Premiers pas pour l'utilisation de l'IA scientifique

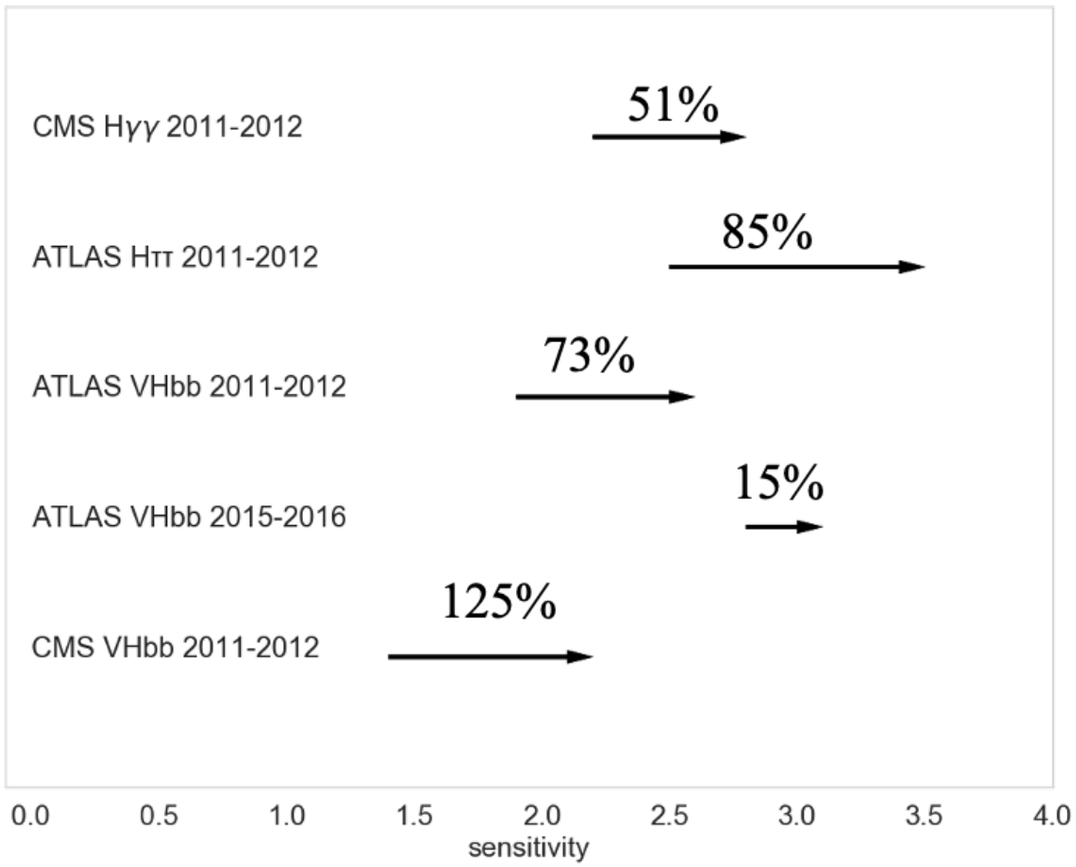
Corentin Allaire
David Rousseau
Françoise Bouvet

