Intelligence artificielle

Jill-Jênn Vie

8 mai 2025

Jill-Jênn Vie

Thiers, Lyon, Cachan, agrég', Orsay, Pix, Tokyo, Inria Lille, Inria Saclay, X

Jill-Jênn Vie

Thiers, Lyon, Cachan, agrég', Orsay, Pix, Tokyo, Inria Lille, Inria Saclay, X

ICPC coach Ecole polytechnique (gold SWERC 2024), ENS Paris-Saclay, Université de Lille, Institut Polytechnique de Paris, CentraleSupélec

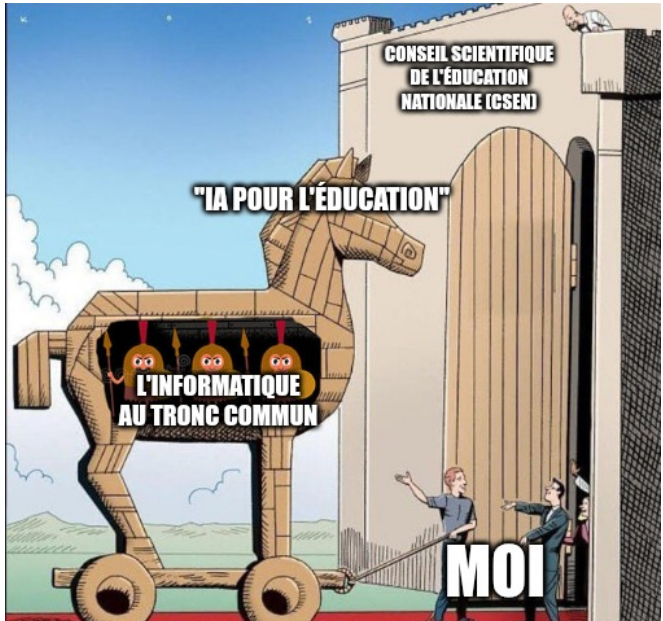
Inria

Équipe Soda à Inria Saclay : understanding society with data

IA en éducation, systèmes de recommandation (TP 1), tabular deep learning

scikit-learn : 3 millions de téléchargements par jour

J'ai rejoint le CSEN en 2013



Outline

Apprentissage

Cheval de troie 1

Cheval de troie 2

Démo : illustration de fairness (biais dans les décisions algorithmiques)

Bonus 1 : efficacité et impact environnemental des LLM

Bonus 2 : risques sociétaux des LLM, notamment pour l'éducation

IA

Une base de données : me renvoie les requêtes que je demande

Une IA : infère les points manquants par interpolation (« BayesDB »)

Apprend à partir d'exemples, ou collecte ses données par l'expérience (par renforcement)

Doit être capable de généraliser

Programme MPI / NSI

Apprentissage supervisé

- Algorithme des k plus proches voisins (NSI 1re)
- Matrice de confusion
- Arbres k-dimensionnels
- Arbre de décision (algorithme ID3)

Apprentissage non supervisé

- Algorithme des k-moyennes
- Algorithme de classification hiérarchique ascendante

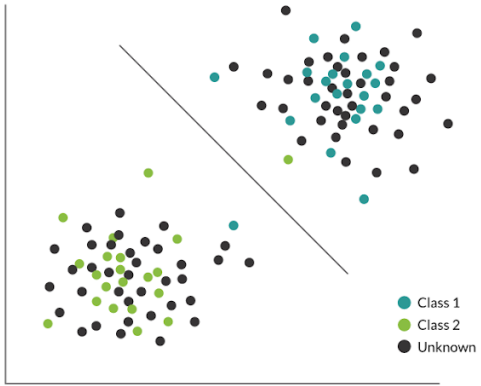
Jeux et heuristiques

- Jeux d'accessibilité à 2 joueurs sur un graphe
- Algorithme min-max avec une heuristique
- Élagage alpha-bêta
- Algorithme A* (ex. projet NSI)

Apprentissage supervisé

On observe des échantillons $\boldsymbol{x}, y$

ex. classification, régression

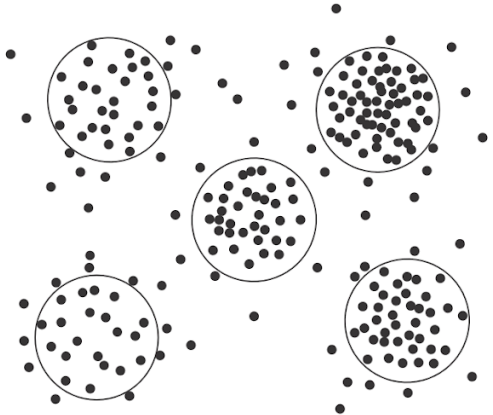



```
model.fit(x_train, y_train)
model.predict(x_test)
```

