# RCP217 —Intelligence artificielle pour des données multimédia

**Responsable UE : Marin FERECATU**
Conservatoire National des Arts et Métiers
Lab. CEDRIC, Equipe Vertigo
http://cedric.cnam.fr/~ferecatu/

le cnam

# Plan de la séance

- Contexte
- Apprentissage en flux  vs. apprentissage statique
- Dérive conceptuelle et oubli catastrophique
- Solutions :
  - Méthodes par régularisation
  - Architectures DNN évolutives
  - Méthodes dual-memory

# Contexte

Anglais : *stream learning*

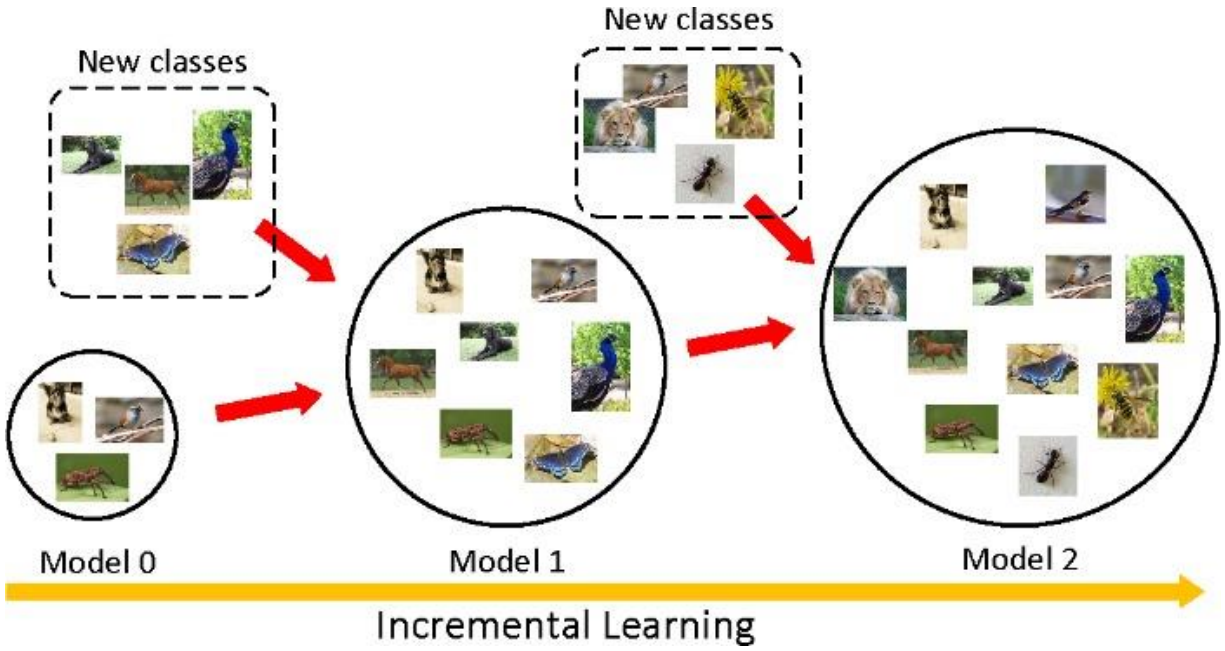
Flux/flot de données :

- Séquence ordonnée d'éléments (dans le temps)
- Lisibles une seule fois (ou nb. de fois limité)
- Données continus, illimités, arrivent avec une grande rapidité
- Système limité en capacité mémoire et stockage.
- Distribution statistique qui change avec le temps.

Exemples : le trafic réseau, les conversations téléphoniques, les transactions ATM, les recherches sur le web, et les données des capteurs (météo, médicales, satellite, etc).

# Apprentissage en flux

- Apprentissage en flux de données / apprentissage incrémental



# Plan de la séance

- Contexte
- Apprentissage en flux  vs. apprentissage statique
- Dérive conceptuelle et oubli catastrophique
- Solutions :
  - Méthodes par régularisation
  - Architectures DNN évolutives
  - Méthodes dual-memory

# Apprentissage statique

**Apprentissage statique (static learning) :**

- Données d'apprentissage disponible dès le départ
- Mise-à-jour des données
  - Apparition des nouvelles classes
  - Disparition des classes
- Pas de problème de disponibilité de donnés

**Solution :**

- Ré-apprentissage
- Fine tuning (pour des données supplémentaires)

# Apprentissage statique

- Pas de problème de disponibilité de donnés
- Pas de contrainte de temps réel

Ré-apprentissage possible

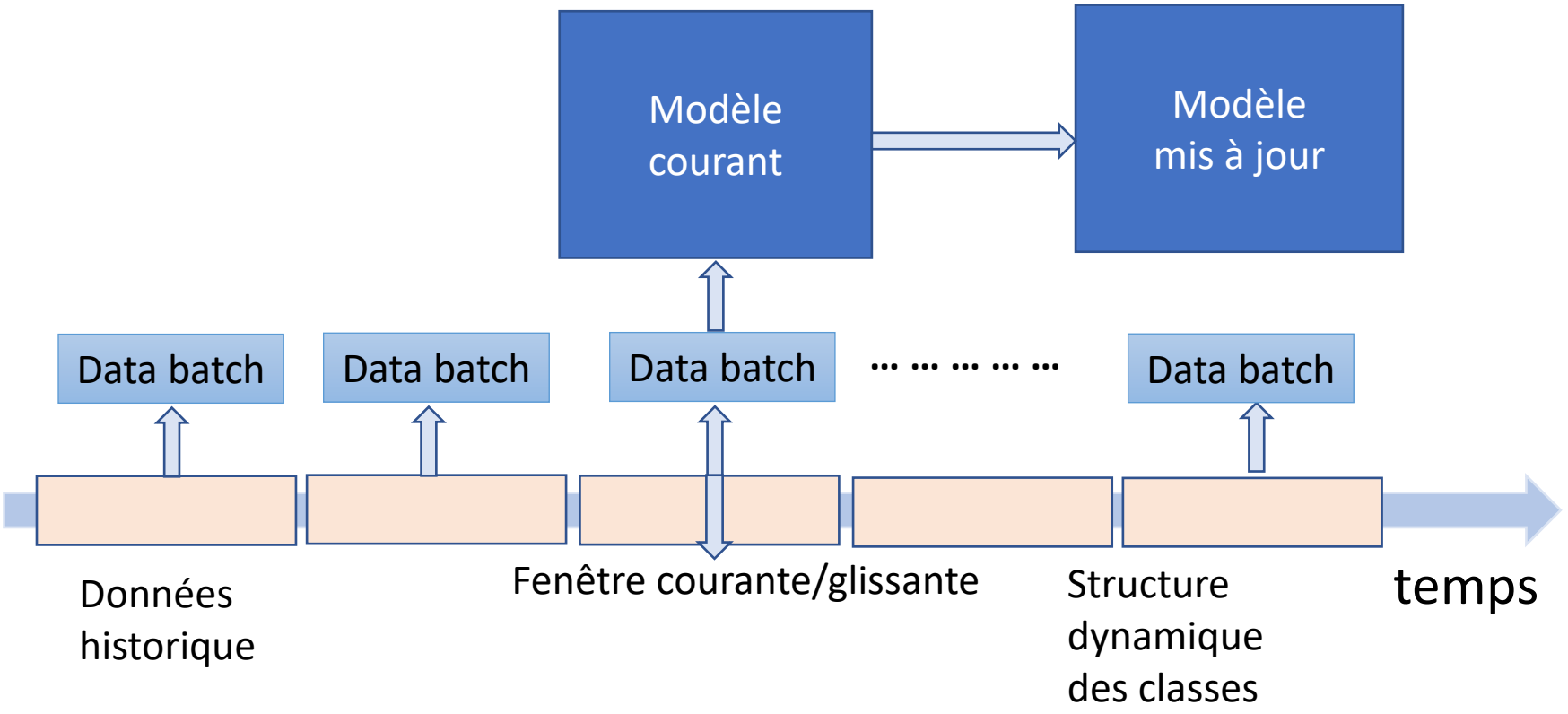
Mise-à-jour du modèle possible

# Apprentissage en flot (EN: stream)

**Contraintes :**

- Les données arrivent de façon continu (incrémentale)
- La composition/structure des données change :
  - Des nouvelles classes peuvent apparaitre
  - Une parties des classes peuvent disparaitre (et réapparaitre plus tard dans le flux)
    - Non-stationarité
- Souvent on veut un modèle à jour à chaque instant (contrainte de temps réel)
- Souvent les données historique ne sont pas stockées

