Réseaux de neurones

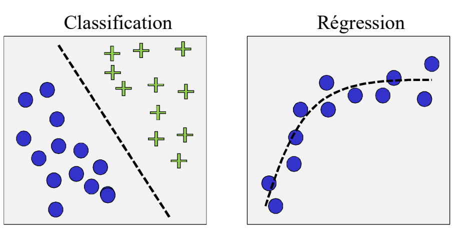
IFT 780

Modèles génératifs

Par

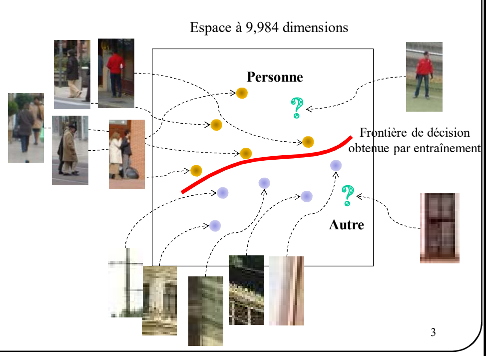
Pierre-Marc Jodoin

Jusqu'à présent : apprentissage supervisé



2

Inria person dataset



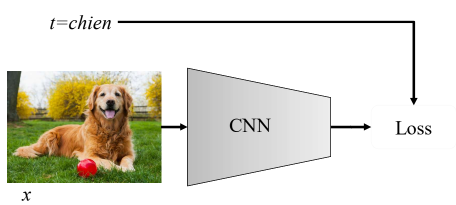
3

Apprentissage supervisé

Deux grandes familles d'applications

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

Apprentissage supervisé avec CNN



Supervisé vs non supervisé

Apprentissage supervisé : il y a une cible

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

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

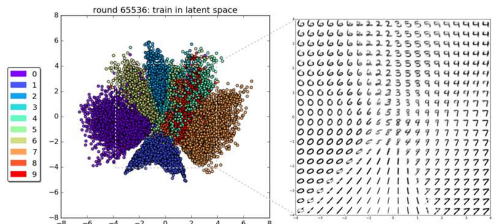
$$D = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$$

Apprentissage non supervisé

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

Applications : clustering, visualization, comprehension, etc.

Exemple : visualization de la distribution des images MNIST



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

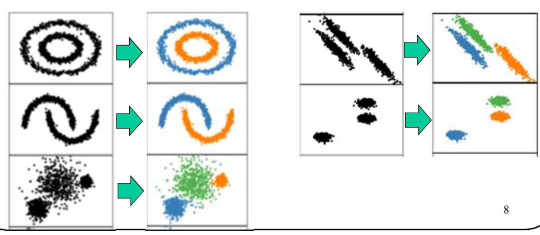
7

Apprentissage non supervisé

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

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

Ex: clustering = retrouver la variable latente "cluster"



8

Pourquoi une variable latente?

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

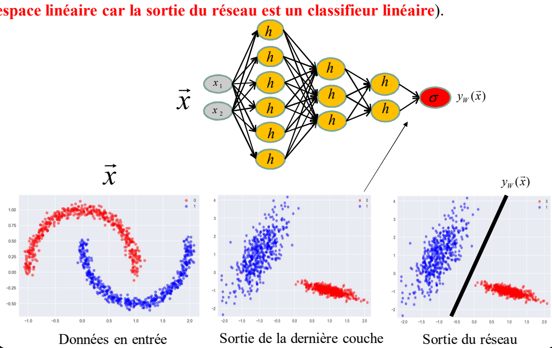
Plus d'info au tableau.

En apprentissage non-supervisé

nous nous appuierons sur
2 propriétés des réseaux de neurones

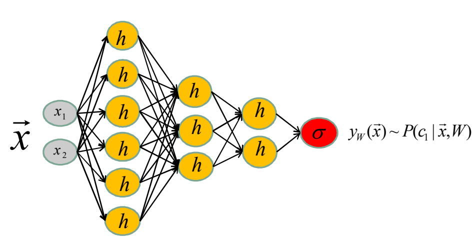
Propriété 1

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**).



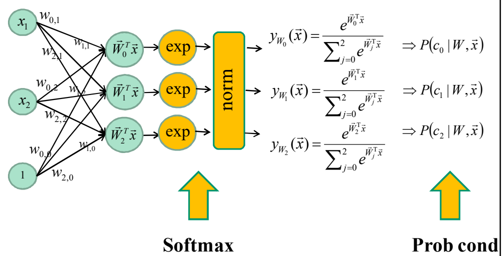
Propriété 2

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



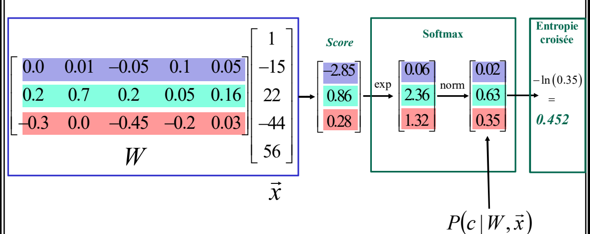
Propriété 2

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

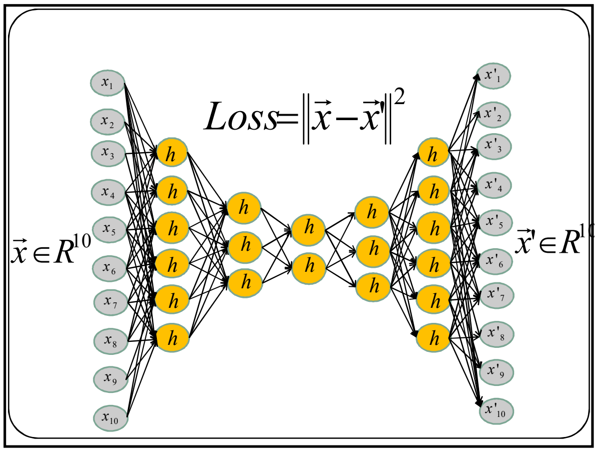


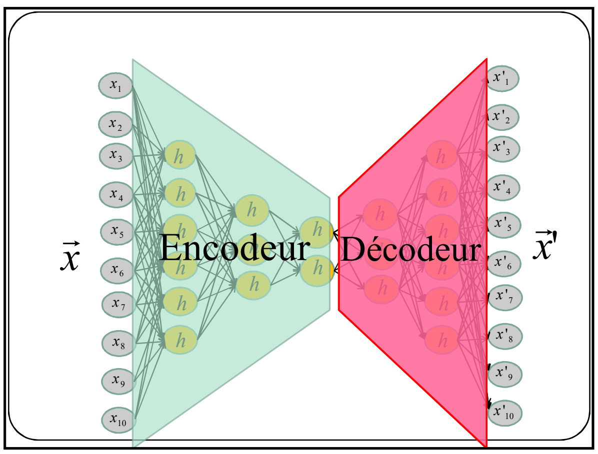
Rappel

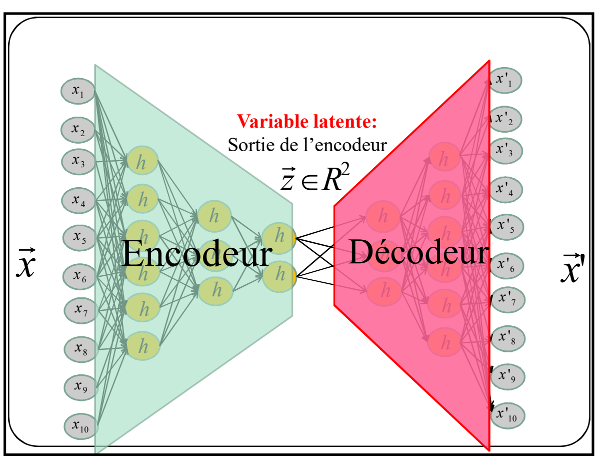
$$\vec{x} = \begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}, t = 2$$