Apprentissage non supervisé

On observe seulement $\boldsymbol{x}$

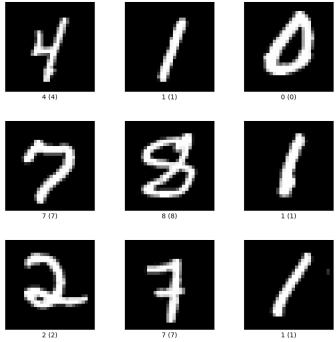
ex. texte brut, du son



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

Exemple : reconnaissance de chiffres manuscrits (TP 2)

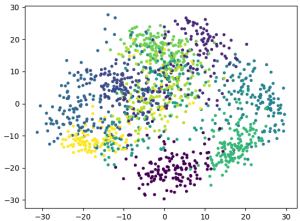
```
from sklearn.datasets import load_digits
```



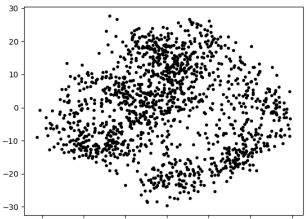
1797 images 8 x 8 en niveaux de gris, $X \in \mathbb{R}^{1797 \times 64}, y \in \{0, \ldots, 9\}^{64}$

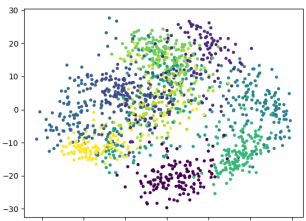
TP 2 : visualisation via projection en 2 dimensions

k plus proches voisins (projection linéaire)

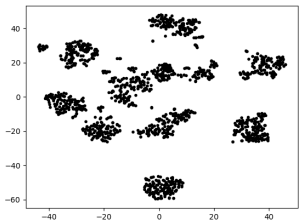


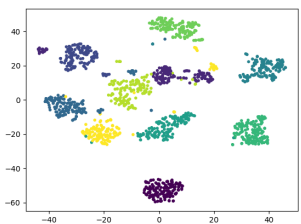
k-moyennes sur PCA (projection linéaire)



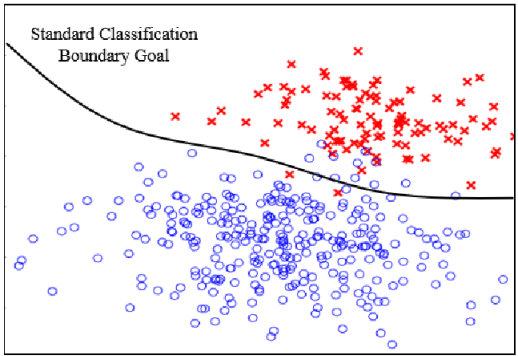


k-moyennes sur t-SNE (non linéaire)

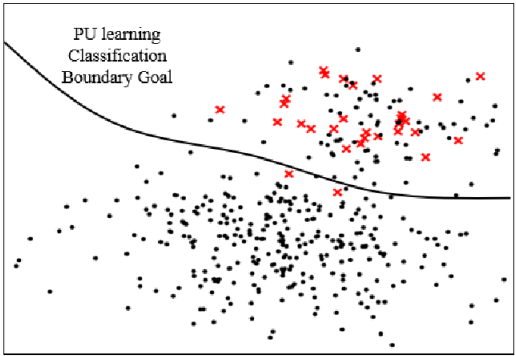




Apprentissage semi-supervisé : certains $\boldsymbol{x}$ n'ont pas de y



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

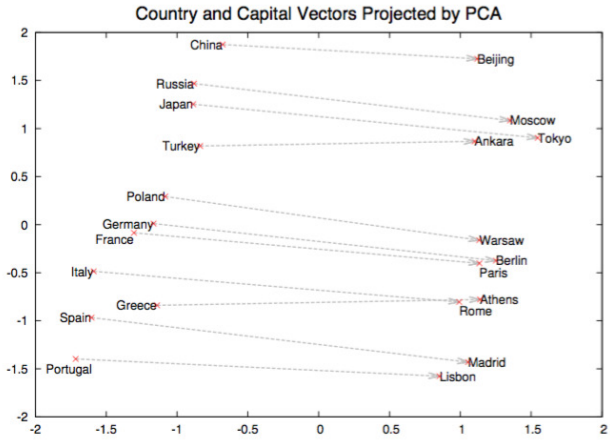


(b) Positive Unlabeled learning problem with only a percentage of known positive labels.