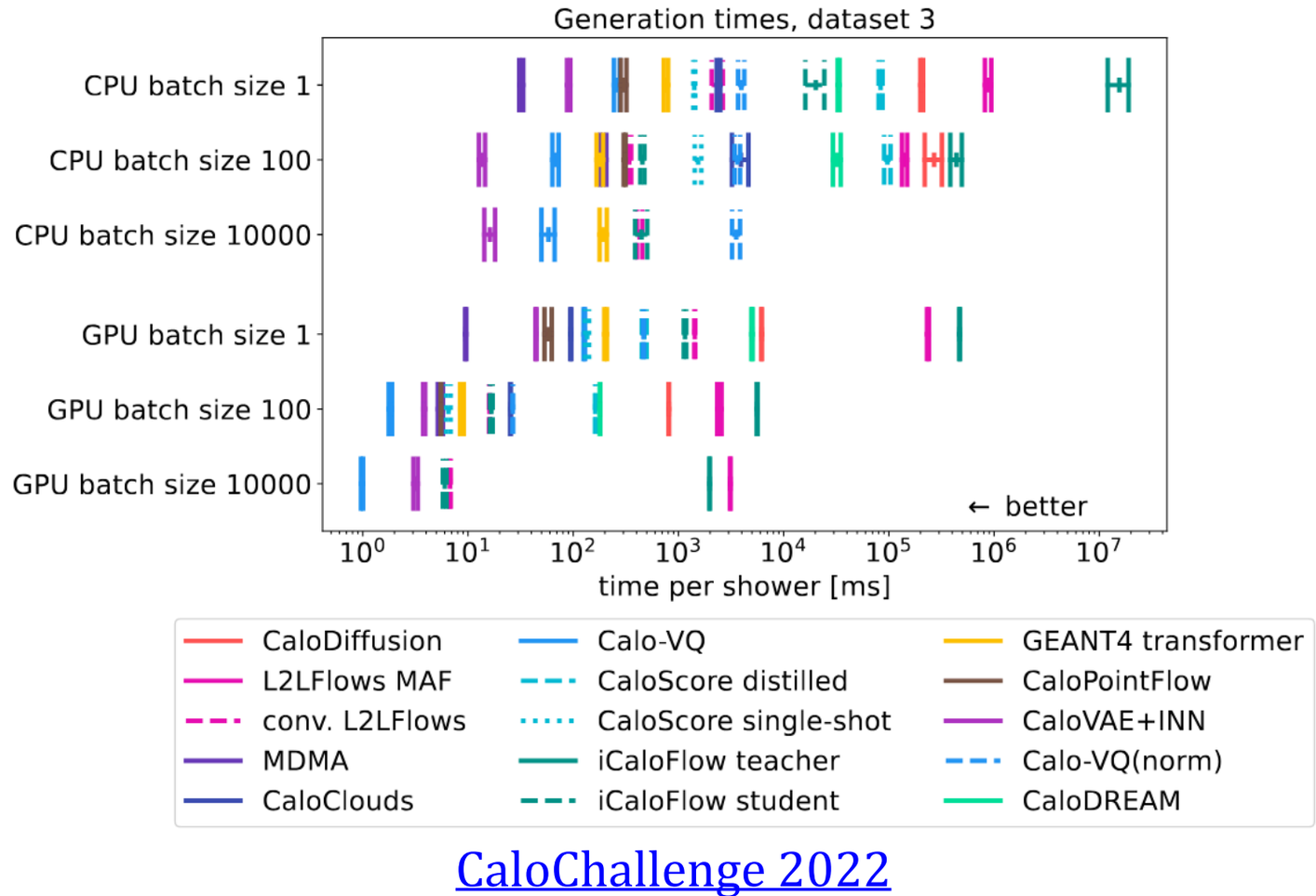
Pourquoi utiliser du Deep Learning ?

- Outil informatique/statistique nous permettant de maximiser l'usage de nos données scientifiques

➡ Au LHC, IA **~ 50% de data en plus** pour nos analyses

Nature 560, 41–48 (2018)





CaloChallenge 2022

- Optimisation de notre consommation en ressource de calcul

➡ CaloChallenge 2022 : simulation de calorimètre. **~10 000 fois moins coûteux** que Geant4

Intensifier son problème

- Pas de magie !
- Les réseaux de neurones: résolvent le **problème** sur lequel ils sont optimisés et uniquement dans le **contexte** où ils sont entraînés
- Objectifs ?
  - **Classifier** des objets ?
  - **Reconstruire** des quantités physiques ?
  - **Générer** de nouveaux objets ? (simulation)
- Pourquoi veut-on utiliser de l'IA ?
  - Algorithme actuel trop peu performant ?
  - Algorithme actuel trop lent ? Utilisation de ressources hétérogènes ?
  - Question insoluble numériquement ?

Intensifier son problème

- Comment déterminer si notre nouvel algorithme est bon ?
- Bien séparer les notion de **fonction de coût / loss** de la notion de **métrique** de performance

Loss :

- Sert à l'optimisation du réseau
- Tend vers 0 quand le réseau fonctionne mieux
- Doit être différentiable

	Fonction d'activation (dernière couche)	Fonction de coût
Régression (quantitatif)	Linéaire	Mean squared error $c(y, \hat{y}) = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$
Classification à 2 classes (régression logistique)	Sigmoïde	Cross-entropy ce(y, $\hat{y}$) = ((y == 1)? -log($\hat{y}$) : -log(1 - $\hat{y}$))
Classification multi-classes	Soft-max (probabilité)	Cross-entropy (categorical_crossentropy)

Métrique :

- Sert à l'évaluation des performances du réseau
- Peut être aussi complexe que nécessaire :
  - Efficacité de reconstruction
  - Vitesse de calcul
  - Faux-positif, faux-négatif, AUC…

Comprendre ses données

- Pour apprendre les réseaux ont besoin de **données représentatives étiquetées**
- Quelle quantité est disponible ? Quelle est la qualité de l'étiquetage ?
- Est-il possible de **simuler des données réalistes** ?
  Si oui, l'entraînement est beaucoup plus facile.
- Le type de donnée influe grandement sur le type d'algorithme:
  - Image → Réseau de convolution
  - Données tabulaire → Perceptron multi-couche (MLP)
  - Séquence → LSTM (réseau récurrent)
  - Données non ordonnées / Graph → GNN, Transformer

Quel Hardware pour l'IA ?

