Deep Learning – Introduction

Jill-Jênn Vie Martin Van Waerebeke
Kimia Nadjahi Marc Lelarge

Sep 26, 2025

About the instructors

Jill-Jênn Vie

- Research scientist at Inria & ICPC coach
- Personalized education, recommender systems
- Soda: understanding society with data; foundation models for tabular learning; ex. scikit-learn (3M downloads a day), now skrub

Martin Van Waerebeke

- PhD student at Inria
- Machine unlearning
- ARGO: learning, graphs, distributed optimization

Kimia Nadjahi

Researcher at CNRS, optimal transport

Marc Lelarge

Director of research at Inria, ARGO team

Goal of this course

- Implement deep learning so that you can use this knowledge for your research
- Understand how automatic differentiation is implemented in frameworks like PyTorch, so that you can do it in any language
- Learn how to convert an equation into code

AI moves very fast but some things don't change

- It's all about objectives, vectors, gradients (derivatives)
- Data (labeled or not)
- Models (pretrained or not)

Deep Learning

- 1957: Perceptron (Rosenblatt): the first (1-layer) neural network… that works
- 1969: Minsky & Papert highlight its limitations $\rightarrow \rightarrow$ loss of funding, AI winter
- 1980: Fukushima proposes the neocognitron, a precursor to CNNs (computer vision)

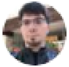
"**Deep Learning Conspiracy**" trio ("Canadian mafia")
Geoffrey Hinton (Nobel Prize in Physics 2024), Yann LeCun, Yoshua Bengio

Deep Learning

- 1957: Perceptron (Rosenblatt): the first (1-layer) neural network… that works
- 1969: Minsky & Papert highlight its limitations → → loss of funding, AI winter
- 1980: Fukushima proposes the neocognitron, a precursor to CNNs (computer vision)

"**Deep Learning Conspiracy**" trio ("Canadian mafia")
Geoffrey Hinton (Nobel Prize in Physics 2024), Yann LeCun, Yoshua Bengio

- 2007: informal workshop organized at Neural Information Processing Systems
- 2012: breakthrough in image recognition – AlexNet (Krizhevsky, Sutskever, Hinton)
- 2013: breakthrough in language processing – word2vec (Mikolov et al.)
- 2014: breakthrough génération d'images – GAN (Goodfellow et al.)
- 2016: reinforcement learning – AlphaGo beats Lee Sedol
- 2017: transformers (Vaswani et al.)
- 2018: BERT & GPT; Canadian mafia wins the 2018 Turing Award
- 2021: latent diffusion by CompVis, Munich
- 2022: stable diffusion, InstructGPT, ChatGPT
- 2024: Sutton & Barto (reinforcement learning) win the 2024 Turing Award
- 2025: OpenAI solves every problem 12/12 of ICPC World Finals



Ian Goodfellow
@goodfellow_ian

4.5 years of GAN progress on face generation. arxiv.org/abs/1406.2661 arxiv.org/abs/1511.06434 arxiv.org/abs/1606.07536 arxiv.org/abs/1710.10196 arxiv.org/abs/1812.04948

Traduire le post

Generative Adversarial Networks (Goodfellow, 2014, Test of Time 2024)

A master student in Kyoto University (Yanghua Jin) did this:

Geoffrey Hinton's MOOC (2012)

AI

A database: answers my queries

An AI: infers the missing points by interpolating

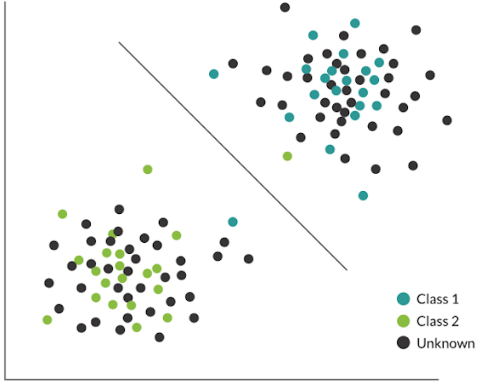
Learns from examples, or collects data by experience (reinforcement learning)