« PU » learning pour « positive unlabeled »

Apprentissage auto-supervisé

Prédire un mot à partir des mots autour (word2vec, Mikolov et al., 2013) → TP 3



Prédire le token suivant (GPT) → aussi TP 3

Apprentissage en contexte

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

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

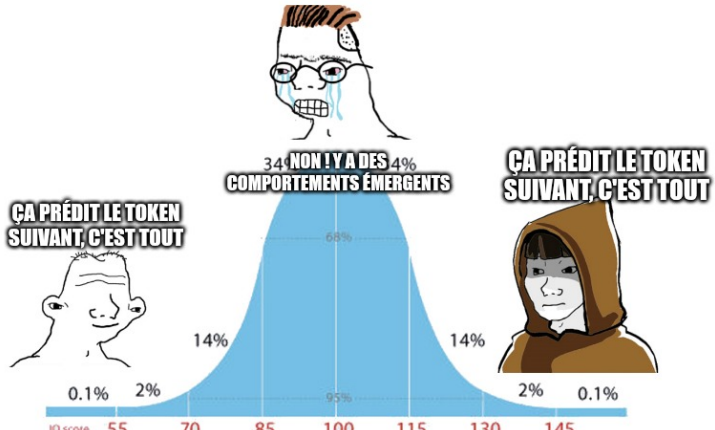
In-context learning: pretrain (LLM), predict(test, train)

Apprentissage en contexte

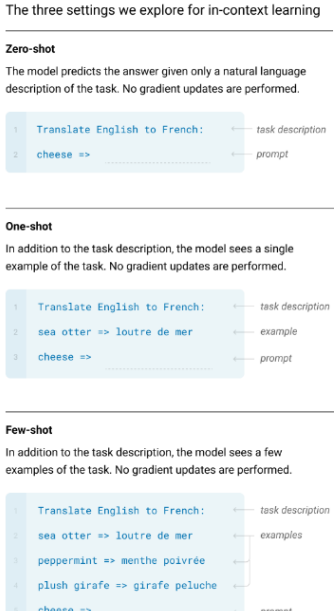
Traditional ML: fit(train), predict(test)

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

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



LLMs 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

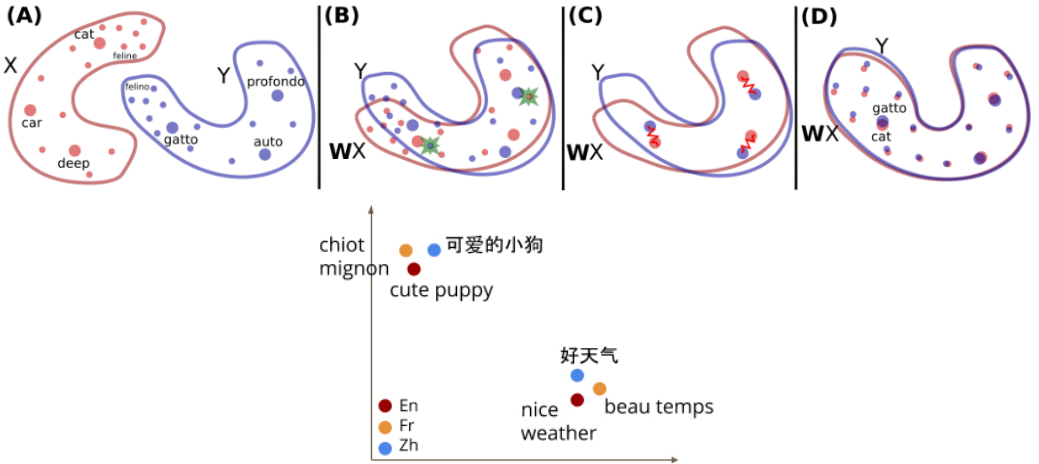
Open access

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.

Multilingual word embeddings (unsupervised)

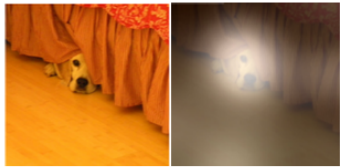


Guillaume Lample et al. (2018). « Word translation without parallel data ». In : *International Conference on Learning Representations*. URL : https://arxiv.org/abs/1710.04087

Légende automatique et attention (Xu and Bengio, 2015)

« C'est une boîte noire »

A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.

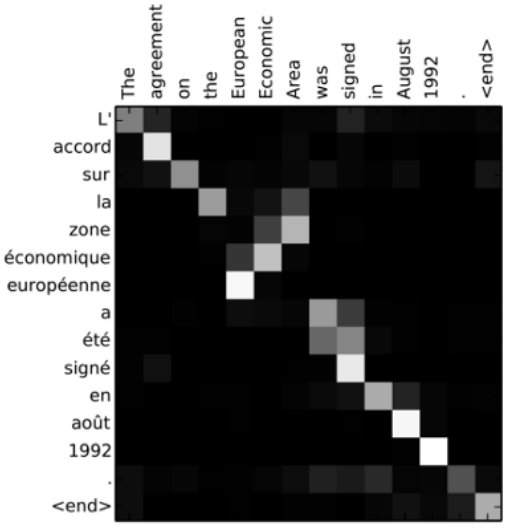
A stop sign is on a road with a mountain in the background.

