

# Café IA

## Feedback on Machine Learning [Reproducibility of trainings]

I2M / Groupe de Travail - IA / June 26, 2023

Jean-Luc.Charles@ENSA.M.EU



# Feedback on Machine Learning



## Some points of interest

- [If needed...] Preliminaries: the historical way...
- Unsupervised / Supervised / Reinforcement learning
- Use a Virtual Python Environment (PVE) to work on ML...
- The repeatability of trainings: the key to compare trainings.

# Feedback on Machine Learning



## Some points of interest

- [If needed...] Preliminaries: the historical way...
- Unsupervised / Supervised / Reinforcement learning
- Use a Virtual Python Environment (PVE) to work on ML...
- The repeatability of trainings: the key to compare trainings.

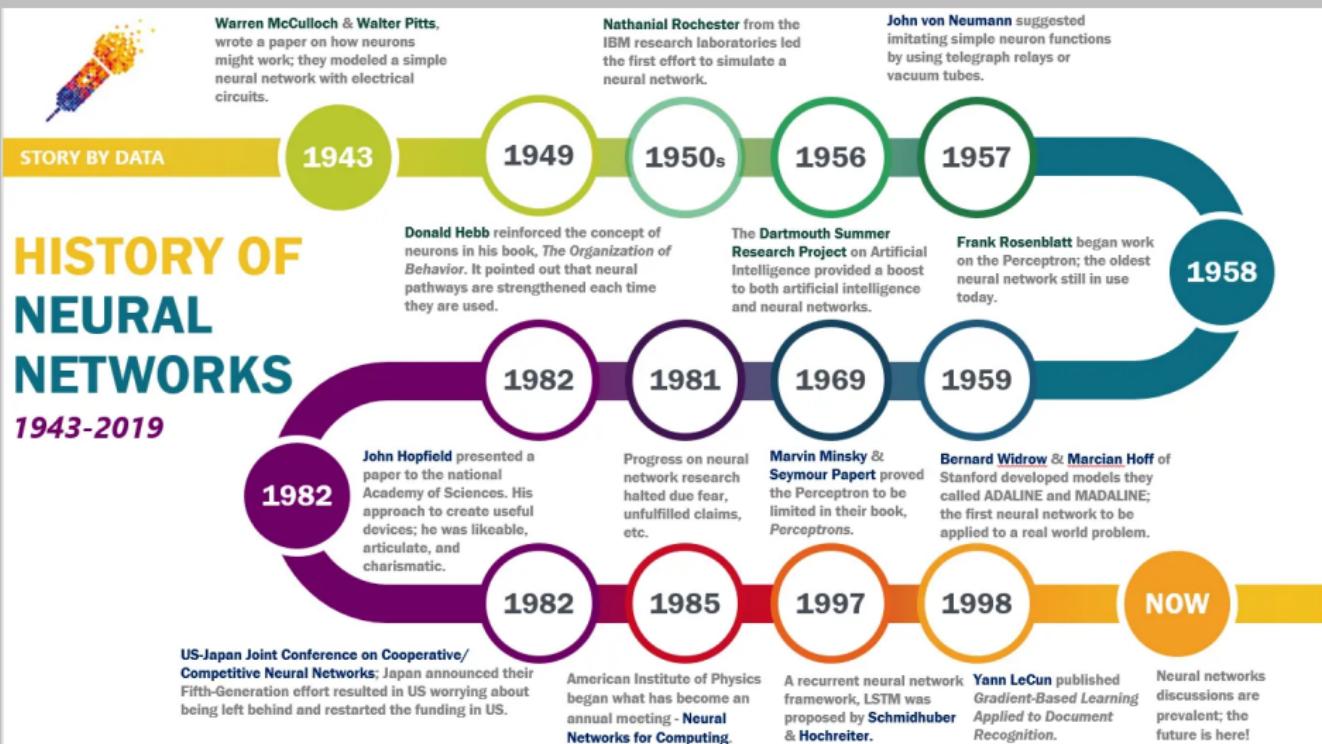


## My profile

- I'm teaching Python programming (Scientific & Object Oriented)
- I started to get interested in ML in 2015
- I wrote several materials in ML : workshop, project & practical work for different schools (ENSAM, ENSEIRB, ENSPIMA, PPU...)