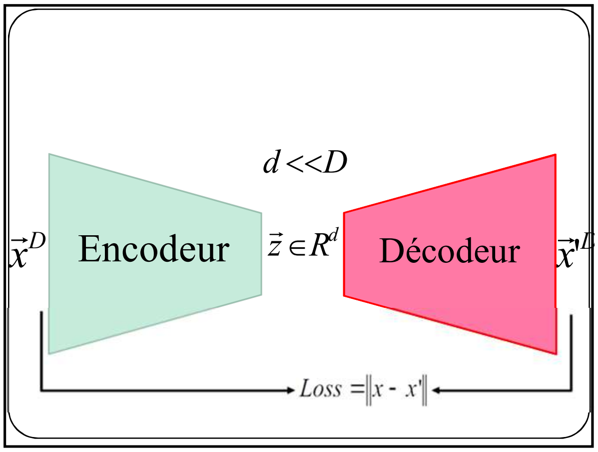


Comment utiliser un réseau de neurones pour apprendre la **configuration sous-jacente** de données non étiquettés?

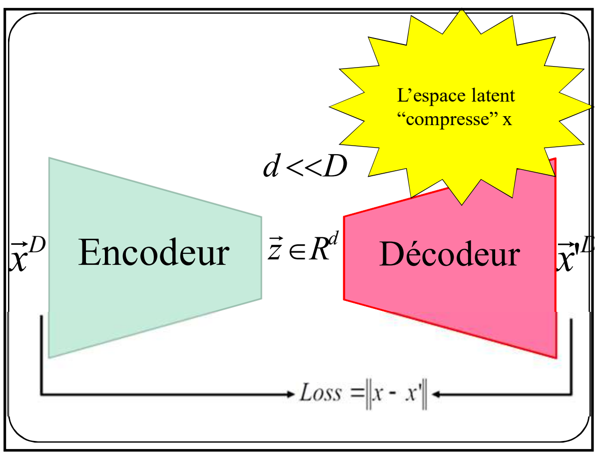

$Loss = \|\vec{x} - \vec{x}'\|^2$
$\vec{x} \in R^{10}$
$\vec{x}' \in R^{10}$


$\vec{x}$
Encodeur
Décodeur
$\vec{x}'$

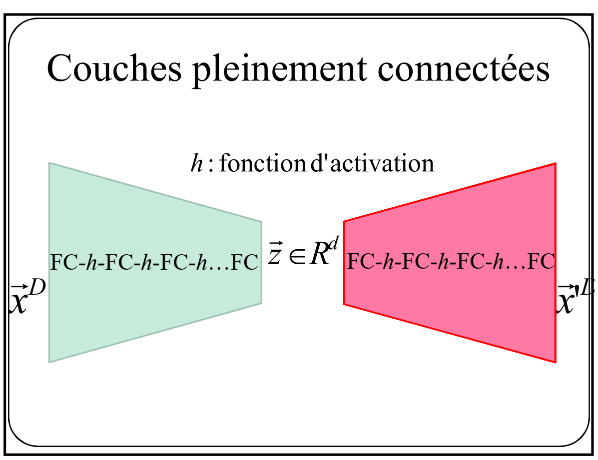

Variable latente:
Sortie de l'encodeur
$\vec{z} \in R^2$
$\vec{x}$
Encodeur
Décodeur
$\vec{x}'$



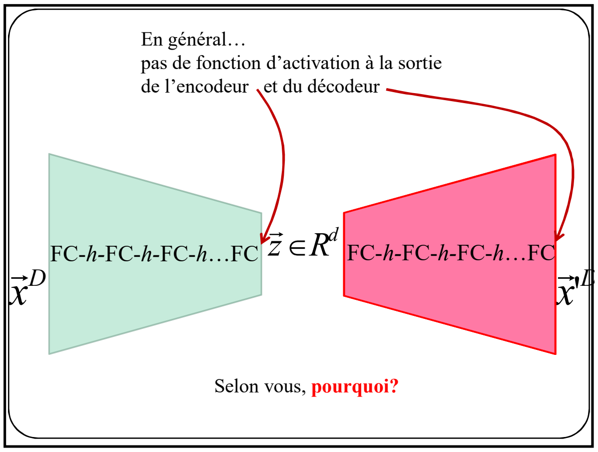
$d << D$
$\vec{x}^D$
Encodeur
$\vec{z} \in R^d$
Décodeur
$\vec{x}'^D$
$Loss = \|x - x'\|$

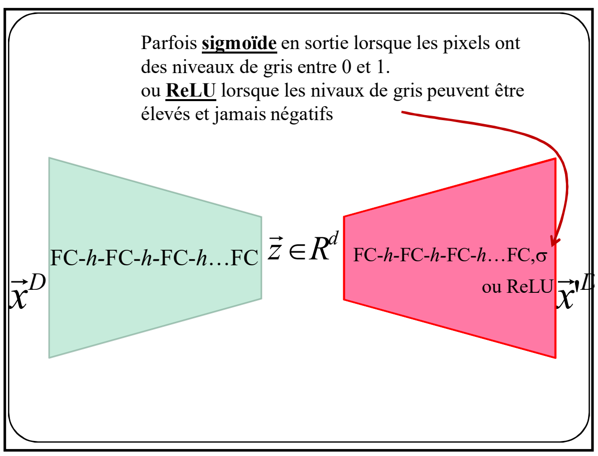


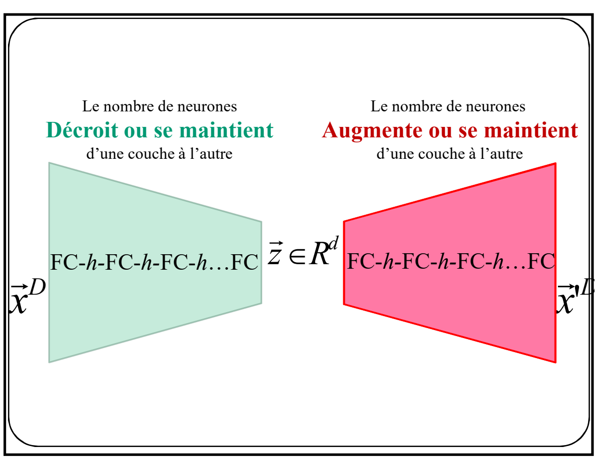
L'espace latent
"compresse" x
$d << D$
$\vec{x}^D$
Encodeur
$\vec{z} \in R^d$
Décodeur
$\vec{x}'^D$
$Loss = \|x - x'\|$