# Apprentissage en flux



# Apprentissage en flux

**Applications :**

- Analyse de tendance sur les réseaux sociaux
- Traitement des flux vidéos
- Surveillance de capteurs
- Trafique en temps réel
- Environnement (météo, pollution, etc.)
- Données de l'espace (images satellite multi-résolution)
- etc.

# Apprentissage en flux

**Exigences pour un système d'apprentissage en flux :**

- Temps réel : apprentissage en continu (*life long learning*)
- Données limitées : apprendre rapidement à partir de quelques exemples et de concepts inégalement représentés.
- Passage à l'échelle : grand nombre de classes
- Protection des connaissances déjà acquises
- Mémoire à long terme : ne pas oublier les connaissances déjà acquises.

# Online learning vs. batch learning

| **Static dataset** | **Data streams** |
| --- | --- |
| Dataset is of a limited size and exact training time per epoch can always be estimated | Data might come with a very high speed → Need **real time data processing, information extraction and learning mechanism.** |
| Sizes of data classes are usually similar or can be equalized before training | Stream length for particular classes can differ a lot from one class to another → Want the learning system to **learn equally well on classes of different sizes**. |

Ref. [1]

# Online learning vs. batch learning

| **Static dataset** | **Data streams** |
|---|---|
| The whole dataset is available for learning and retraining at any moment. | Streams are potentially of a very big (infinite) size → Want to **avoid storing historical data**. |
| All the data classes are available at every training epoch, also training is slow with gradient-based methods, usually need multiple training epochs | At some point, several data classes can never appear again in the stream → Need to **avoid catastrophic forgetting** of already learned information. |
| Number of classes together with labels for each data sample are provided before training | Data classes, never seen by the learning system before, can appear → Need a model, able to **adapt to new data classes** |

Ref. [1]

# Plan de la séance

- Contexte
- Apprentissage en flux  vs. apprentissage statique
- Dérive conceptuelle et oubli catastrophique
- Solutions :
  - Méthodes par régularisation
  - Architectures DNN évolutives
  - Méthodes dual-memory

# Apprentissage en flux

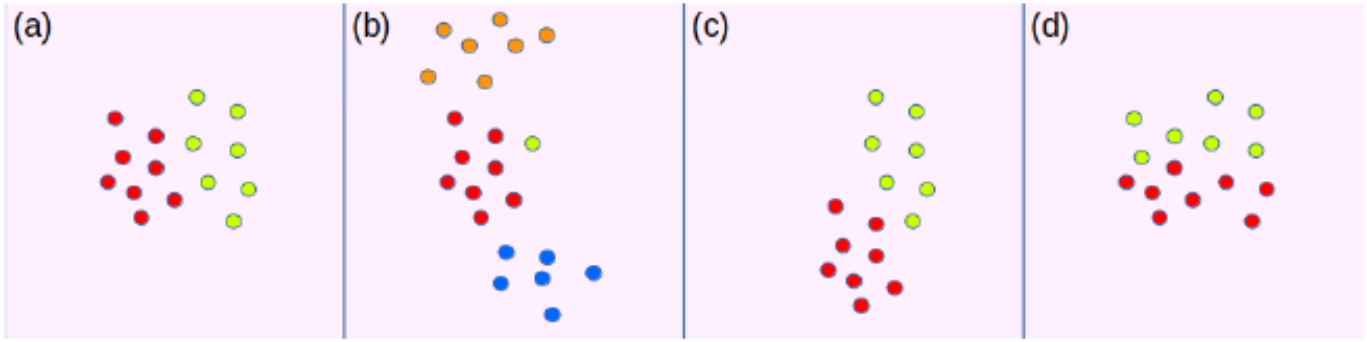
**Les données en flux :**

- Non i.i.d. : données échantillonnés à partir d'une distribution changeante
- Données dynamiques : gérer les dérives dans la distribution des données (**concept drift**) :

$$p_{t_0}(X, y) \neq p_{t_1}(X, y)$$

# Dérive conceptuelle (concept drift)

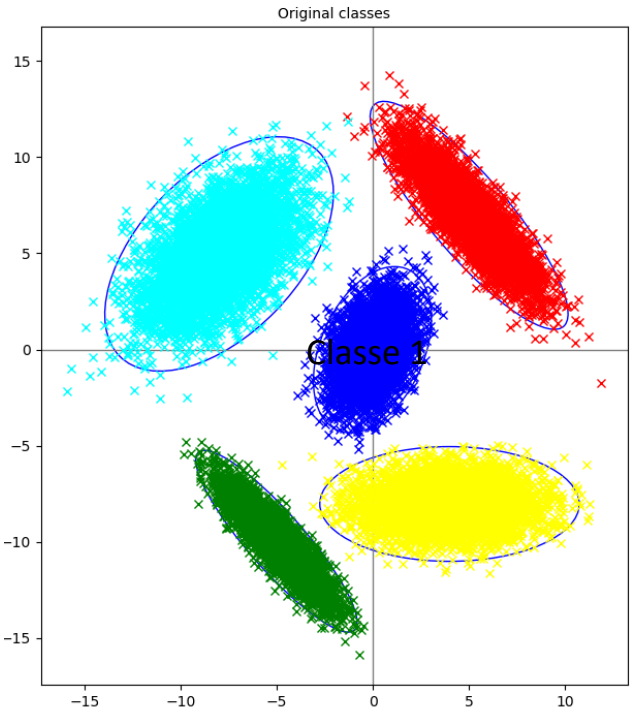
Données dynamiques : gérer les dérives dans la distribution des données (**concept drift**) :

