

# Artificial Intelligence day @IRCER Limoges

## Deep Learning feedbacks: Reproducibility, Hyperparameters

Jean-Luc CHARLES

Retired from ENSAM Bordeaux,  
I2M, UMR 5295 CNRS, TALENCE.

Jean-Luc.Charles@mailo.com

14 mars 2025



## Presentation Summary

- Historical way, definitions & concepts of AI & Deep Learning (DL)
- Reproducibility in Deep Learning
- Hyperparameters
- Sharing reproducible DL training



## Presentation Summary

- Historical way, definitions & concepts of AI & Deep Learning (DL)
- Reproducibility in Deep Learning
- Hyperparameters
- Sharing reproducible DL training



## My professional career

- First industrial career ~10 years (Signal Processing at Thomson-SINTRA then OO C++ development at Capgemini)
- I joined the I2M lab at ENSAM (Bordeaux) as a lecturer, working on "The Contribution of OO to Simulation in Mechanics"
- I taught Scientific & OO programing in Python at ENSAM
- I started to work on AI & Deep Learning in 2015
- Now I go on working on AI as a self-worker...



## Presentation Summary

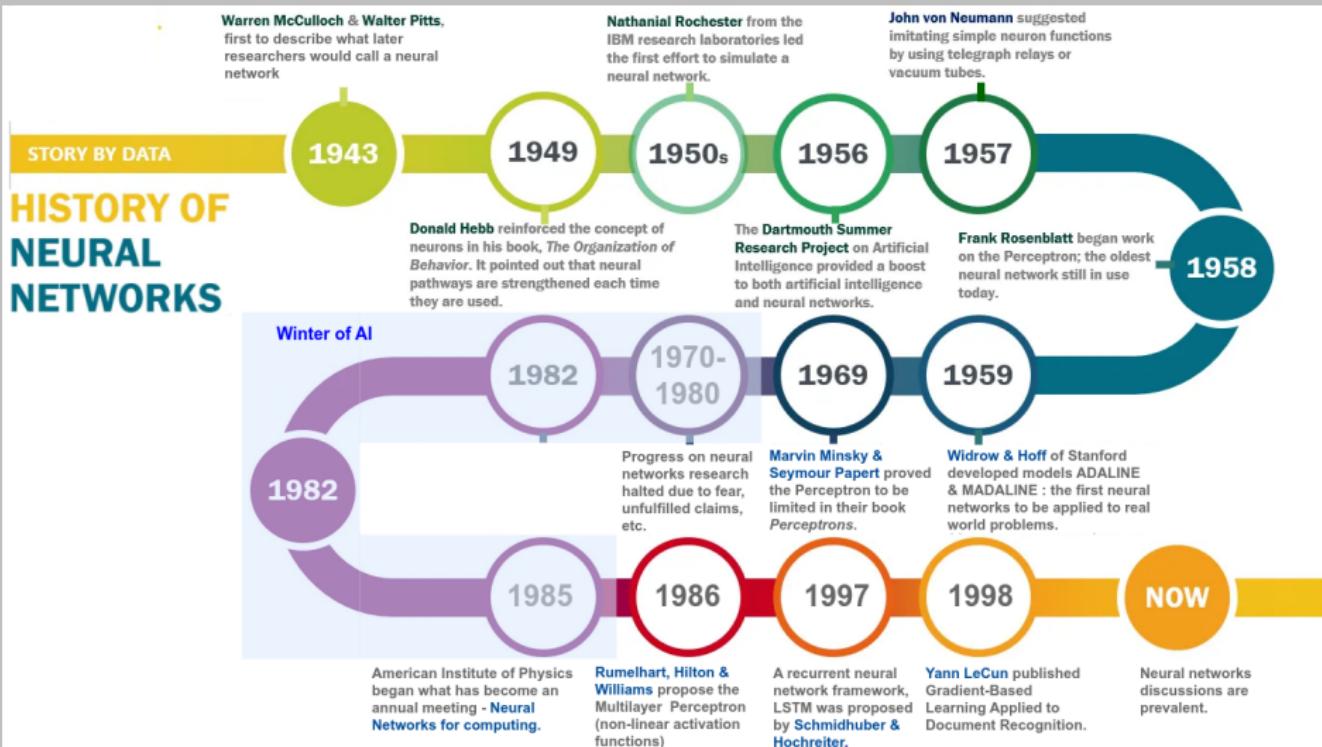
- Historical way, definitions & concepts of AI & Deep Learning (DL)
- Reproducibility in Deep Learning
- Hyperparameters
- Sharing reproducible DL training



## Ressources

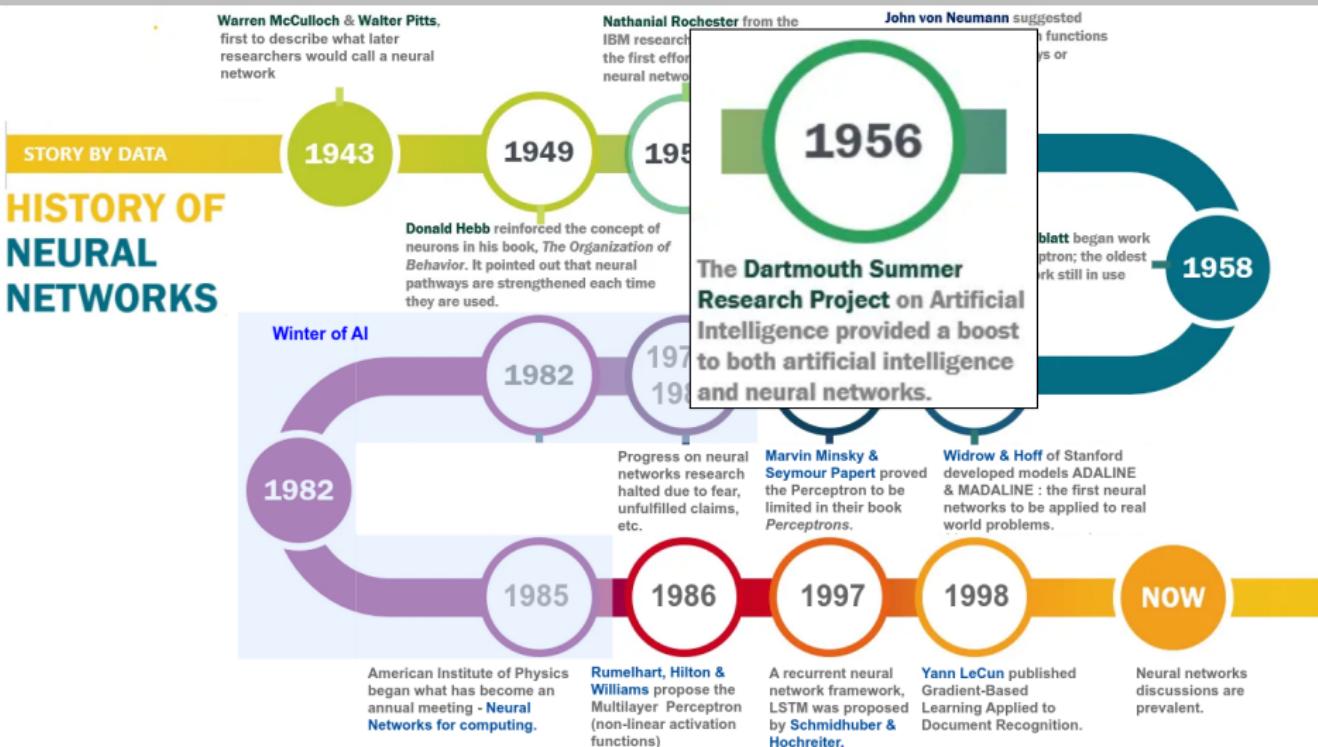
- GitHub repository for the PDF slides:
- GitHub repository for the example notebooks:  
[/Reproducibility-MNIST-DNN-CPU](#) and  
[/Reproducibility-MNIST-CNN-GPU](#)

# AI : the historical way... from 1950 to 2000s



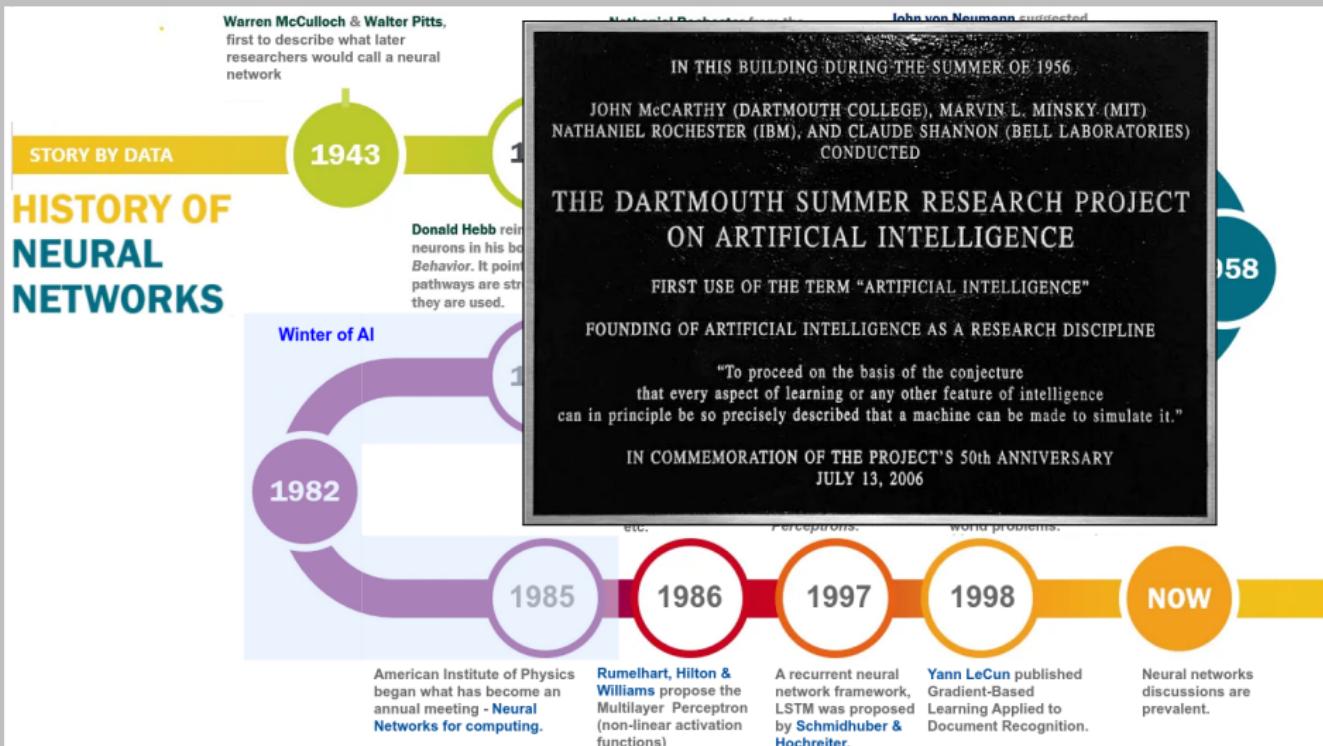
Adapted from : Kate Strachni: "Brief History of Neural Netwrks", medium.com

# AI : the historical way... from 1950 to 2000s



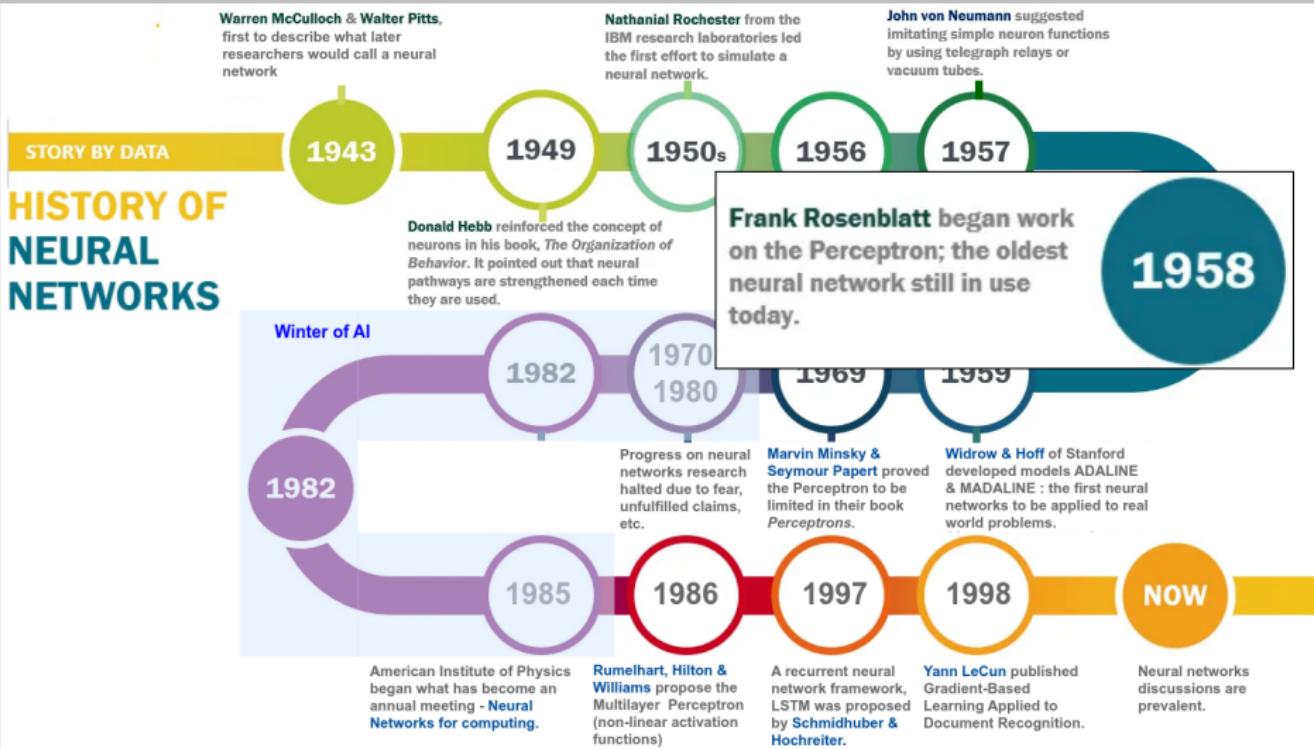
Adapted from : [Kate Strachni: "Brief History of Neural Netwrks"](#), [medium.com](#)

# AI : the historical way... from 1950 to 2000s



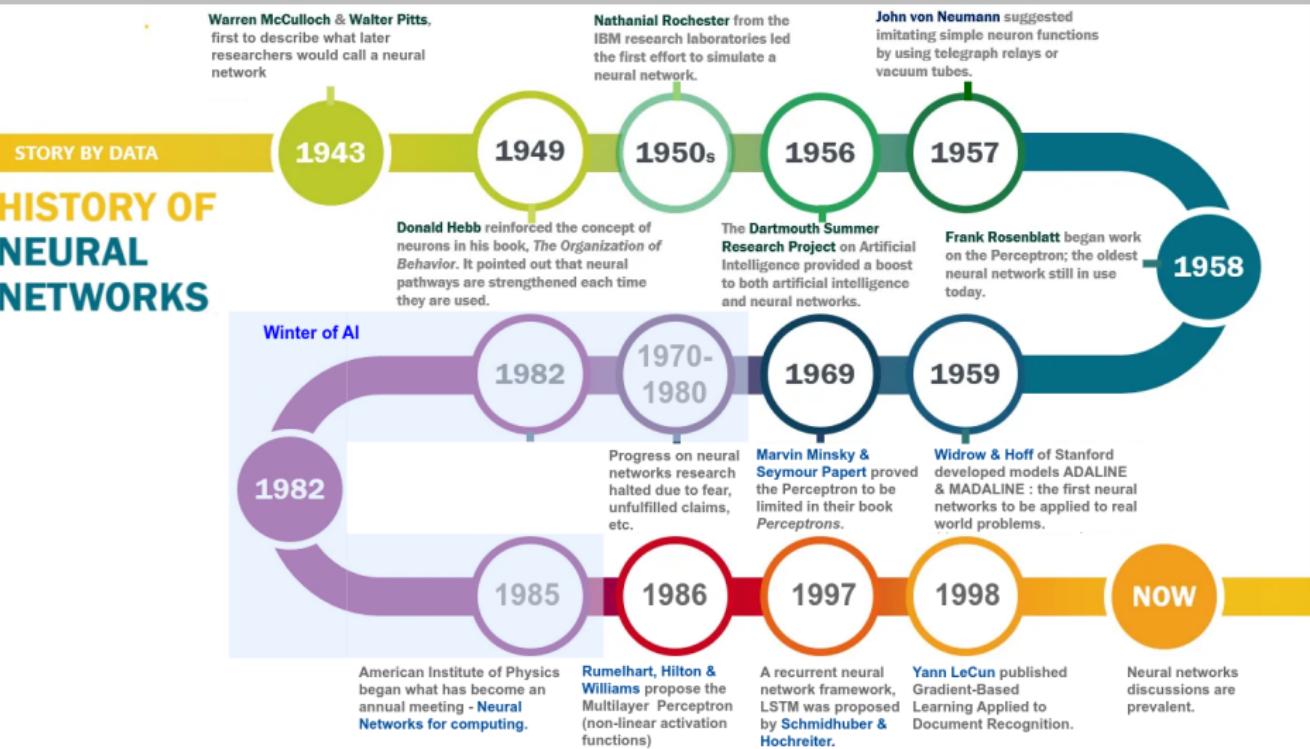
Adapted from : Kate Strachni: "Brief History of Neural Netwrks", medium.com