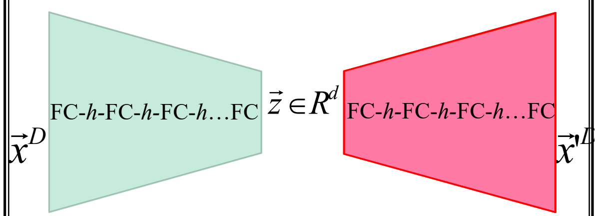
Couches pleinement connectées
h : fonction d'activation
FC-h-FC-h-FC-h…FC
$\vec{z} \in R^d$
FC-h-FC-h-FC-h…FC
$\vec{x}^D$
$\vec{x}'^D$


En général…
pas de fonction d'activation à la sortie
de l'encodeur et du décodeur
FC-*h*-FC-*h*-FC-*h*…FC
$\vec{z} \in R^d$
FC-*h*-FC-*h*-FC-*h*…FC
$\vec{x}^D$
$\vec{x}'^D$
Selon vous, **pourquoi?**


Parfois **sigmoïde** en sortie lorsque les pixels ont
des niveaux de gris entre 0 et 1.
ou **ReLU** lorsque les niveaux de gris peuvent être
élevés et jamais négatifs
FC-*h*-FC-*h*-FC-*h*…FC
$\vec{z} \in R^d$
FC-*h*-FC-*h*-FC-*h*…FC,σ
ou ReLU
$\vec{x}^D$
$\vec{x}'^D$


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
FC-*h*-FC-*h*-FC-*h*…FC
$\vec{z} \in R^d$
FC-*h*-FC-*h*-FC-*h*…FC
$\vec{x}^D$
$\vec{x}'^D$