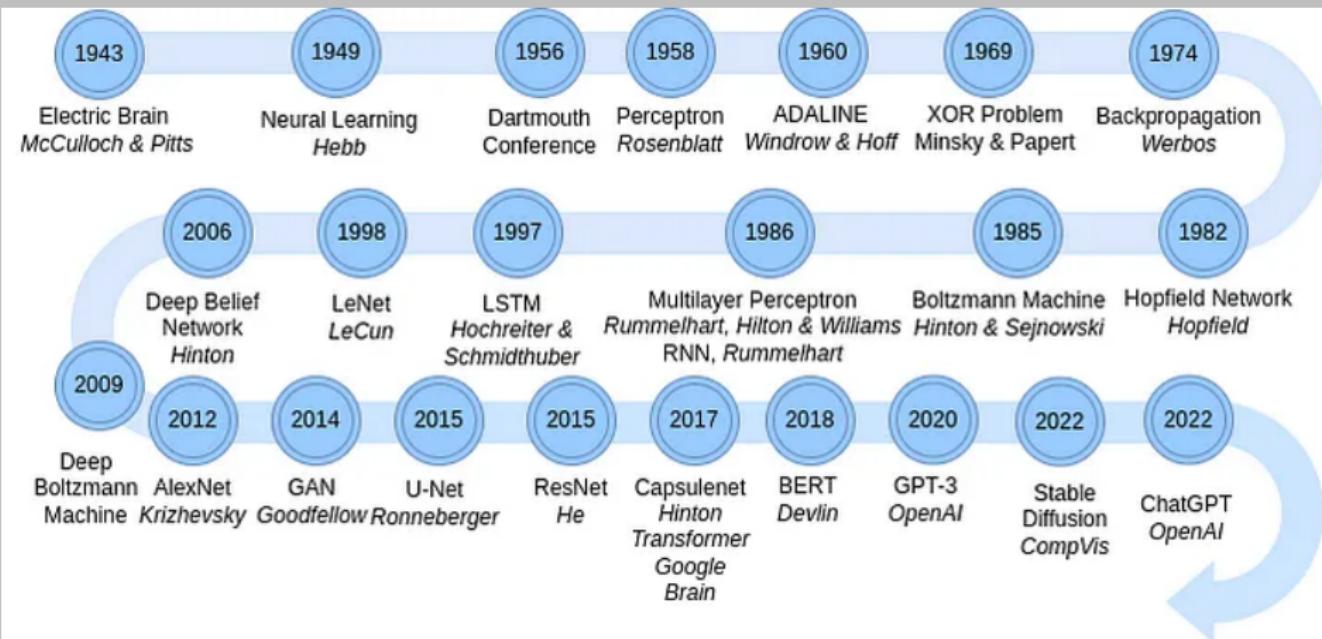
# The historical way...

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



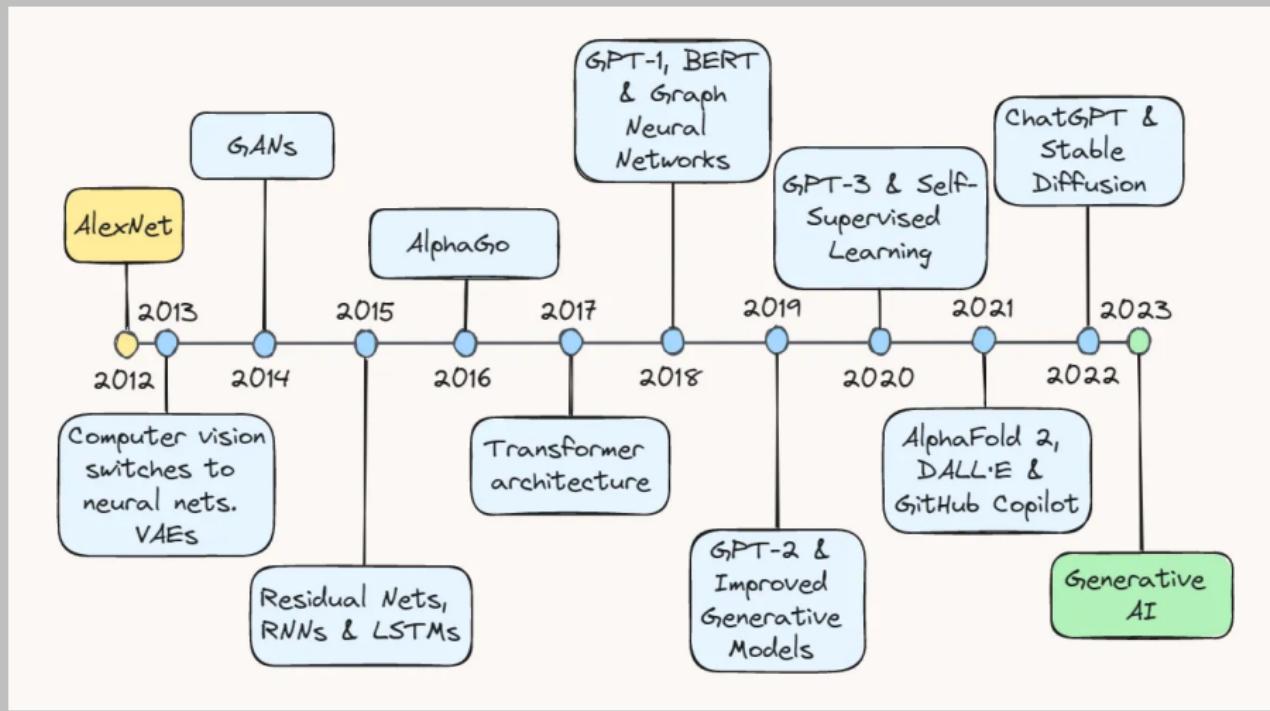
# The historical way...

from: Pumalin: "A Brief History of Neural Nets", medium.com



# The historical way...

from: Thomas A Dorfe: "Ten Years of AI in Review", medium.com



# Artificial Intelligence ?



Historically<sup>a</sup> *badly chosen term*

Ambiguous current meaning

Many (contradictory) definitions depending on periods and authors...

---

<sup>a</sup>first used in 1956 by [John McCarthy](#), researcher at Stanford during the Dartmouth conference

- "...the science of making computers do things that require intelligence when done by humans." [Alan Turing, 1940](#)
- "the field of study that gives computers the ability to learn without being explicitly programmed." [Arthur Samuel, 1960](#)
- "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [Tom Mitchell, 1997](#)
- Notion of *intelligent agent* or *rational agent*  
"...agent that acts in such a way as to reach the best solution or, in an uncertain environment, the best predictable solution."