# AI : the historical way... from 1950 to 2000s



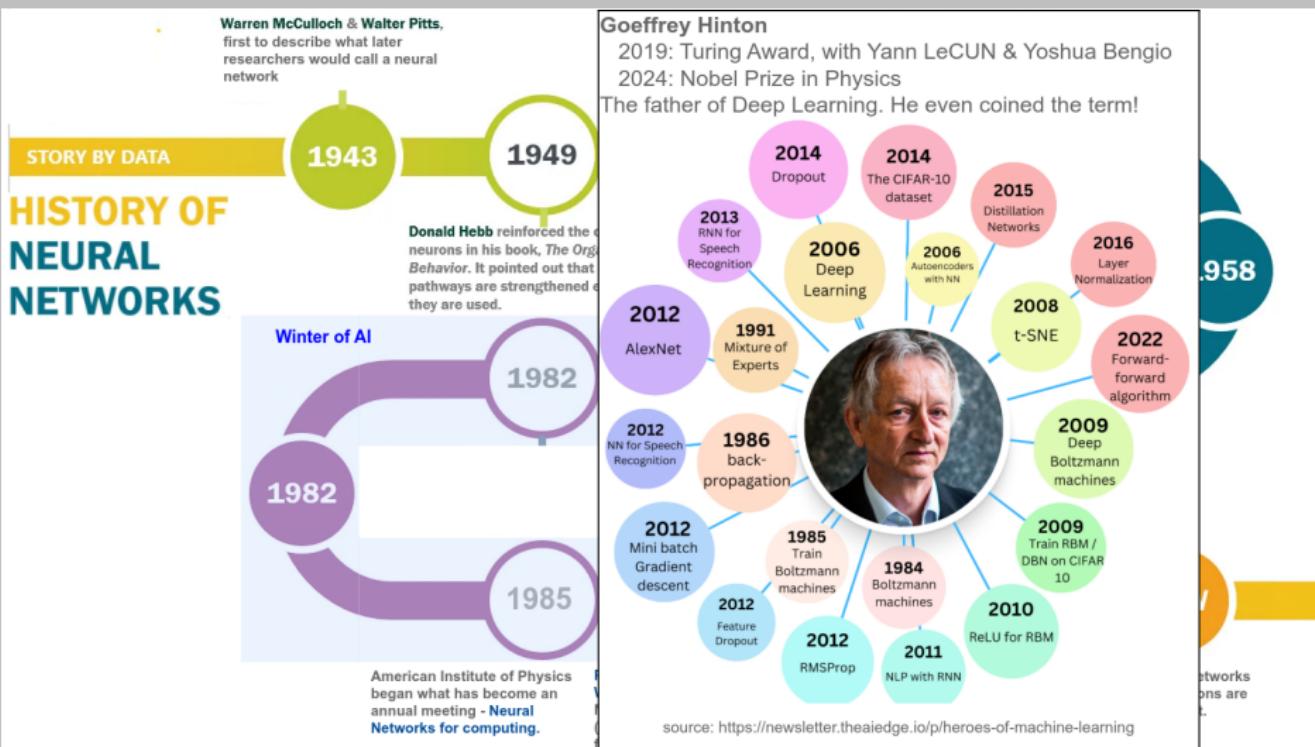
Adapted from : Kate Strachni: "Brief History of Neural Networks", medium.com

# AI : the historical way... from 1950 to 2000s



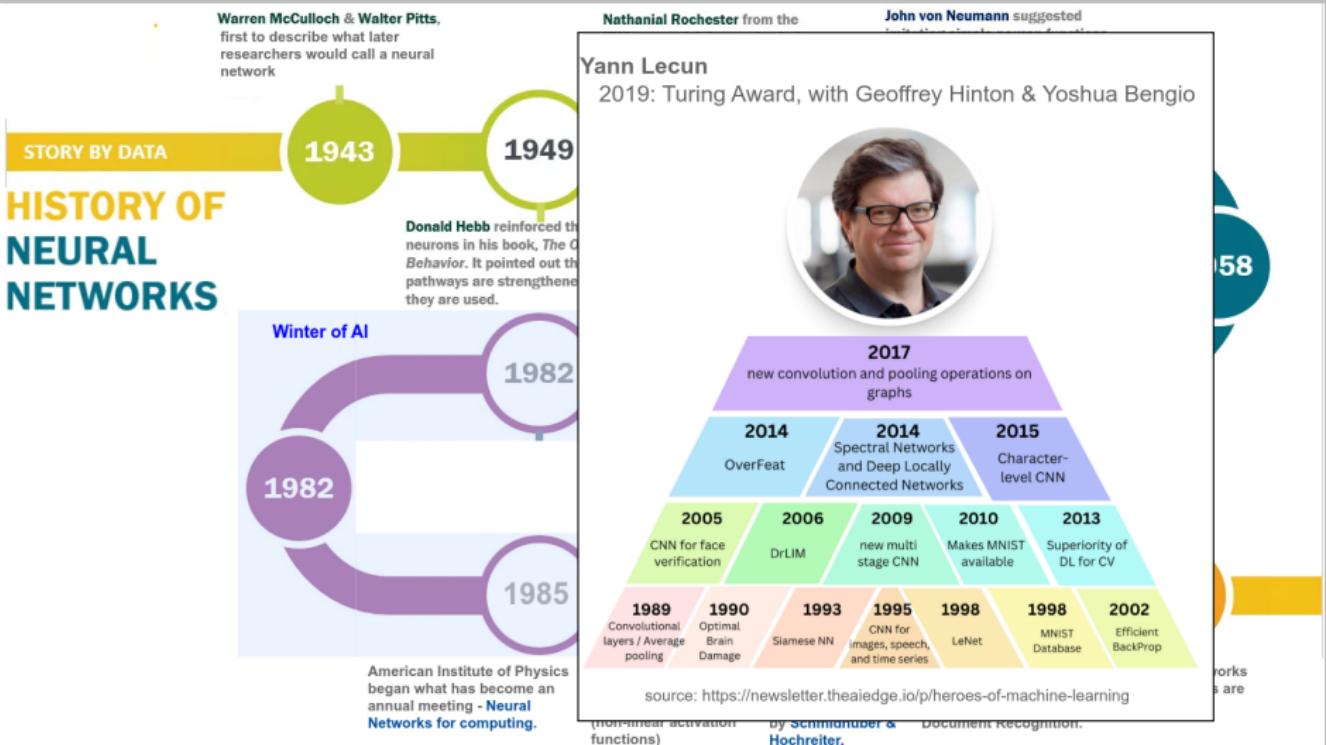
Adapted from : Kate Strachni: "Brief History of Neural Netwrks", medium.com

# AI : the historical way... from 1950 to 2000s



Adapted from : Kate Strachni: "Brief History of Neural Netwrks", medium.com

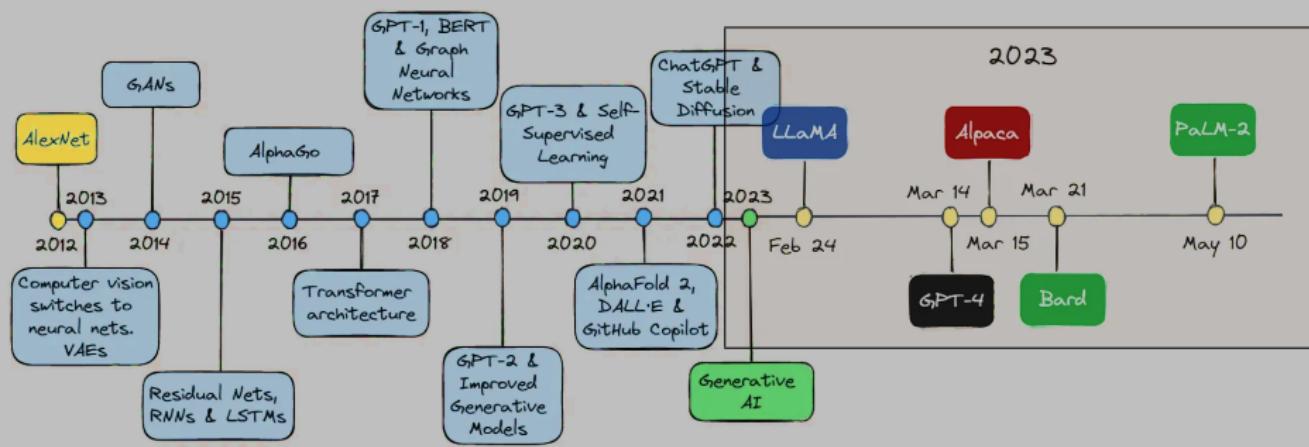
# AI : the historical way... from 1950 to 2000s



Adapted from : Kate Strachni: "Brief History of Neural Networks", medium.com



# The historical way... post 2023s, crazy acceleration



# The historical way... post 2023s, crazy acceleration



# Artificial Intelligences ?

**Narrow/Weak AI**

**Strong/General AI**

# Artificial Intelligences ?

## Narrow/Weak AI

- AI systems designed for specific, narrow tasks:  
*AlphaGo, self-driving cars, robot systems in the medical field...*
- The most common form of AI that we encounter today.
- Builds systems that **behave like humans**.
- **We already are there...** We use it every day!  
*anti-spam, facial/voice recognition, language translation...*

## Strong/General AI

# Artificial Intelligences ?

## Narrow/Weak AI

- AI systems designed for specific, narrow tasks:  
*AlphaGo, self-driving cars, robot systems in the medical field...*
- The most common form of AI that we encounter today.
- Builds systems that **behave like humans**.
- **We already are there...** We use it every day!  
*anti-spam, facial/voice recognition, language translation...*

## Strong/General AI

- Builds systems with the ability to reason in general.
- Could explain how humans think?
- **Whe are not yet here...** but closer and closer every day

# Machine Learning: a field of AI

## ARTIFICIAL INTELLIGENCE

### MACHINE LEARNING

SUPERVISED  
LEARNING

UNSUPERVISED  
LEARNING

DEEP  
REINFORCEMENT  
LEARNING

SELF-  
SUPERVISED  
LEARNING

# Branches of Machine Learning

## Supervised learning

A **labeled dataset** is used to train algorithms (models):

- **Classification**

- Images classification
- Objects detection in images
- Speech recognition...

- **Regression**

- Predict a value...

- **Anomaly detection**

- Spam detection
- Manufacturing: finding known (learned) defects
- Weather prediction
- Diseases classification...

...

# Branches of Machine Learning

## Unsupervised learning

The algorithm attempts to identify patterns / structures within **unlabeled datasets**:

- **Clustering & Grouping**

- Data mining, web data grouping, news grouping...
- Market segmentation
- Astronomical data analysis...

- **Anomaly Detection**

- Manufacturing: finding defects (even new ones)
- Monitoring abnormal activity: failure, hacker...
- Fraud detection: fake account on Internet...

- **Dimensionality reduction**

- Data compressions...

...

# Branches of Machine Learning

## Deep Reinforcement Learning DRL

An agent (the model) learns how to drive an environment within a trial and error process by maximising a cumulative **reward**:

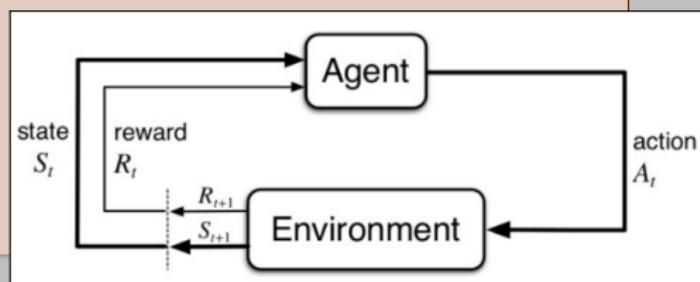
- **Control/command**

- Controlling drones, mechatronic systems, electric VTVL rocket, robots
- Factory optimization
- Financial (stock) trading...

- **Decision making**

- games (video games)
- financial analysis...

...



# Branches of Machine Learning

## Self-Supervised Learning [LM23]

- Aims to learn features/patterns from **unlabeled data without relying on human annotations.**
- Can extract data features that exhibit robust generalization.
- Technically a subset of Unsupervised Learning.
- But closely related to Supervised Learning
  - ~ optimizes performance against a self-learned ground truth.

# Branches of Machine Learning

## Self-Supervised Learning [LM23]

- Aims to learn features/patterns from **unlabeled data without relying on human annotations.**
- Can extract data features that exhibit robust generalization.
- Technically a subset of Unsupervised Learning.
- But closely related to Supervised Learning  
~ optimizes performance against a self-learned ground truth.

# Branches of Machine Learning

## Self-Supervised Learning [LM23]

- Aims to learn features/patterns from **unlabeled data without relying on human annotations.**
- Can extract data features that exhibit robust generalization.
- Technically a subset of Unsupervised Learning.
- But closely related to Supervised Learning  
~ optimizes performance against a self-learned ground truth.

# Branches of Machine Learning

## Self-Supervised Learning [LM23]

- Aims to learn features/patterns from **unlabeled data without relying on human annotations.**
- Can extract data features that exhibit robust generalization.
- Technically a subset of Unsupervised Learning.
- But closely related to Supervised Learning  
~~ optimizes performance against a self-learned ground truth.

# Branches of Machine Learning

## Self-Supervised Learning

2 main stages of SSL:

- **Pre-training** (Unsupervised):

- **Fine-Tuning** (Supervised Task):

# Branches of Machine Learning

## Self-Supervised Learning

2 main stages of SSL:

- **Pre-training** (Unsupervised):

The model learns useful representations from unlabeled data by solving *pretext tasks*:

- Image-Based SSL: Predicting missing parts of an image, similarity between original and augmented images
- Text-Based SSL: Predicting missing words in a sentence
- Audio-Based SSL: Predicting the next segment of audio.

- **Fine-Tuning** (Supervised Task):

# Branches of Machine Learning

## Self-Supervised Learning

2 main stages of SSL:

- **Pre-training** (Unsupervised):

The model learns useful representations from unlabeled data by solving *pretext tasks*:

- Image-Based SSL: Predicting missing parts of an image, similarity between original and augmented images
- Text-Based SSL: Predicting missing words in a sentence
- Audio-Based SSL: Predicting the next segment of audio.

- **Fine-Tuning** (Supervised Task):

The model is adapted to a specific task (image classification, speech recognition) using a smaller labeled dataset.

## Various approaches for ML algorithms

### Supervised learning:

- Neural Networks
- Bayesian inference
- Random forest
- Decision Tree
- Support Vector Machine (SVM)
- K-Nearest Neighbor (KNN)
- Linear regression
- Logistic regression...

### Unsupervised learning:

- Neural Networks
- Principal Composant Analysis (PCA)
- Singular Value Decomposition (SVD)
- K-mean & Prob. clustering...

### Reinforcement learning:

- Neural Networks (Q-learning, Actor-Critic, DDPG, PPO...)
- Monte Carlo...

### Self-Supervised learning:

- Neural Networks (SimCLR, Dino...)

## Various approaches for ML algorithms

### Supervised learning:

- Neural Networks
- Bayesian inference
- Random forest
- Decision Tree
- Support Vector Machine (SVM)
- K-Nearest Neighbor (KNN)
- Linear regression
- Logistic regression...

### Unsupervised learning:

- Neural Networks
- Principal Composant Analysis (PCA)
- Singular Value Decomposition (SVD)
- K-mean & Prob. clustering...

### Reinforcement learning:

- Neural Networks
- Monte Carlo...