Should be able to generalize

Supervised learning

We observe $\boldsymbol{x}$, y

i.e. classification, regression



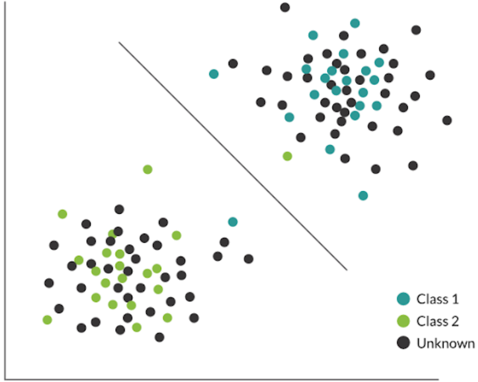

```
model.fit(x_train, y_train)
model.predict(x_test)
```

Unsupervised learning

Supervised learning

We observe $\boldsymbol{x}$, y

i.e. classification, regression

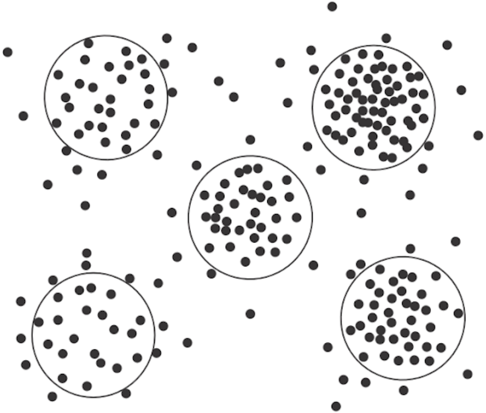



```
model.fit(x_train, y_train)
model.predict(x_test)
```

Unsupervised learning

We just observe $\boldsymbol{x}$

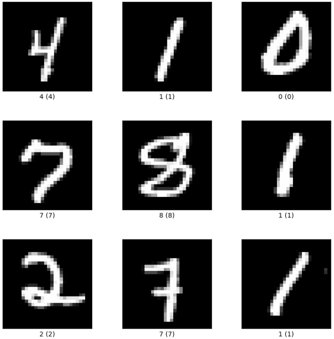
i.e. raw text, music, ratings



```
x_tr = pca.fit_transform(x_train)
x_te = pca.transform(x_test)
```

Example: handwritten digit recognition

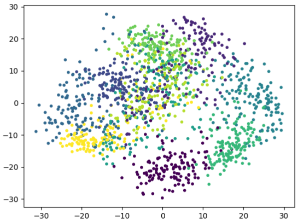
```
from sklearn.datasets import load_digits
```



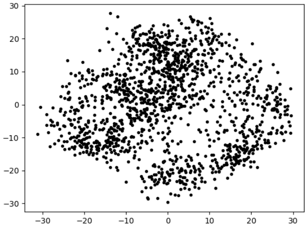
1797 images 8 x 8 grayscale, $X \in \mathbb{R}^{1797\times 64}, y \in \{0, \dots, 9\}^{64}$

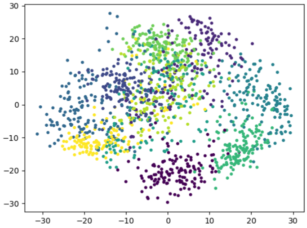
Visualization by projecting over 2 dimensions

k nearest neighbor (linear projection)

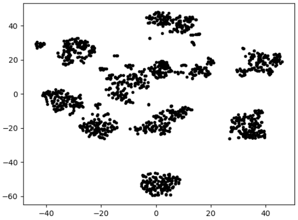


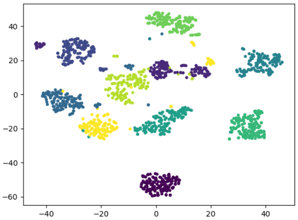
k-means (linear projection)



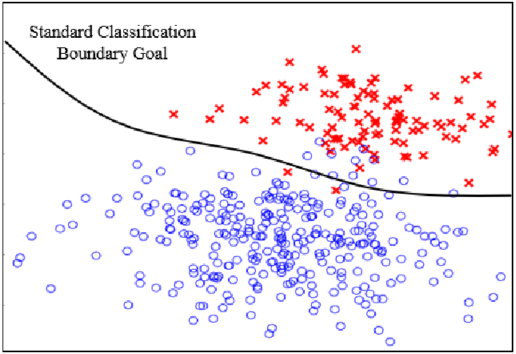


k-means on t-SNE (non-linear)

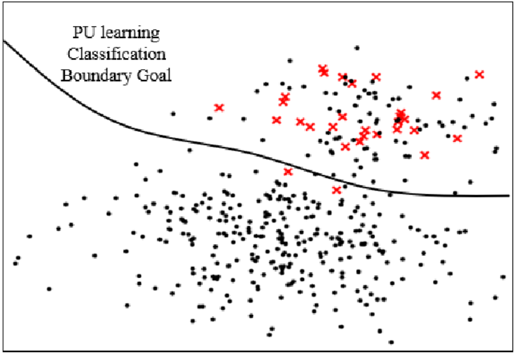




Semi-supervised learning: some $\boldsymbol{x}$ do not have y



(a) Supervised classification learning problem with all labels known.