# Artificial Intelligences ?

**Strong AI**

**Weak AI**

**General AI**

**Narrow AI**

# Artificial Intelligences ?

## Strong AI

- Build systems that think exactly the same way that people do.
- Try also to explain how humans think... **Whe are not yet here.**

## Weak AI

## General AI

## Narrow AI

# Artificial Intelligences ?

## Strong AI

- Build systems that think exactly the same way that people do.
- Try also to explain how humans think... **Who are not yet here.**

## Weak AI

- Build systems that can behave like humans.
- The results will tell us nothing about how humans think.
- **We already are there...** We use it every day!  
(anti-spam, facial/voice recognition, language translation...)

## General AI

## Narrow AI

# Artificial Intelligences ?

## Strong AI

- Build systems that think exactly the same way that people do.
- Try also to explain how humans think... **Who are not yet here.**

## Weak AI

- Build systems that can behave like humans.
- The results will tell us nothing about how humans think.
- **We already are there...** We use it every day!  
(anti-spam, facial/voice recognition, language translation...)

## General AI

- AI systems designed for the ability to reason in general.

## Narrow AI

- AI systems designed for specific tasks.

# Branches of Machine Learning

Page from [medium.com/machine-learning-for-humans/...](https://medium.com/machine-learning-for-humans/)

Machine learning  $\subseteq$  artificial intelligence

## ARTIFICIAL INTELLIGENCE

Design an intelligent agent that perceives its environment and makes decisions to maximize chances of achieving its goal.

Subfields: vision, robotics, machine learning, natural language processing, planning, ...

## MACHINE LEARNING

Gives "computers the ability to learn without being explicitly programmed" (Arthur Samuel, 1959)

### SUPERVISED LEARNING

Classification, regression

### UNSUPERVISED LEARNING

Clustering, dimensionality reduction, recommendation

### REINFORCEMENT LEARNING

Reward maximization

# Branches of Machine Learning

## Supervised learning applications

**labeled dataset** is used to train algorithms to classify data:

- **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 application

Analyze and cluster **unlabeled datasets**:

- **Clustering & Grouping**

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

- **Anomaly Detection**

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

- **Dimensionality reduction**

- Compress data using fewer numbers...

...

# Branches of Machine Learning

## Reinforcement learning

An agent learns how to drive an environment by maximising a **reward**:

- **Control/command**

- Controlling robots, drones...
- Factory optimization
- Financial (stock) trading...

- **Decision making**

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

...

# Various approaches for ML algorithms

## Supervised learning:

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

## Unsupervised learning:

- Neural Networks
- Principal Composant Analysis
- Singular Value Decomposition
- K-mean & Prob. clustering...

## Reinforcement learning:

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

# Various approaches for ML algorithms

## Supervised learning:

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

## Unsupervised learning:

- **Neural Networks**
- Principal Composant Analysis
- Singular Value Decomposition
- K-mean & Prob. clustering...

## Reinforcement learning:

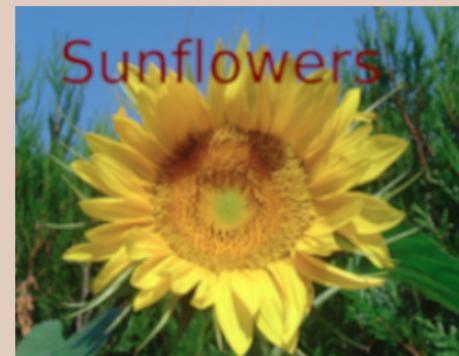
- **Neural Networks**(Q-learning, Actor-Critic, DDPG, PPO...)
- Monte Carlo
- SARSA

The following deals only with **Artificial Neural Networks**.

# Fields of ML

## Computer Vision

- Image Classification
- Object Detection
- (Semantic) Segmentation
- Image Generation  
Les 10 Meilleurs Génératrices d'Images
- Pose Estimation
- ...



# Fields of ML

## Computer Vision

- Image Classification
- Object Detection
- (Semantic) Segmentation
- Image Generation  
Les 10 Meilleurs Générateurs d'Images
- Pose Estimation
- ...

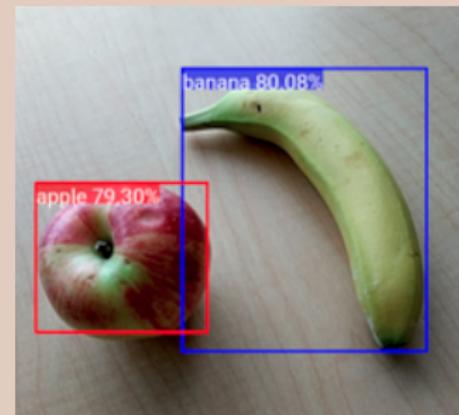


Image credit: [Tensorflow](#)

# Fields of ML

## Computer Vision

- Image Classification
- Object Detection
- (Semantic) Segmentation
- Image Generation  
Les 10 Meilleurs Génératrices d'Images
- Pose Estimation
- ...



# Fields of ML

## Computer Vision

- Image Classification
- Object Detection
- (Semantic) Segmentation
- Image Generation  
[Les 10 Meilleurs Générateurs d'Images](#)
- Pose Estimation
- ...



Image credit: [stylegan](#)

# Fields of ML

## Computer Vision

- Image Classification
- Object Detection
- (Semantic) Segmentation
- Image Generation  
[Les 10 Meilleurs Générateurs d'Images](#)
- Pose Estimation
- ...



Image credit: [Tensorflow-Pose Estimation](#)

# Fields of ML

## Natural Language Processing: NLP

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

# Neural Network Architectures

Common NN architectures:

- **Feed Forward**: the simplest NN made of successive layers of neurones, with *Feed Forward* and *Back Propagation* algorithms.
- **Convolutional** (CNN): Mostly used for analyzing and classifying images.
- **Recurrent** (RNN): Used to learn from time series, like the Long Short-Term Memory (LSTM) algorithm.
- **Transformers** : Recently used for Natural Language Processing and then for image classification.
- **Auto Encoder** (AEN): Dimensionality reduction, Feature extraction, Denoising of data/images, Imputing missing data.
- **Generative Adversarial** (GAN): to generate text, images, music...