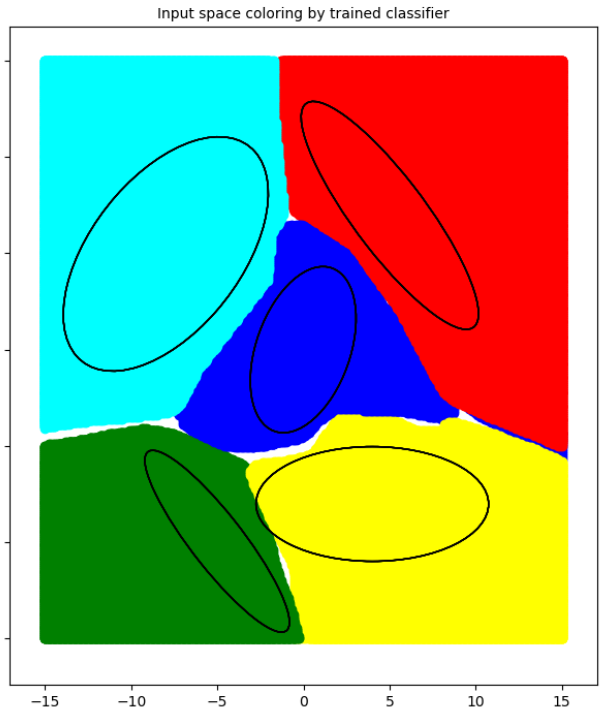



$$p_{t_0}(X, y) \neq p_{t_1}(X, y)$$

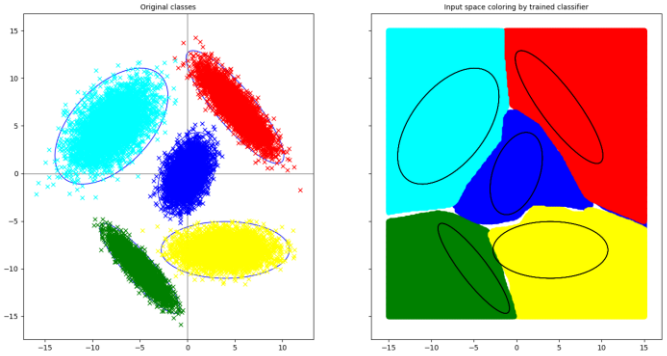
# Dérive conceptuelle (concept drift)

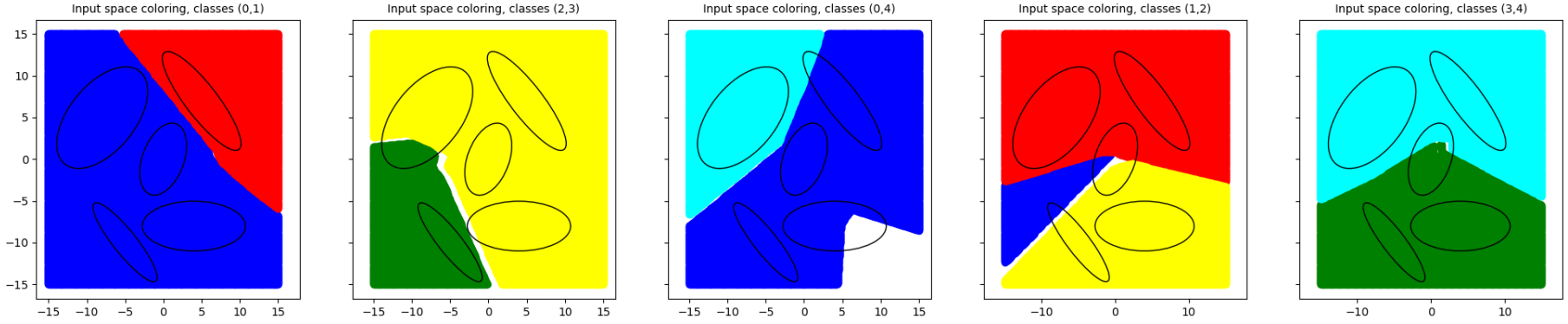
- 5-class classifier (**Linear(2, 6)** → ReLU → **Linear(6, 6)** → ReLU() → Linear(6, 5))



# Dérive conceptuelle (concept drift)



# Dérive conceptuelle (concept drift)

Données dynamiques : gérer les dérives dans la distribution des données (**concept drift**) :

$$p_{t_0}(X, y) \neq p_{t_1}(X, y)$$

Conséquence :

- Catastrophic forgetting : oubli catastrophique – les nouvelles classes (par la rétropropagation) vont effacer progressivement les anciennes classes

# Oubli catastrophique

**Solution non-raisonnable :**

- Conserver toutes les données historiques tout en acquérant les nouvelles données
  - Demande trop de place pour tout stocker
  - Demande trop de temps pour tout ré-entrainer

**Etat de l'art :**

- Approches basées sur la régularisation
- Architectures DNN évolutives
- Méthodes « dual-memory »

# Plan de la séance

- Contexte
- Apprentissage en flux  vs. apprentissage statique
- Dérive conceptuelle et oubli catastrophique
- Solutions :
  - Méthodes par régularisation
  - Architectures DNN évolutives
  - Méthodes dual-memory

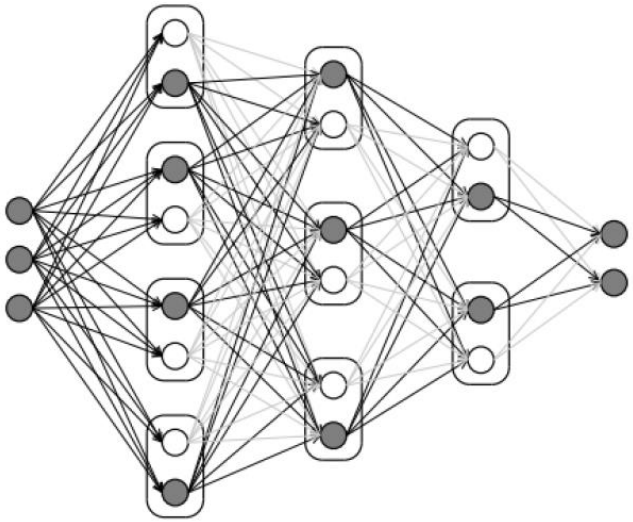
# Régularisation

**Régularisation** : mis à jour du réseau (des poids)

- Eviter le sur-apprentissage
- Conséquence : meilleure généralisation

# Régularisation

**Local Winner Takes All (LWTA)** [4] Schmidhuber et al.



A new type of “non-linearity” : seulement les neurones le plus actifs propagent l’information et sont mis à jour pour un batch d’apprentissage.

MNIST : Split 50%(A)-50%(B) : train on A, retrain on B, test on A.

57.84% (sigmoid), 16.63% (ReLU) to 6.12% (LWTA)

# Régularisation

**Learning without Forgetting (LwF)** [5] :

- Initialiser un ensemble de paramètres pour chaque nouvelle tâche : mis à jour sans aucune restriction.
- Un grand pool de paramètres partagés entre toutes les tâches : mis à jour très lentement pour éviter de changer les performances sur le taches déjà apprises
- Désavantages :
  - Nécessite des stockage de données pour chaque tâche
  - Complexité qui croît linéairement avec le nombre de tâches

# Régularisation

**Elastic Weight Consolidation** [6] :

- Cout quadratique supplémentaire pour la différence entre le modèle mis à jour et sa version historique, apprise sur les tâches précédentes.
- La pénalité ralentit les mises à jour des pondérations pertinentes pour les anciennes tâches.
- Désavantages : assez lourd à mettre en œuvre (calcul de Ficher Information Matrix - FIM), possible seulement hors ligne

# Régularisation

**PathNet** [1] : contrôler le flux d'informations dans le réseau en créant des (bandes de) chemins spécifiques pour chaque tache à l'intérieur du réseau.