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

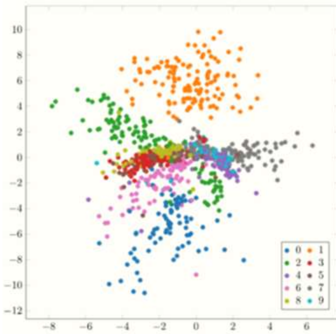
```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128), nn.ReLU(True),
            nn.Linear(128, 64), nn.ReLU(True),
            nn.Linear(64, 12), nn.ReLU(True),
            nn.Linear(12,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

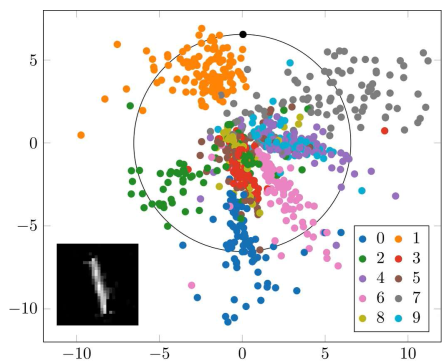
Visualisation de l'espace latent pour 1000 images MNIST

chaque image correspond à 1 point 2D



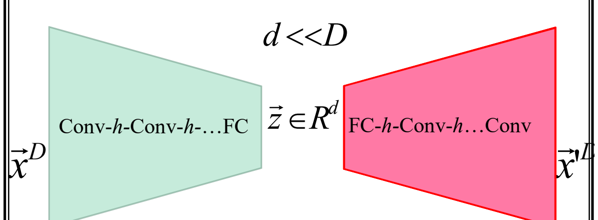
https://gertjanvandenburg.com/blog/autoencoder/

Générer des images avec le décodeur.

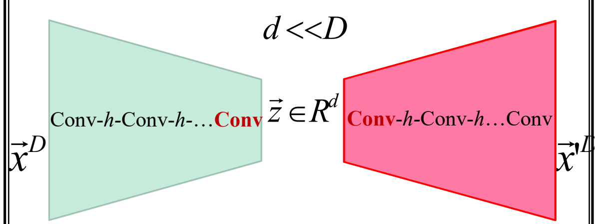


https://gertjanvandenburg.com/blog/autoencoder/

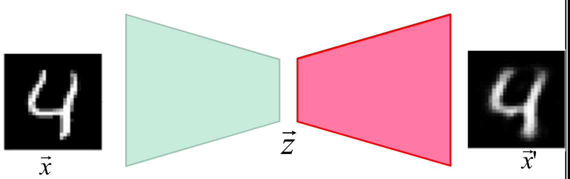
Couches convolutives



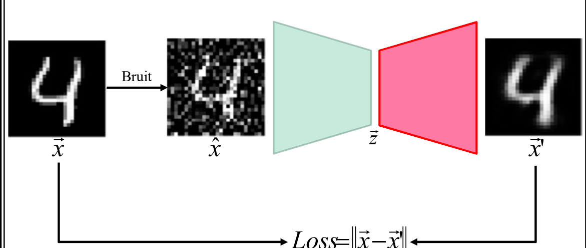
Autoencodeur pleinement convolutif



Autoencodeur de base

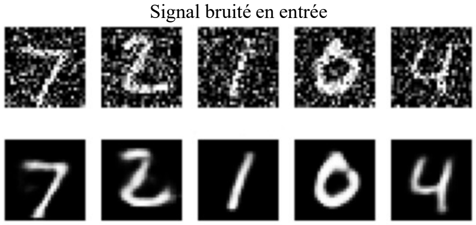


Autoencodeur pour débruitage

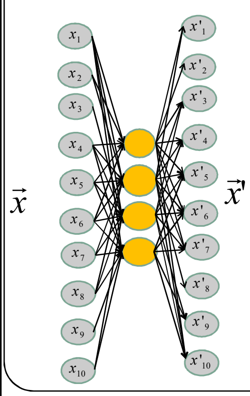


Une fois entraîné…

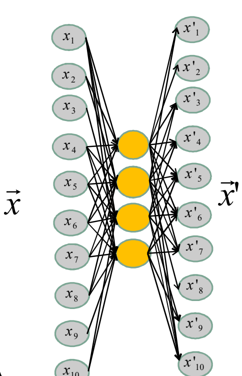
Signal bruité en entrée



Signal reconstruit et débruité



Autoencodeur à **une couche** et **sans fonction d'activation**

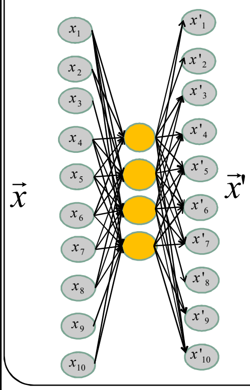


$W_1 \in R^{4\times10}$

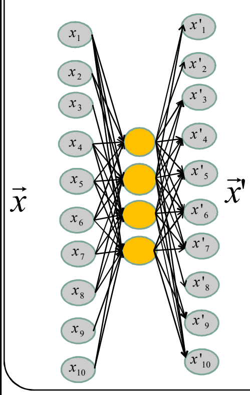
$W_2 \in R^{10\times4}$

$$\vec{z} = W_1\vec{x}$$

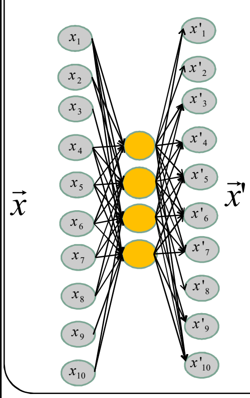
$$\vec{x}' = W_2\vec{z}$$



$$\left.\begin{aligned}\vec{z} &= W_1\vec{x}\\ \vec{x}' &= W_2\vec{z}\end{aligned}\right\}\ \vec{x}' = W_2W_1\vec{x}$$

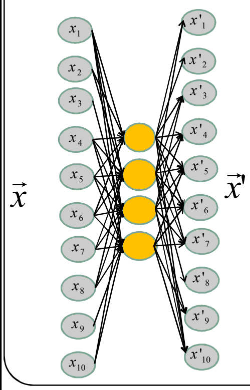


$$Loss = \left\|\vec{x} - W_2W_1\vec{x}\right\|^2$$



On peut démontrer que la solution est

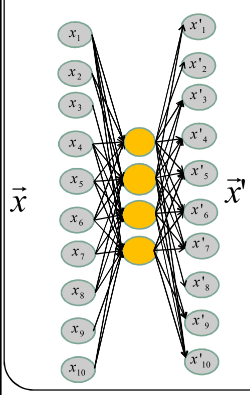
$$W_1 = \left(W_2^T W_2\right)^{-1} W_2^T$$



On peut démontrer que la solution est

$$W_1$$

Matrice constituée des principaux
vecteurs propres
de la matrice de
variance-covariance.
des données



DONC

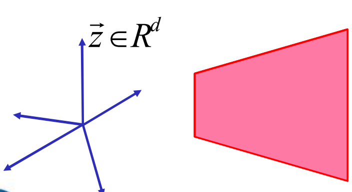
Autoencodeur linéaire

=

Analyse en composante principale
(ACP)

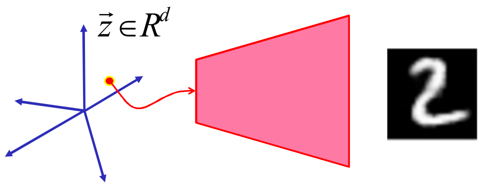
E. Plaut, **From Principal Subspaces to Principal Components with Linear Autoencoders,** arXiv:1804.10253v3, 2018

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

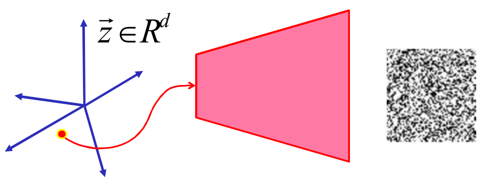


Comment, dans cet espace infini, trouver un point $\vec{z}$
qui donnera une image valable un fois décodée?

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



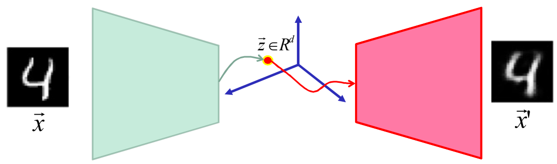
Malheureusement, la vaste majorité du temps, on reproduira du bruit



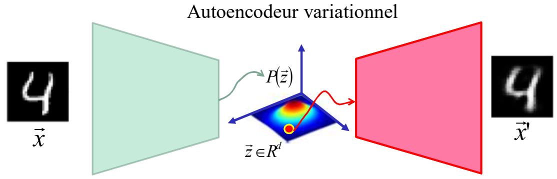
Au lieu d'apprendre à **reproduire un signal d'entrée**…

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

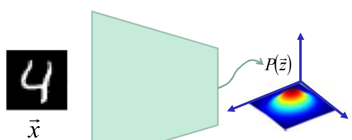
Autoencodeur de base



Autoencodeur variationnel

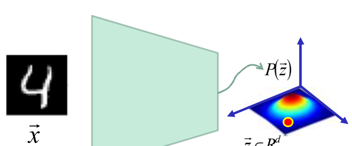


Autoencodeur variationnel



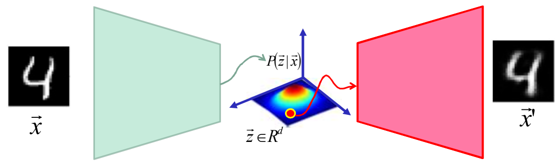
L'encodeur produit une distribution $P(\vec{z})$ et non juste un point $\vec{z}$

Autoencodeur variationnel



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

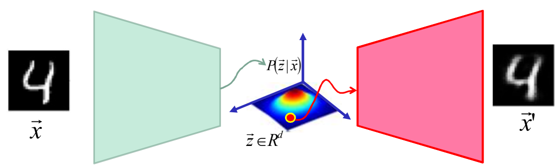
Autoencodeur variationnel



On reconstruit $\vec{x}'$

Autoencodeur variationnel

Remarque 1

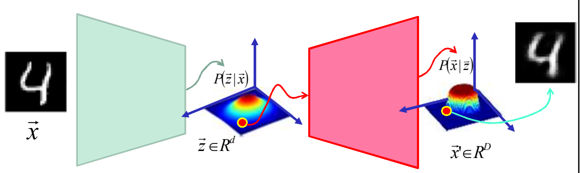


Puisque la distribution de $\vec{z}$ dépend de $\vec{x}$
on dira que la distribution apprise est

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

Autoencodeur variationnel

Remarque 2



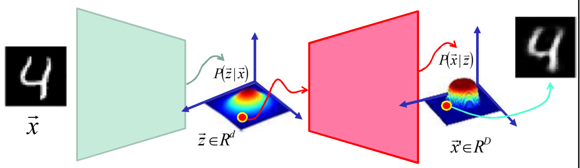
Le **décodeur** peut également produire une distribution de probabilités

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