Graphical chart: [from Saul Dobilas on Medium](#)

# The need for PVE (Python Virtual Environment) to program NN training with Python

# The need for PVE (Python Virtual Environment) to program NN training with Python

## Advantages

- Dedicated environment (disk tree) with fixed version of the Python interpreter and modules
- Easy to create and destroy if needed
- Each Python project can have its own PVE
  
- PVEs protect your projects against operating system updates or hazardous manipulations...

# The need for PVE (Python Virtual Environment) to program NN training with Python

## Advantages

- Dedicated environment (disk tree) with fixed version of the Python interpreter and modules
- Easy to create and destroy if needed
- Each Python project can have its own PVE
  
- PVEs protect your projects against operating system updates or hazardous manipulations...

# The need for PVE (Python Virtual Environment) to program NN training with Python

## Advantages

- Dedicated environment (disk tree) with fixed version of the Python interpreter and modules
  - Easy to create and destroy if needed
  - Each Python project can have its own PVE
- 
- PVEs protect your projects against operating system updates or hazardous manipulations...

# The need for PVE (Python Virtual Environment) to program NN training with Python

## Advantages

- Dedicated environment (disk tree) with fixed version of the Python interpreter and modules
- Easy to create and destroy if needed
- Each Python project can have its own PVE
  - ~ you can load/update modules for a project without breaking other projects
- PVEs protect your projects against operating system updates or hazardous manipulations...

# The need for PVE (Python Virtual Environment) to program NN training with Python

## Advantages

- Dedicated environment (disk tree) with fixed version of the Python interpreter and modules
- Easy to create and destroy if needed
- Each Python project can have its own PVE
  - ~ you can load/update modules for a project without breaking other projects
- PVEs protect your projects against operating system updates or hazardous manipulations...

# The need for PVE (Python Virtual Environment) to program NN training with Python

## Advantages

- Dedicated environment (disk tree) with fixed version of the Python interpreter and modules
- Easy to create and destroy if needed
- Each Python project can have its own PVE
  - ~ you can load/update modules for a project without breaking other projects
- PVEs protect your projects against operating system updates or hazardous manipulations...

## Disadvantages

- ? (just do it)

# PVE: simple to create & use

## with conda

- Download & install [miniconda](#)
- `conda create -n pyml python=3.8` ~ creates the `pyml` PVE
- `conda activate pyml` ~ activates the `pyml` PVE
- `conda install <module>` ~ installs a module  
`pip install <module>`

# PVE: simple to create & use

## with conda

- Download & install [miniconda](#)
- `conda create -n pyml python=3.8` ↪ creates the **pyml** PVE
- `conda activate pyml` ↪ activates the **pyml** PVE
- `conda install <module>` ↪ installs a module  
`pip install <module>`

# PVE: simple to create & use

## with conda

- Download & install [miniconda](#)
- `conda create -n pyml python=3.8` ~ creates the **pyml** PVE
- `conda activate pyml` ~ activates the **pyml** PVE
- `conda install <module>` ~ installs a module  
`pip install <module>`

# PVE: simple to create & use

## with conda

- Download & install [miniconda](#)
- `conda create -n pyml python=3.8` ~ creates the **pyml** PVE
- `conda activate pyml` ~ activates the **pyml** PVE
- `conda install <module>` ~ installs a module  
`pip install <module>`

# PVE: simple to create & use

## with conda

- Download & install [miniconda](#)
- `conda create -n pyml python=3.8` ~ creates the **pyml** PVE
- `conda activate pyml` ~ activates the **pyml** PVE
- `conda install <module>` ~ installs a module  
`pip install <module>`

## advantages / disadvantages

- 😊 The version of Python in a conda PVE can be  $\neq$  version of the interpreter that comes with the OS installation

# PVE: simple to create & use

## with conda

- Download & install [miniconda](#)
- `conda create -n pyml python=3.8` ~ creates the **pyml** PVE
- `conda activate pyml` ~ activates the **pyml** PVE
- `conda install <module>` ~ installs a module  
`pip install <module>`

## advantages / disadvantages

- 😊 The version of Python in a conda PVE can be  $\neq$  version of the interpreter that comes with the OS installation
- 😊 Some modules installed with **conda** can be more performant (numpy with MKL)

# PVE: simple to create & use

## with conda

- Download & install [miniconda](#)
- `conda create -n pyml python=3.8` ~ creates the **pyml** PVE
- `conda activate pyml` ~ activates the **pyml** PVE
- `conda install <module>` ~ installs a module  
`pip install <module>`

## advantages / disadvantages

- 😊 The version of Python in a conda PVE can be  $\neq$  version of the interpreter that comes with the OS installation
- 😊 Some modules installed with **conda** can be more performant (numpy with MKL)
- 😢 `conda install` and `pip install` can be conflicting...