### Self-Supervised learning:

- Neural Networks

A large variety of possibilities ∼ success of **Deep Learning** and **Neural Networks**

# Fields & applications of DL

## Computer Vision

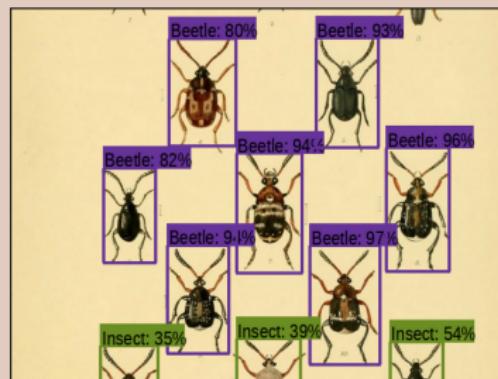
- Image Classification.
- Object Detection .
- (Semantic) Segmentation.
- Pose Estimation.
- Style transfer.
- OCR (Optical Character Recognition)
- Image Generation
- ...



# Fields & applications of DL

## Computer Vision

- Image Classification.
- Object Detection .
- (Semantic) Segmentation.
- Pose Estimation.
- Style transfer.
- OCR (Optical Character Recognition)
- Image Generation
- ...



source : [Tensorflow tutorials object\\_detection](#)

# Fields & applications of DL

## Computer Vision

- Image Classification.
- Object Detection .
- (Semantic) Segmentation.

- Pose Estimation
- Style Transfer
- Optical Flow
- Image Enhancement
- Video Processing



source : [Tensorflow](#)

# Fields & applications of DL

## Computer Vision

- Image Classification.
- Object Detection .
- (Semantic) Segmentation.
- Pose Estimation.
- Style transfer.
- OCR (Optical Character Recognition)
- Image Generation
- ...

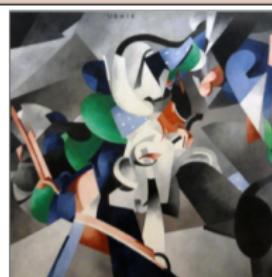


source : [Tensorflow pose\\_detection](#)

# Fields & applications of DL

## Computer Vision

- Image Classification.
- Object Detection .
- (Semantic) Segmentation.
- Pose Estimation.
- Style transfer.
- OCR (Optical Character Recognition)
- Image Generation
- ...



source : [Tensorflow style\\_transfer](#)

# Fields & applications of DL

## Computer Vision

- Image Classification.
- Object Detection .
- (Semantic) Segmentation.
- Pose Estimation.
- Style transfer.
- OCR (Optical Character Recognition).
- Image Generation
- ...

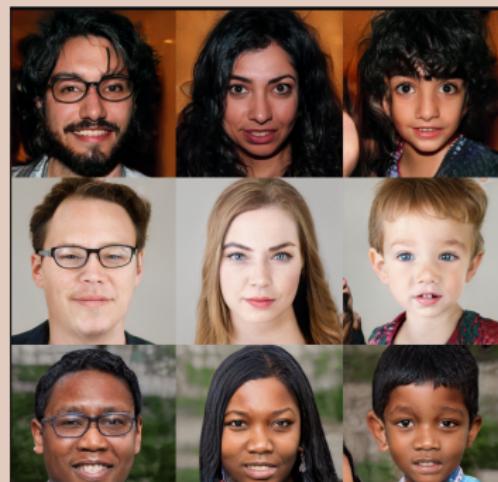


source : [pyimagesearch.com](http://pyimagesearch.com)

# Fields & applications of DL

## Computer Vision

- Image Classification.
- Object Detection .
- (Semantic) Segmentation.
- Pose Estimation.
- Style transfer.
- OCR (Optical Character Recognition).
- Image Generation
- ...



source : [stylegan](#)

# Fields & applications of ML

## *NLP(Natural Language Processing): some applications...*

- Natural Language Understanding (NLU)
- Natural Language Generation (NLG)
- Speech recognition / Speech Synthesis (Text To Speech)
- Machine Translation (languages)
- Virtual agents and ChatBots
- LLM ChatBots
- ...

## Societal issues: Explainability

**Explainability** is becoming an important aspect in research, referred to as

**XAI**    **Explainable Artificial Intelligence** [MM25] [Ali+23]

**IML**    **Interpretable Machine learning** [MCB20] [Mol24] [Rud+21]

# Societal issues: Explainability

**Explainability** is becoming an important aspect in research, referred as

XAI      **Explainable** Artificial Intelligence [MM25] [Ali+23]

IML      **Interpretable** Machine learning [MCB20] [Mol24] [Rud+21]

## Explainability

- **Unexplainability** of the DL results still constitutes an obstacle to their dissemination today.
- Neural Networks are often denigrated as a **Black box** by scientists with a Cartesian approach.
- The growing complexity of NNs (LLM for example) makes the simple explanation of their outputs extremely difficult.

# Societal issues: Decision-making, Certification

## Decision-making

- **Decisions** in sensible fields (justice, insurance, defense...) are being ceded to ML algorithms to the detriment of human control, raising concern for loss of fairness and equity.
- Decision-making algorithms rely inevitably on assumptions (quality of training data) not always verified

# Societal issues: Decision-making, Certification

## Decision-making

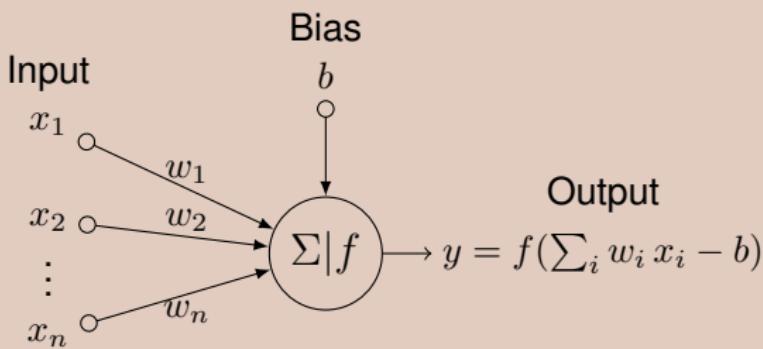
- **Decisions** in sensible fields (justice, insurance, defense...) are being ceded to ML algorithms to the detriment of human control, raising concern for loss of fairness and equity.
- Decision-making algorithms rely inevitably on assumptions (quality of training data) not always verified

## Certification

- The evaluation and the certification of AI systems remains a major issue for their integration in the industry
- Formal certification of ML algorithms still a subject of research
- Examples:  
[LNE: Process certification for AI](#),  
[Fraunhofer: Auditing and Certification of AI Systems](#)

# NN technical aspects

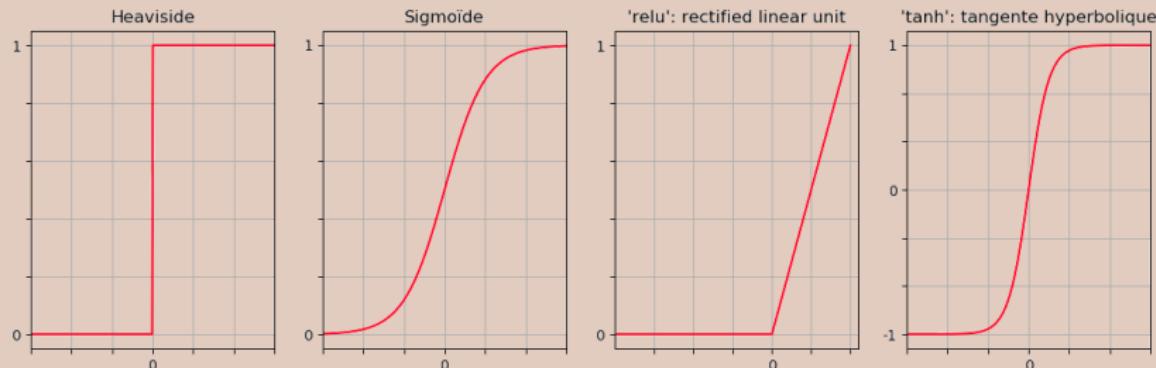
## The artificial neurone



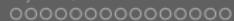
- Receives the input stimuli  $(x_i)_{i=1..n}$  affected by **weights** ( $w_i$ )
- Computes the **weighted sum** of the input:  $\sum_i w_i x_i - b$
- Outputs its activation  $f(\sum_i w_i x_i - b)$ , computed with a non-linear **activation function**  $f$ .

# NN technical aspects

## Common activation functions



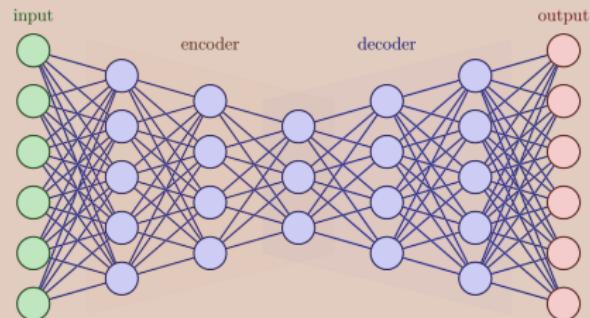
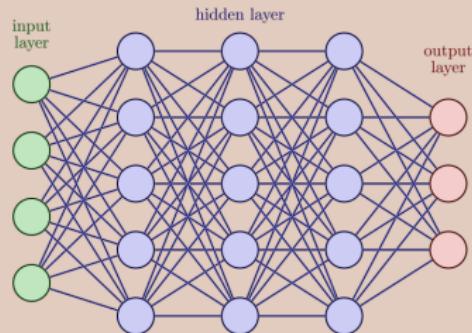
- Introduces a non-linear behavior.
- Sets the range of the neuron output:  $[-1, 1]$ ,  $[0, 1]$ ,  $[0, \infty[$ ...
- The bias  $b$  sets the activation threshold of the neuron.



# NN Computing aspects

## Neural network

Neurons are grouped by layers to form a **Neural Network**



# Neural Network Architectures

Many NN architectures for many applications (non-exhaustive list)

- **Dense Neural Network**

the simplest architecture made of successive layers of neurones, with *FeedForward Neural Network (FNN)* and *Back Propagation* algorithms.

- **Convolutional (CNN)**

- **Recurrent (RNN)**

- **Auto Encoder (AEN)**

- **Generative Adversarial (GAN)**

- **Transformers**

- **Large Language Model (LLM)**

- ...

# Neural Network Architectures

Many NN architectures for many applications (non-exhaustive list)

- **Dense Neural Network**
- **Convolutional (CNN)** - Mostly used for analyzing and classifying images.
- **Recurrent (RNN)**
- **Auto Encoder (AEN)**
- **Generative Adversarial (GAN)**
- **Transformers**
- **Large Language Model (LLM)**
- ...

# Neural Network Architectures

Many NN architectures for many applications (non-exhaustive list)

- **Dense Neural Network**
- **Convolutional (CNN)**
- **Recurrent (RNN)** - Used for time series, like the Long Short-Term Memory (LSTM) algorithm.
- **Auto Encoder (AEN)**
- **Generative Adversarial (GAN)**
- **Transformers**
- **Large Language Model (LLM)**
- ...

# Neural Network Architectures

Many NN architectures for many applications (non-exhaustive list)

- **Dense Neural Network**
- **Convolutional (CNN)**
- **Recurrent (RNN)**
- **Auto Encoder (AEN)**- Dimensionality reduction, Feature extraction, Denoising of data/images, Inputting missing data...
- **Generative Adversarial (GAN)**
- **Transformers**
- **Large Language Model (LLM)**
- ...

# Neural Network Architectures

Many NN architectures for many applications (non-exhaustive list)

- **Dense Neural Network**
- **Convolutional (CNN)**
- **Recurrent (RNN)**
- **Auto Encoder (AEN)**
- **Generative Adversarial (GAN)**  
to generate text, images, music...
- **Transformers**
- **Large Language Model (LLM)**
- ...

# Neural Network Architectures

Many NN architectures for many applications (non-exhaustive list)

- **Dense Neural Network**
- **Convolutional (CNN)**
- **Recurrent (RNN)**
- **Auto Encoder (AEN)**
- **Generative Adversarial (GAN)**
- **Transformers**  
(2017) Natural language processing, and also image classification
- **Large Language Model (LLM)**
- ...

# Neural Network Architectures

Many NN architectures for many applications (non-exhaustive list)

- **Dense Neural Network**
- **Convolutional (CNN)**
- **Recurrent (RNN)**
- **Auto Encoder (AEN)**
- **Generative Adversarial (GAN)**
- **Transformers**
- **Large Language Model (LLM)**

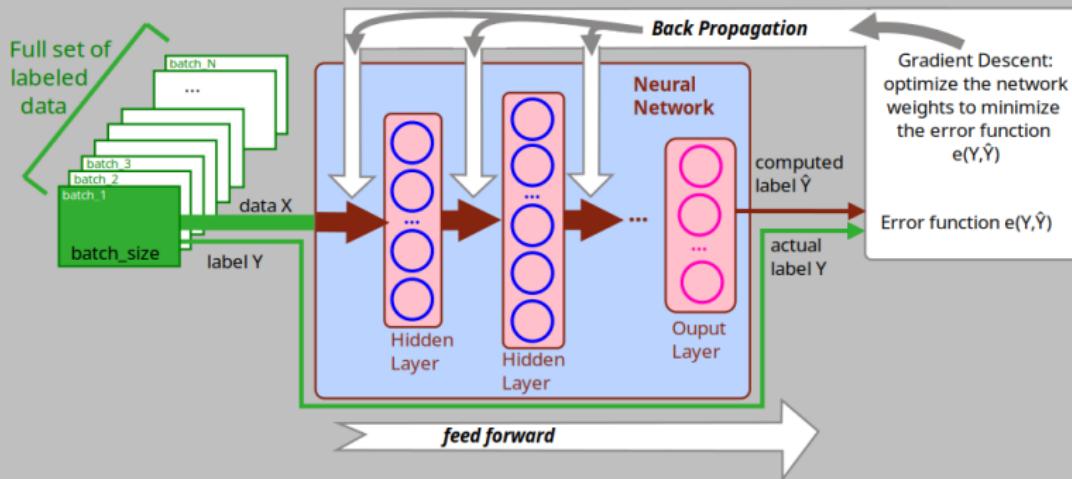
read text, sound, write books, images, speak, make music ...

*OpenAI: ChatGPT - Anthropic: Claude, Sonnet - Google: Gemini -  
Mistral: mistral - Meta: LLama...*

- ...

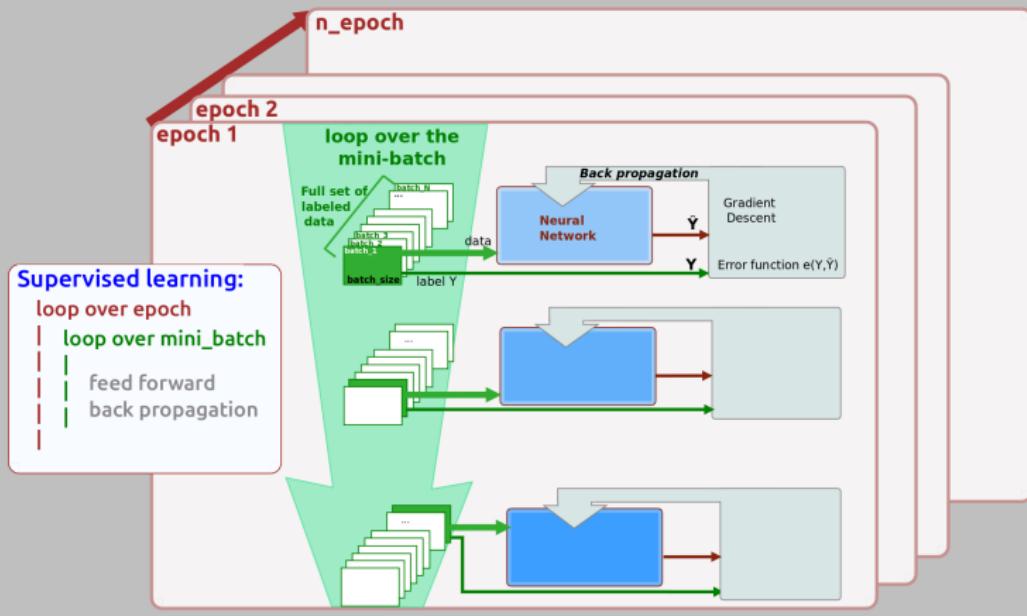
# NN Computing aspects

## Supervised learning : Feed Forward and Back Propagation



- The dataset is split into (mini) **batches** of size **batch\_size**
- After each *feed forward* the *Back Propagation* algorithm modifies the neurons weights to minimize the error  $e$ .

# NN Computing aspects



- Training with the whole dataset is repeated **n\_epoch** times,
- The network state after epoch **n** becomes the initial state for epoch **n+1**.

# Reproducibility

The reproducibility crisis in AI & ML is not new...