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

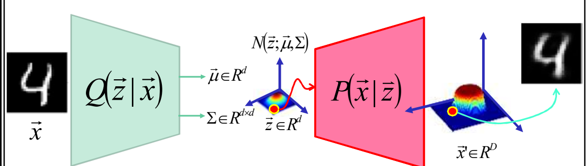
Autoencodeur variationnel

Remarque 3



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

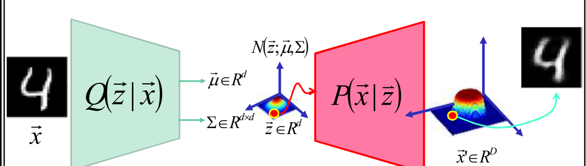
$$Q(\vec{z}\,|\,\vec{x}) \approx P(\vec{z}\,|\,\vec{x})$$



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

Autoencodeur variationnel

Remarque 4

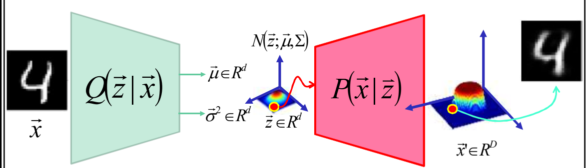


Pour simplifier les calculs, on va supposer que Σ est diagonale

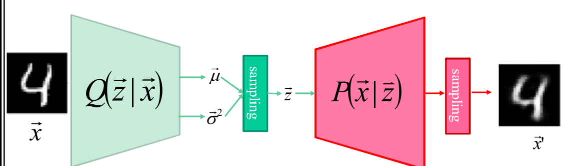
$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3^2 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \sigma_d^2 \end{pmatrix}$$

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

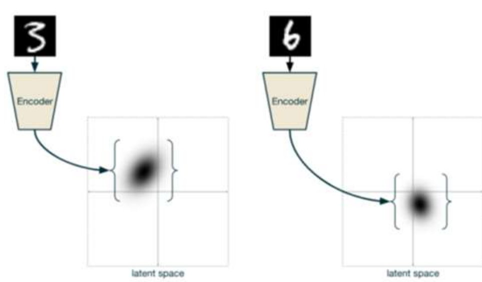
Autoencodeur variationnel



Autoencodeur variationnel

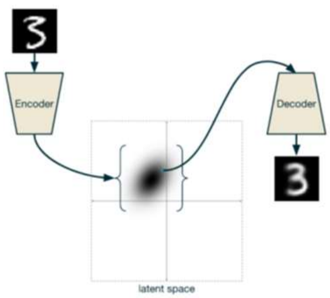


Autre façon de voir les choses…



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

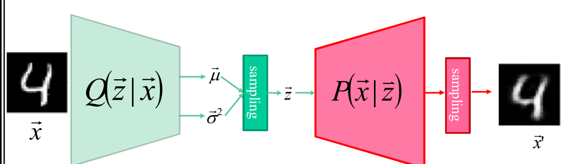
Autre façon de voir les choses…



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

Autoencodeur variationnel

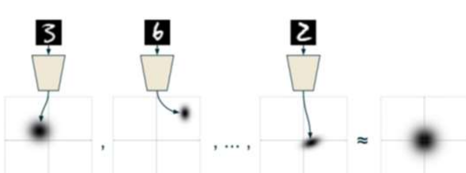
Remarque 5



Distribution *a priori* de $\vec{z}$: gaussienne centrée à 0 et de variance 1

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

Autre façon de voir les choses…

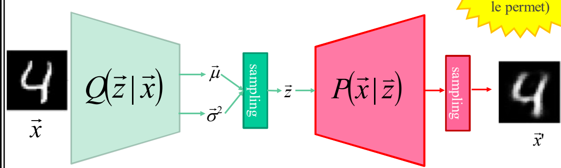


$N(\vec{z};0,1)$

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

Autoencodeur variationnel

Preuve au tableau (si le temps le permet)



ELBO loss : Evidence Lower Bound

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

Autoencodeur variationnel

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

ELBO loss : Evidence Lower Bound

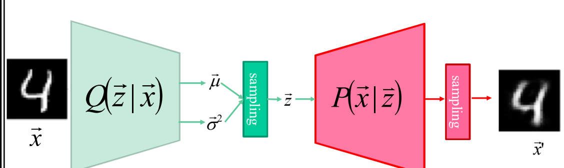
$$Loss = \underbrace{KL\big(N(z;0,1), N(z;\mu,\Sigma)\big)}_{\text{Perte encodeur}} - \underbrace{\log\big(P(x\,|\,z)\big)}_{\text{Perte décodeur}}$$

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

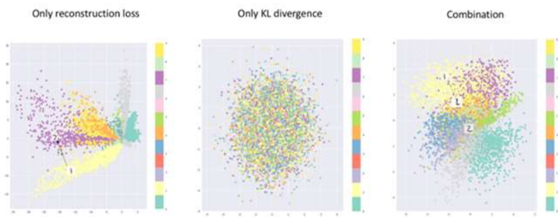
$$Loss = \frac{1}{2}\sum_{i=1}^{d}\underbrace{\left(1+\log\left(\sigma_i^2\right)-\mu_i^2-\sigma_i^2\right)}_{\text{Perte encodeur}} \underbrace{- \lambda\|\vec{x}-\vec{x}'\|^2}_{\text{Perte décodeur}}$$

Autoencodeur variationnel

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

Evidence Lower Bound

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

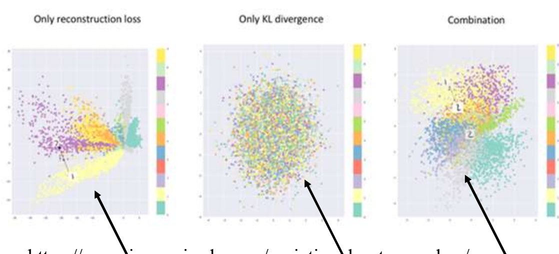


https://www.jeremyjordan.me/variational-autoencoders/

Autoencodeur variationnel

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

ELBO loss : Evidence Lower Bound



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

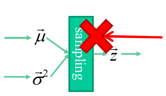
Pas gaussien, difficile à échantillonner

Trop gaussien, tout est indistinguable

Ensemble de distributions séparées qui forment une gaussienne

Autoencodeur variationnel

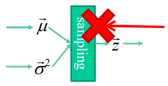
Remarque 6



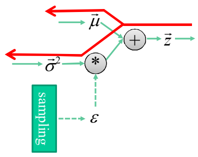
Pas de rétro-propagation à travers un processus d'échantionnage

$$\vec{z} \sim N(\vec{z};\vec{\mu},\Sigma)$$

Autoencodeur variationnel
Remarque 6



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



Reparameterization trick
$\vec{z} = \vec{\mu} + \varepsilon\vec{\sigma}^2$

Autoencodeur variationnel jouet MNIST : d=32 dim

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

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

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

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

Reparameterization trick

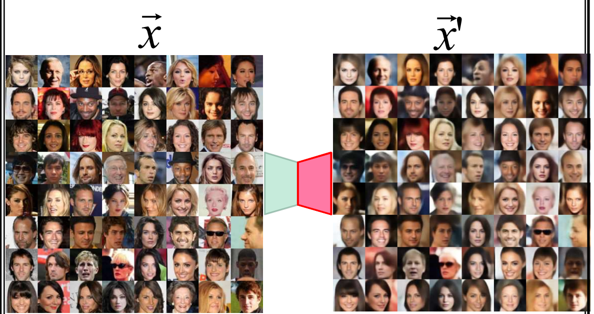
Autoencodeur variationnel jouet MNIST : d=32 dim