# PVE: simple to create & use

## with Python env

- Install the `venv` Python module
- `python -m venv pyml` ~ creates the `pyml` PVE
- `./pyml/Scripts/activate` ~ activates the `pyml` PVE
- `pip install <module>` ~ installs a module

# PVE: simple to create & use

## with Python env

- Install the `venv` Python module
- `python -m venv pyml` ~ creates the `pyml` PVE
- `./pyml/Scripts/activate` ~ activates the `pyml` PVE
- `pip install <module>` ~ installs a module

# PVE: simple to create & use

## with Python env

- Install the `venv` Python module
- `python -m venv pyml` ~ creates the `pyml` PVE
- `./pyml/Scripts/activate` ~ activates the `pyml` PVE
- `pip install <module>` ~ installs a module

# PVE: simple to create & use

## with Python env

- Install the `venv` Python module
- `python -m venv pyml` ~ creates the `pyml` PVE
- `./pyml/Scripts/activate` ~ activates the `pyml` PVE
- `pip install <module>` ~ installs a module

# PVE: simple to create & use

## with Python env

- Install the `venv` Python module
- `python -m venv pym1` ~ creates the `pym1` PVE
- `./pym1/Scripts/activate` ~ activates the `pym1` PVE
- `pip install <module>` ~ installs a module

## advantages / disadvantages

- :( The version of Python for the PVE is the version of the interpreter that comes with the OS installation

# PVE: simple to create & use

## with Python env

- Install the `venv` Python module
- `python -m venv pym1` ~ creates the `pym1` PVE
- `./pym1/Scripts/activate` ~ activates the `pym1` PVE
- `pip install <module>` ~ installs a module

## advantages / disadvantages

- :( The version of Python for the PVE is the version of the interpreter that comes with the OS installation
- :) No conflict with `conda` install

# Reproducibility of training

## Where to find randomness in NN training

- **Random initialization of the weights** of the network before the training starts.
- Regularization, e.g. **dropout**, which involves randomly dropping nodes in the network while training.
- Optimization process like **stochastic gradient descent** or **Adam** also include randomness.
- Shuffling of data to build batches to train the NN
- and many others stages in ML algorithms where random generators are used...

# Reproducibility of training

## Where to find randomness in NN training

- **Random initialization of the weights** of the network before the training starts.
- Regularization, e.g. **dropout**, which involves randomly dropping nodes in the network while training.
- Optimization process like **stochastic gradient descent** or **Adam** also include randomness.
- Shuffling of data to build batches to train the NN
- and many others stages in ML algorithms where random generators are used...

# Reproducibility of training

## Where to find randomness in NN training

- **Random initialization of the weights** of the network before the training starts.
- Regularization, e.g. **dropout**, which involves randomly dropping nodes in the network while training.
- Optimization process like **stochastic gradient descent** or **Adam** also include randomness.
- Shuffling of data to build batches to train the NN
- and many others stages in ML algorithms where random generators are used...

# Reproducibility of training

## Where to find randomness in NN training

- **Random initialization of the weights** of the network before the training starts.
- Regularization, e.g. **dropout**, which involves randomly dropping nodes in the network while training.
- Optimization process like **stochastic gradient descent** or **Adam** also include randomness.
- Shuffling of data to build batches to train the NN
- and many others stages in ML algorithms where random generators are used...

# Reproducibility of training

## Where to find randomness in NN training

- **Random initialization of the weights** of the network before the training starts.
- Regularization, e.g. **dropout**, which involves randomly dropping nodes in the network while training.
- Optimization process like **stochastic gradient descent** or **Adam** also include randomness.
- Shuffling of data to build batches to train the NN
- and many others stages in ML algorithms where random generators are used...

# Reproducibility of training

## Practical work with tensorflow

- We will run a **Jupyter notebook** to illustrate the *Reproducibility of training* with **tensorflow**.
- See the PVE usage
- Create a simple *Feed Forward* dense network and train it to classify *hand written* images from the MNIST
- Explore where and how the question of the Reproducibility occurs... and how to solve it 😊

# Reproducibility of training

## Practical work with tensorflow

- We will run a **Jupyter notebook** to illustrate the *Reproducibility of training* with **tensorflow**.
- See the PVE usage
- Create a simple *Feed Forward* dense network and train it to classify *hand written* images form the MNIST
- Explore where and how the question of the Reproducibility occurs... and how to solve it 😊

# Reproducibility of training

## Practical work with tensorflow

- We will run a **Jupyter notebook** to illustrate the *Reproducibility of training* with **tensorflow**.
- See the PVE usage
- Create a simple *Feed Forward* dense network and train it to classify *hand written* images from the MNIST
- Explore where and how the question of the Reproducibility occurs... and how to solve it 😊

# Reproducibility of training

## Practical work with tensorflow

- We will run a **Jupyter notebook** to illustrate the *Reproducibility of training* with **tensorflow**.
- See the PVE usage
- Create a simple *Feed Forward* dense network and train it to classify *hand written* images from the MNIST
- Explore where and how the question of the Reproducibility occurs... and how to solve it 😊

[1] *Artificial Intelligence: A Modern Approach (4th Edition)*, By Stuart Russell & Peter Norvig. Pearson, 2020. ISBN 978-0134610993. [aima.cs.berkeley.edu](http://aima.cs.berkeley.edu)

*Intelligence artificielle – Une approche moderne – 4e éd.*, By Stuart Russell & Peter Norvig. Translated by L. Miclet, F. Popineau, & C. Cadet. Paris: Pearson Education France, 2021. ISBN 978-2326002210.