- Deux types de ***hardware*** à considérer : entrainement / inférence

Entrainement :

- Ordinateur portable pour petit algorithme (BDT, petit réseau de neurones)
- *Data center* local sur GPU:
  - Ruche (Paris-Saclay)
  - Jean Zay (IDRIS)
  - Centre de calcul des expériences (Grille…)

Inférence :

- Dépend énormément de l'objectif : analyse, acquisition…
- CPU même si entrainé sur GPU
- Peuvent être déployés « *online* » dans les expériences sur FPGA
- Taille de réseau à adapter au support d'inférence

Bibliothèque d'IA : Ne pas réinventer la roue

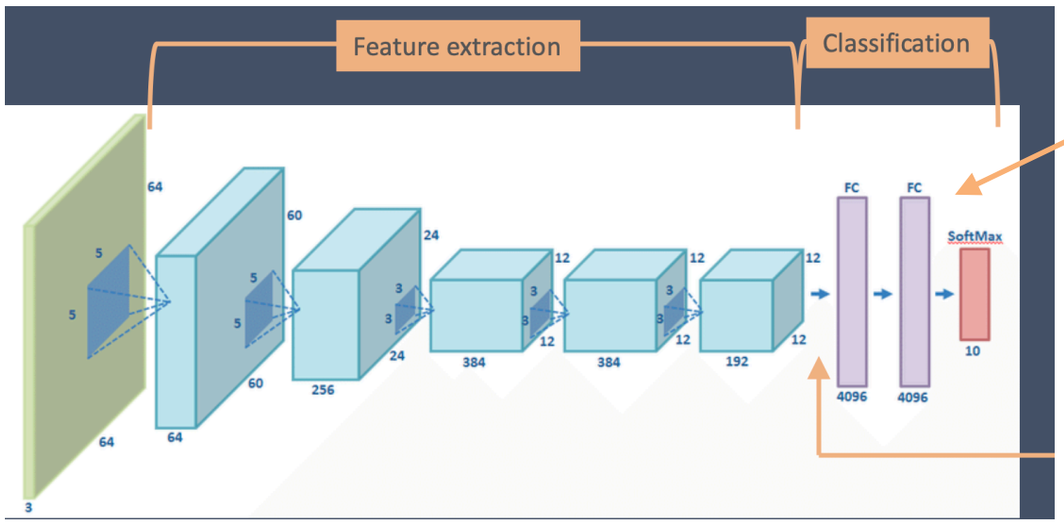
- Pour le machine learning en Python : [scikit-learn]
- Tout les outils nécessaires pour le ML (pas/peu de DL)
- Bibliothèque spécifique dans les autres langages (penser à regarder si elles sont **maintenues**)

- Pour le Deep Learning : [Pytorch] et [Tensorflow]
- Utilisation simplifiée de l'un ou l'autre : [Keras]
- Les « **briques** » de la plupart des réseaux sont disponibles

Example : réseau de neurones avec Keras

```
from keras.models import Sequential
from keras.layers import Input, Dense, Normalization

layer_norm = Normalization()
layer_norm.adapt(x_train)

model = Sequential()
model.add(Input(np.shape(x_train)[1:], ))
model.add(layer_norm)
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy', 'AUC'])

model.summary()
```