- *1500 scientists lift the lid on reproducibility*  
Nature, volume 533, pages 452–454 (2016)
- *Artificial intelligence faces reproducibility crisis*  
Science, 16 Feb 2018, Vol 359, Issue 6377, pp. 725-726
- *Reproducibility Workshops* at major AI conferences:
  - NeurIPS NeurIPS Reproducibility Challenge 2018, 2019, 2020...
  - ICML International Conference on Machine Learning: 2017, 2018, 2019...
  - AAAI Association for the Advancement of Artificial Intelligence: reproducibility checklist

# Reproducibility

The reproducibility crisis in AI & ML is not new...

- *1500 scientists lift the lid on reproducibility*

Nature, volume 533, pa

- *Artificial intelligence fact*

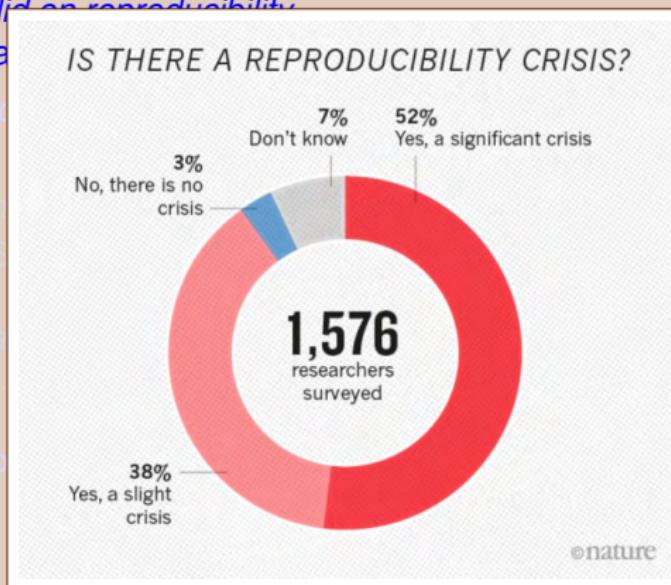
Science, 16 Feb 2018,

- *Reproducibility Worksh*

- NeurIPS NeurIPS  
2019, 2020...

- ICML International  
2017, 2018, 2019

- AAAI Association  
Intelligence: repro



# Reproducibility

The reproducibility crisis in AI & ML is not new...

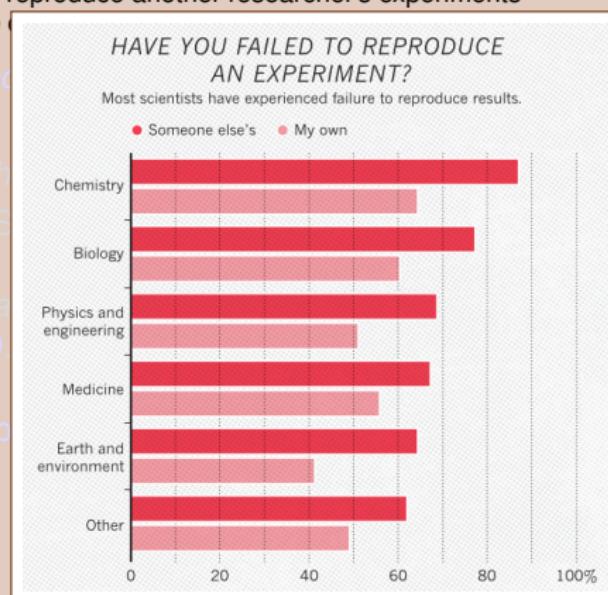
- *1500 scientists lift the lid on reproducibility*

Nature, volume 533, pages 452–454 (2016)

70% of researchers failed to reproduce another researcher's experiments  
50% failed to reproduce one of their own

- *Artificial intelligence fails the reproducibility test*  
Science, 16 Feb 2018,

- *Reproducibility Workshop*
  - NeurIPS NeurIPS 2019, 2020...
  - ICML International Conference on Machine Learning 2017, 2018, 2019
  - AAAI Association for the Advancement of Artificial Intelligence: reproducibility workshop



# Reproducibility

The reproducibility crisis in AI & ML is not new...

- *1500 scientists lift the lid on reproducibility*  
Nature, volume 533, pages 452–454 (2016)
- *Artificial intelligence faces reproducibility crisis*  
Science, 16 Feb 2018, Vol 359, Issue 6377, pp. 725-726
- *Reproducibility Workshops at major AI conferences:*
  - NeurIPS NeurIPS Reproducibility Challenge 2018, 2019, 2020...
  - ICML International Conference on Machine Learning: 2017, 2018, 2019...
  - AAAI Association for the Advancement of Artificial Intelligence: reproducibility checklist

# Reproducibility

The reproducibility crisis in AI & ML is not new...

- *1500 scientists lift the lid on reproducibility*  
Nature, volume 533, pages 452–454 (2016)
- *Artificial intelligence faces reproducibility crisis*  
Science, 16 Feb 2018, Vol 359, Issue 6377, pp. 725-726
- *Reproducibility Workshops* at major AI conferences:
  - NeurIPS NeurIPS Reproducibility Challenge [2018](#), [2019](#), [2020](#)...
  - ICML International Conference on Machine Learning: [2017](#), [2018](#), [2019](#)...
  - AAAI Association for the Advancement of Artificial Intelligence: [reproducibility checklist](#)

# Reproducibility

The reproducibility crisis in AI & ML is not new...

- *1500 scientists lift the lid on reproducibility*  
Nature, volume 533, pages 452–454 (2016)
- *Artificial intelligence faces reproducibility crisis*  
Science, 16 Feb 2018, Vol 359, Issue 6377, pp. 725-726
- *Reproducibility Workshops* at major AI conferences:
  - NeurIPS NeurIPS Reproducibility Challenge [2018](#), [2019](#), [2020](#)...
  - ICML *International Conference on Machine Learning*: [2017](#), [2018](#), [2019](#)...
  - AAAI *Association for the Advancement of Artificial Intelligence*: [reproducibility checklist](#)
- Many papers relate the reproducibility question in AI & ML:  
[\[Bak16\]](#)[\[Alb+23\]](#)[\[Pin+20\]](#)[\[TVD18\]](#)[\[Sha19\]](#)[\[eal20\]](#)[\[GK18\]](#)[\[Hut18\]](#)..
- Currently still a problem for trusting ML outputs.

# Reproducibility - The DL context

# Reproducibility - The DL context

## Reproducibility for Deep Learning

refers to the ability to achieve the **same or similar results**:

- Using the same **training dataset**
- Using the same **algorithms**  
model architecture, parameters and hyperparameters, and other code
- Within the same computing **environment**  
software (Python modules) used to run the algorithm,  
data-preprocessing... and all pieces of code.

# Reproducibility - The DL context

## Reproducibility: an important issue for confidence in DL

- Necessary to trust DL outputs
- Required for the validation of published results & claims
- Helps to promote open and accessible research
- Makes published matter an educational tool for students

# Reproducibility - The DL context

## Reproducibility: an important issue for confidence in DL

- Necessary to trust DL outputs
- Required for the validation of published results & claims
- Helps to promote open and accessible research
- Makes published matter an educational tool for students

# Reproducibility - The DL context

## Reproducibility: an important issue for confidence in DL

- Necessary to trust DL outputs
- Required for the validation of published results & claims
- Helps to promote open and accessible research
- Makes published matter an educational tool for students

# Reproducibility - The DL context

## Reproducibility: an important issue for confidence in DL

- Necessary to trust DL outputs
- Required for the validation of published results & claims
- Helps to promote open and accessible research
- Makes published matter an educational tool for students

# Reproducibility - The DL context

2 main reasons why DL training is not reproducible

- NN training involves random draws at many stages:
- Non-determinism in hardware operations  
(floating-point ops are sensitive to computation order...)
  - Most of these cases occur on GPUs  
*auto-tuning*: libraries CUDA and cuDNN auto-select the optimal primitive operations at runtime ~ different ops may occur on subsequent run
  - Forcing deterministic ops possible but ~ slower computation 😊

# Reproducibility - The DL context

## 2 main reasons why DL training is not reproducible

- NN training involves random draws at many stages:
  - Data augmentation [SK19]
  - Shuffling training data
  - Batch ordering at each epoch
  - Initialisation of the NN weights
  - Exploration noise during training (DRL)
  - Specific algorithm treatments: *dropout* [Sri+14]
  - Optimisation algorithms: *stochastic gradient* (adam)...
- Non-determinism in hardware operations  
(floating-point ops are sensitive to computation order...)
  - Most of these cases occur on GPUs  
*auto-tuning*: libraries CUDA and cuDNN auto-select the optimal primitive operations at runtime ~ different ops may occur on subsequent run
  - Forcing deterministic ops possible but ~ slower computation 😬

# Reproducibility - The DL context

## 2 main reasons why DL training is not reproducible

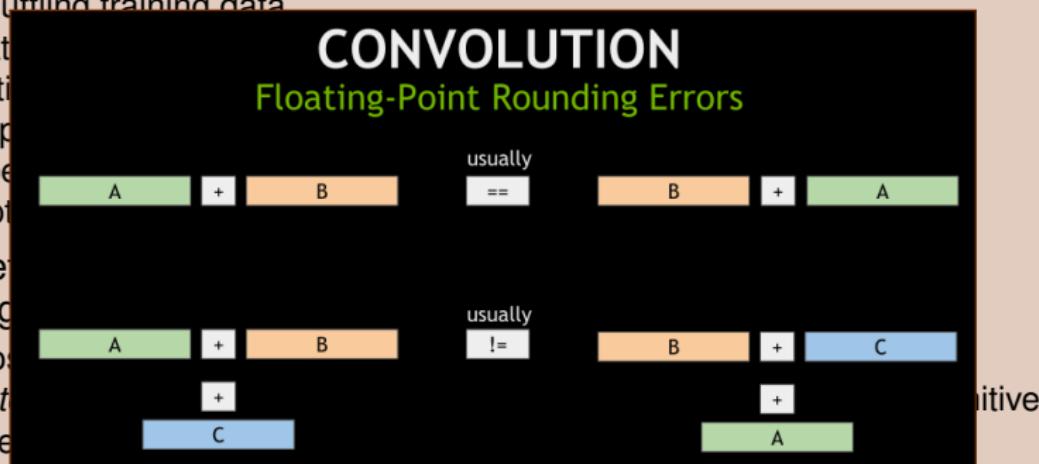
- NN training involves random draws at many stages:
  - Data augmentation [SK19]
  - Shuffling training data
  - Batch ordering at each epoch
  - Initialisation of the NN weights
  - Exploration noise during training (DRL)
  - Specific algorithm treatments: *dropout* [Sri+14]
  - Optimisation algorithms: *stochastic gradient* (adam)...
- Non-determinism in hardware operations  
(floating-point ops are sensitive to computation order...)
  - Most of these cases occur on GPUs  
*auto-tuning*: libraries CUDA and cuDNN auto-select the optimal primitive operations at runtime  $\leadsto$  different ops may occur on subsequent runs
  - Forcing deterministic ops possible but  $\sim$  slower computation 😬

# Reproducibility - The DL context

2 main reasons why DL training is not reproducible

- NN training involves random draws at many stages:

- Data augmentation [SK19]
- Shuffling training data
- Batching
- Initialization
- Experiments
- Special cases
- Optimizations



- Non-deterministic operations (floating-point)
- Most operations are non-deterministic, e.g., auto-tuned operations
- source: [NVIDIA-1]
- Forcing deterministic ops possible but ~ slower computation 😊

# Reproducibility - The DL context

## 2 main reasons why DL training is not reproducible

- NN training involves random draws at many stages:
  - Data augmentation [SK19]
  - Shuffling training data
  - Batch ordering at each epoch
  - Initialisation of the NN weights
  - Exploration noise during training (DRL)
  - Specific algorithm treatments: *dropout* [Sri+14]
  - Optimisation algorithms: *stochastic gradient* (adam)...
- Non-determinism in hardware operations  
(floating-point ops are sensitive to computation order...)
  - Most of these cases occur on GPUs  
*auto-tuning*: libraries CUDA and cuDNN auto-select the optimal primitive operations at runtime  $\leadsto$  different ops may occur on subsequent run
  - Forcing deterministic ops possible but  $\leadsto$  slower computation 😞

# Reproducibility - The DL context

some practices to make DL reproducible

1/ set the **SEED** ↪ makes random generators reproducible

- tensorflow
- pytorch

[ten24a][ten24c][ten24b]

[Pyt24][PyT24]

# Reproducibility - The DL context

some practices to make DL reproducible

1/ set the **SEED** ↳ makes random generators reproducible

- tensorflow

[ten24a][ten24c][ten24b]

```
tf.keras.utils.set_random_seed(<SEED>) # sets Python, NumPy and TensorFlow seed
```

- pytorch

[Pyt24][PyT24]

# Reproducibility - The DL context

some practices to make DL reproducible

1/ set the **SEED** ~ makes random generators reproducible

- tensorflow

[ten24a][ten24c][ten24b]

```
tf.keras.utils.set_random_seed(<SEED>) # sets Python, NumPy and TensorFlow seed
```

- pytorch

[Pyt24][PyT24]

```
import random
import numpy as np
import torch

random.seed(<SEED>)           # Sets the seed for Python built-in random module
np.random.seed(<SEED>)         # Sets the seed for NumPy random number generator

torch.manual_seed(<SEED>)      # Sets the seed for PyTorch CPU random number generator
```

```
torch.cuda.manual_seed(<SEED>)    # Sets the seed for the current GPU device
torch.cuda.manual_seed_all(<SEED>)  # Sets the seed for all available GPU devices
```

# Reproducibility - The DL context

some practices to make DL reproducible

2/ Set the operations deterministic  $\rightsquigarrow$  makes GPU ops deterministic at the cost of performance

- tensorflow
- pytorch

[ten24a][ten24c][ten24b]

[Pyt24][PyT24]

# Reproducibility - The DL context

some practices to make DL reproducible

2/ Set the operations deterministic  $\rightsquigarrow$  makes GPU ops deterministic at the cost of performance

- tensorflow

[ten24a] [ten24c] [ten24b]

```
tf.config.experimental.enable_op_determinism()  
# this will make GPU ops as deterministic as possible,  
# but it will affect the overall performance
```

- pytorch

[Pyt24] [PyT24]

# Reproducibility - The DL context

some practices to make DL reproducible

2/ Set the operations deterministic  $\rightsquigarrow$  makes GPU ops deterministic at the cost of performance

- tensorflow

[ten24a] [ten24c] [ten24b]

```
tf.config.experimental.enable_op_determinism()  
# this will make GPU ops as deterministic as possible,  
# but it will affect the overall performance
```

- pytorch

[PyT24] [PyT24]

```
# try this:  
import torch.backends.cudnn  
torch.backends.cudnn.deterministic = True  
torch.backends.cudnn.benchmark = False
```

but it can be more tricky... seed [PyT24].