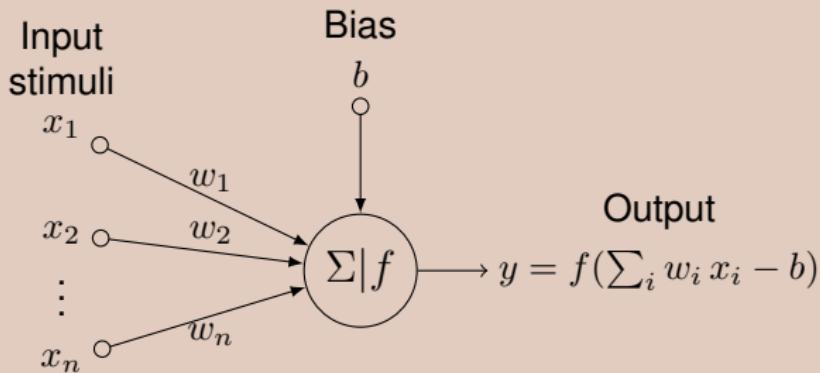
[2] *What is artificial intelligence (AI), and what is the difference between general AI and narrow AI?*, Kris Hammond, 2015  
[www.computerworld.com/article/2906336/what-is-artificial-intelligence.html](http://www.computerworld.com/article/2906336/what-is-artificial-intelligence.html)

[3] *Stanford Encyclopedia of Philosophy*, [plato.stanford.edu/entries/artificial-intelligence](http://plato.stanford.edu/entries/artificial-intelligence)

[4] *Deep Learning.*, Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016), MIT Pres, ISBN 9780262035613

# Artificial Neural Networks

## The Artificial neuron model

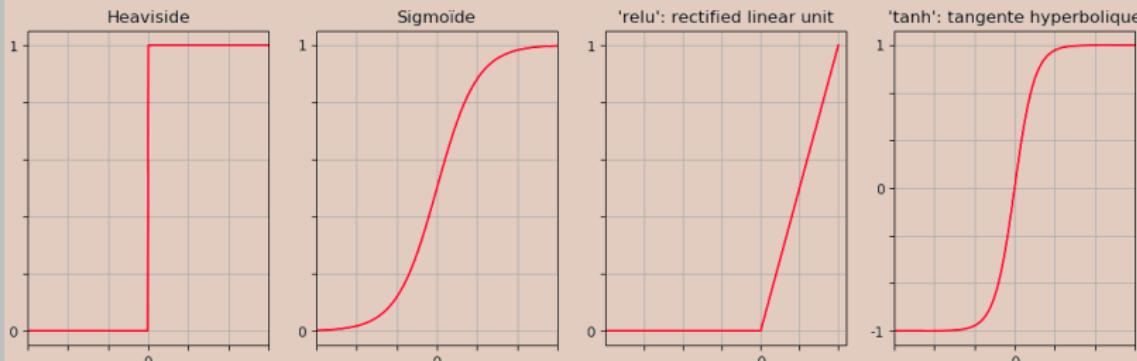


An **artificial neuron**:

- receives the input stimuli  $(x_i)_{i=1..n}$  with **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**  $f$ .

# Artificial Neural Networks

## Common activation functions

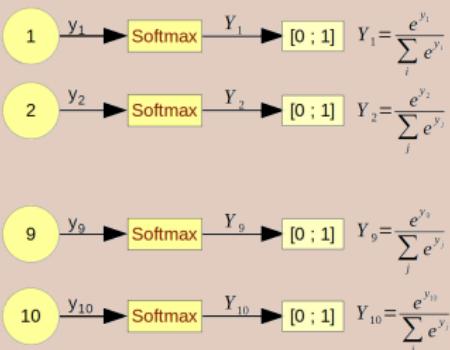


- Introduces a non-linear behavior.
- Sets the range of the neuron output:  $[-1, 1]$ ,  $[0, 1]$ ,  $[0, \infty[$ ...
- The bias  $b$  sets the activation threshold of the neuron.

# Réseau de neurones dense

- Dans les couches intermédiaires la fonction d'activation **relu** favorise l'apprentissage du réseau<sup>1</sup>.
- La classification (dernière couche) utilise la fonction **softmax** :

## Fonction d'activation *softmax*



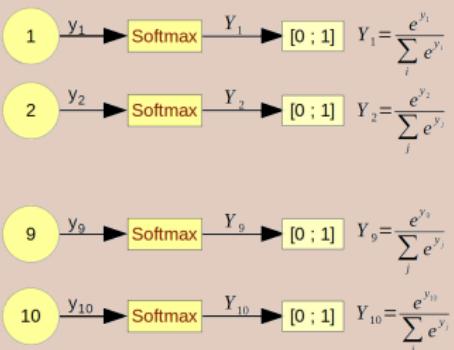
- L'activation du neurone  $k$  est  $Y_k = e^{y_k} / \sum_i e^{y_i}$  avec  $y_k = \sum_i \omega_i x_i - b$  calculé par le neurone  $k$ .
- Les sorties des neurones s'interprètent comme des probabilités dans l'intervalle  $[0,1]$ .

<sup>1</sup> évite le *vanishing gradient* qui apparaît dans l'algorithme de *back propagation*

# Réseau de neurones dense

- Dans les couches intermédiaires la fonction d'activation **relu** favorise l'apprentissage du réseau<sup>1</sup>.
- La classification (dernière couche) utilise la fonction **softmax** :

## Fonction d'activation *softmax*



- L'activation du neurone  $k$  est  $Y_k = e^{y_k} / \sum_i e^{y_i}$  avec  $y_k = \sum_i \omega_i x_i - b$  calculé par le neurone  $k$ .
- Les sorties des neurones s'interprètent comme des probabilités dans l'intervalle  $[0,1]$ .

Réponse du réseau  $\leadsto$  label associé au neurone de plus grande probabilité.