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.

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



k plus proches voisins (Russell et Norvig, 2020, p. 687)

Algorithme k plus proches voisins parmi n en d dimensions

pour tout i de 1 à n **faire**
    calculer la distance de $\boldsymbol{x}$ à $\boldsymbol{x}_i$
identifier les k plus proches voisins de $\boldsymbol{x}$
calculer la moyenne de leurs valeurs, ou la classe majoritaire

- Complexité de trouver les voisins de $\boldsymbol{x}$ dans X ? $O(nd + kn)$

Pour une version recommandation dans des bases de données, cf. (Abiteboul et al., 2011, p. 365)

k plus proches voisins (Russell et Norvig, 2020, p. 687)

Algorithme k plus proches voisins parmi n en d dimensions

pour tout i de 1 à n **faire**
    calculer la distance de $\boldsymbol{x}$ à $\boldsymbol{x}_i$
identifier les k plus proches voisins de $\boldsymbol{x}$
calculer la moyenne de leurs valeurs, ou la classe majoritaire

- Complexité de trouver les voisins de $\boldsymbol{x}$ dans X ? $O(nd + kn)$
- Avec *median of medians* : $O(nd)$

Pour une version recommandation dans des bases de données, cf. (Abiteboul et al., 2011, p. 365)

k plus proches voisins (Russell et Norvig, 2020, p. 687)

Algorithme k plus proches voisins parmi n en d dimensions

pour tout i de 1 à n **faire**
    calculer la distance de $\boldsymbol{x}$ à $\boldsymbol{x}_i$
identifier les k plus proches voisins de $\boldsymbol{x}$
calculer la moyenne de leurs valeurs, ou la classe majoritaire

- Complexité de trouver les voisins de $\boldsymbol{x}$ dans X ? $O(nd + kn)$
- Avec *median of medians* : $O(nd)$
- Peut être accéléré avec un K-*d tree* en $O((n + K) \log n)$

Pour une version recommandation dans des bases de données, cf. (Abiteboul et al., 2011, p. 365)

k plus proches voisins (Russell et Norvig, 2020, p. 687)

Algorithme k plus proches voisins parmi n en d dimensions

pour tout i de 1 à n **faire**
    calculer la distance de $\boldsymbol{x}$ à $\boldsymbol{x}_i$
identifier les k plus proches voisins de $\boldsymbol{x}$
calculer la moyenne de leurs valeurs, ou la classe majoritaire

- Complexité de trouver les voisins de $\boldsymbol{x}$ dans X ? $O(nd + kn)$
- Avec *median of medians* : $O(nd)$
- Peut être accéléré avec un K-*d tree* en $O((n + K) \log n)$
- Product quantization : $O(n + k \log k \log \log n)$ (algorithme d'approximation)

Pour une version recommandation dans des bases de données, cf. (Abiteboul et al., 2011, p. 365)

Product Quantization (2010) Vector Databases (2022)

Une structure de données pour calculer des k plus proches voisins $\text{argmax}_{i \in S} q^T x_i$ (MIPS : *maximum inner product search*)

- K-dim tree
- en grande dimension, k plus proches voisins approchés (ex. FAISS) : *product quantization* (Jégou, Douze and Schmid, 2010), des objets proches doivent avoir des hash proches[1] (*locality-sensitive hashing*)

Variante : k plus proches voisins pondérés

Similarité $w_i = \boldsymbol{x}_{test}^T \boldsymbol{x}_{train_i} / N_z$ (renormalisés pour sommer à 1)
Je prédis pour $\boldsymbol{x}_{test}$: $\hat{y} = \sum_i w_i y_{train_i}$
Tiens mais l'attention c'est quoi ? $softmax(QK^T)V$
Une combinaison linéaire de valeurs V à poids correspondant à des produits scalaires (similarités) entre la requête Q et les clés K
Comprendre les KNN, c'est une bonne étape pour comprendre l'attention

[1] À ne pas utiliser pour hacher des mots de passe SVP

Récapitulatif et bonus

Apprentissage supervisé : caractéristiques X et étiquettes y

Apprentissage non supervisé : apprendre une représentation des données X, ex. réduire la dimension

Apprentissage par renforcement : apprendre en interagissant avec un environnement

Apprentissage semi-supervisé : certains X n'ont pas de y associé

Apprentissage auto-supervisé : prédire des parties de X à partir de lui-même

Apprentissage en contexte : prédire y_{test} à partir de $X_{test}, X_{train}, y_{train}$, pas d'entraînement, seulement de l'inférence

Méthodes paramétriques (réseaux de neurones)

Méthodes non paramétriques (k plus proches voisins, arbres de décision)

Formalisation apprentissage supervisé

Échantillons : $(\boldsymbol{x}_i, y_i) \in \boldsymbol{X} \times \mathcal{Y}$

Les caractéristiques $\boldsymbol{x}_i$ et les étiquettes y_i.