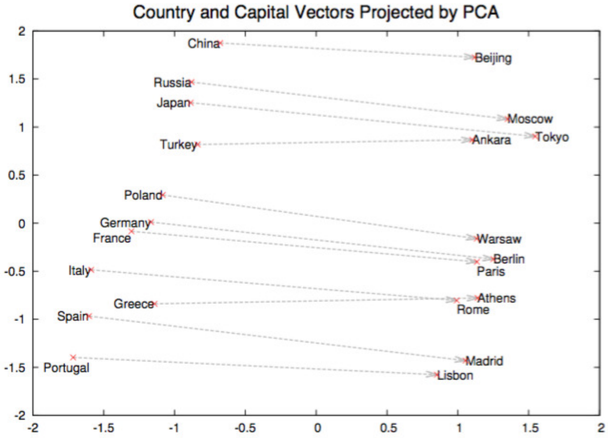


(b) Positive Unlabeled learning problem with only a percentage of known positive labels.

"PU" learning: "positive unlabeled"

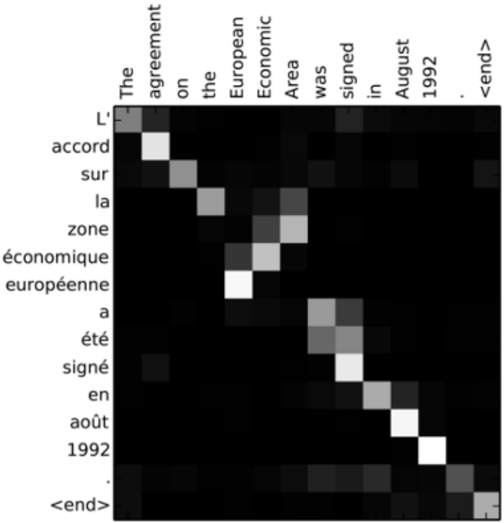
Self-supervised learning

Predicting a word using its neighbors (word2vec, Mikolov et al., 2013, Test of Time 2023)



Predicting the next token (GPT)

Attention (Bahdanau, Cho and Bengio, 2014, Test of Time 2024)



Automatic captioning using attention (Xu and Bengio, 2015)

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

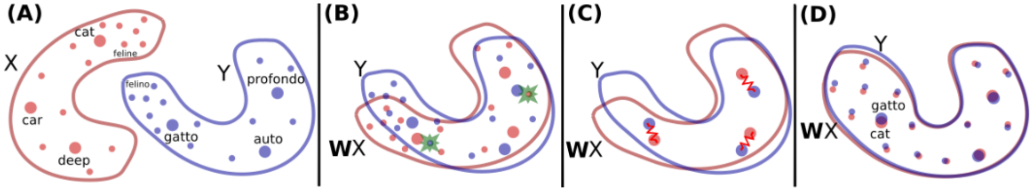
A stop sign is on a road with a mountain in the background.

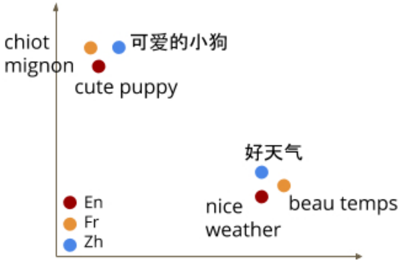
A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Multilingual word embeddings (unsupervised)





Guillaume Lample et al. (2018). "Word translation without parallel data". In: *International Conference on Learning Representations*. URL: https://arxiv.org/abs/1710.04087

In-context learning

Traditional ML: fit(train), predict(test)

Transfer learning: pretrain, finetune(train), predict(test) (e.g. word embeddings)