# Reproducibility - Simple examples

Simple example: Dense NN to classify MNIST hand-written digits [GitHub](#)



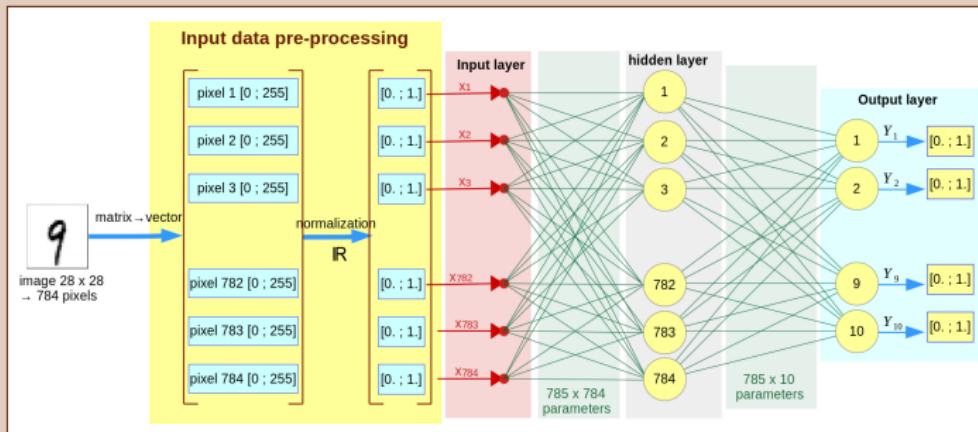
[MNIST dataset](#): 70 000 labeled grayscale images  $28 \times 28$  pixels (60000 training, 10000 validation)

# Reproducibility - Simple examples

Simple example: Dense NN to classify MNIST hand-written digits [GitHub](#)



**MNIST dataset:** 70 000 labeled grayscale images  $28 \times 28$  pixels (60000 training, 10000 validation)



Dense NN: 784 inputs, 1 hidden layer (784 neurones, *relu*), 1 output layer (10 neurones, *softmax*)

# Reproducibility - Simple examples

## A simple example: Dense NN

tensorflow/keras snippet to build the model:

```
def build_DNN(seed=None):
    if seed is not None:
        ##### # Deterministic training #
        ##### # 1/ sets the Python seed, the NumPy seed, and the TensorFlow seed:
        tf.keras.utils.set_random_seed(seed)

        # 2/ make the tf ops deterministic
        # this will make GPU ops as deterministic as possible, but it will affect
        # the overall performance:
        tf.config.experimental.enable_op_determinism()

    # 5 keras lines to build the dense network.
    model = Sequential()
    model.add(Input(shape=(NB_INPUT,), name='input'))                      # INPUT layer
    model.add(Dense(NB_NEURON, activation='relu', name='c1'))                 # First hidden layer
    model.add(Dense(NB_CLASS, activation='softmax', name='c2'))                # OUTPUT layer
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model
```

# Reproducibility - Simple examples

## A simple example: Dense NN **no seed fixed**

Simple experiment: write a loop where the model is built **without setting seed**, trained & evaluated once at each iteration (*epochs=1*) :

# Reproducibility - Simple examples

## A simple example: Dense NN no seed fixed $\leadsto$ not reproducible

Simple experiment: write a loop where the model is built **without setting seed**, trained & evaluated once at each iteration (*epochs=1*) :

```
print(get_cpu_info()['brand_raw'])
for _ in range(5):
    model = build_DNN() # Build a new model without setting seed
    hist = model.fit(x_train, y_train, epochs=1, batch_size=16, validation_data=(x_valid, y_valid), verbose=2)

Intel(R) Xeon(R) W-1390P @ 3.50GHz
3750/3750 - 8s - 2ms/step - accuracy: 0.9448 - loss: 0.1822 - val_accuracy: 0.9582 - val_loss: 0.1349
3750/3750 - 8s - 2ms/step - accuracy: 0.9468 - loss: 0.1789 - val_accuracy: 0.9690 - val_loss: 0.0936
3750/3750 - 8s - 2ms/step - accuracy: 0.9459 - loss: 0.1800 - val_accuracy: 0.9754 - val_loss: 0.0798
3750/3750 - 8s - 2ms/step - accuracy: 0.9462 - loss: 0.1771 - val_accuracy: 0.9690 - val_loss: 0.1082
3750/3750 - 8s - 2ms/step - accuracy: 0.9469 - loss: 0.1777 - val_accuracy: 0.9698 - val_loss: 0.0956
```

# Reproducibility - Simple examples

A simple example: Dense NN no seed fixed  $\leadsto$  not reproducible

Simple experiment: write a loop where the model is built **without setting seed**, trained & evaluated once at each iteration (*epochs=1*) :

```
print(get_cpu_info()['brand_raw'])
for _ in range(5):
    model = build_DNN() # Build a new model without setting seed
    hist = model.fit(x_train, y_train, epochs=1, batch_size=16, validation_data=(x_valid, y_valid), verbose=2)

Intel(R) Xeon(R) W-1390P @ 3.50GHz
3750/3750 - 8s - 2ms/step - accuracy: 0.9448 - loss: 0.1822 - val_accuracy: 0.9582 - val_loss: 0.1349
3750/3750 - 8s - 2ms/step - accuracy: 0.9468 - loss: 0.1789 - val_accuracy: 0.9690 - val_loss: 0.0936
3750/3750 - 8s - 2ms/step - accuracy: 0.9459 - loss: 0.1800 - val_accuracy: 0.9754 - val_loss: 0.0798
3750/3750 - 8s - 2ms/step - accuracy: 0.9462 - loss: 0.1771 - val_accuracy: 0.9690 - val_loss: 0.1082
3750/3750 - 8s - 2ms/step - accuracy: 0.9469 - loss: 0.1777 - val_accuracy: 0.9698 - val_loss: 0.0956
```

$\leadsto$  Same behaviour on Google Collab platform (Xeon processor):

```
1 print(get_cpu_info()['brand_raw'])
2 for _ in range(5):
3     model = build_DNN() # Build a new model without setting seed
4     hist = model.fit(x_train[:1000], y_train[:1000], epochs=1, batch_size=16,
5                       validation_data=(x_valid[:200], y_valid[:200]), verbose=2)

Intel(R) Xeon(R) CPU @ 2.20GHz
63/63 - 2s - 27ms/step - accuracy: 0.7410 - loss: 0.8808 - val_accuracy: 0.8550 - val_loss: 0.5447
63/63 - 2s - 27ms/step - accuracy: 0.7470 - loss: 0.8468 - val_accuracy: 0.8150 - val_loss: 0.5601
63/63 - 2s - 34ms/step - accuracy: 0.7520 - loss: 0.8321 - val_accuracy: 0.8350 - val_loss: 0.5183
63/63 - 2s - 39ms/step - accuracy: 0.7490 - loss: 0.8592 - val_accuracy: 0.8600 - val_loss: 0.4838
63/63 - 2s - 28ms/step - accuracy: 0.7360 - loss: 0.8955 - val_accuracy: 0.8150 - val_loss: 0.5690
```

# Reproducibility - Simple examples

A simple example: Dense NN no seed fixed  $\leadsto$  not reproducible

Non-reproducibility worsens if the dataset is reduced:

```
1 print(get_cpu_info()['brand_raw'])
2 for _ in range(5):
3     model = build_DNN() # Build a new model without setting seed
4     hist = model.fit(x_train[:1000], y_train[:1000], epochs=1, batch_size=16,
5                       validation_data=(x_valid[:200], y_valid[:200]), verbose=2)
6
7 Intel(R) Xeon(R) W-1390P @ 3.50GHz
8 63/63 - 1s - 10ms/step - accuracy: 0.7490 - loss: 0.8607 - val_accuracy: 0.8300 - val_loss: 0.5360
9 63/63 - 1s - 10ms/step - accuracy: 0.7510 - loss: 0.8562 - val_accuracy: 0.8300 - val_loss: 0.5765
10 63/63 - 1s - 10ms/step - accuracy: 0.7490 - loss: 0.8049 - val_accuracy: 0.8600 - val_loss: 0.4851
11 63/63 - 1s - 9ms/step - accuracy: 0.7250 - loss: 0.8943 - val_accuracy: 0.8550 - val_loss: 0.4981
12 63/63 - 1s - 11ms/step - accuracy: 0.7500 - loss: 0.8424 - val_accuracy: 0.8300 - val_loss: 0.5456
```

$\leadsto$  Same behaviour on Google Collab platform (Xeon processor):

```
1 print(get_cpu_info()['brand_raw'])
2 for _ in range(5):
3     model = build_DNN() # Build a new model without setting seed
4     hist = model.fit(x_train[:1000], y_train[:1000], epochs=1, batch_size=16,
5                       validation_data=(x_valid[:200], y_valid[:200]), verbose=2)
6
7 Intel(R) Xeon(R) CPU @ 2.20GHz
8 63/63 - 2s - 27ms/step - accuracy: 0.7410 - loss: 0.8808 - val_accuracy: 0.8550 - val_loss: 0.5447
9 63/63 - 2s - 27ms/step - accuracy: 0.7470 - loss: 0.8468 - val_accuracy: 0.8150 - val_loss: 0.5601
10 63/63 - 2s - 34ms/step - accuracy: 0.7520 - loss: 0.8321 - val_accuracy: 0.8350 - val_loss: 0.5183
11 63/63 - 2s - 39ms/step - accuracy: 0.7490 - loss: 0.8592 - val_accuracy: 0.8600 - val_loss: 0.4838
12 63/63 - 2s - 28ms/step - accuracy: 0.7360 - loss: 0.8955 - val_accuracy: 0.8150 - val_loss: 0.5690
```

# Reproducibility - Simple example

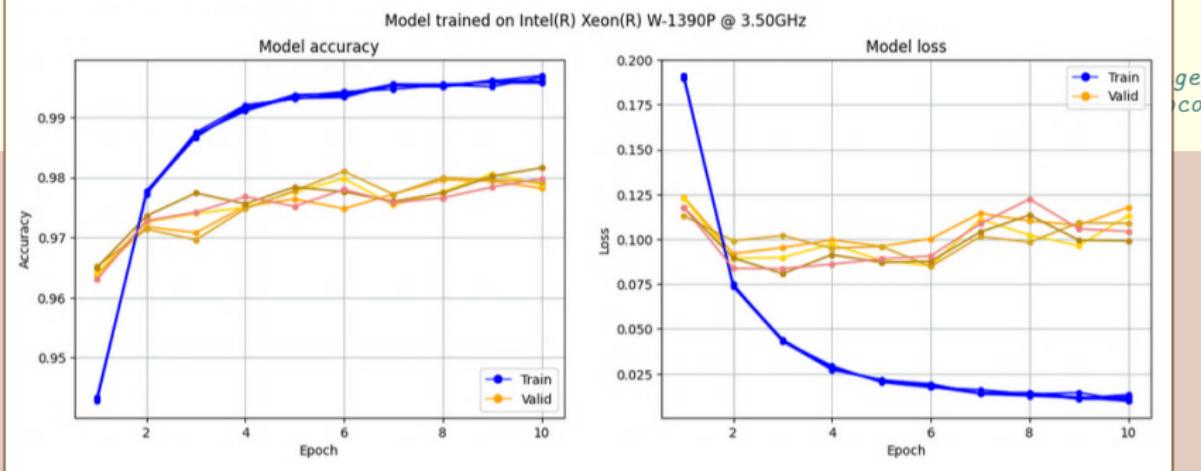
## A simple example: Dense NN no seed fixed

```
for i in range(5):
    model = build_DNN()                      # build a new mode without setting seed
    hist  = model.fit(x_train, y_train,          # images, labels
                      epochs=10,                 # the total number of successive trainings
                      batch_size=32,              # input images by batch of <batche_size> images
                      validation_data=(x_valid, y_valid), # validation after each epcoh
                      verbose=0)
```

# Reproducibility - Simple example

A simple example: Dense NN no seed fixed  $\leadsto$  not reproducible

```
for i in range(5):
```

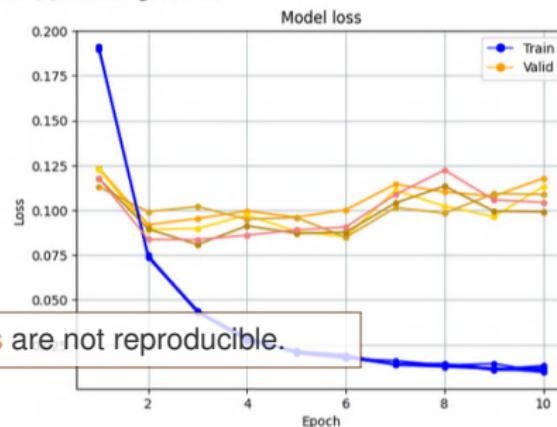
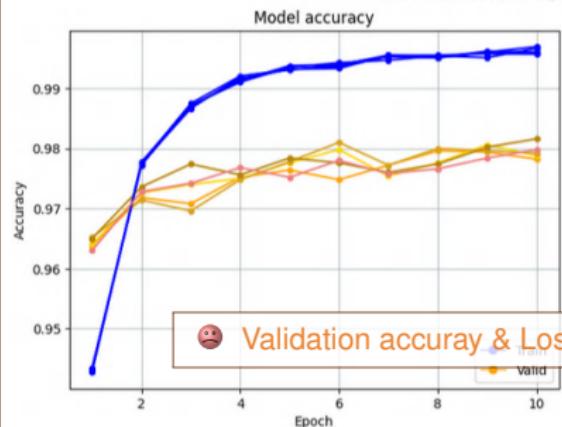


# Reproducibility - Simple example

A simple example: Dense NN no seed fixed  $\leadsto$  not reproducible

```
for i in range(5):
```

Model trained on Intel(R) Xeon(R) W-1390P @ 3.50GHz



# Reproducibility - Simple examples

## A simple example: Dense NN **seed set** $\leadsto$ reproducible

A model is built with **seed set** at each iteration:

```
print(get_cpu_info()['brand_raw'])
for _ in range(5):
    model = build_DNN(seed=1234) # Build a new model with seed set
    hist = model.fit(x_train, y_train, epochs=1, batch_size=16, validation_data=(x_valid, y_valid), verbose=2)

Intel(R) Xeon(R) W-1390P @ 3.50GHz
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 7s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 7s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
```

# Reproducibility - Simple examples

## A simple example: Dense NN **seed set** $\leadsto$ reproducible

A model is built with **seed set** at each iteration:

```
print(get_cpu_info()['brand_raw'])
for _ in range(5):
    model = build_DNN(seed=1234) # Build a new model with seed set
    hist = model.fit(x_train, y_train, epochs=1, batch_size=16, validation_data=(x_valid, y_valid), verbose=2)

Intel(R) Xeon(R) W-1390P @ 3.50GHz
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 7s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 7s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
```

$\leadsto$  Same behaviour on Google Collab platform (Xeon processor):

```
1 print(get_cpu_info()['brand_raw'])
2 for _ in range(5):
3     model = build_DNN(seed=1234) # Build a new model with seed set
4     hist = model.fit(x_train, y_train, epochs=1, batch_size=16, validation_data=(x_valid, y_valid), verbose=2)

Intel(R) Xeon(R) W-1390P @ 3.50GHz
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 7s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 7s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
```

# Reproducibility - Simple examples

A simple example: reload model structure & weights  $\leadsto$  **reproducible**

The model built once with **seed set** is saved and then just re-loaded (structure & weights) at each iteration:

```
model = build_DNN(seed=1234)
model.save('models/DNN_relu_seed1234.keras')

print(get_cpu_info()['brand_raw'])
for _ in range(5):
    model = tf.keras.models.load_model('models/DNN_relu_seed1234.keras') # reload the model structure & weights
    hist = model.fit(x_train, y_train, epochs=1, batch_size=16, validation_data=(x_valid, y_valid), verbose=2)

Intel(R) Xeon(R) W-1390P @ 3.50GHz
3750/3750 - 9s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 7s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
```

# Reproducibility - Simple examples

A simple example: reload model structure & weights  $\leadsto$  reproducible

The model built once with **seed set** is saved and then just re-loaded (structure & weights) at each iteration:

```
model = build_DNN(seed=1234)
model.save('models/DNN_relu_seed1234.keras')
```

print(get\_ipython().run\_line\_magic('time', ''))  
**for** \_ **in** range(5):
 model = build\_DNN(seed=1234)
 model.save('models/DNN\_relu\_seed1234.keras')
 hist = model.fit(x\_train, y\_train, epochs=1, batch\_size=16, validation\_data=(x\_valid, y\_valid), verbose=2)

Same behaviour on Google Collab platform (Xeon processor):

```
Intel(R) Xeon(R) W-1390P @ 3.50GHz
3750/3750 - 7s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 9s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 9s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 8s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
3750/3750 - 7s - 2ms/step - accuracy: 0.9457 - loss: 0.1790 - val_accuracy: 0.9688 - val_loss: 0.1067
```

# Reproducibility - Simple example

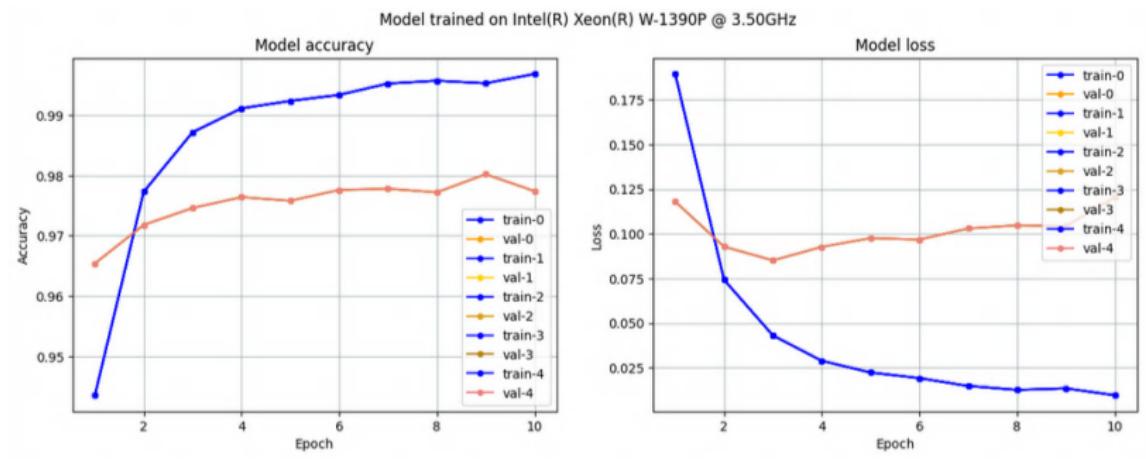
## A simple example: Dense NN **seed fixed**

```
for i in range(5):
    model = build_DNN(1234)                  # build a new mode without setting seed
    hist  = model.fit(x_train, y_train, # images, labels
                      epochs=10,          # the total number of successive trainings
                      batch_size=32,       # input images by batch of <batche_size> images
                      validation_data=(x_valid, y_valid), # validation after each epcoh
                      verbose=0)
```