Habituellement $\boldsymbol{X} = \mathbb{R}^d$
Si $\mathcal{Y}$ est discret : classification, sinon régression.

Modèle $f_\theta : \boldsymbol{X} \to \mathcal{Y}$ ayant des paramètres θ

Une fonction d'erreur $\mathcal{L}$ à minimiser qui dépend des paramètres θ.

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

Régression linéaire (ATTENTION, HORS PROGRAMME !!!)

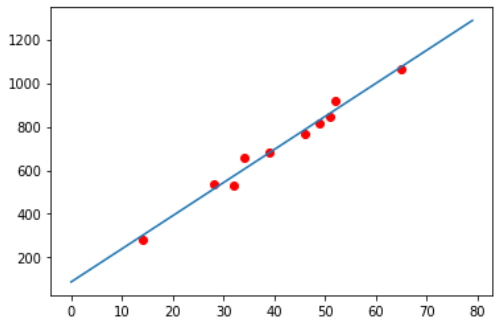
Modèle $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = \boldsymbol{x}_i^T \boldsymbol{\theta} \simeq y_i$

Erreur : moindres carrés
$\mathcal{L} = \sum_i ||\boldsymbol{x}_i^T \boldsymbol{\theta} - y_i||^2$

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

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

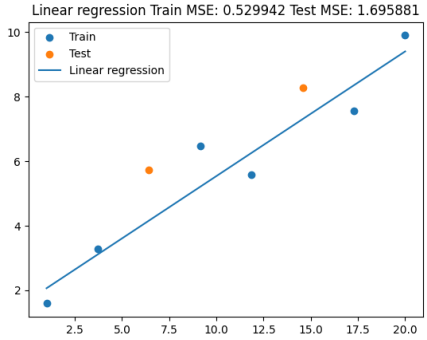
$$\boldsymbol{\theta} = (X^T X)^{-1} X^T \boldsymbol{y}$$



Risque de surapprentissage

Fitting

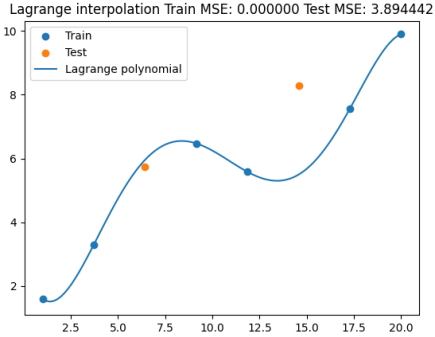
Polynomial degree 1 $Y = wX + b$



Linear regression
`scipy.stats.linreg`

Overfitting

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



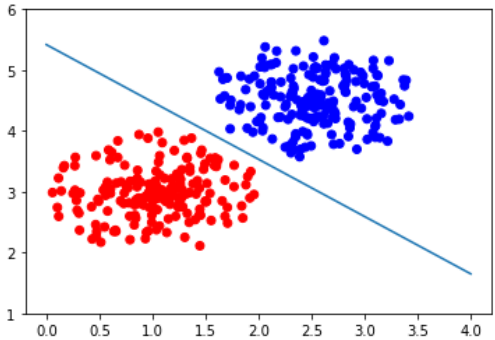
Lagrange interpolation
`scipy.interpolation.lagrange`

Régression logistique (classification)

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

Pratique car $\sigma' = \sigma(1 - \sigma)$

Pas de formule close pour $\boldsymbol{\theta}$ donc apprentissage de $\boldsymbol{\theta}$ par méthode de Newton



Algorithme du perceptron (Rosenblatt, 1957) est une couche linéaire (remplacer σ par la fonction en escalier Heaviside)

Variante : reconnaissance de caractères
Remplacer σ par un softmax : régression logistique multinomiale

Réseau de neurones

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

La ℓ-ième couche a d_ℓ neurones. Couche $\ell = 0$ d'entrée, $\ell = L$ de sortie.
σ est la fonction d'activation (non linéaire), ex. $\sigma = \text{ReLU} = \max(\mathbf{0}, \boldsymbol{x})$. Le code :

```
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)
)
```