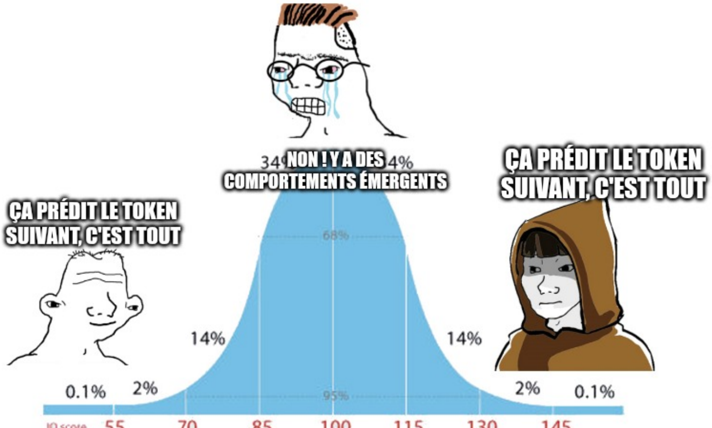
In-context learning: pretrain (LLM), predict(test, train)

In-context learning

Traditional ML: fit(train), predict(test)

Transfer learning: pretrain, finetune(train), predict(test) (e.g. word embeddings)

In-context learning: pretrain (LLM), predict(test, train)



Large language models are few-shot learners

Article

Accurate predictions on small data with a tabular foundation model

https://doi.org/10.1038/s41586-024-08328-6

Received: 17 May 2024

Accepted: 31 October 2024

Published online: 8 January 2025

Open access

Check for updates

Noah Hollmann[1,2,3,7✉], Samuel Müller[1,7✉], Lennart Purucker[1], Arjun Krishnakumar[1], Max Körfer[1], Shi Bin Hoo[1], Robin Tibor Schirrmeister[4,5] & Frank Hutter[1,3,6✉]

Tabular data, spreadsheets organized in rows and columns, are ubiquitous across scientific fields, from biomedicine to particle physics to economics and climate science[1,2]. The fundamental prediction task of filling in missing values of a label column based on the rest of the columns is essential for various applications as diverse as biomedical risk models, drug discovery and materials science. Although deep learning has revolutionized learning from raw data and led to numerous high-profile success stories[3–5], gradient-boosted decision trees[6–9] have dominated tabular data for the past 20 years. Here we present the Tabular Prior-data Fitted Network (TabPFN), a tabular foundation model that outperforms all previous methods on datasets with up to 10,000 samples by a wide margin, using substantially less training time. In 2.8 s, TabPFN outperforms an ensemble of the strongest baselines tuned for 4 h in a classification setting. As a generative transformer-based foundation model, this model also allows fine-tuning, data generation, density estimation and learning reusable embeddings. TabPFN is a learning algorithm that is itself learned across millions of synthetic datasets, demonstrating the power of this approach for algorithm development. By improving modelling abilities across diverse fields, TabPFN has the potential to accelerate scientific discovery and enhance important decision-making in various domains.

Summing up

Supervised learning: features X and labels y

Unsupervised learning: learning a representation of X (e.g. dimensionality reduction)

Reinforcement learning: learning by interacting with an environment

Semi-supervised learning: some X are not mapped to a y, missing values

Self-supervised learning: predict parts of X from itself

In-context learning: predict y_{test} from $X_{test}, X_{train}, y_{train}$, no training, only inference

Supervised learning

Samples: $(\boldsymbol{x}_i, y_i) \in \boldsymbol{X} \times \mathcal{Y}$
Features $\boldsymbol{x}_i$ and labels y_i.

Usually $\boldsymbol{X} = \mathbb{R}^d$
If $\mathcal{Y}$ is discrete: classification, otherwise regression.

Model $f_\theta : \boldsymbol{X} \to \mathcal{Y}$ has parameters θ
Find the best parameters $\boldsymbol{\theta}$ to reach a goal: making $f_{\boldsymbol{\theta}}(x)$ close to y

A differentiable loss function $\mathcal{L}$ is minimized, and depends on parameters θ.