# Reproducibility - Simple example

A simple example: Dense NN **seed fixed**  $\leadsto$  reproducible

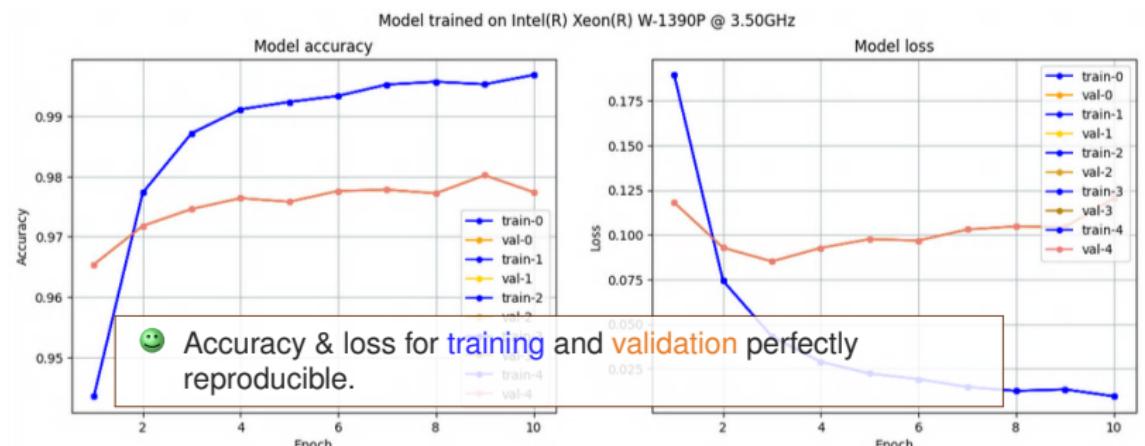
```
for i in range(5):
```



# Reproducibility - Simple example

A simple example: Dense NN **seed fixed**  $\leadsto$  reproducible

```
for i in range(5):
```

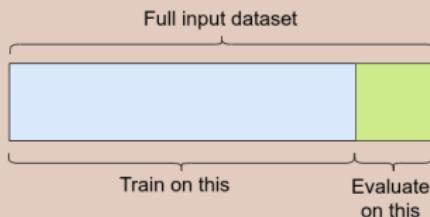


# Training & Evaluation strategy

Validation techniques are essential for assessing how well a model generalizes to unseen data.

## Split the input Dataset

Common practice: Simple hold-out validation split



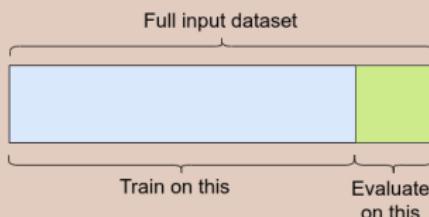
Input Dataset  $\leadsto$  **Train** and **Valid** (test) datasets

# Training & Evaluation strategy

Validation techniques are essential for assessing how well a model generalizes to unseen data.

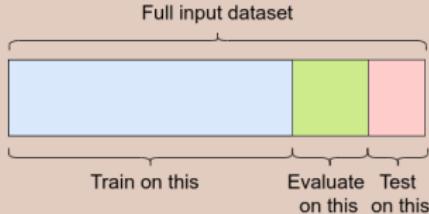
## Split the input Dataset

Common practice: Simple hold-out validation split



Input Dataset  $\leadsto$  **Train** and **Valid** (test) datasets

State of the art: Train/Validation/Test



Input Dataset  $\leadsto$  **Train**, **Valid** and **test** datasets.

# Training & Evaluation strategy

## Split the input Dataset

**Random Split:** most used method  
unless dealing with ordered data like time-series

- **Training Dataset**
- **Validation Dataset**
- **Test Dataset**

# Training & Evaluation strategy

## Split the input Dataset

**Random Split:** most used method  
unless dealing with ordered data like time-series

- **Training Dataset** ~ to train the model
- **Validation Dataset**
  
- **Test Dataset**

# Training & Evaluation strategy

## Split the input Dataset

**Random Split:** most used method

unless dealing with ordered data like time-series

- **Training Dataset** → to train the model
- **Validation Dataset** → to get unbiased evaluation (loss function...) of the model during training , used to tune:
  - **model hyper-parameters**: nb layers, nb neurons/layer, activation function/layer, optimisation algorithm...
  - **training hyper-parameters**: split ratio, batch size, nb epochs, learning rate
  - But training and validation dataset are known to the trained DL model
- **Test Dataset**

# Training & Evaluation strategy

## Split the input Dataset

**Random Split:** most used method

unless dealing with ordered data like time-series

- **Training Dataset** → to train the model
- **Validation Dataset** → to get unbiased evaluation (loss function...) of the model during training , used to tune:
  - **model hyper-parameters**: nb layers, nb neurons/layer, activation function/layer, optimisation algorithm...
  - **training hyper-parameters**: split ratio, batch size, nb epochs, learning rate
  - But training and validation dataset are known to the trained DL model
- **Test Dataset** completely new to the model → to get performance of the trained model.

# Training & Evaluation strategy

## Split the input Dataset: different strategies

### ● Random Splitting

- Randomly shuffle dataset and assign samples to train, valid or test sets according to given ratios
- Supervised Learning ~ may not be indicated with *imbalanced dataset* (some classes have significantly fewer items than others)

### ● Stratified Splitting (Supervised Learning)

- The dataset is divided while preserving the relative proportions of each class across the subsets

# Training & Evaluation strategy

Split the input Dataset: different strategies

## ● Random Splitting

- Randomly shuffle dataset and assign samples to train, valid or test sets according to given ratios
- Supervised Learning  $\leadsto$  may not be indicated with *imbalanced dataset* (some classes have significantly fewer items than others)

## ● Stratified Splitting (Supervised Learning)

- The dataset is divided while preserving the relative proportions of each class across the subsets

# Training & Evaluation strategy

Split the input Dataset: different strategies

## ● Random Splitting

- Randomly shuffle dataset and assign samples to train, valid or test sets according to given ratios
- Supervised Learning  $\leadsto$  may not be indicated with *imbalanced dataset* (some classes have significantly fewer items than others)

## ● Stratified Splitting (Supervised Learning)

- The dataset is divided while preserving the relative proportions of each class across the subsets

# Training & Evaluation strategy

Split the input Dataset: different strategies

## ● Random Splitting

- Randomly shuffle dataset and assign samples to train, valid or test sets according to given ratios
- Supervised Learning  $\leadsto$  may not be indicated with *imbalanced dataset* (some classes have significantly fewer items than others)

## ● Stratified Splitting (Supervised Learning)

- The dataset is divided while preserving the relative proportions of each class across the subsets
- For example : [scikit train\\_test\\_split](#) function  $\leadsto$  random & stratified split.

# Training & Evaluation strategy

## Split the input Dataset: strategies

- **K-fold Cross-Validation** K-Fold CV
- **Stratified K-Fold Cross-Validation**
- And yet other strategies depending on the dataset...

# Training & Evaluation strategy

## Split the input Dataset: strategies

### ● **K-fold Cross-Validation** K-Fold CV

- Recommended with small datasets
- One of the ways to reduce training overfit
- Shuffle then split the input dataset into  $K$  (often 10) partitions (*folds*)
- Loop over  $K$ :
  - fold  $k \sim$  validation and the  $k - 1$  remaining folds  $\sim$  training
  - At the end, compute average metrics (val loss, val accuracy...) over  $k$

### ● **Stratified K-Fold Cross-Validation**

- And yet other strategies depending on the dataset...

# Training & Evaluation strategy

## Split the input Dataset: strategies

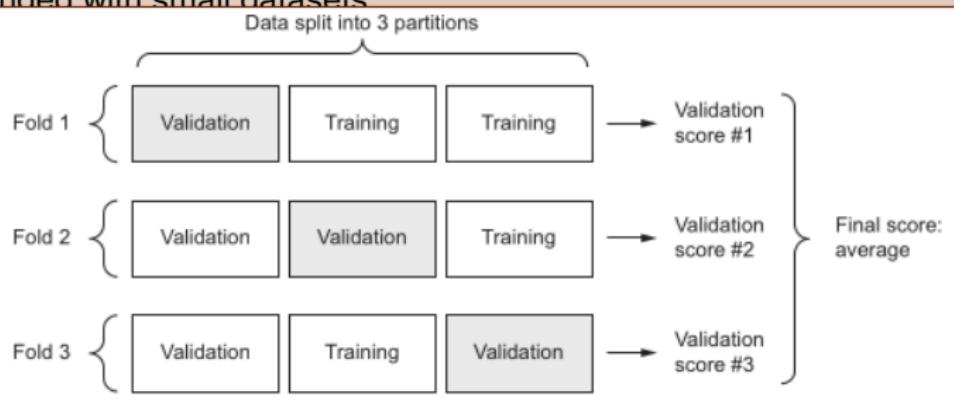
### ● K-fold Cross-Validation K-Fold CV

- Recommended with small datasets
- One of the
- Shuffle the
- Loop over fold  $k$

At the end

### ● Stratified K-F

### ● And yet other



source: [Cho18] p. 99

# Training & Evaluation strategy

## Split the input Dataset: strategies

### ● **K-fold Cross-Validation** K-Fold CV

- Recommended with small datasets
- One of the ways to reduce training overfit
- Shuffle then split the input dataset into  $K$  (often 10) partitions (*folds*)
- Loop over  $K$ :
  - fold  $k \sim$  validation and the  $k - 1$  remaining folds  $\sim$  training
  - At the end, compute average metrics (val loss, val accuracy...) over  $k$

### ● **Stratified K-Fold Cross-Validation**

- K-Fold CV + stratifying  $\sim$  each fold has the same proportion of classes as the entire dataset.
- And yet other strategies depending on the dataset...

# Training & Evaluation strategy

## Split the input Dataset: strategies

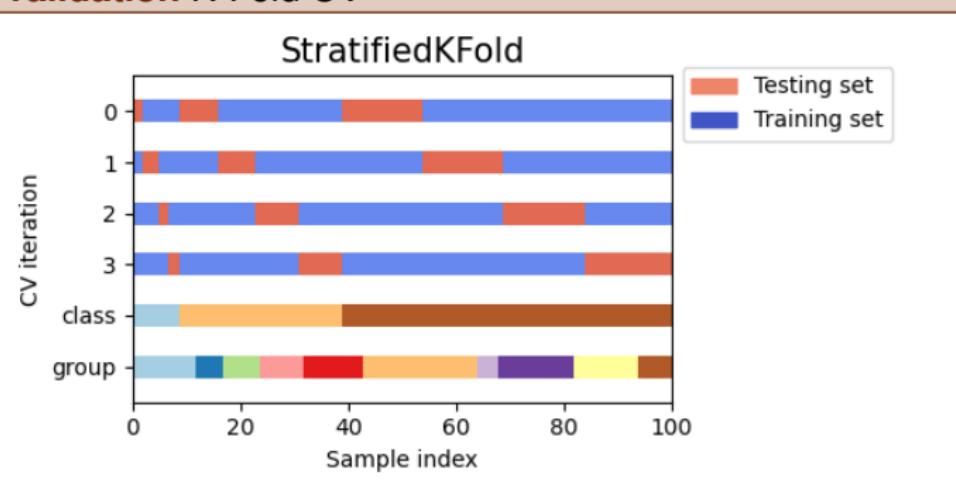
### ● K-fold Cross-Validation K-Fold CV

- Recommended
- One of the best
- Shuffle the data
- Loop over fold  $k$
- At the end

### ● Stratified K-Fold CV

- K-Fold CV as the entire process

### ● And yet other



source: [scikit-learn Cross-validation pages](#)

# Training & Evaluation strategy

## Split the input Dataset: strategies

### ● K-fold Cross-Validation K-Fold CV

- Recommended with small datasets
- One of the ways to reduce training overfit
- Shuffle then split the input dataset into  $K$  (often 10) partitions (*folds*)
- Loop over  $K$ :
  - fold  $k \sim$  validation and the  $k - 1$  remaining folds  $\sim$  training
  - At the end, compute average metrics (val loss, val accuracy...) over  $k$

### ● Stratified K-Fold Cross-Validation

- K-Fold CV + stratifying  $\sim$  each fold has the same proportion of classes as the entire dataset.
- And yet other strategies depending on the dataset...
  - Exple Python : **scikit** **KFold** function  $\sim$  random, stratified, grouped...

# Training & Evaluation strategy

## Split the input Dataset: strategies

### ● K-fold Cross-Validation K-Fold CV

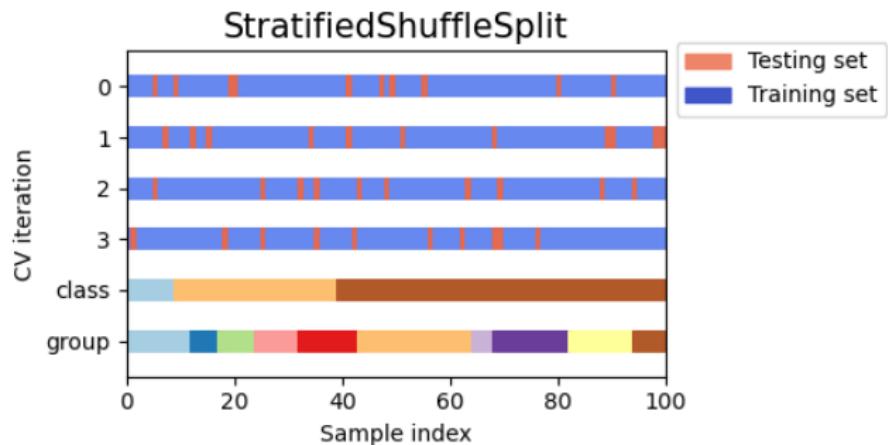
- Recommended with small datasets
- One of the
- Shuffle the
- Loop over
- fold  $k$
- At the end

### ● Stratified K-F

- K-Fold CV
- as the enti

### ● And yet other

- Exple Pyth



source: [Cho18] p. 99

# Hyper parameters

## The training Hyper-Parameters

- Have a determinant influence on:
  - the whole training process
  - the performance of the trained network
- Many hyper-parameters are involved in the training of a Neural Network
  - Some are *well known* and clearly visible in the computer code:
    - the `ratios` used to split the input Dataset into train/valid/test sets
    - `epochs` ~ the number of epochs
    - `batch_size` ~ the size of the mini-batch
    - `learning_rate` ~ the learning rate of the optimizer (SGD, ADAM...)
  - Some are less *highlighted* in published papers &tutos

# Hyper parameters

## The training Hyper-Parameters

- Have a determinant influence on:
  - the whole training process
  - the performance of the trained network
- **Many** hyper-parameters are involved in the training of a Neural Network
  - Some are *well known* and clearly visible in the computer code:
    - the `ratios` used to split the input Dataset into train/valid/test sets
    - `epochs` ~ the number of epochs
    - `batch_size` ~ the size of the mini-batch
    - `learning_rate` ~ the learning rate of the optimizer (SGD, ADAM...)
  - Some are less *highlighted* in published papers &tutos

# Hyper parameters

## The training Hyper-Parameters

- Have a determinant influence on:
  - the whole training process
  - the performance of the trained network
- **Many** hyper-parameters are involved in the training of a Neural Network
  - Some are *well known* and clearly visible in the computer code:
    - the **ratios** used to split the input Dataset into train/valid/test sets
    - **epochs** ~ the number of epochs
    - **batch\_size** ~ the size of the mini-batch
    - **learning\_rate** ~ the learning rate of the optimizer (SGD, ADAM...)
  - Some are less *highlighted* in published papers &tutos

# Hyper parameters

## The training Hyper-Parameters

- Have a determinant influence on:
  - the whole training process
  - the performance of the trained network
- **Many** hyper-parameters are involved in the training of a Neural Network
  - Some are *well known* and clearly visible in the computer code:
    - the **ratios** used to split the input Dataset into train/valid/test sets
    - **epochs** ~ the number of epochs
    - **batch\_size** ~ the size of the mini-batch
    - **learning\_rate** ~ the learning rate of the optimizer (SGD, ADAM...)
  - Some are less *highlighted* in published papers &tutos

# Hyper parameters

## The training Hyper-Parameters

- Have a determinant influence on:

- the whole training process
- the performance of the learning network

- Many hyper-parameters of a Neural Network

- Some are visible in the computer code:
  - the **ratio** of the dataset into train/valid/test sets
  - batch** size
  - learning rate** of the optimizer (SGD, ADAM...)

```
fit(  
    x=None,  
    y=None,  
    batch_size=None,  
    epochs=1,  
    verbose='auto',  
    callbacks=None,  
    validation_split=0.0,  
    validation_data=None,  
    shuffle=True,  
    class_weight=None,  
    sample_weight=None,  
    initial_epoch=0,  
    steps_per_epoch=None,  
    validation_steps=None,  
    validation_batch_size=None,  
    validation_freq=1  
)
```

tensorflow model fit

well known

and in the training of a Neural

visible in the computer code:

Dataset into train/valid/test sets

mini-batch

rate of the optimizer (SGD, ADAM...)

published papers &tutos

# Hyper parameters

## The training Hyper-Parameters

- Have a determinant influence on:
  - the whole training process
  - the performance of the trained network

- Many hyper-parameters** involved in the training of a Neural Network

- Some are
  - the **rate**
  - epoch**
  - batch**
  - learning\_rate**
- Some are

```
tf.keras.optimizers.Adam(
    learning_rate=0.001, well known
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=None,
    loss_scale_factor=None,
    gradient_accumulation_steps=None,
    name='adam',
    **kwargs
)
```

tensorflow optimizers Adam

visible in the computer code:

Dataset into train/valid/test sets

mini-batch

type of the optimizer (SGD, ADAM...)

published papers &tutos

# Hyper parameters

## The training Hyper-Parameters

- Have a determinant influence on:
  - the whole training process
  - the performance of the trained network
- Many hyper-parameters are involved in the training of a Neural Network
  - Some are
    - the **rate**
    - **epoch**
    - **batch**
    - **learning\_rate**
  - Some are

```
tf.keras.optimizers.SGD(  
    learning_rate=0.01, well known  
    momentum=0.0,  
    nesterov=False,  
    weight_decay=None,  
    clipnorm=None,  
    clipvalue=None,  
    global_clipnorm=None,  
    use_ema=False,  
    ema_momentum=0.99,  
    ema_overwrite_frequency=None,  
    loss_scale_factor=None,  
    gradient_accumulation_steps=None,  
    name='SGD',  
    **kwargs  
)
```

tensorflow optimizers SGD

visible in the computer code:

Dataset into train/valid/test sets

mini-batch

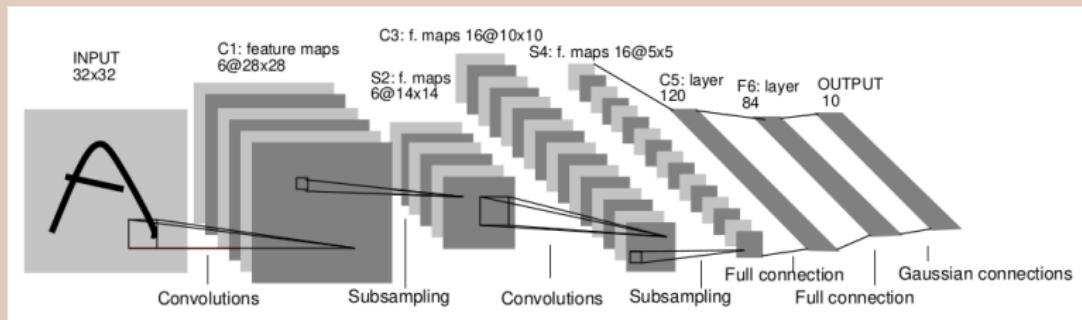
type of the optimizer (SGD, ADAM...)

published papers &tutos

# Hyper parameters

## The **batch\_size** hyper-parameter

- Exemple: MNISTdataset, CNN model LeNet5 build with SEED set [GitHub](#)



LeNet5 from LeCun

# Hyper parameters

## The `batch_size` hyper-parameter

- Example: MNIST dataset, CNN model LeNet5 build with SEED set
- 6 trainings with `batch_size` in (16, 32, 64, 128, 256, 512, 1024)

# Hyper parameters

## The `batch_size` hyper-parameter

- Example: MNIST dataset, CNN model LeNet5 build with SEED set
- 6 trainings with `batch_size` in (16, 32, 64, 128, 256, 512, 1024)

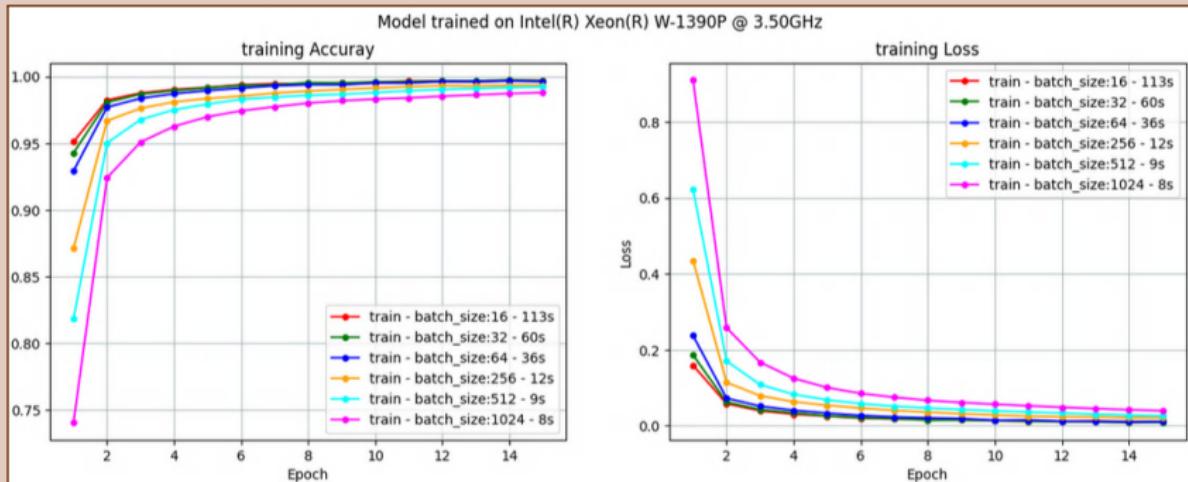
tensorflow/keras snippet:

```
nb_epoch = 15
for bs in (16, 32, 64, 256, 512, 1024):
    print(f'batch_size={bs}, {SEED=}')
    ...
    # Build a new model at each lopp lap, with seed set:
    model = build_CNN(IM_SHAPE, seed=SEED)

    hist = model.fit(x_train, y_train, # images, labels
                      epochs=nb_epoch, # the total number of successive trainings
                      batch_size=bs,   # split the train dataset in multiple batches
                      validation_data=(x_valid, y_valid),
                      verbose=0)
    ...
    ...
```

# Hyper parameters

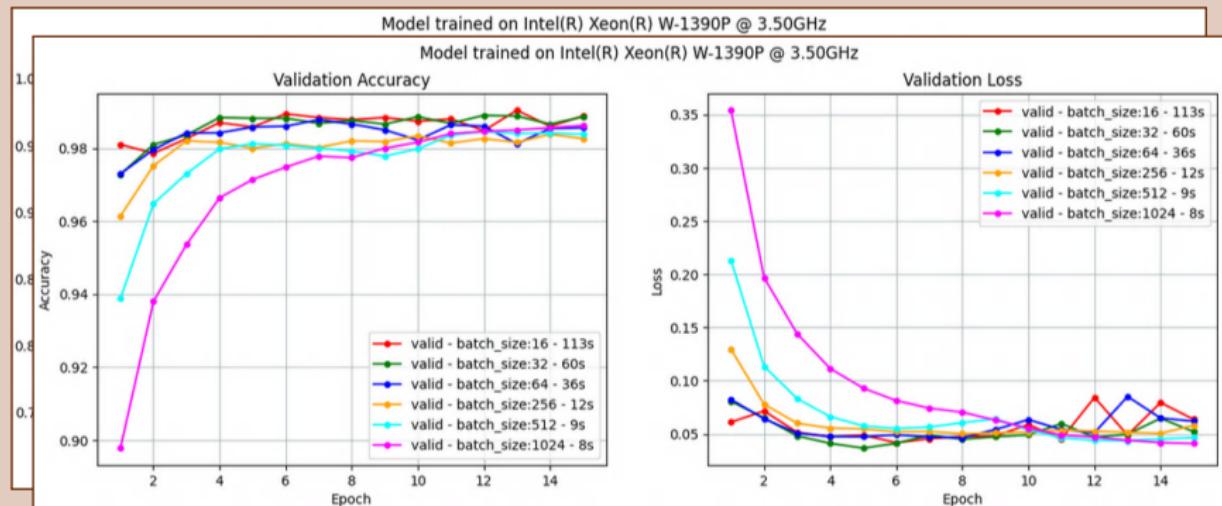
## The **batch\_size** hyper-parameter



- Training metrics (accuracy & loss)  $\leadsto$  right shape but not relevant.
- Validation metrics  $\leadsto$  interesting minimum loss for batch\_size=32 at epoch=5

# Hyper parameters

## The **batch\_size** hyper-parameter



- Training metrics (accuracy & loss)  $\sim$  right shape but not relevant.
- Validation metrics  $\sim$  interesting minimum loss for **batch\_size**=32 at **epoch**=5

# Hyper parameters

## The **patience** hyper-parameter in the **EarlyStopping** algorithm

- Given the previous Validation Loss curves, it should be wise to stop the training for `batch_size=32` at `epoch=5` because the validation loss increases.
- `tensorflow` & `PyTorch` offer such mechanism known as **Early Stopping**

# Hyper parameters

## The **patience** hyper-parameter in the **EarlyStopping** algorithm

- Given the previous Validation Loss curves, it should be wise to stop the training for `batch_size=32` at `epoch=5` because the validation loss increases.
- `tensorflow` & `PyTorch` offer such mechanism known as **Early Stopping**

`tensorflow/keras` snippet: using `EarlyStopping` *callback* to stop the training

```
for bs in (16, 32, 64, 256, 512, 1024):
    ...
    # Build a new model at each lopp lap, with seed set:
    model = build_CNN(IM_SHAPE, seed=SEED)
    callbacks_list = [
        EarlyStopping(monitor='val_loss', # The parameter to monitor
                      patience=1,           # accept that 'val_loss' increases 1 time
                      restore_best_weights=True, # important !!!
                      verbose=1) ]
    hist = model.fit(x_train, y_train, # images, labels
                      epochs=nb_epoch, # the total number of successive trainings
                      batch_size=bs,   # split the train dataset in multiple batches
                      validation_data=(x_valid, y_valid),
                      verbose=0,
                      callbacks = callbacks_list)
```

# Hyper parameters

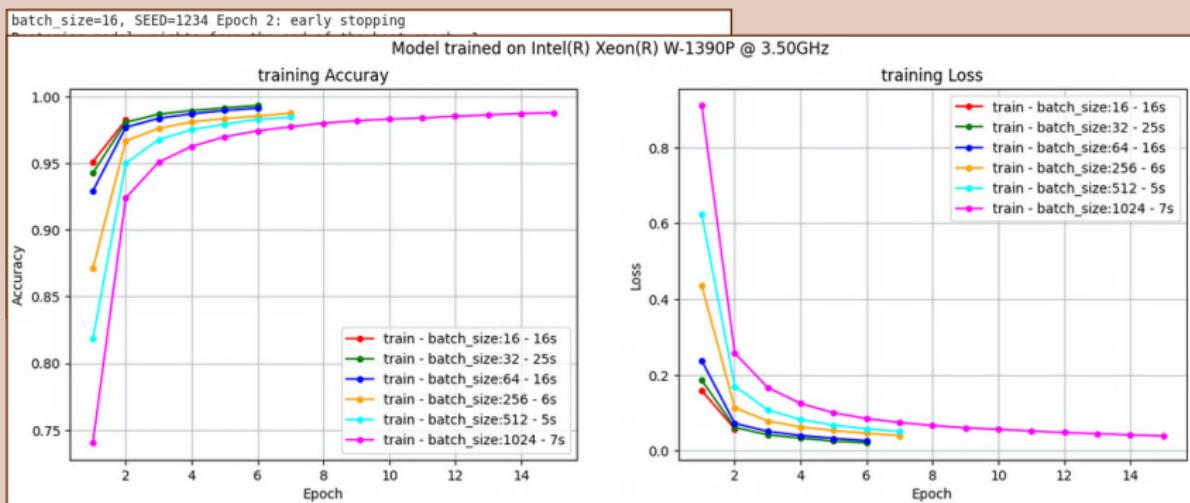
## The **patience** hyper-parameter in the **EarlyStopping** algorithm

```
batch_size=16, SEED=1234 Epoch 2: early stopping
Restoring model weights from the end of the best epoch: 1.
Train Elapsed time 17s -> 00:00:17
batch_size=32, SEED=1234 Epoch 6: early stopping
Restoring model weights from the end of the best epoch: 5. ←
Train Elapsed time 27s -> 00:00:27
batch_size=64, SEED=1234 Epoch 6: early stopping
Restoring model weights from the end of the best epoch: 5.
Train Elapsed time 15s -> 00:00:15
batch_size=256, SEED=1234 Epoch 7: early stopping
Restoring model weights from the end of the best epoch: 6.
Train Elapsed time 6s -> 00:00:06
batch_size=512, SEED=1234 Epoch 7: early stopping
Restoring model weights from the end of the best epoch: 6.
Train Elapsed time 5s -> 00:00:05
batch_size=1024, SEED=1234 Restoring model weights from the end of the best epoch: 15.
Train Elapsed time 7s -> 00:00:07
```

- Training for **batch\_size=32** stopped at **epoch=6** (model is saved for **epoch=5**)

# Hyper parameters

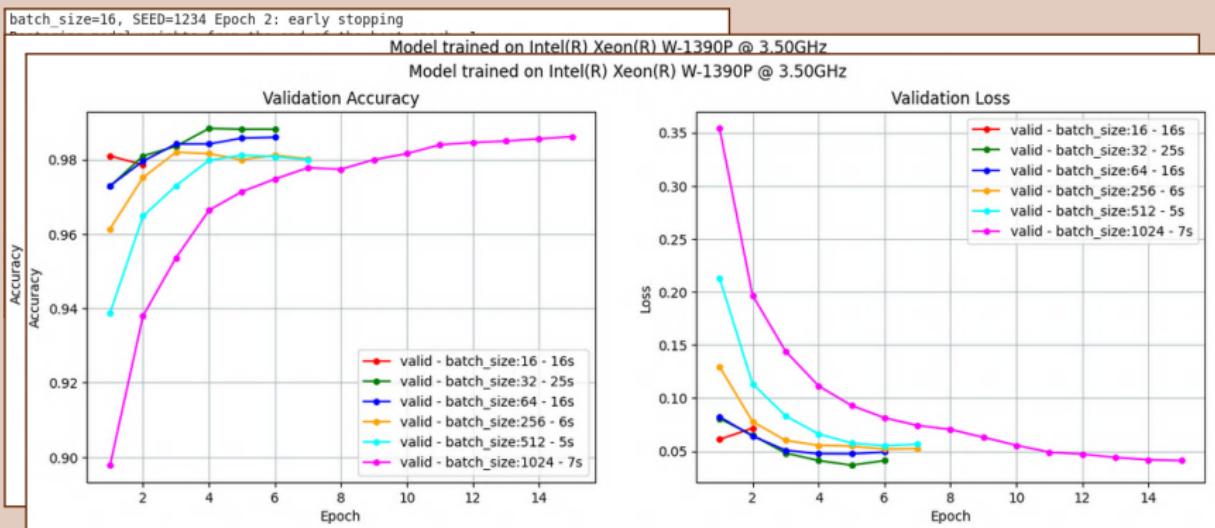
## The **patience** hyper-parameter in the **EarlyStopping** algorithm



- Training for **batch\_size**=32 stopped at **epoch**=6 (model is saved for **epoch**=5)

# Hyper parameters

## The **patience** hyper-parameter in the **EarlyStopping** algorithm



- Training for `batch_size=32` stopped at `epoch=6` (model is saved for `epoch=5`)

# Hyper parameters

## The **patience** hyper-parameter in the **EarlyStopping** algorithm

```
batch_size: 16 time:16s test acc: 0.981 test_loss:0.064
batch_size: 32 time:25s test acc: 0.990 test loss:0.035
batch_size: 64 time:16s test_acc: 0.986 test_loss:0.044
batch_size: 256 time:6s test_acc: 0.983 test_loss:0.050
batch_size: 512 time:5s test_acc: 0.981 test_loss:0.054
batch_size:1024 time:7s test_acc: 0.987 test_loss:0.040
```

- The model performance for **batch\_size**=32 at **epoch**=5 has the best score when tested with the **test Dataset**.
- The training elapsed time is now only 25 sec.
- The whole notebook is reproducible.