```
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')

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

    return KLD + self.lambda*BCE
```

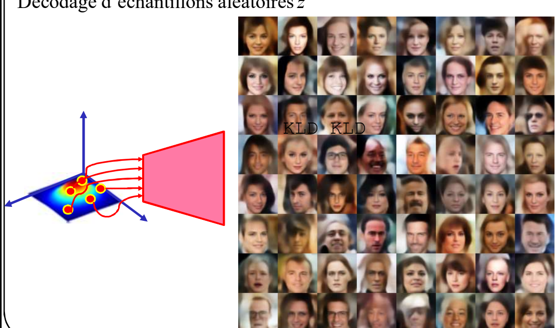
Ex.: base de données *CelebA*



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

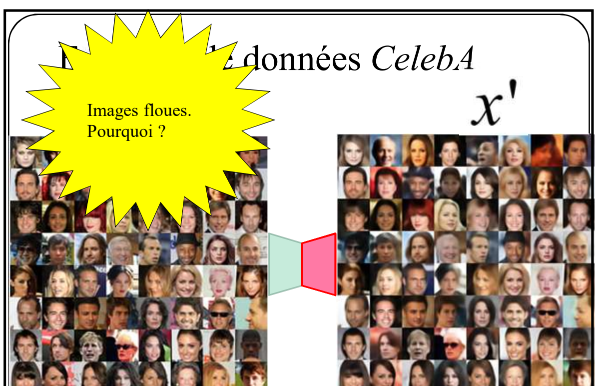
Ex.: base de données *CelebA*

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



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

… données *CelebA*



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

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

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

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

Modèle génératif   Distribution *a priori*

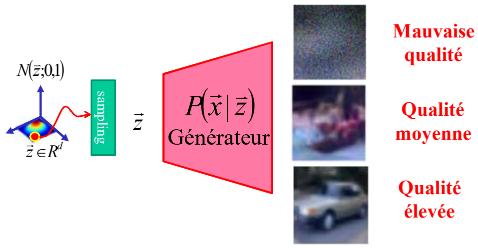
Comme pour les VAE, on utilisera une **distribution *a priori*** facile à échantillonner : une **gaussienne**!

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

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

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

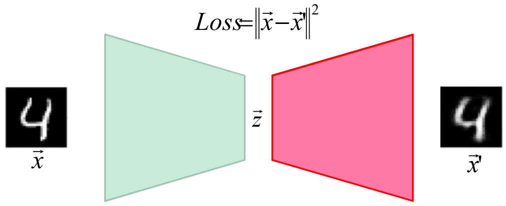
Dépendamment des performances du décodeur, les images générées **seront de qualité très variable.**



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

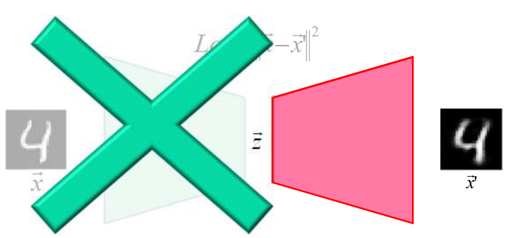
Pour un autoencodeur (variationnel ou non) c'est facile! car on a un encodeur et une image de référence

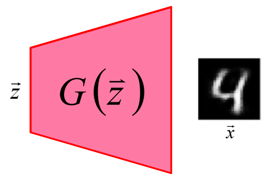
$$Loss = \|\vec{x} - \vec{x}'\|^2$$



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**?





Loss sans cible de référence?

$\vec{z}$

Approximer la perte à l’aide d’un

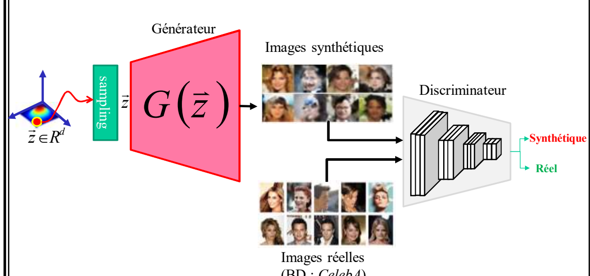
2^{e} réseau de neurones

et d’une

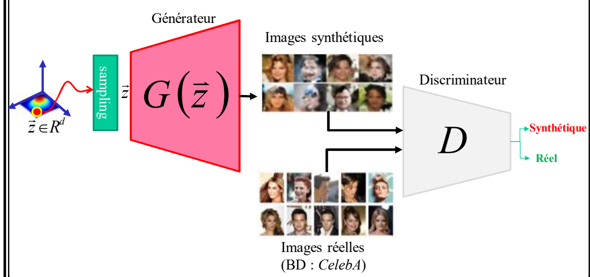
Base de données de référence

référence?

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

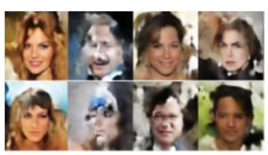


Pour entraîner un décodeur, il faut une **fonction de perte** (loss) qui **mesure la qualité des images produites**



Données étiquetées

Produites par le générateur



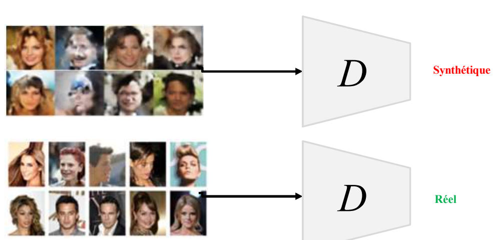
$t = 0 \quad ('synthétique')$

$t = 1 \quad ('réel')$

Issues d'une vraie BD

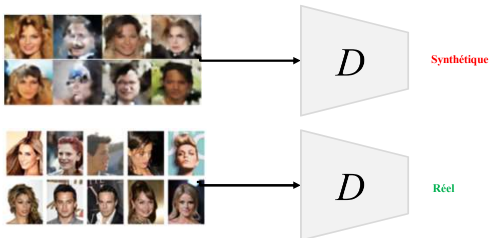
Deux réseaux aux **objectifs différents** :

Discriminateur : différentie les images synthétiques des images réelles



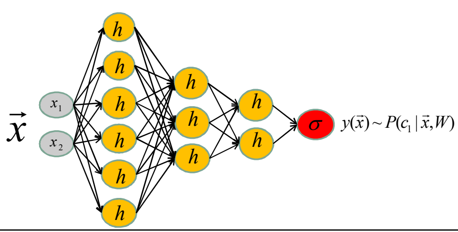
Discriminateur : classifieur binaire (régression logistique)

=> Perte **l'entropie croisée**



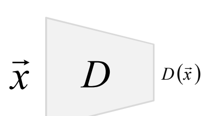
Rappel, entropie croisée pour une régression logistique binaire:

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



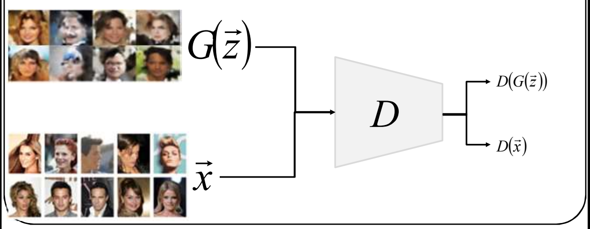
Le réseau discriminateur est représenté par la **lettre D**

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



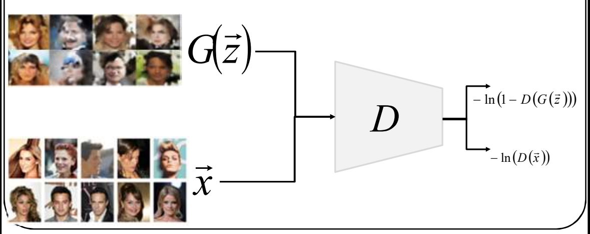
Puisque les images **synthétiques** ont été générées par le **générateur**

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



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

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



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

$$IE[x] = \int xp(x)dx$$
$$\approx \frac{1}{N}\sum_{i=1}^{N} x_i \quad \text{où } x_i \sim p(x)$$

approximation Monte Carlo

$$IE[f(x)] = \int f(x)p(x)dx$$
$$\approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \quad \text{où } x_i \sim p(x)$$

Rappel: Espérance mathématique et estimateur Monte Carlo

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

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

(Loss de GAN dans la littérature)

Objectif du discriminateur

Paramètres du discriminateur

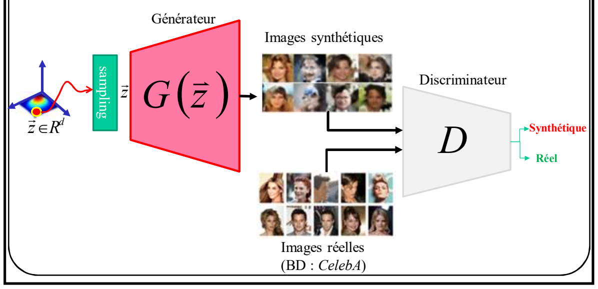
$$W_D = \arg\min_{W_D} -IE_{\vec{x}\sim P_{reel}}\left[\ln\left(D(\vec{x})\right)\right] - IE_{\vec{z}\sim P_z}\left[\ln\left(1 - D\left(G(\vec{z})\right)\right)\right]$$