Goal: find $\theta = \text{argmin}_\theta \mathcal{L}(\theta, \boldsymbol{x}_i, y_i)$

Examples of loss functions

Regression: continuous target

Ex. $y \in \mathbb{R} \quad \mathcal{L} = \sum_i (\hat{y}_i - y_i)^2$ squared error

Classification: discrete target

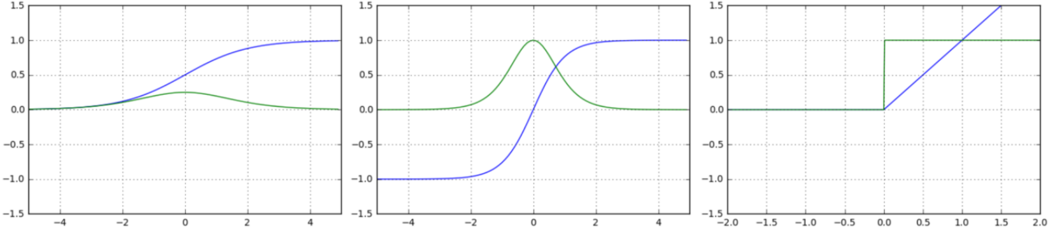
Ex. $y \in \{0, 1\}$

- Logistic regression $\hat{y} = \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{x}) \in \{0, 1\}$
- Logistic loss: $\mathcal{L}_i(p, y) = y \log p + (1 - y) \log(1 - p)$

Ex. $y \in \{1, \ldots, C\}$

- Multinomial regression $\hat{y} = \text{softmax}(\boldsymbol{W}\boldsymbol{x} + b) \in \{0, 1\}^C$
- Cross-entropy loss: $\mathcal{L}_i(\boldsymbol{p}, y) = \sum_{c=1}^{C} y_c \log p_c = \log p_y$

Activation / link functions; why sigmoid, softmax?



$$\mathrm{sigm}(x) = \frac{1}{1+e^{-x}} \qquad \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \qquad \mathrm{relu}(x) = \max(0,x)$$

$$\mathrm{sigm}'(x) = \mathrm{sigm}(x)(1-\mathrm{sigm}(x)) \qquad \tanh'(x) = 1-\tanh(x)^2 \qquad \mathrm{relu}'(x) = 1_{x>0}$$

(Ollion & Grisel)

	Binary	Multiclass
f	softplus : $x \mapsto \log(1+\exp(x))$	logsumexp$^+$: $\boldsymbol{x} \mapsto \log(1+\sum_c \exp(x_c))$
f'	sigmoid : $x \mapsto 1/(1+\exp(-x))$	softmax : $\boldsymbol{x} \mapsto \exp(\boldsymbol{x})/\sum_c \exp(x_c)$
f''	$x \mapsto \mathrm{sigmoid}(x)(1-\mathrm{sigmoid}(x))$	$\boldsymbol{x} \mapsto \mathrm{diag}(s(\boldsymbol{x})) - s(\boldsymbol{x})s(\boldsymbol{x})^T$

Linear regression

Model $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = \boldsymbol{x}_i^T \boldsymbol{\theta} \simeq y_i$

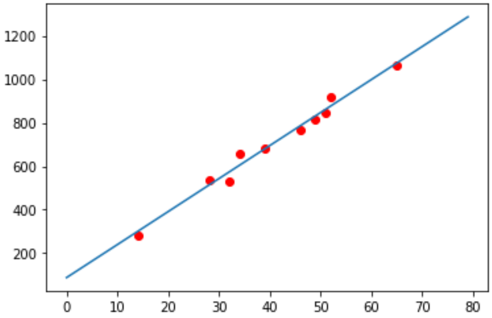
Loss: squared error $\mathcal{L} = \sum_i ||\boldsymbol{x}_i^T \boldsymbol{\theta} - y_i||^2$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \sum_i 2\boldsymbol{x}_i(\boldsymbol{x}_i^T \boldsymbol{\theta} - y_i) = 0$$

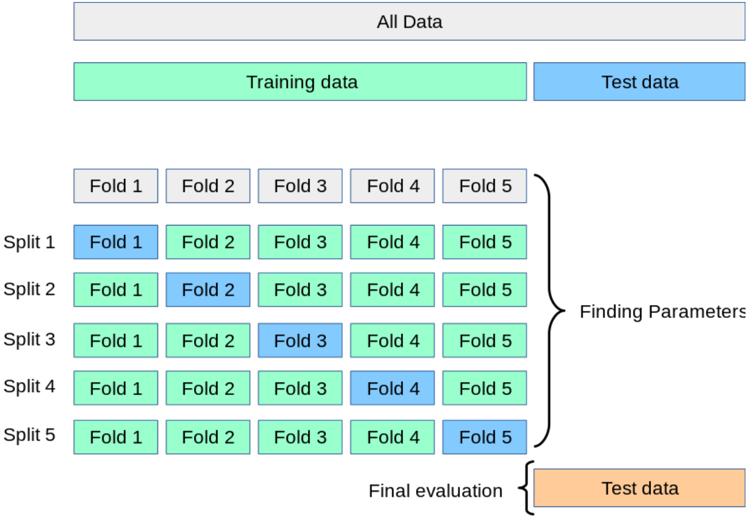
$$\left(\sum_i \boldsymbol{x}_i \boldsymbol{x}_i^T\right) \boldsymbol{\theta} = \sum_i \boldsymbol{x}_i y_i$$