Mais du coup ? Je peux le représenter comme $d_0 \to d_1 \to \cdots \to d_L$

Réseau de neurones

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

La ℓ-ième couche a d_ℓ neurones. Couche $\ell = 0$ d'entrée, $\ell = L$ de sortie.
σ est la fonction d'activation (non linéaire), ex. $\sigma = \text{ReLU} = \max(\mathbf{0}, \boldsymbol{x})$. Le code :

```
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)
)
```

Mais du coup ? Je peux le représenter comme $d_0 \to d_1 \to \cdots \to d_L$

Cette notation simplifie cet exposé

Régression linéaire ou logistique : $n \to 1$

Perceptron pour la reconnaissance des chiffres :

Cette notation simplifie cet exposé

Régression linéaire ou logistique : $n \to 1$

Perceptron pour la reconnaissance des chiffres : $n \to 10$

L'apprentissage de représentations

Autoencodeur $n \to \cdots \to k \to \cdots \to n$

Analyse en composantes principales PCA :

Cette notation simplifie cet exposé

Régression linéaire ou logistique : $n \to 1$

Perceptron pour la reconnaissance des chiffres : $n \to 10$

L'apprentissage de représentations

Autoencodeur $n \to \cdots \to k \to \cdots \to n$

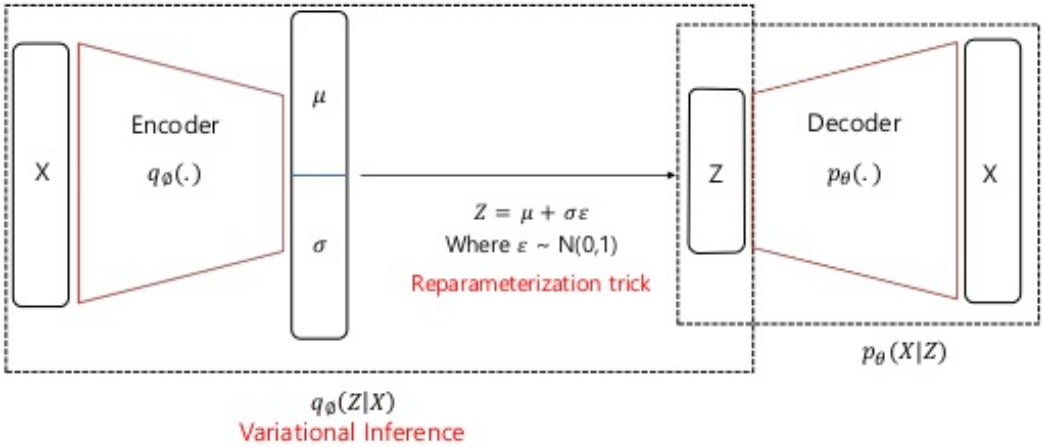
Analyse en composantes principales PCA : $n \to k \to n$ (sans les σ non linéaires)

Ainsi, une PCA est un autoencodeur linéaire à une couche

(k-SVD : meilleure approximation de rang k, on tronque aux k plus grandes valeurs singulières, cf. TP 1 et TP 3)

Extrait du MOOC de Hinton (2012)

Variational autoencoders (Kingma 2014)



Contrôler la génération par une classe, ou un prompt

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

Un étudiant de master à l'université de Kyoto (Yanghua Jin) a fait ceci :

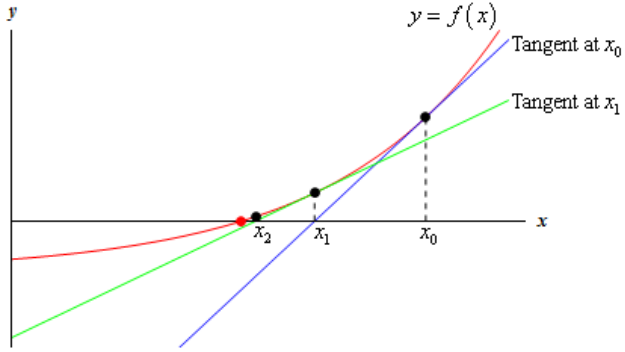
Why SGD?

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 $\nabla_{\boldsymbol{x}} g$.