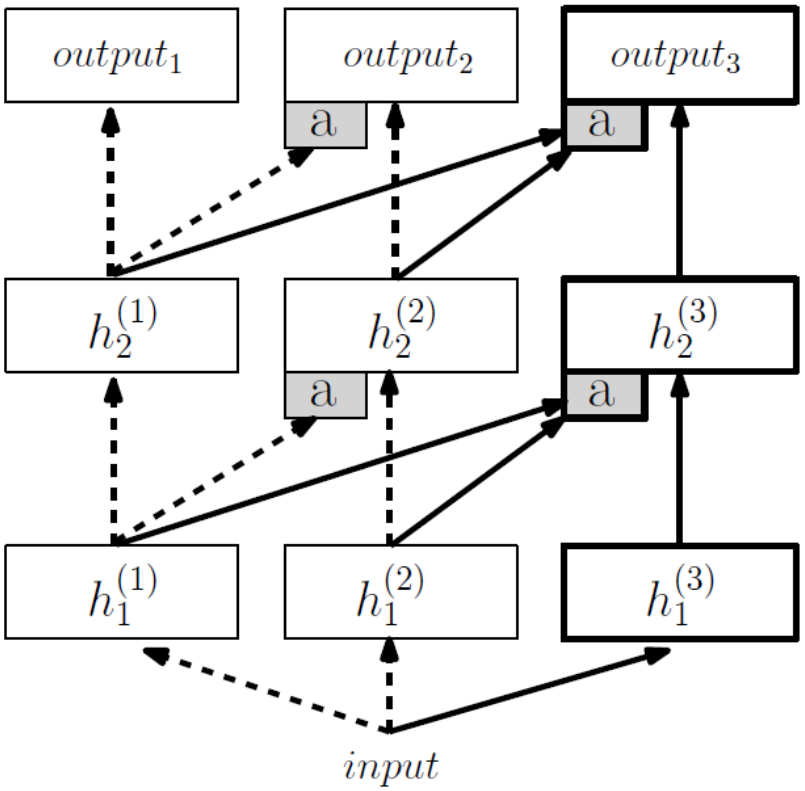
Désavantages : des modèles assez lourds, longs à entrainer, possible seulement hors ligne

**Hard Attention Mechanism (HAT)** [6] : masques/poids à valeurs réelles attachés à chaque couche et utilisées pour faire une rétropropagation sélective/pondérée.

# Plan de la séance

- Contexte
- Apprentissage en flux  vs. apprentissage statique
- Dérive conceptuelle et oubli catastrophique
- Solutions :
  - Méthodes par régularisation
  - Architectures DNN évolutives
  - Méthodes dual-memory

# Architectures DNN évolutives

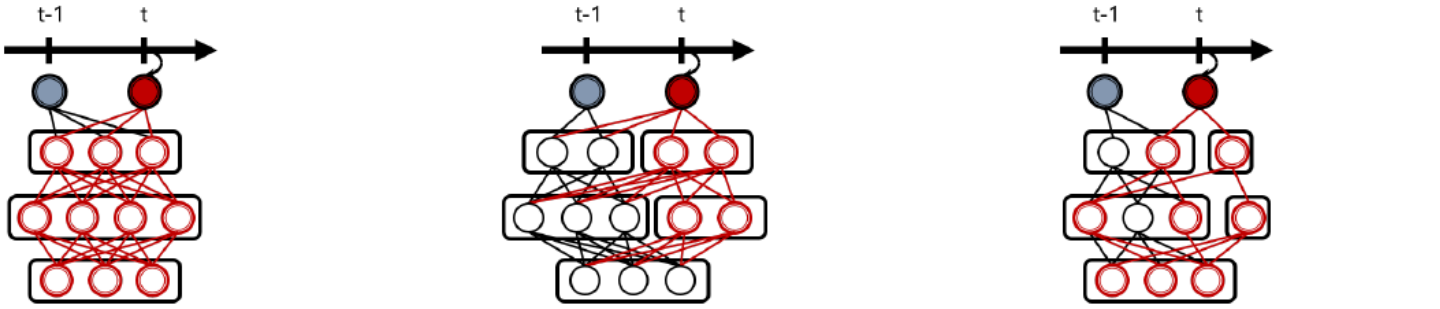



**Progressive NN** [7]

- Chaque nouvelle tache clone le réseau initial avec
  - Initialisation aléatoire
  - Connexion latérales pour favoriser le transfert et accélérer l'apprentissage.

# Architectures DNN évolutives

**Dynamically Extendable Networks (DEN)** [8]

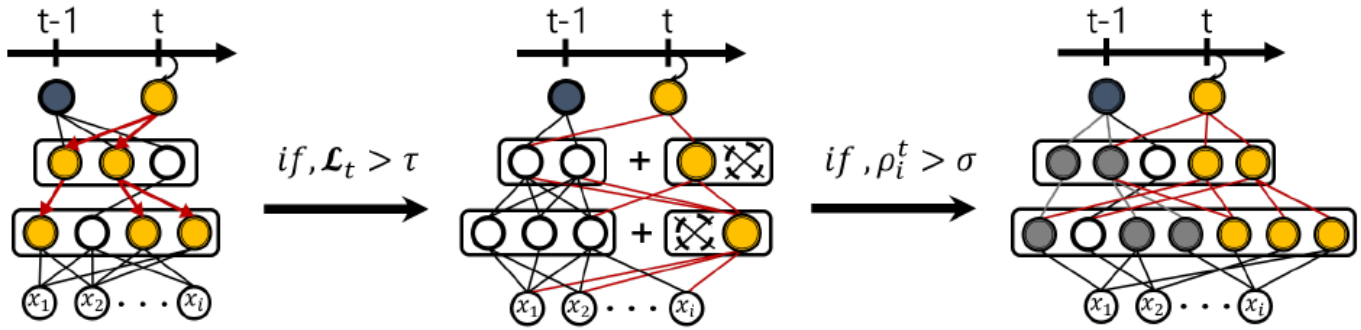



(a) Retraining w/o expansion (b) No-retraining w/ expansion (c) Partial retraining w/ expansion

a. Elastic Weight Consolidation (régularisation)
b. Progressive NN
c. DEN : ré-entraînement partiel et expansion sélective

# Architectures DNN évolutives

**Dynamically Extendable Networks (DEN)** [8]




Gauche : réentraînement sélectif

Milieu : expansion

Droite : Split / Duplication

# Plan de la séance

- Contexte
- Apprentissage en flux  vs. apprentissage statique
- Dérive conceptuelle et oubli catastrophique
- Solutions :
  - Méthodes par régularisation
  - Architectures DNN évolutives
  - Méthodes dual-memory

# Approches Dual-Memory

**Modèles génératifs :**

- Ne pas stocker les données historiques, mais générer des donnés quand le besoin se présente, données qui ont les mêmes caractéristique que les données manquantes.