Ou encore, de façon équivalente

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

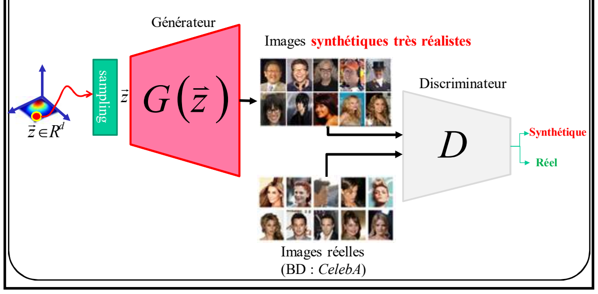
Objectif du générateur

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

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

Objectif du générateur :

produire des images synthétiques indistinguables des images réelles

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

« 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

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\overbrace{[\log D(\boldsymbol{x})]}^{\text{Discriminateur veux } D(x)=1 \text{ pour les vrais données}} + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - \overbrace{\underbrace{D(G(\boldsymbol{z})))]}_{\text{Générateur veux } D(G(x)) = 1 \text{ pour les données synthétiques}}}^{\text{Discriminateur veux } D(G(x)) = 0 \text{ pour les données synthétiques}}.$$

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

NOTE

dans les faits, on ne minimise pas cette loss

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

on maximise plutôt celle-ci

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

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

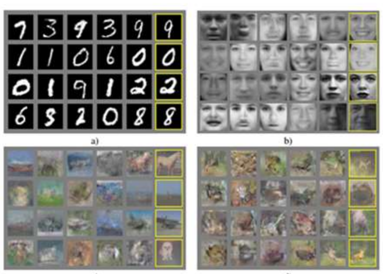
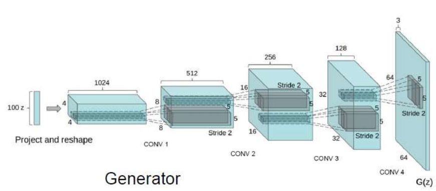


Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative 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

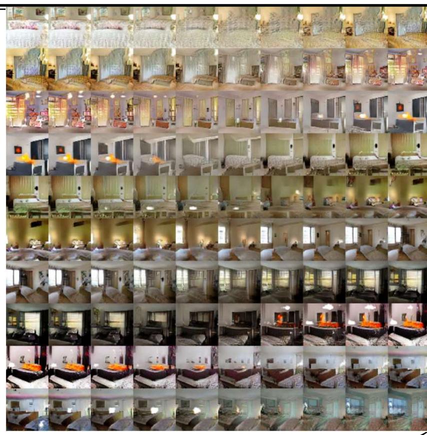
https://github.com/soumith/ganhacks

Autre recommandations
- BatchNorm partout
- Pas de FC, juste des conv

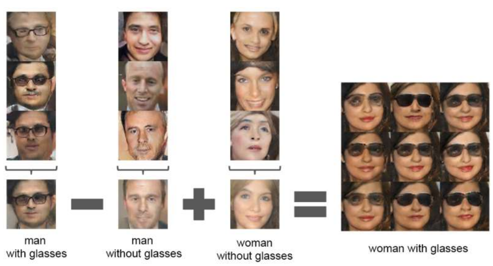
Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

Deep Convolution Generative Adversarial Net (DCGAN)

Interpolation entre 9 vecteurs latents aléatoires



"Vector arithmetic for visual concepts"



Problèmes d'instabilité

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

111

Problèmes d’instabilité

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

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

112

Problèmes d’instabilité

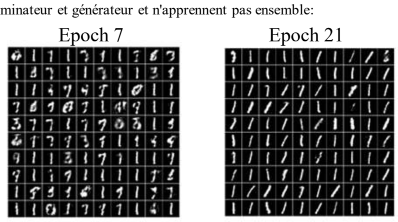
- Si discriminateur et générateur et n'apprennent pas ensemble:
  - disparition des gradients
  - **effondrement des modes**
  - on ne peut générer d’images à haute résolution

Le générateur peut apprendre à générer tout le temps la même image qui bat le discriminateur

113

Problèmes d’instabilité

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



Le gén… bat
le disc…

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

114

LS GAN

Problème des GANs de base

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

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

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

115

“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

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

116

“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

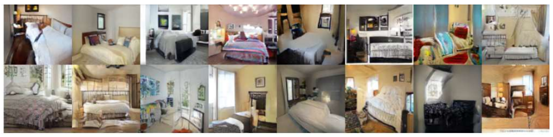
Pour LS GAN, la sortie du réseau n’est plus une sigmoïde

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

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

“Least Squares GAN” Mao et al. ICCV’17

LS GAN



(a) Generated by LSGANs.

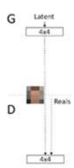
(b) Generated by DCGANs (Reported in [11]).

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

progressive GAN

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

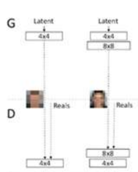
On commence avec des images de **faible résolution : 4x4 pixels**



"Progessive GAN" Karras et al. ICLR'18

progressive GAN

Et progressivement, on augmente la résolution de l'image

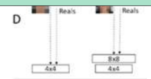


"Progessive GAN" Karras et al. ICLR'18

progressive GAN

Et progressivement, on augmente la résolution de l'image

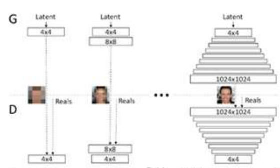