$$\boldsymbol{\theta} = (X^T X)^{-1} X^T \boldsymbol{y}$$

(in closed form)

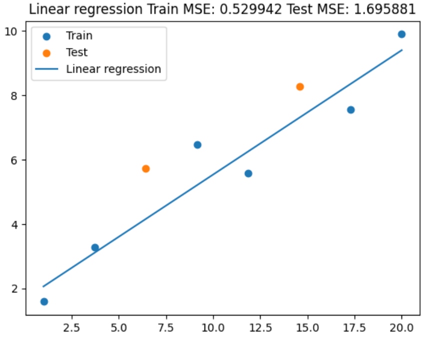


Cross-validation



Fitting

Polynomial degree 1 $Y = wX + b$

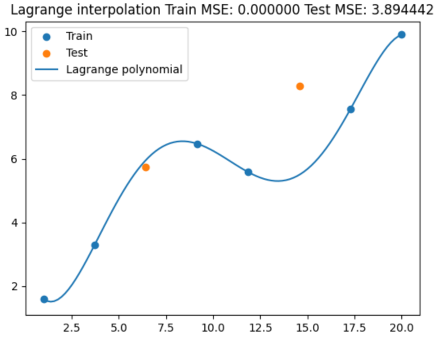


Linear regression

`scipy.stats.linreg`

Overfitting

Polynomial degree 6 $Y = \sum_{k=0}^{6} w_k X^k$



Lagrange interpolation

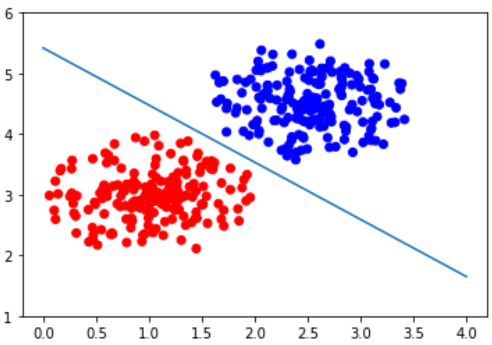
`scipy.interpolation.lagrange`

Logistic regression (classification)

$f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = \sigma(\boldsymbol{x}_i{}^T \boldsymbol{\theta}) = \Pr(Y_i = 1)$
where $\sigma : x \mapsto 1/(1 + e^{-x})$

Convenient because $\sigma' = \sigma(1 - \sigma)$

No closed form for $\boldsymbol{\theta}$ then learning $\boldsymbol{\theta}$ by
Newton's method



Perceptron (Rosenblatt, 1957) is a linear layer (replace σ with step function Heaviside)

Variant: handwritten character recognition

Replace σ with softmax: multinomial logistic regression

Multilayer Perceptrons (MLP)

$$\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x} \\
\boldsymbol{x}^{(\ell+1)} &= \sigma\big(\boldsymbol{W}^{(\ell)}\boldsymbol{x}^{(\ell)} + \boldsymbol{b}^{(\ell)}\big) \quad \ell = 0, \ldots, L-2 \\
y = \boldsymbol{x}^{(L)} &= \boldsymbol{W}^{(L-1)}\boldsymbol{x}^{(L-1)} + \boldsymbol{b}^{(L-1)}
\end{aligned}$$

Multilayer Perceptrons (MLP)

$$\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x} \in \mathbb{R}^{d_0} \\
\boldsymbol{x}^{(\ell+1)} &= \sigma(\boldsymbol{W}^{(\ell)}\boldsymbol{x}^{(\ell)} + \boldsymbol{b}^{(\ell)}) \in \mathbb{R}^{d_\ell} \quad \ell = 0, \ldots, L-2 \\
y = \boldsymbol{x}^{(L)} &= \boldsymbol{W}^{(L-1)}\boldsymbol{x}^{(L-1)} + \boldsymbol{b}^{(L-1)} \in \mathbb{R}^{d_L}
\end{aligned}$$

The ℓth layer has d_ℓ neurons. Input layer $\ell = 0$, output layer $\ell = L$.
σ is the link function. Usually, $\sigma = \mathrm{ReLU} = \max(\mathbf{0}, \boldsymbol{x})$. #params?