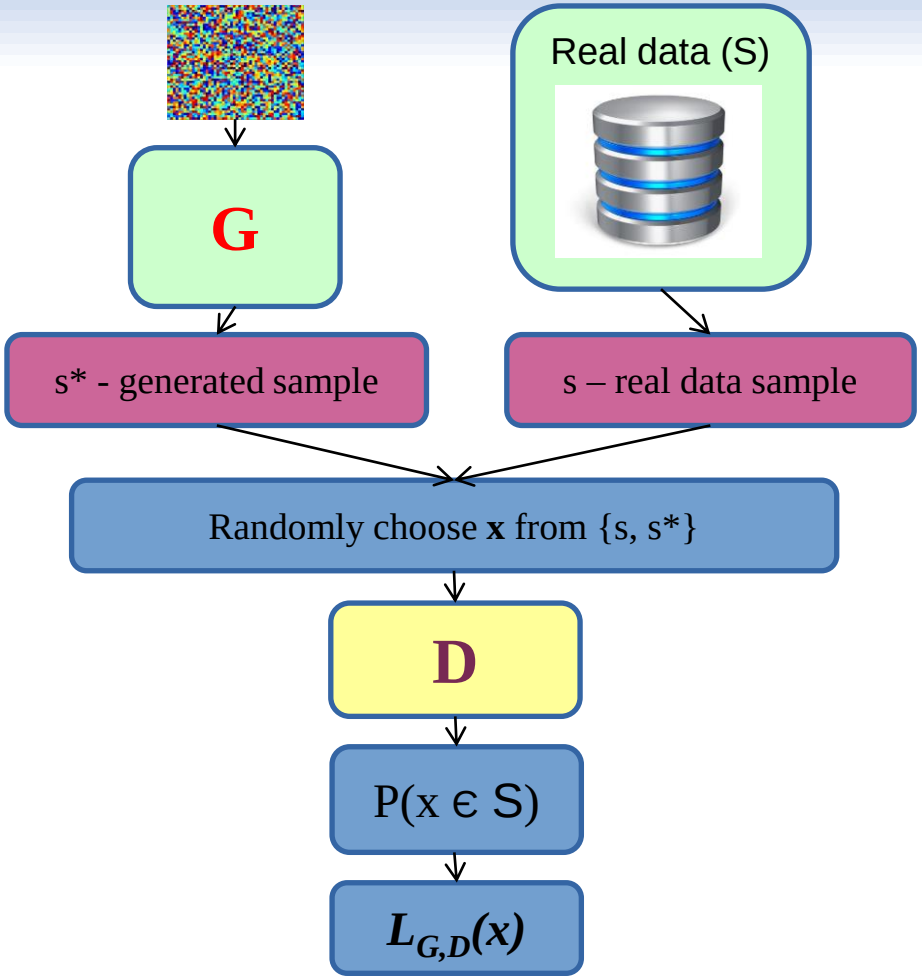
**Stockage :**

- Utilisation de unités de stockage supplémentaires pour préserver **une partie** des données historiques
- Réutilisation de ces données stockées pour renforcer le classes absentes du flux/stream

# Modelés génératifs

- Utilisation des modèles génératifs dans un scénario de classification en ligne : élimine la nécessité de stocker des données historiques
- Le prix à payer est l'entraînement des modèles génératifs
- GAN (Generative adversarial network) (reseaux antagonistes/adverses génératifs)

# Rappel GAN




G : réseau générateur

D : réseau discriminateur

La rétro-propagation «force» G à générer des échantillons plus réalistes et D à mieux distinguer s de s*

# Rappel GAN

Objectif GAN [A] :

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

Generator gradient for update (gradient descent):

predict badly on fake images
=> probability close to 1

$$\nabla_{\mathbf{w}_G} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

# Rappel GAN

Objectif GAN [A] :

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

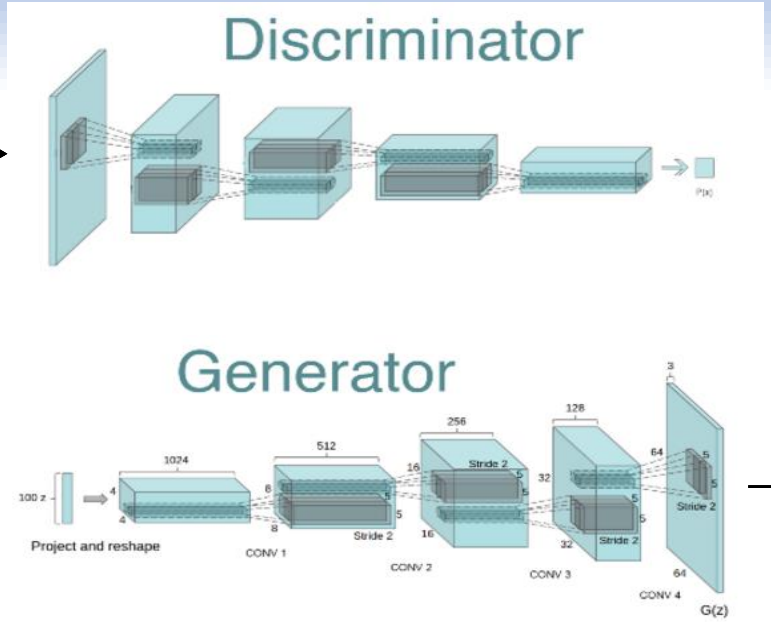
Discriminator gradient for update (gradient ascent):

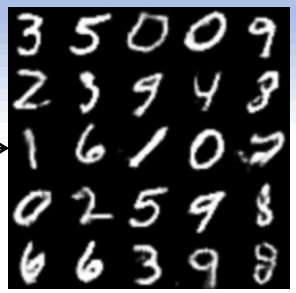
predict well on real images
=> probability close to 1

predict well on fake images
=> probability close to 0

$$\nabla_{\mathbf{w}_D} \frac{1}{n} \sum_{i=1}^{n} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

