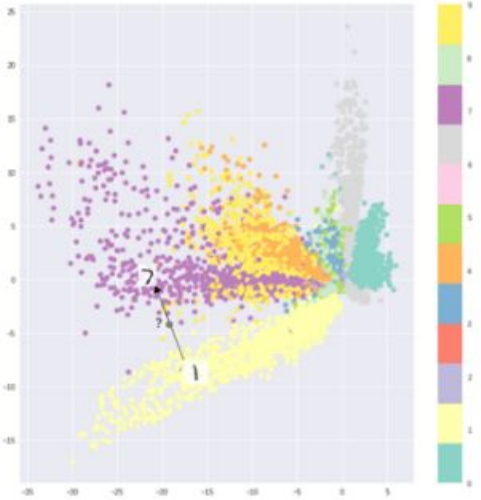
Conclusion

Autoencodeurs:

- Permettent d'apprendre des caractéristiques des données sans étiquettes de classe
- Peuvent être utilisés par des modèles supervisés
- Difficiles à échantillonner



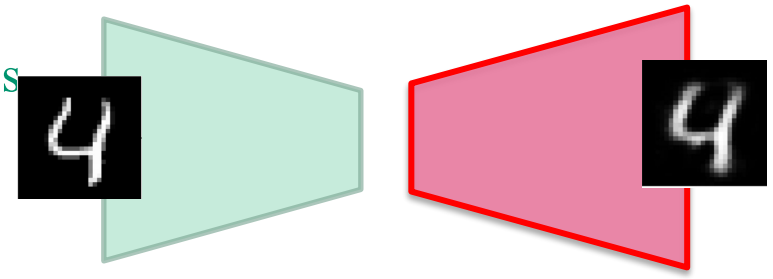
Only reconstruction loss



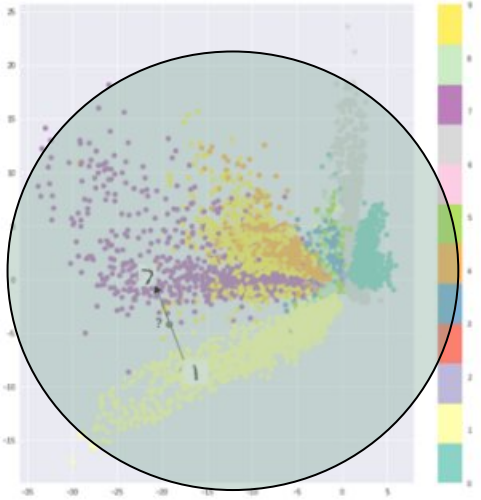
Conclusion

Autoencodeurs:

- Permettent d'apprendre des caractéristiques des données sans étiquettes de classe
- Peuvent être utilisés par des modèles supervisés
- Difficiles à échantillonner