<sup>1</sup> évite le *vanishing gradient* qui apparaît dans l'algorithme de *back propagation*

# Réseau de neurones dense

## Codage *One-hot* des labels

Mettre les labels des images au format de la sortie du réseau :

- Labels des images :  **nombres entiers** de 0 à 9.
- Sortie du réseau : **vecteur de 10 float** dans l'intervalle [0,1] calculés par les fonctions *softmax* des 10 neurones de sortie.
- Codage *one-hot* d'un ensemble ordonné  $\mathcal{L}$  de  $N$  labels :
  - chaque label est représenté par un vecteur à  $N$  composantes toutes nulles, sauf une (égale à 1),
  - le rang du 1 du vecteur associé à un label est le rang du label dans  $\mathcal{L}$ .

# Réseau de neurones dense

## Codage *One-hot* des labels

Mettre les labels des images au format de la sortie du réseau :

- Labels des images :  **nombres entiers** de 0 à 9.
- Sortie du réseau : **vecteur de 10 float** dans l'intervalle [0,1] calculés par les fonctions *softmax* des 10 neurones de sortie.
- Codage one-hot** d'un ensemble ordonné  $\mathcal{L}$  de  $N$  labels :

chiffre	vecteur <i>one-hot</i>
0	[1 0 0 0 0 0 0 0 0]
1	[0 1 0 0 0 0 0 0 0]
2	[0 0 1 0 0 0 0 0 0]
3	[0 0 0 1 0 0 0 0 0]
4	[0 0 0 0 1 0 0 0 0]
5	[0 0 0 0 0 1 0 0 0]
6	[0 0 0 0 0 0 1 0 0]
7	[0 0 0 0 0 0 0 1 0]
8	[0 0 0 0 0 0 0 0 1]
9	[0 0 0 0 0 0 0 0 1]

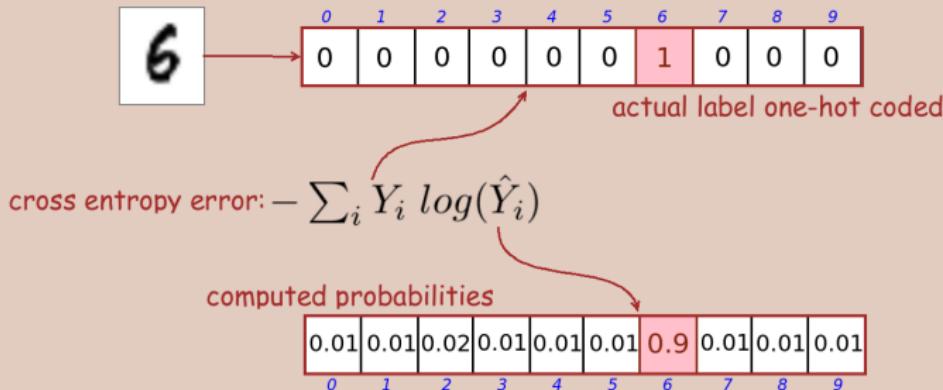
Le label est représenté par un vecteur à  $N$  composantes, sauf une (égale à 1), la 1 du vecteur associé à un label est le rang du label

Le codage *one-hot* des labels '0' à '9' donne un vecteur à 10 composantes, comme celui calculé par le réseau de neurones.

# 1 - Réseau de neurones dense

## Fonction d'erreur : *Cross entropy error*

- L'image traitée par le réseau  $\leadsto$  vecteur  $\hat{Y}$  de 10 float à comparer au vecteur  $Y$  du codage *hot-one* du label de l'image.
- La fonction d'erreur (de perte) *cross entropy* est adaptée au codage *one-hot* :  $e(Y, \hat{Y}) = -\sum_i Y_i \log(\hat{Y}_i)$



# 1 - Réseau de neurones dense

## Optimisation et *Back Propagation*

- Pendant la phase d'apprentissage un algorithme d'optimisation calcule le gradient de la fonction d'erreur par rapport aux poids du réseau.

# 1 - Réseau de neurones dense

## Optimisation et *Back Propagation*

- Pendant la phase d'apprentissage un algorithme d'optimisation calcule le gradient de la fonction d'erreur par rapport aux poids du réseau.
- L'algorithme de *Back Propagation* **modifie** les poids du réseau couche par couche grâce au gradient de la fonction d'erreur, en itérant de la dernière couche à la première couche.

# 1 - Réseau de neurones dense

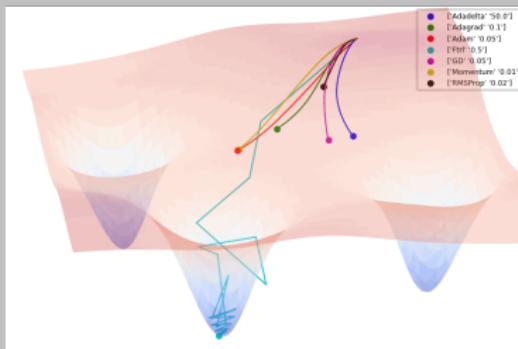
## Optimisation et *Back Propagation*

- Pendant la phase d'apprentissage un algorithme d'optimisation calcule le gradient de la fonction d'erreur par rapport aux poids du réseau.
- L'algorithme de *Back Propagation* **modifie** les poids du réseau couche par couche grâce au gradient de la fonction d'erreur, en itérant de la dernière couche à la première couche.
- Exemples d'algorithme d'optimisation :
  - Descente de Gradient (*Gradient Descent (GD)*)
  - Descente de Gradient Stochastique (*Stochastic Gradient Descent (SGD)*)
  - *Adam* (version améliorée de descente de gradient)...

Le module `tf.keras.optimizers` propose l'implémentation Python de plusieurs algorithmes d'optimisation.