# Deep Convolutional GAN (DCGAN) [8]

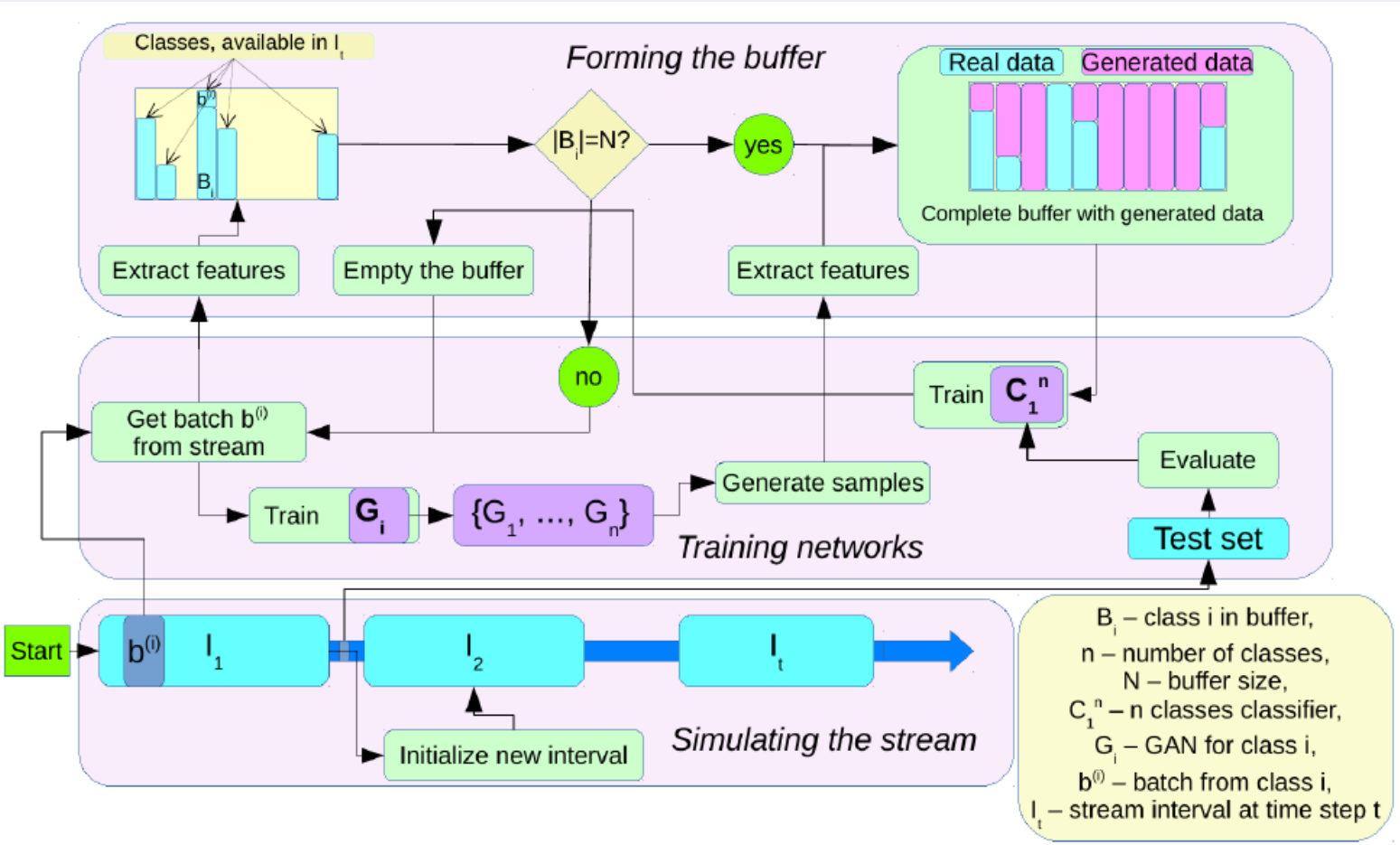


**DCGAN vs. GAN:**

- Pooling replaced by strided convolutions
- Batch normalization
- No fully connected hidden layers
- ReLU activation (for G) and
- LeakyReLU activation (for D)

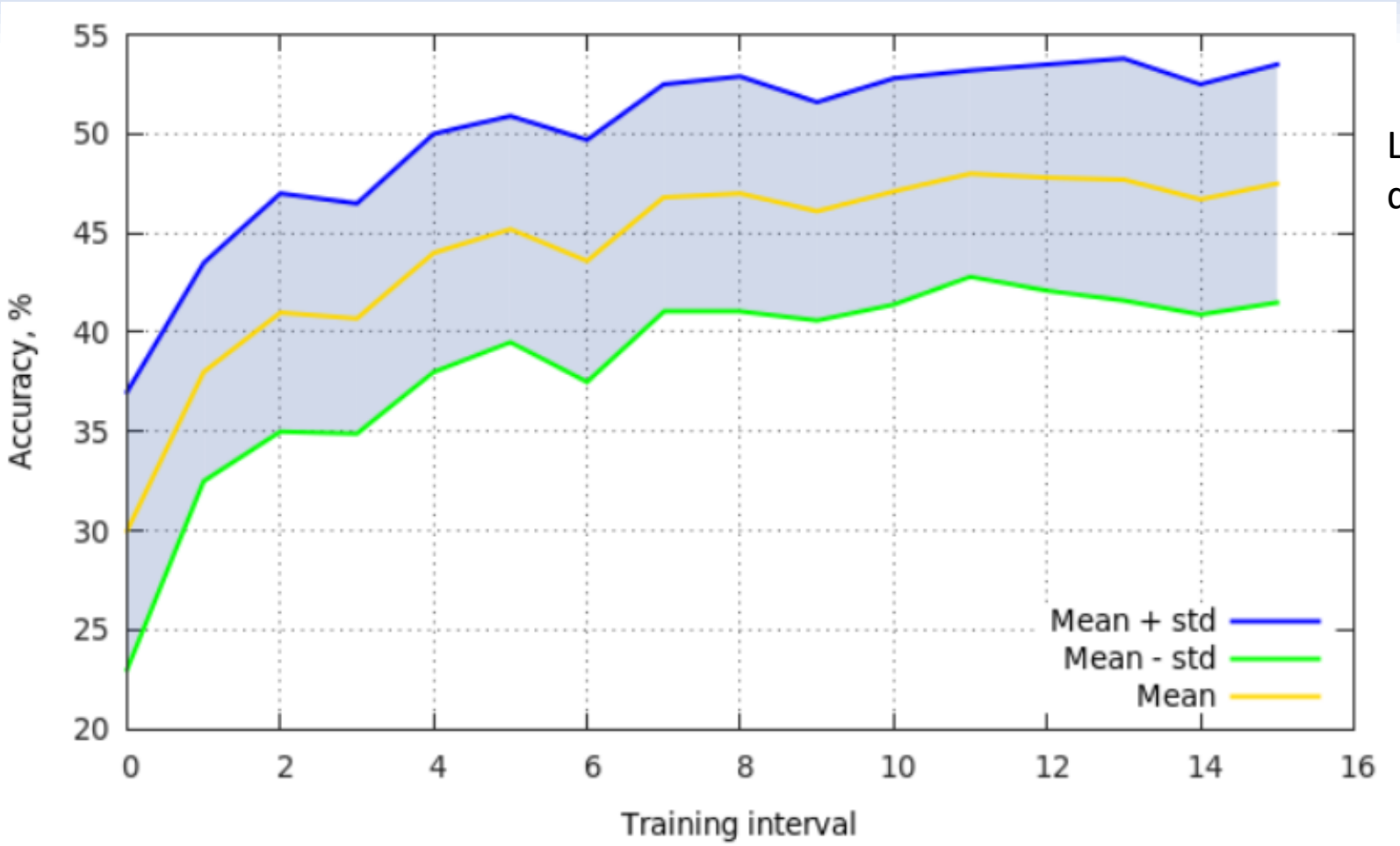
# Modelés génératifs : GAN [1]



# Modelés génératifs : GAN [1]


MNIST dataset

# Modelés génératifs : GAN [1]




LSUN dataset

# Modelés génératifs : GAN [1]

- Les GAN nécessitent beaucoup de données d'entraînement
- Les GAN marchent moins bien pour des données complexes (comme le dataset LSUN)
- GAN : entraînement souvent pas stable (convergence lente)