Transfert learning et modèle de fondation



Ré-entrainement du classificateur

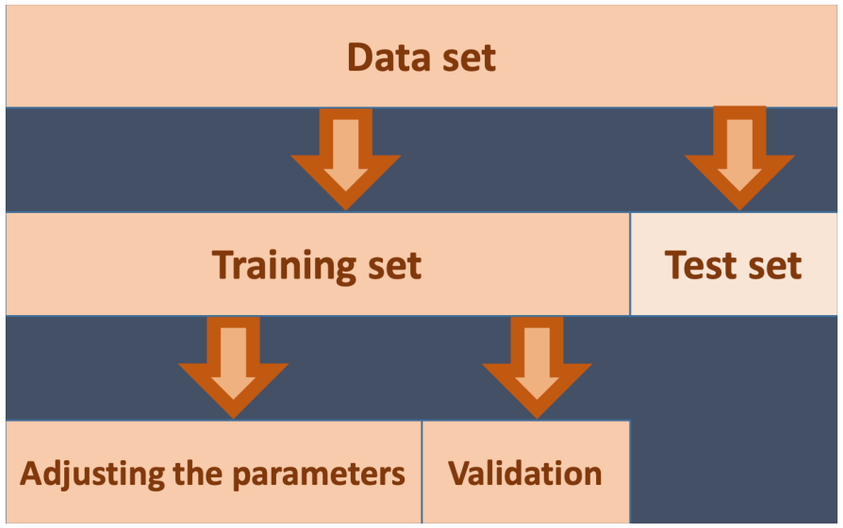
Réutilise l'extraction de feature

- Certains réseaux doivent apprendre 2 choses :
  - **Extraire** des *features* de l'entrée
  - **Calculer** une quantité avec ces *features*
    - Typiquement : Images
    - Réutilisation de réseaux déjà entraînés sur des images

- **Modèle de fondation** : même idée pour domaines spécifiques
- Entraînement sur des données du détecteur ATLAS utilisé pour différentes analyses/reconstructions d'objets

Bonne pratique d'entrainement

- Bien séparer ses données **d'entraînement**, de **validation** et de **test**.
- Attention aux corrélations !

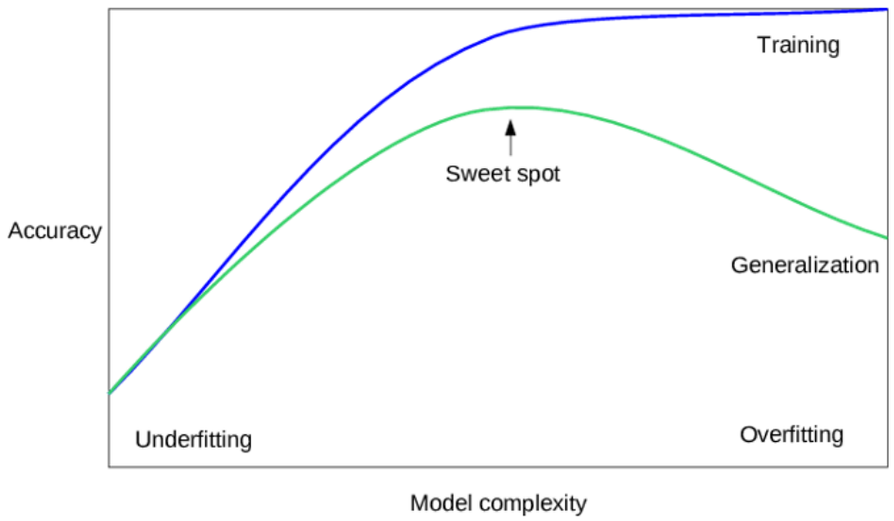


- **Normalisation**
  1. $X = \{x^m\}, m \in \{1, M\}, x^m \in R^N$
  2. $\mu_i = \frac{1}{M}\sum_{m=1}^{M} x_i^m$
  3. $\sigma_i^2 = \frac{1}{M}\sum_{m=1}^{M}(x_i^m - \mu_i)^2$
  4. $x_i^m \leftarrow \frac{x_i^m - \mu_i}{\sigma_i}$

blue -> $\begin{pmatrix}1\\0\\0\end{pmatrix}$; green -> $\begin{pmatrix}0\\1\\0\end{pmatrix}$; red -> $\begin{pmatrix}0\\0\\1\end{pmatrix}$

- Bien préparer ses données:
  - Normalisation
  - Encodage des classes

- Suivre l'évolution de sa **courbe d'entraînement**
- Outil utile : **Tensorboard**



Déploiement de son réseau

- Une fois le réseau entraîné, les poids peuvent être **sauvegardés** sous forme de fichier (souvent de très petite taille)
- Ces fichiers vont vous permettre de réutiliser le modèle sur vos futurs données
- Deux déploiements possible :
  - Dans un **script Python**, en utilisant la bibliothèque d'entraînement: le réseau est chargé et utilisé
  - Dans un autre langage : utilisation d'un **moteur d'inférence**
- onnxruntime : Inférence d'un modèle déjà entraîné depuis la plupart des langages de programmation
- Nécessite juste d'écrire une l'interface et de fournir les fichiers de poids