Multilayer Perceptrons (MLP)

$$
\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x} \in \mathbb{R}^{d_0} \\
\boldsymbol{x}^{(\ell+1)} &= \sigma(\boldsymbol{W}^{(\ell)}\boldsymbol{x}^{(\ell)} + \boldsymbol{b}^{(\ell)}) \in \mathbb{R}^{d_\ell} \quad \ell = 0, \ldots, L-2 \\
y = \boldsymbol{x}^{(L)} &= \boldsymbol{W}^{(L-1)}\boldsymbol{x}^{(L-1)} + \boldsymbol{b}^{(L-1)} \in \mathbb{R}^{d_L}
\end{aligned}
$$

The ℓth layer has d_ℓ neurons. Input layer $\ell = 0$, output layer $\ell = L$.
σ is the link function. Usually, $\sigma = \text{ReLU} = \max(\boldsymbol{0}, \boldsymbol{x})$. #params?

```python
from torch import nn
mlp = nn.Sequential(
    nn.Linear(d_0, d_1),
    nn.ReLU(),
    nn.Linear(d_1, d_2),
    ...
    nn.Linear(d_{L-1}, d_L)
)
```

But then? I can represent a L-layer perceptron as $d_0 \to d_1 \to \cdots \to d_L$

This notation simplifies expression

Supervised learning

Linear or logistic regression: $n \to 1$

Perceptron for handwritten digit recognition:

This notation simplifies expression

Supervised learning

Linear or logistic regression: $n \to 1$

Perceptron for handwritten digit recognition: $n \to 10$

Unsupervised learning

Principal components analysis:

This notation simplifies expression

Supervised learning

Linear or logistic regression: $n \to 1$
Perceptron for handwritten digit recognition: $n \to 10$

Unsupervised learning

Principal components analysis: $n \to k \to n$ (without nonlinear sigmoid σ)
Autoencoder $n \to \cdots \to k \to \cdots \to n$
Therefore, a PCA is a 1-layer linear autoencoder
(k-SVD: best k-rank approximation, truncate at k biggest singular values)

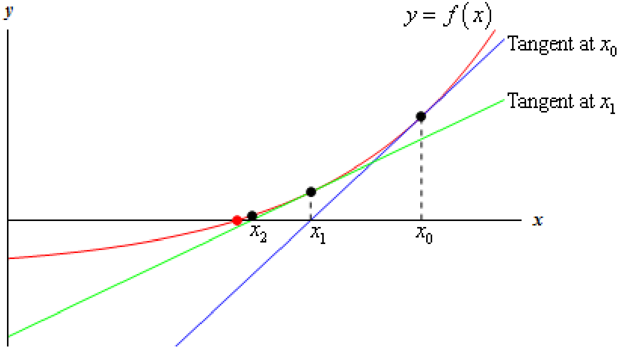
Why SGD?

To minimize a function

Find the zeroes of its derivative

How to find the zeroes of a function?

Newton's method: find x such that $f(x) = 0$



$f : \mathbb{R} \to \mathbb{R}$ differentiable, $\nexists x, f'(x) = 0$

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

Quadratic convergence

$$\exists C > 0, |x_{t+1} - \ell| \leq C|x_t - \ell|^2$$

In higher dimension

Let $g : \mathbb{R}^n \to \mathbb{R}$ twice differentiable, with n big

In higher dimension

Let $g : \mathbb{R}^n \to \mathbb{R}$ twice differentiable, with n big

What is the size of $g'(\boldsymbol{x})$? Usually noted $\dfrac{\partial g}{\partial \boldsymbol{x}}$ or $\nabla_{\boldsymbol{x}} g$.