# Hyper parameters

## The **patience** hyper-parameter in the **EarlyStopping** algorithm

```
batch_size: 16 time:16s test acc: 0.981 test_loss:0.064
batch_size: 32 time:25s test acc: 0.990 test loss:0.035
batch_size: 64 time:16s test_acc: 0.986 test_loss:0.044
batch_size: 256 time:6s test_acc: 0.983 test_loss:0.050
batch_size: 512 time:5s test_acc: 0.981 test_loss:0.054
batch_size:1024 time:7s test_acc: 0.987 test_loss:0.040
```

- The model performance for **batch\_size**=32 at **epoch**=5 has the best score when tested with the **test Dataset**.
- The training elapsed time is now only 25 sec.
- The whole notebook is reproducible.

# Hyper parameters

## The **patience** hyper-parameter in the **EarlyStopping** algorithm

```
batch_size: 16 time:16s test acc: 0.981 test_loss:0.064
batch_size: 32 time:25s test acc: 0.990 test loss:0.035
batch_size: 64 time:16s test_acc: 0.986 test_loss:0.044
batch_size: 256 time:6s test_acc: 0.983 test_loss:0.050
batch_size: 512 time:5s test_acc: 0.981 test_loss:0.054
batch_size:1024 time:7s test_acc: 0.987 test_loss:0.040
```

- The model performance for **batch\_size**=32 at **epoch**=5 has the best score when tested with the **test Dataset**.
- The training elapsed time is now only 25 sec.
- The whole notebook is reproducible.

# Sharing reproducible DL training

## Use a **Python Virtual Environment** (PVE)

- A PVE defines a dedicated file tree where to store the Python modules & tools of a DL project
- Ensures that different projects do not interfere with each other
- Isolate project dependencies
- Protects the Python modules from system updates or from modules updates in other projects
- Provides tools to easily delete/duplicate/reproduce it
- Allows efficient sharing
- Use a dedicated PVE for each DL project, even if small project

# Sharing reproducible DL training

## Use a **Python Virtual Environment** (PVE)

- A PVE defines a dedicated file tree where to store the Python modules & tools of a DL project
- Ensures that different projects do not interfere with each other
- Isolate project dependencies
- Protects the Python modules from system updates or from modules updates in other projects
- Provides tools to easily delete/duplicate/reproduce it
- Allows efficient sharing
- Use a dedicated PVE for each DL project, even if small project

# Sharing reproducible DL training

## Use a **Python Virtual Environment** (PVE)

- A PVE defines a dedicated file tree where to store the Python modules & tools of a DL project
- Ensures that different projects do not interfere with each other
- Isolate project dependencies
- Protects the Python modules from system updates or from modules updates in other projects
- Provides tools to easily delete/duplicate/reproduce it
- Allows efficient sharing
- Use a dedicated PVE for each DL project, even if small project

# Sharing reproducible DL training

## Use a **Python Virtual Environment** (PVE)

- A PVE defines a dedicated file tree where to store the Python modules & tools of a DL project
- Ensures that different projects do not interfere with each other
- Isolate project dependencies
- Protects the Python modules from system updates or from modules updates in other projects
- Provides tools to easily delete/duplicate/reproduce it
- Allows efficient sharing
- Use a dedicated PVE for each DL project, even if small project

# Sharing reproducible DL training

## Use a **Python Virtual Environment** (PVE)

- A PVE defines a dedicated file tree where to store the Python modules & tools of a DL project
- Ensures that different projects do not interfere with each other
- Isolate project dependencies
- Protects the Python modules from system updates or from modules updates in other projects
- Provides tools to easily delete/duplicate/reproduce it
- Allows efficient sharing
- Use a dedicated PVE for each DL project, even if small project

# Sharing reproducible DL training

## Use a **Python Virtual Environment** (PVE)

- A PVE defines a dedicated file tree where to store the Python modules & tools of a DL project
- Ensures that different projects do not interfere with each other
- Isolate project dependencies
- Protects the Python modules from system updates or from modules updates in other projects
- Provides tools to easily delete/duplicate/reproduce it
- Allows efficient sharing
- Use a dedicated PVE for each DL project, even if small project

# Sharing reproducible DL training

## Use a **Python Virtual Environment** (PVE)

- A PVE defines a dedicated file tree where to store the Python modules & tools of a DL project
- Ensures that different projects do not interfere with each other
- Isolate project dependencies
- Protects the Python modules from system updates or from modules updates in other projects
- Provides tools to easily delete/duplicate/reproduce it
- Allows efficient sharing
- Use a dedicated PVE for each DL project, even if small project

# Sharing reproducible DL training

Use a tool to manage your DL projects & PVEs

- Minimalist: `venv`, `virtualenv`...
- More efficient: `conda`, `poetry`...
- And more recently (more powerful) : `uv`

# Sharing reproducible DL training

## Use a tool to manage your DL projects & PVEs

- Minimalist: `venv`, `virtualenv`...
- More efficient: `conda`, `poetry`...
- And more recently (more powerful) : `uv`
  - From the Web page of `uv`:
  - A single tool to replace pip, pipx, poetry, pyenv, virtualenv...
  - 10-100x faster than pip !
  - Provides comprehensive project management, with a universal lockfile.
  - Installs and manages Python versions.
  - Includes a pip-compatible interface for a performance boost with a familiar CLI.
  - Disk-space efficient, with a global cache for dependency deduplication.
  - Supports macOS, Linux, and Windows....

# Sharing reproducible DL training

## Use an **Integrated Development Environment (IDE) aware of PVE**

- VSCode
- PyCharm
- Atom
- PyDev
- Jupyter notebook
- Jupyter Lab
- ...

# Sharing reproducible DL training

## Use a **versionning** tool

- To keep track of **code** and **data** versions
- To share **code** and **data**
- Example: **git** repository for code and data ( < 5 GB)
  - Maintain the repository !
  - If your data files are too big, use the **Git Large File** (GLF) extension to version the data with Git.

Thank you for your attention.

# Bibliography I

## Deep Learning (DL)

- [Cho18] F. Cholet, “Deep learning in python,” MANNING, Shelter Island. Downloadable PDF. (2018), <https://archive.org/download/deep-learning-collection-pdf/Deep%20Learning%20with%20Python%20-%20Fran%C3%A7ois%20Chollet%20-%20Manning%20%282018%29.pdf> (cit. on pp. 105–111).
- [Cho22] F. Cholet, “Machine learning articles,” Articles with tensorflow/pytorch code examples. (2022), <https://github.com/christianversloot/machine-learning-articles/blob/main/README.md>.
- [He+21] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, “Masked autoencoders are scalable vision learners,” (2021), <https://arxiv.org/abs/2111.06377>.
- [OM98] G. Orr and K.-R. Müller, “Neural networks : Tricks of the trade,” Published by Springer Berlin Heidelberg, ISBN: 978-3-540-65311-0. (1998), [https://archive.org/details/springer\\_10.1007-3-540-49430-8/mode/2up](https://archive.org/details/springer_10.1007-3-540-49430-8/mode/2up).
- [SK19] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” Journal of Big Data, volume 6, Article number: 60 (2019). (2019), <https://doi.org/10.1186/s40537-019-0197-0> (cit. on pp. 67–71).
- [Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” Journal of Machine Learning Research 15 (2014) 1929–1958. (2014), <https://dl.acm.org/doi/abs/10.5555/2627435.2670313> (cit. on pp. 67–71).

## Reproducibility

# Bibliography II

- [Alb+23] R. Albertoni, S. Colantonio, P. Skrzypczyński, and J. Stefanowski, “Reproducibility of machine learning: Terminology, recommendations and open issues,” (2023), <https://arxiv.org/pdf/2302.12691.pdf> (cit. on pp. 55–60).
- [Ami24] H. Amit, “Pytorch reproducibility: A practical guide,” (2024), <https://medium.com/@heyamit10/pytorch-reproducibility-a-practical-guide-d6f573cba679>.
- [Bak16] M. Baker, “1,500 scientists lift the lid on reproducibility,” *Nature* 533, 452–454. (2016), <https://www.nature.com/articles/533452a#Sec4> (cit. on pp. 55–60).
- [Che+22] B. Chen, M. Wen, Y. Shi, D. Lin, G. K. Rajbahadur, and Z. M. ( Jiang, “Towards training reproducible deep learning models,” 44th International Conference on Software Engineering (ICSE ’22), May 21–29, 2022, Pittsburgh, PA, USA. ACM, New York, NY, USA. (2022), <https://arxiv.org/abs/2202.02326>.
- [eal20] B. H.-K. et al., “Transparency and reproducibility in artificial intelligence,” (2020), [https://cris.maastrichtuniversity.nl/files/75369807/Aerts\\_2020\\_Transparency\\_and\\_reproducibility\\_in\\_artificial.pdf](https://cris.maastrichtuniversity.nl/files/75369807/Aerts_2020_Transparency_and_reproducibility_in_artificial.pdf) (cit. on pp. 55–60).
- [GK18] O. E. Gundersen and S. Kjensmo, “State of the art: Reproducibility in artificial intelligence,” Vol. 32 No. 1 (2018): Thirty-Second AAAI Conference on Artificial Intelligence. (2018), <https://doi.org/10.1609/aaai.v32i1.11503> (cit. on pp. 55–60).
- [Hut18] M. Hutson, “Artificial intelligence faces reproducibility crisis,” *Science*, 2018, Vol 359, Issue 6377, pp. 725-726, PDF:[www.researchgate.net](http://www.researchgate.net). (2018), <https://www.science.org/doi/10.1126/science.359.6377.725> (cit. on pp. 55–60).

# Bibliography III

- [Pha+20] H. V. Pham *et al.*, “Problems and opportunities in training deep learning software systems: An analysis of variance,” ASE ’20, September 21–25, Virtual Event, Australia. (2020), <https://par.nsf.gov/servlets/purl/10252876>.
- [Pin+20] J. Pineau *et al.*, “Improving reproducibility in machine learning research,” (2020), <https://arxiv.org/abs/2003.12206> (cit. on pp. 55–60).
- [Pin18] J. Pineau, “Reproducible, reusable, and robust reinforcement learning,” Neural Information Processing Systems (NeurIPS) December 5, 2018. (2018), <https://media.neurips.cc/Conferences/NIPS2018/Slides/jpineau-NeurIPS-dec18-fb.pdf>.
- [PyT24] PyTorch, “Use deterministic algorithms,” (2024), [https://pytorch.org/docs/stable/generated/torch.use\\_deterministic\\_algorithms.html](https://pytorch.org/docs/stable/generated/torch.use_deterministic_algorithms.html) (cit. on pp. 72–77).
- [Pyt24] Pytorch, “Reproducibility,” (2024), <https://pytorch.org/docs/stable/notes/randomness.html#reproducibility> (cit. on pp. 72–77).
- [Ria19] D. Riach, “Determinism in deep learning (s9911),” GPU Technologie Conference (GTC) 2019 talk. (2019), <https://developer.nvidia.com/gtc/2019/video/s9911>.
- [Ria24] D. Riach, “Framework reproducibility: Determinism (d9m),” (2024), <https://github.com/NVIDIA/framework-reproducibility/blob/master/doc/d9m/README.md>.
- [Sha19] C. Shao, “Properly setting the random seed in ml experiments. not as simple as you might imagine,” (2019), <https://opendatascience.com/properly-setting-the-random-seed-in-ml-experiments-not-as-simple-as-you-might-imagine/> (cit. on pp. 55–60).

# Bibliography IV

- [ten24a] tensorflow, “Enable op determinism,” (2024), [https://www.tensorflow.org/api\\_docs/python/tf/config/experimental/enable\\_op\\_determinism](https://www.tensorflow.org/api_docs/python/tf/config/experimental/enable_op_determinism) (cit. on pp. 72–77).
- [ten24b] tensorflow, “Set random seed,” (2024), [https://www.tensorflow.org/api\\_docs/python/tf/keras/utils/set\\_random\\_seed](https://www.tensorflow.org/api_docs/python/tf/keras/utils/set_random_seed) (cit. on pp. 72–77).
- [ten24c] tensorflow, “What’s new in tensorflow 2.9?” (2024), <https://blog.tensorflow.org/2022/05/whats-new-in-tensorflow-29.html> (cit. on pp. 72–77).
- [TVD18] R. Tatman, J. VanderPlas, and S. Dane, “A practical taxonomy of reproducibility for machine learning research,” Reproducibility in Machine Learning Workshop at ICML 2018, Stockholm, Sweden. (2018), <https://openreview.net/pdf?id=B1eYYK5QgX> (cit. on pp. 55–60).

## Self-Supervised Learning (SSL)

- [Bal+23] R. Balestrieri *et al.*, “A cookbook of self-supervised learning,” (2023), <https://arxiv.org/abs/2304.12210>.
- [Gui+24] J. Gui *et al.*, “A survey on self-supervised learning: Algorithms, applications, and future trends,” (2024), <https://arxiv.org/abs/2301.05712>.
- [LM23] Y. LeCun and I. Misra, “Self-supervised learning: The dark matter of intelligence,” (2023), <https://arxiv.org/abs/2304.12210> (cit. on pp. 22–25).
- [Mei23] B. Meinardus, “What is self-supervised learning and why should you care?” (2023), <https://medium.com/@boris.meinardus/what-is-self-supervised-learning-and-why-should-you-care-1c6dfc0e0f57>.

# Bibliography V

## Explainable Artificial Intelligence (XAI)

- [Ali+23] S. Ali *et al.*, “Explainable artificial intelligence (xai): What we know and what is left to attain trustworthy artificial intelligence,” *Information Fusion, Volume 99, November 2023, 101805A Preprint.* (2023), <https://www.sciencedirect.com/science/article/pii/S1566253523001148> (cit. on pp. 39, 40).

- [MM25] F. Mumuni and A. Mumuni, “Explainable artificial intelligence (xai): From inherent explainability to large language models,” *A Preprint.* (2025), <https://arxiv.org/abs/2501.09967> (cit. on pp. 39, 40).

## Interpretable Machine learning (IML)

- [MCB20] C. Molnar, G. Casalicchio, and B. Bischl, “Interpretable machine learning – a brief history, state-of-the-art and challenges,” Submitted on 19 Oct 2020, published by ECML PKDD 2020 Workshops. ECML PKDD 2020. Communications in Computer and Information Science, vol 1323. Springer, Cham. (2020), <https://arxiv.org/abs/2010.09337> (cit. on pp. 39, 40).

- [Mol24] C. Molnar, “Interpretable machine learning. a guide for making black box models explainable,” book on line. (2024), <https://christophm.github.io/interpretable-ml-book/> (cit. on pp. 39, 40).

- [Rud+21] C. Rudin, C. Chen2, Z. Chen1, H. Huang1, L. Semenova1, and C. Zhong1, “Interpretable machine learning: Fundamental principles and 10 grand challenges,” *Statistics Surveys, 2021 [last revised 10 Jul 2021 (this version, v2)].* (2021), <https://arxiv.org/abs/2103.11251> (cit. on pp. 39, 40).

## Misc programation tips (IML)

# Bibliography VI

- [Cha24] A. Chawla, "A detailed and beginner-friendly introduction to pytorch lightning: The supercharged pytorch," (2024), <https://www.dailydoseofds.com/a-detailed-and-beginner-friendly-introduction-to-pytorch-lightning-the-supercharged-pytorch>.
- [PyT24] PyTorch, "Use deterministic algorithms," (2024), [https://pytorch.org/docs/stable/generated/torch.use\\_deterministic\\_algorithms.html](https://pytorch.org/docs/stable/generated/torch.use_deterministic_algorithms.html) (cit. on pp. 72–77).
- [Pyt24] Pytorch, "Reproducibility," (2024), <https://pytorch.org/docs/stable/notes/randomness.html#reproducibility> (cit. on pp. 72–77).
- [Ria19] D. Riach, "Determinism in deep learning (s9911)," GPU Technologie Conference (GTC) 2019 talk. (2019), <https://developer.nvidia.com/gtc/2019/video/s9911>.
- [Ria24] D. Riach, "Framework reproducibility: Determinism (d9m)," (2024), <https://github.com/NVIDIA/framework-reproducibility/blob/master/doc/d9m/README.md>.
- [sldev25] scikit-learn developers, "Cross-validation: Evaluating estimator performance," (2025), [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html).
- [ten24b] tensorflow, "Set random seed," (2024), [https://www.tensorflow.org/api\\_docs/python/tf/keras/utils/set\\_random\\_seed](https://www.tensorflow.org/api_docs/python/tf/keras/utils/set_random_seed) (cit. on pp. 72–77).
- [ten24c] tensorflow, "What's new in tensorflow 2.9?" (2024), <https://blog.tensorflow.org/2022/05/whats-new-in-tensorflow-29.html> (cit. on pp. 72–77).