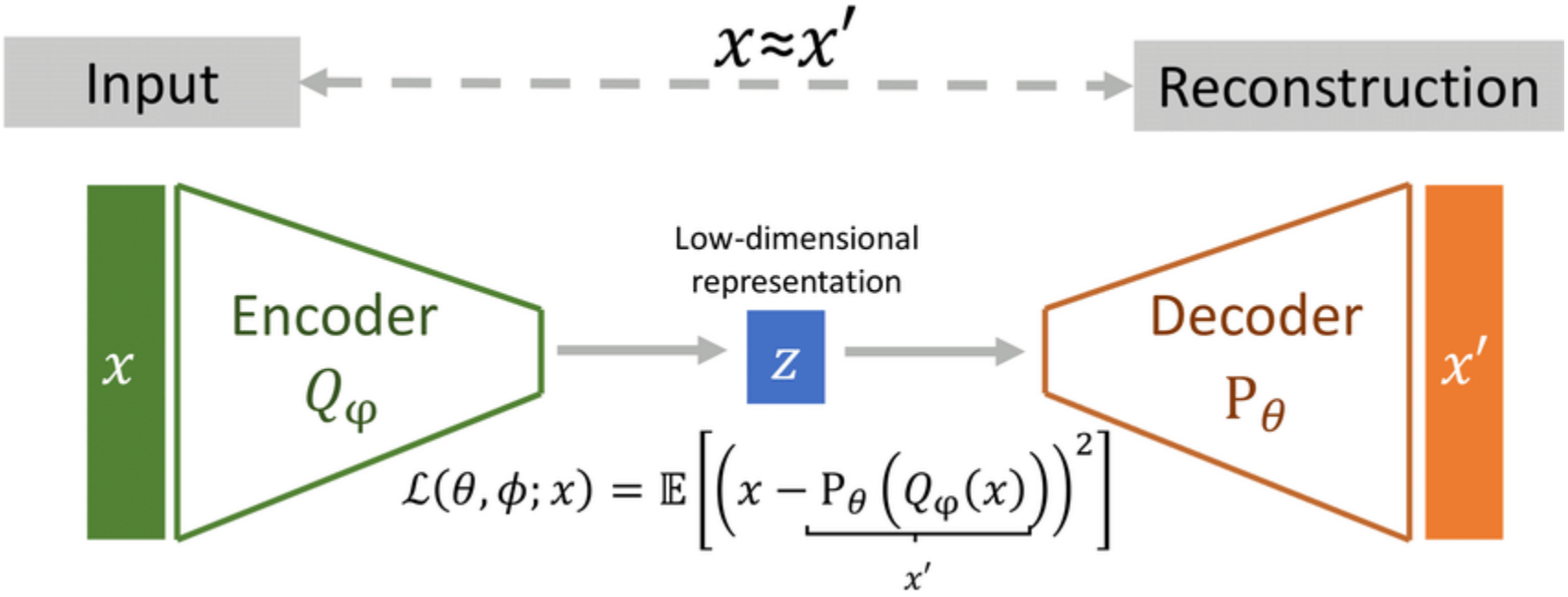
# Approches Dual-Memory

**Stockage** (stocker plutôt que générer les données manquantes) :

- Une partie des données du flux
- Comprimer / réduire la dimension des éléments pour pouvoir stocker plus d'éléments

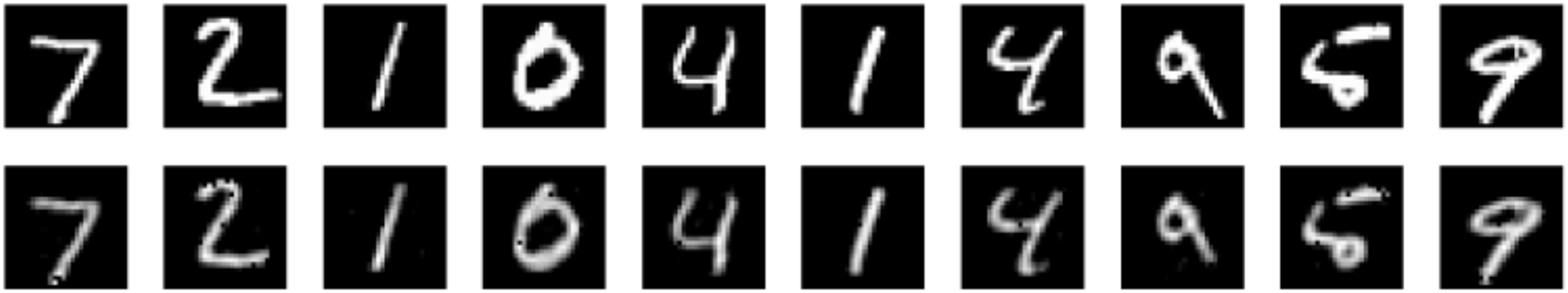
Solution : Auto-encodeurs utilisés comme modèles pseudo-génératifs.

# Rappel auto-encodeurs



# Rappel auto-encodeurs

MNIST : vanilla AE
Input size : 784
Code size : 64



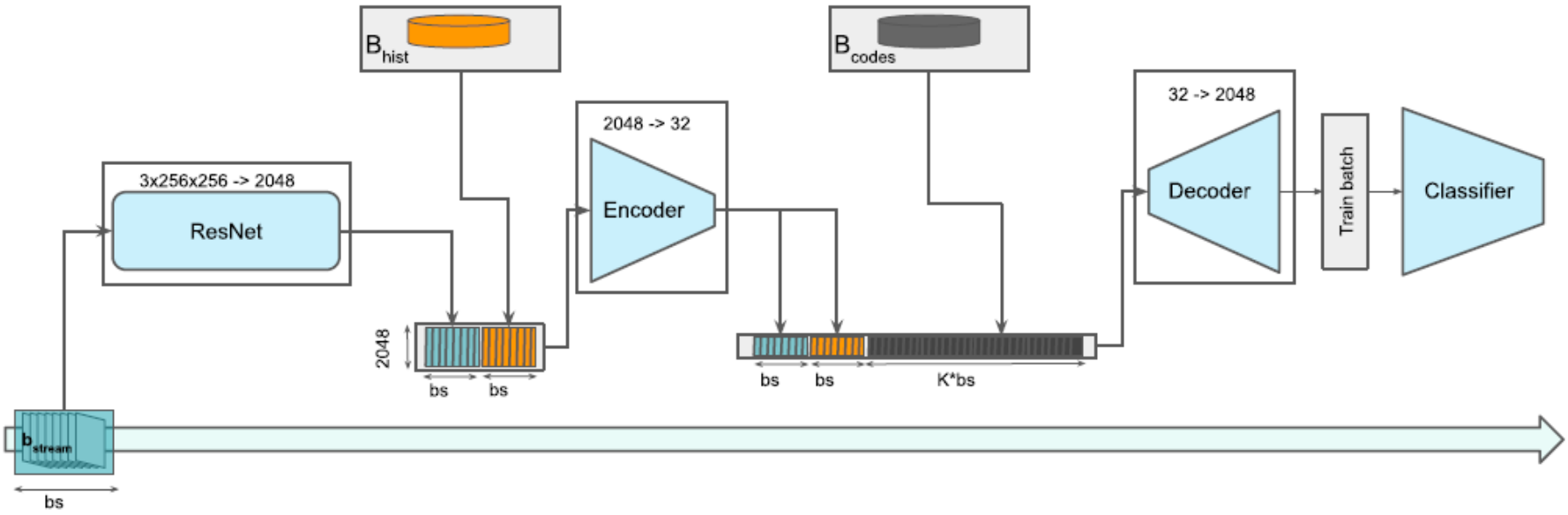
# Auto-encodeur: perte (loss)