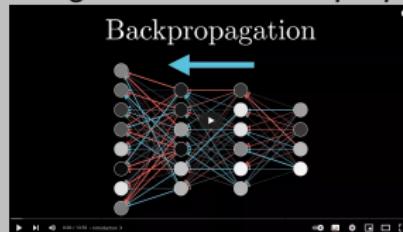
# 1 - Réseau de neurones dense

Visualisation des itérations d'algorithmes de descente de gradient pour une fonction de perte ultra-simple à seulement 2 variables :

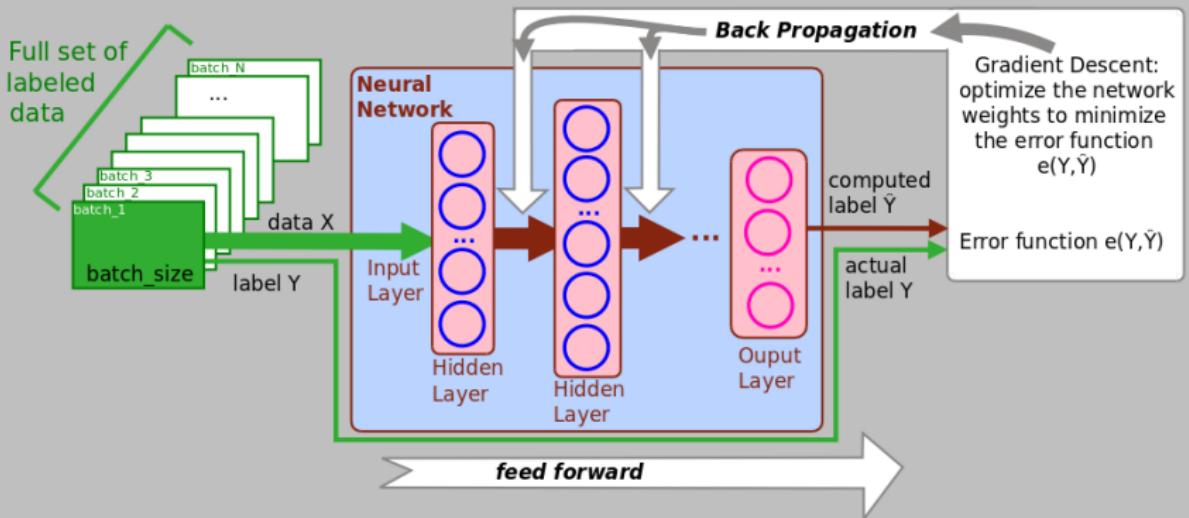


(source : [github.com/Jaewan-Yun/optimizer-visualization](https://github.com/Jaewan-Yun/optimizer-visualization))

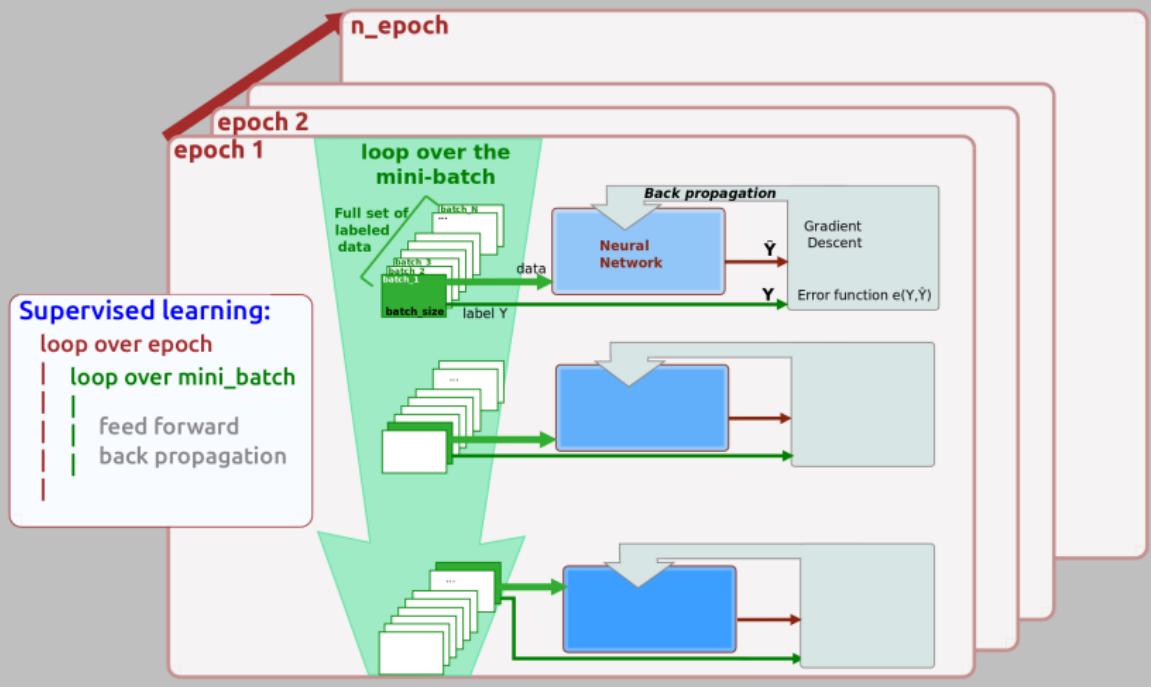
Vidéo d'explication de l'algorithme de *back propagation* :



# Supervised learning : Feed Forward and Back Propagation



- Le jeu de données est découpé en (mini) **batches** de taille **batch\_size**
- Après chaque *feed forward* l'algorithme de *Back Propagation* modifie les poids des neurones pour minimiser l'erreur  $e$ .



- L'entraînement avec le jeu de données est répété  $n_{epoch}$  fois,
  - L'état du réseau à la fin de l'époque  $n$  sert d'état initial pour l'époque  $n+1$ .