"*Progressive Growing GAN requires that the capacity of both the generator and discriminator model be expanded by adding layers during the training process*"



"Progessive GAN" Karras et al. ICLR'18

progressive GAN

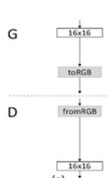
Et progressivement, chaque couche qu'on ajoute vient bonifier la couche précédente : cela se fait à l'aide d'une **opération « résiduelle ».**



"Progessive GAN" Karras et al. ICLR'18

Ajout de couches

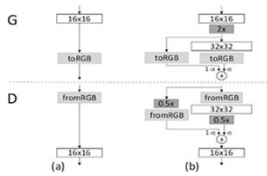
Lorsque **l'entraînement** d'une couche de résolution RxR (ici 16x16) est **terminé**…



"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.

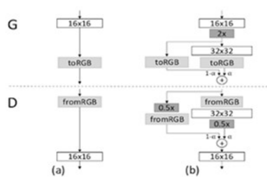


"Progessive GAN" Karras et al. ICLR'18

Ajout de couches

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

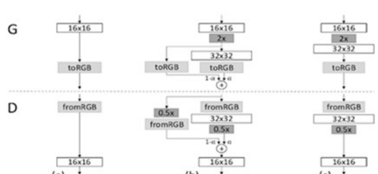
Au début de l'entrainement, α=0 et progressivemnt α augmente pour atteindre α=1 à la fin



"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**

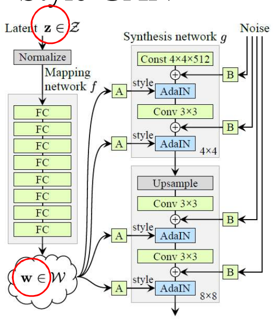
Table 2: Generator and discriminator that we use with CELEBA-HQ to generate 1024×1024 images.

"Progessive GAN" Karras et al. ICLR'18

https://youtu.be/XOxxPcy5Gr4

128

Style GAN

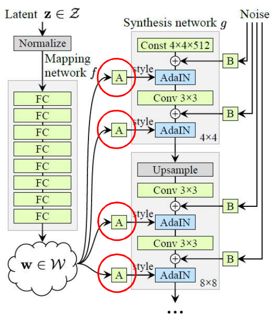


(b) Style-based generator

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



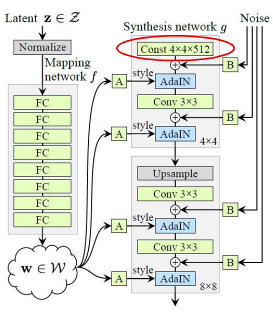
(b) Style-based generator

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

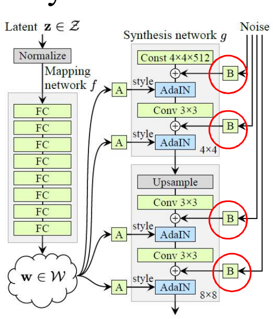


(b) Style-based generator

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

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

Style GAN



(b) Style-based generator

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

132

Effet du bruit

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

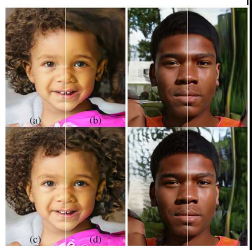
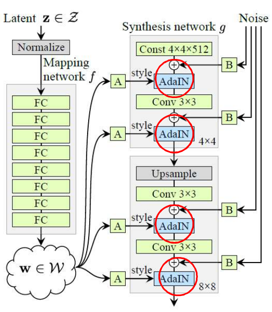


Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only ($64^2 - 1024^2$). (d) Noise in coarse layers only ($4^2 - 32^2$). We can see that the artificial omission of noise leads to featureless "painterly" look.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

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**

134

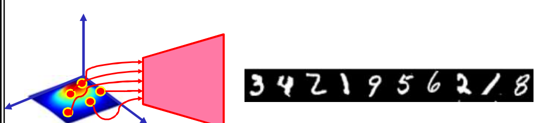
Style GAN

Entraînement progressif comme pour **progressive GAN**

Style GAN

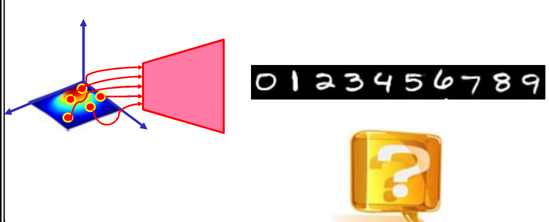
Défi avec les GAN

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



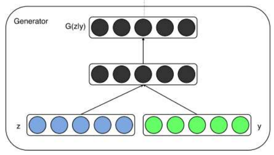
Défi avec les GAN

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?



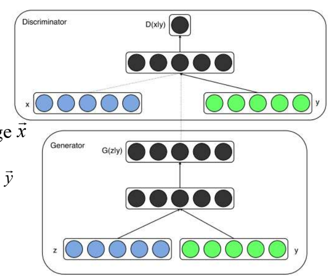
Gan conditionnel

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

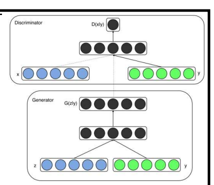


Et de discriminer une image $\vec{x}$

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

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

Gan conditionnel



GAN de base

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

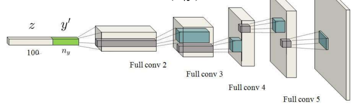
GAN conditionnel

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

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

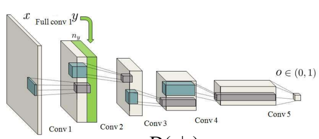
Version convolutionnelle

G(z|y)



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

Version convolutionnelle

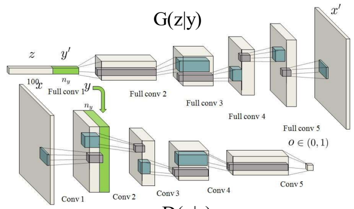


D(x|y)

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

Version convolutionnelle

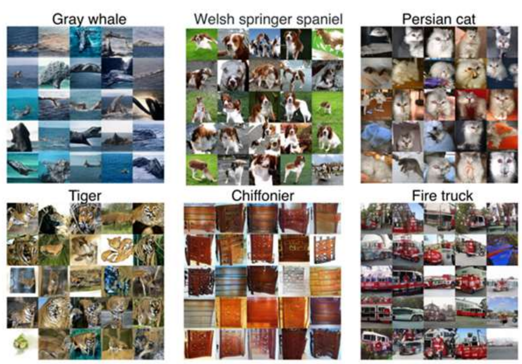
G(z|y)



D(x|y)

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

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



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

Code pytorch pour plus de 30 modèles de GANs

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

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

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