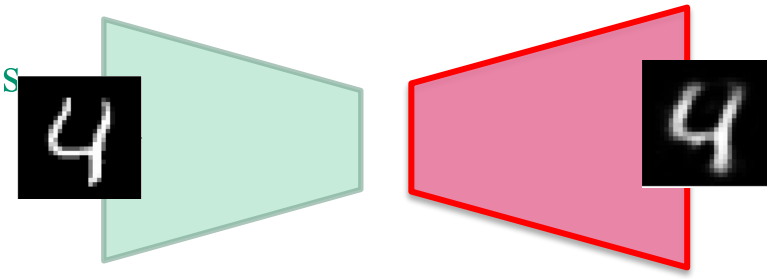
Only reconstruction loss

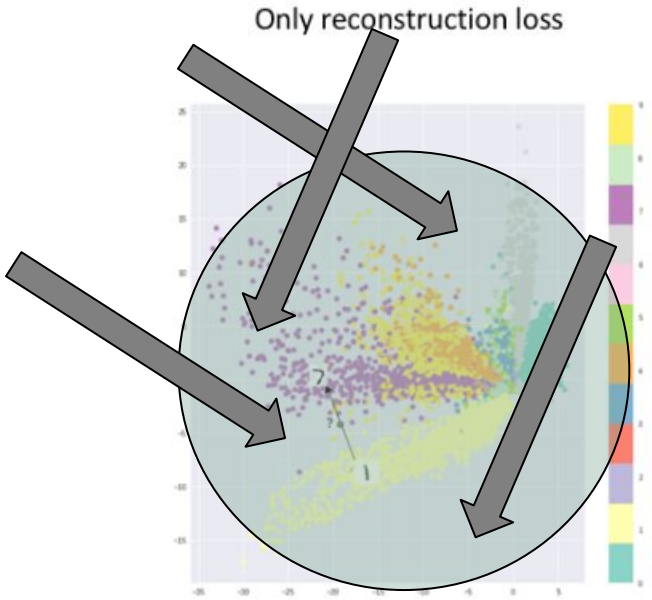


Conclusion

Autoencodeurs:

- Permettent d'apprendre des caractéristiques des données sans étiquettes de classe
- Peuvent être utilisés par des modèles supervisés
- Difficiles à échantillonner

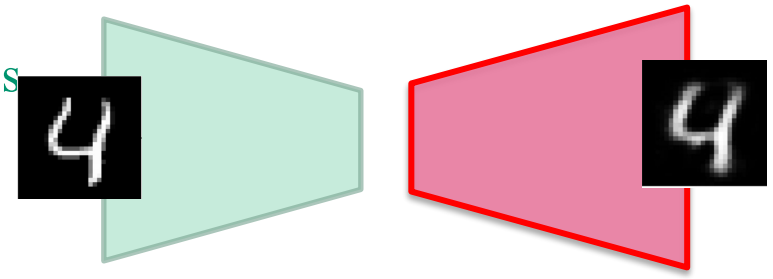




Conclusion

Autoencodeurs:

- Permettent d'apprendre des caractéristiques des données sans étiquettes de classe
- Peuvent être utilisés par des modèles supervisés
- Difficiles à échantillonner



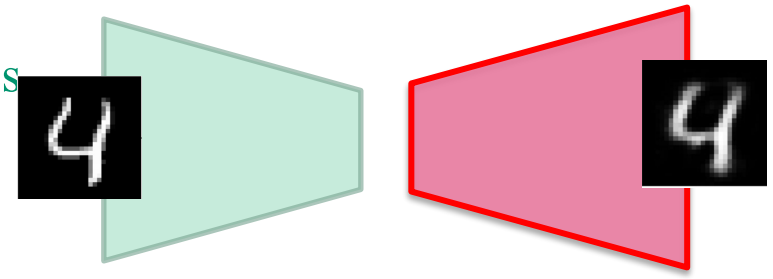
Autoencodeurs variationnels:

- Approche théoriquement intéressante aux AE
- q(z|x) appris est échantillonnable et utile
- Génère des images floues

Conclusion

Autoencodeurs:

- Permettent d'apprendre des caractéristiques des données sans étiquettes de classe
- Peuvent être utilisés par des modèles supervisés
- Difficiles à échantillonner



Autoencodeurs variationnels:

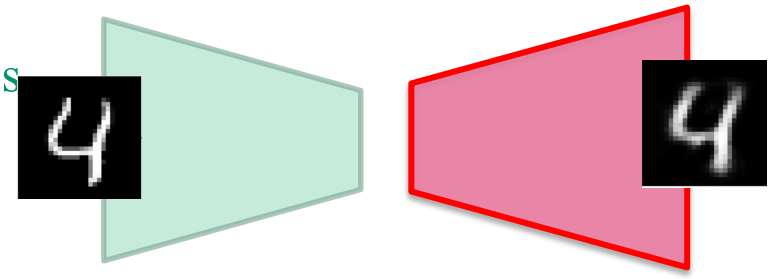
- Approche théoriquement intéressante aux AE
- q(z|x) appris est échantillonnable et utile
- Génère des images floues

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

Conclusion

Autoencodeurs:

- Permettent d'apprendre des caractéristiques des données sans étiquettes de classe
- Peuvent être utilisés par des modèles supervisés
- Difficiles à échantillonner



Autoencodeurs variationnels:

- Approche théoriquement intéressante aux AE
- q(z|x) appris est échantillonnable et utile
- Génère des images floues

Generative adversarial networks (GANs):

- Abandonne l'idée d'apprendre p(x)
- Meilleurs résultats en génération
- Difficiles à entraîner