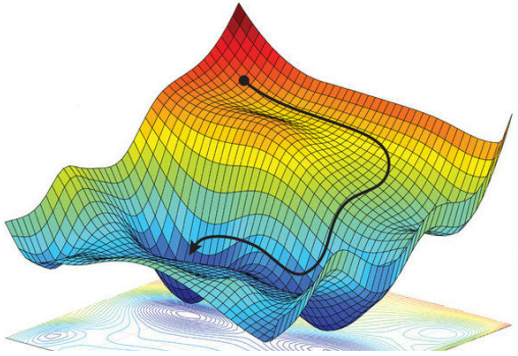
In higher dimension

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



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

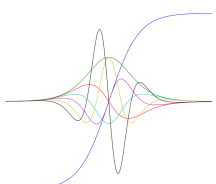
```
for each epoch:
    for x, y in dataset:
        compute gradients ∂L/∂θ (f_θ(x), y)   # also noted ∇_θ L
        θ ← θ − γ ∇_θ L   # γ is the learning rate
```

SGD in PyTorch

```python
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()
```

How to compute gradients automatically? What's in `loss.backward()`?

```
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()
```



Existing methods and their limitations

Numerical differentiation

$$\frac{f(x+h)-f(x)}{h}$$

Round-off errors

Existing methods and their limitations

Numerical differentiation

$$\frac{f(x+h) - f(x)}{h}$$

Round-off errors

Symbolic differentiation

Have to keep symbolic expressions at each step of the process

Existing methods and their limitations

Numerical differentiation

$$\frac{f(x+h) - f(x)}{h}$$

Round-off errors

Symbolic differentiation

Have to keep symbolic expressions at each step of the process

Automatic differentiation

Chain rule

$$(f \circ g)' = g' \cdot (f' \circ g)$$

Generalized chain rule

$$\frac{df_1}{dx} = \frac{df_1}{df_2}\frac{df_2}{df_3}\cdots\frac{df_n}{dx}$$

Reminder: Jacobians

Consider differentiable $f : \mathbb{R}^n \to \mathbb{R}^m$, its Jacobian $J_f \in \mathbb{R}^{m \times n}$ contains its first-order partial derivatives $(J_f)_{ij} = \dfrac{\partial f_i}{\partial x_j}$:

$$J_f = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} & \cdots & \dfrac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Reminder: Jacobians

Consider differentiable $f : \mathbb{R}^n \to \mathbb{R}^m$, its Jacobian $J_f \in \mathbb{R}^{m \times n}$ contains its first-order partial derivatives $(J_f)_{ij} = \dfrac{\partial f_i}{\partial x_j}$:

$$J_f = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} & \cdots & \dfrac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Example: linear layer

W is a $m \times n$ matrix

If $f(\boldsymbol{x}) = W\boldsymbol{x}$

$f_i = W_i \boldsymbol{x} = \sum_j W_{ij} x_j$ where W_i is ith row of W

$(J_f)_{ij} = \dfrac{\partial f_i}{\partial x_j} = W_{ij}$ so $J_f = W$

Generalized multivariate chain rule

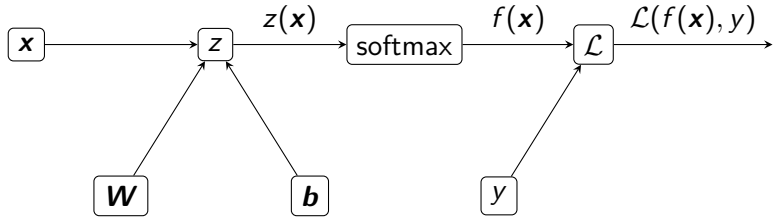
Consider differentiable $f : \mathbb{R}^m \to \mathbb{R}^k, g : \mathbb{R}^n \to \mathbb{R}^m$ and $\boldsymbol{a} \in \mathbb{R}^n$.

$$D_{\boldsymbol{a}}(f \circ g) = D_{g(\boldsymbol{a})} f \circ D_{\boldsymbol{a}} g$$

So the Jacobians verify:

$$J_{f \circ g} = (J_f \circ g) J_g$$

Graphe orienté acyclique (DAG) de calcul dérivable

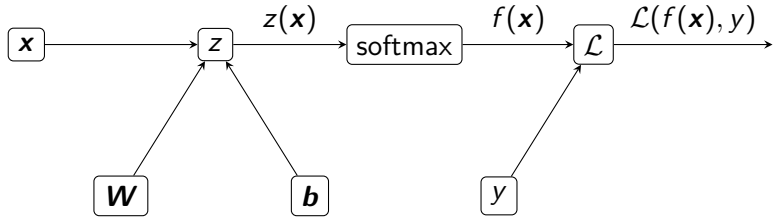


$$\frac{d\mathcal{L}}{db_1} = \frac{d\mathcal{L}}{df}\frac{df}{db_1} = \frac{d\mathcal{L}}{df}\left(\frac{df}{dz}\frac{dz}{db_1}\right) \text{ (forward)}$$
$$= \frac{d\mathcal{L}}{dz}\frac{dz}{db_1} = \left(\frac{d\mathcal{L}}{df}\frac{df}{dz}\right)\frac{dz}{db_1} \text{ (backward)}$$

Properly written: $J_{\mathcal{L}\circ\text{softmax}\circ z} = (J_{\mathcal{L}} \circ f)(J_{\text{softmax}} \circ z)J_z$

Given that $\boldsymbol{x} \in \mathbb{R}^d, z(\boldsymbol{x}), f(\boldsymbol{x}) \in \mathbb{R}^{d_2}, \mathcal{L}(f(\boldsymbol{x}), y) \in \mathbb{R}$,
which order is better?