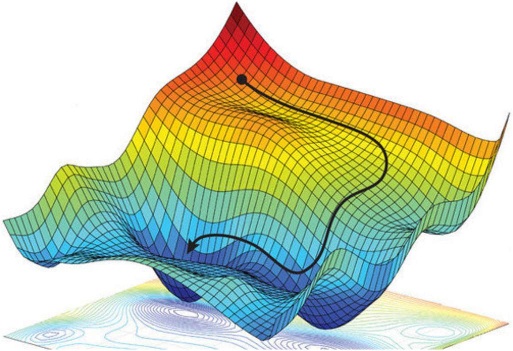
In higher dimension

Let $g : \mathbb{R}^n \to \mathbb{R}$ twice differentiable, with n big

What is the size of $g'(\boldsymbol{x})$? Usually noted $\dfrac{\partial g}{\partial \boldsymbol{x}}$ or $\nabla_{\boldsymbol{x}} g$.

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \underbrace{g''(\boldsymbol{x}_t)^{-1}}_{\in \mathbf{R}^{n \times n},\ O(n^3)} \underbrace{g'(\boldsymbol{x}_t)}_{\in \mathbf{R}^n}$$

Gradient descent



$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}_t)$$

for each epoch:
    for $\boldsymbol{x}$, y in dataset:
        compute gradients $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$ # also noted $\nabla_\theta \mathcal{L}$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} \mathcal{L}$ # γ is the learning rate

A typical training loop in PyTorch

```
import torch

# define model, n_epochs, trainloader
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)

for _ in range(n_epochs):
    for batch_inputs, batch_labels in trainloader:
        outputs = model(batch_inputs)
        loss = criterion(outputs, batch_labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Variants

Batch gradient descent

Compute the gradient of loss on all examples and update parameters

Variants

Batch gradient descent

Compute the gradient of loss on all examples and update parameters

Gradient descent

For each example $\boldsymbol{x}_i, y$ update parameters

Variants

Batch gradient descent

Compute the gradient of loss on all examples and update parameters

Gradient descent

For each example $\boldsymbol{x}_i, y$ update parameters

Stochastic gradient descent

Sample examples $\boldsymbol{x}_i, y$ and update parameters

Variants

Batch gradient descent

Compute the gradient of loss on all examples and update parameters

Gradient descent

For each example $\boldsymbol{x}_i, y$ update parameters

Stochastic gradient descent

Sample examples $\boldsymbol{x}_i, y$ and update parameters

Minibatch gradient descent

Sample a batch of examples, estimate full gradient from batch and update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \gamma \frac{n}{B} \sum_{i \in B} \mathcal{L}(\boldsymbol{x}_i, y)$$

Natural questions you should ask yourself

Data $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ targets $\boldsymbol{y} \in \mathcal{Y}^n$

Model $f : \boldsymbol{x} \mapsto \hat{y} = f(\boldsymbol{x})$

How to represent data as X, y?

How to build f?

Which loss $\mathcal{L}$ should I choose?

Which optimizer?

Stochastic gradient descent? Nesterov? Adam? AdamW?

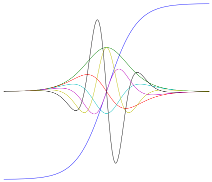
Automatic differentiation

Why backpropagation? What does `loss.backward()` do?
(We will cover this next week.)

@untitled01ipynb

How to compute gradients automatically?

```python
from autograd import elementwise_grad as egrad
import matplotlib.pyplot as plt
x = np.linspace(-7, 7, 200)
plt.plot(x, tanh(x),
        x, egrad(tanh)(x),                                     # first
        x, egrad(egrad(tanh))(x),                              # second
        x, egrad(egrad(egrad(tanh)))(x),                       # third
        x, egrad(egrad(egrad(egrad(tanh))))(x),                # fourth
        x, egrad(egrad(egrad(egrad(egrad(tanh)))))(x),         # fifth
        x, egrad(egrad(egrad(egrad(egrad(egrad(tanh))))))(x))  # sixth
plt.show()
```



Please come to class

This course requires your active participation

I am taking an RER B from Massy-Palaiseau every Friday to see you

I took an earlier flight from a conference in Prague (RecSys 2025) to see you today

Lessons on Fridays 8:30-11:30 (incl. 10 min break)

Check room on https://diplome.di.ens.fr/calendar_fr.html

Validation

- Some TDs will be homeworks (/10)
- One project (/10) by teams of 3 maximum (preferably 2); defense end of January

Communication

Moodle & Discord

- Lample, Guillaume et al. (2018). "Word translation without parallel data". In: *International Conference on Learning Representations*. URL: https://arxiv.org/abs/1710.04087.
- Mikolov, Tomas et al. (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.