$$\mathcal{L}_{cl} = \left\| C(x) - C(G(x)) \right\|_2$$

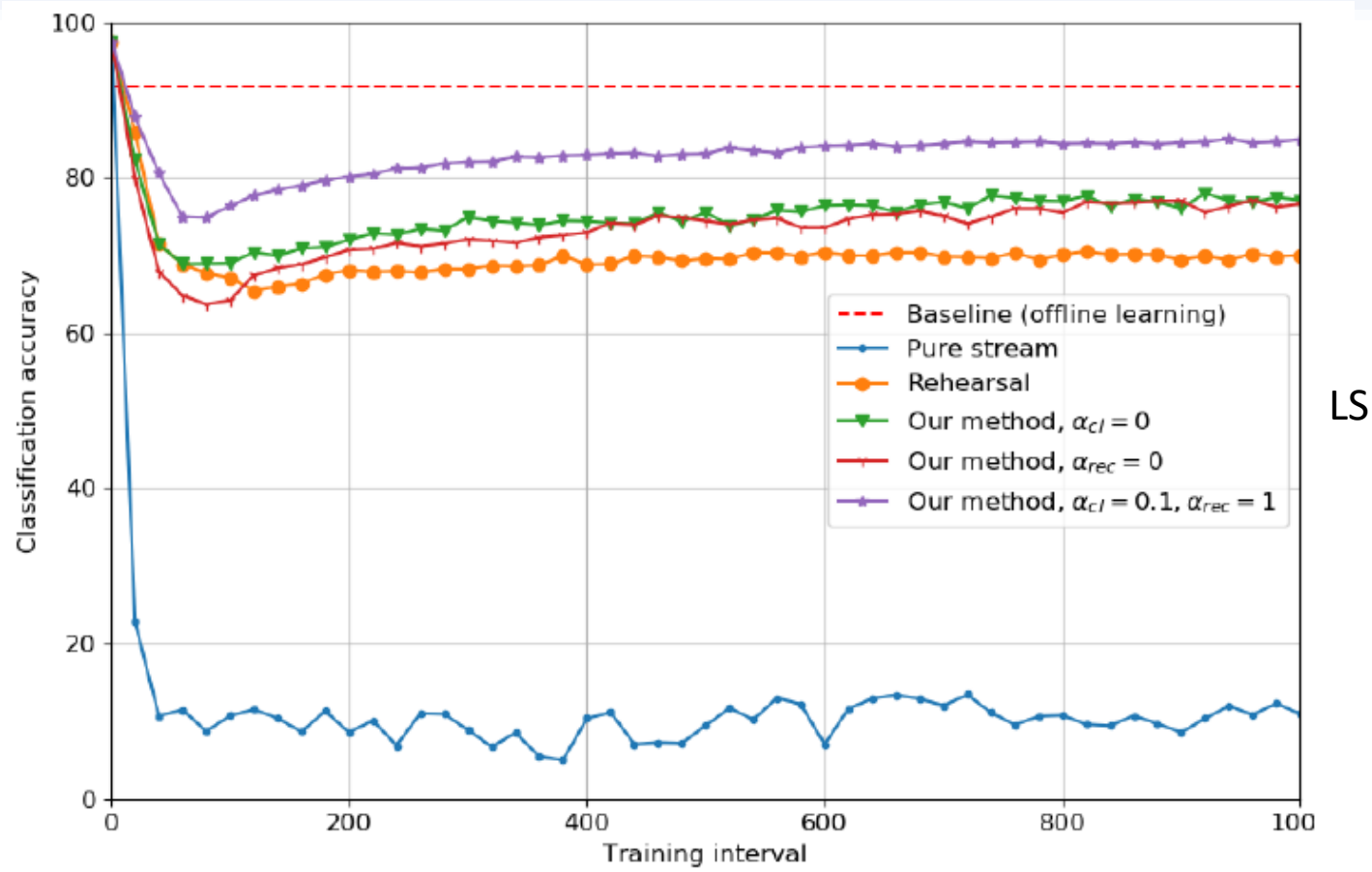
$$\mathcal{L} = \alpha_{cl}\mathcal{L}_{cl} + \alpha_{rec}\mathcal{L}_{rec}$$

| Code | 2 | 4 | 8 | 16 | 32 | Base |
|---|---|---|---|---|---|---|
| MNIST | 91.3 | 95.3 | 97.1 | **97.7** | 98.2 | 99.4 |
| LSUN | 57.9 | 72.9 | 79.5 | 83.3 | **89.1** | 90.7 |
| Syn-1000 | 79.8 | 91.1 | 93.3 | **94.5** | 94.6 | 96.1 |

# Apprentissage en flux



# Apprentissage en flux

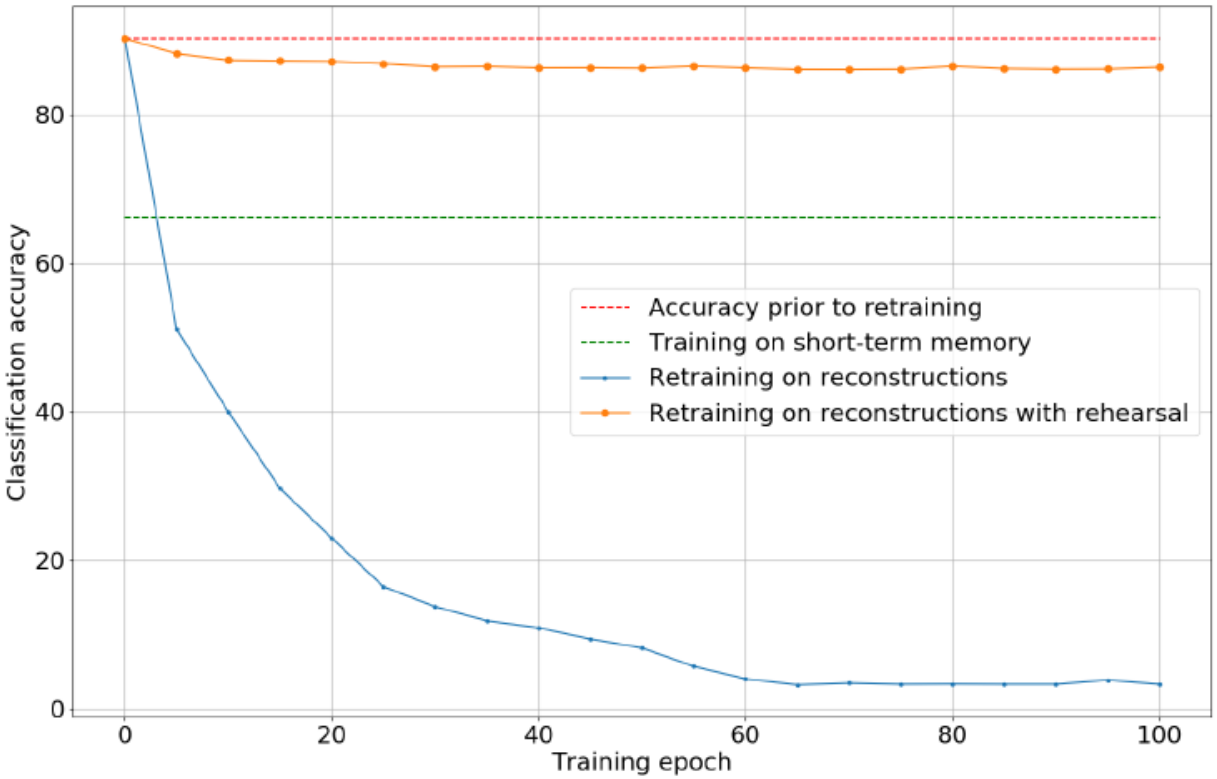



LSUN dataset

# Auto-encodeur: perte (loss)

Phénomène d'oubli dans les auto-encodeurs

$$L = \left(\sum_{i=1}^{k} \frac{1}{r_i}\right)^{-1} \cdot \left(\sum_{i=1}^{k} \frac{l_i}{r_i}\right)$$



# Bibliographie

1. Besedin A., Continual Forgetting-Free Deep Learning from High-dimensional Data Streams, PhD Thesis, 2019

2. Bifet. Et al, *Machine Learning for Data Streams: with Practical Examples in MOA*, The MIT Press 2018

3. Parisi et al., *Continual lifelong learning with neural networks: A review.* arXiv:1802.07569, 2018.

4. Srivastava et al., Compete to compute. Advances in neural information processing systems, pages 2310–2318, 2013.

5. Li et al. *Learning without forgetting.* IEEE Transactions on PAMI, 2017

6. Serra et al., *Overcoming catastrophic forgetting with hard attention to the task.* arXiv:1801.01423, 2018.

7. Rusu et al. *Progressive neural networks.* arXiv:1606.04671, 2016

8. Yoon et al., *Lifelong learning with dynamically expandable networks.* ICLR, 2018.

9. Radford et al., *Unsupervised representation learning with deep convolutional generative adversarial networks.* arXiv:1511.06434.

Web :

A. http://pages.stat.wisc.edu/~sraschka/teaching/stat479-ss2019/