Graphe orienté acyclique (DAG) de calcul dérivable



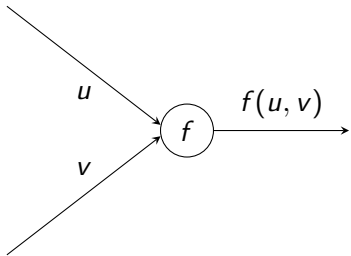
$$\frac{d\mathcal{L}}{db_1} = \frac{d\mathcal{L}}{df}\frac{df}{db_1} = \frac{d\mathcal{L}}{df}\left(\frac{df}{dz}\frac{dz}{db_1}\right) \text{ (forward)}$$
$$= \frac{d\mathcal{L}}{dz}\frac{dz}{db_1} = \left(\frac{d\mathcal{L}}{df}\frac{df}{dz}\right)\frac{dz}{db_1} \text{ (backward)}$$

Properly written: $J_{\mathcal{L}\circ\text{softmax}\circ z} = (J_{\mathcal{L}} \circ f)(J_{\text{softmax}} \circ z)J_z$

Given that $\boldsymbol{x} \in \mathbb{R}^d, z(\boldsymbol{x}), f(\boldsymbol{x}) \in \mathbb{R}^{d_2}, \mathcal{L}(f(\boldsymbol{x}), y) \in \mathbb{R}$,
which order is better? This is why backpropagation.

Reverse accumulation (in $\mathbb{R}$)

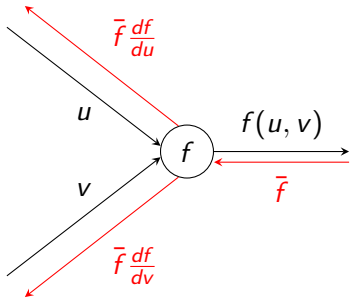
Let us note the adjoint $\overline{f} \triangleq \frac{d\mathcal{L}}{df}$.



Éviter de recalculer deux fois la même chose (mémoïsation)

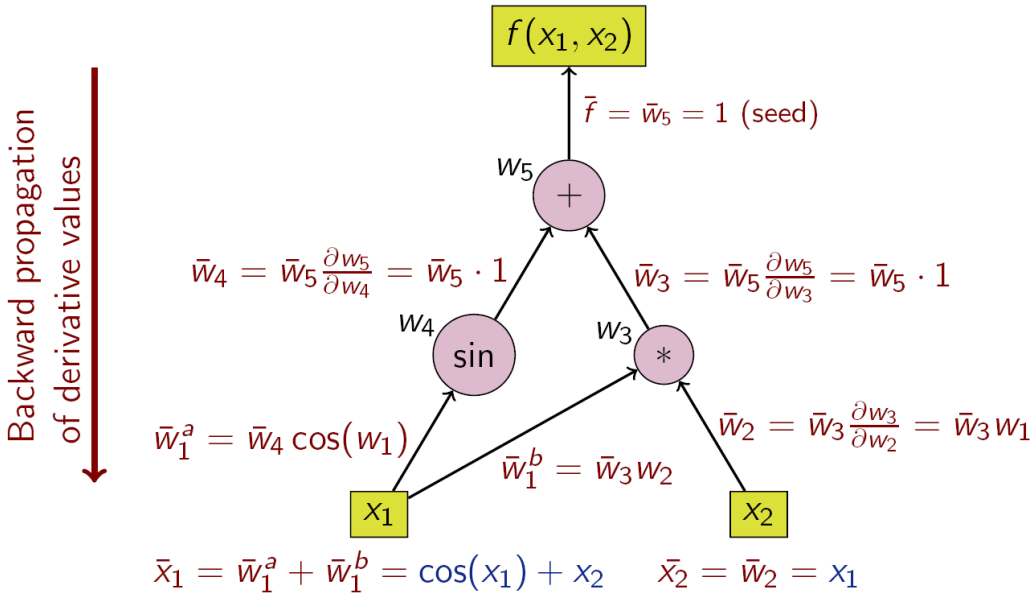
Reverse accumulation (in $\mathbb{R}$)

Let us note the adjoint $\overline{f} \triangleq \frac{d\mathcal{L}}{df}$.



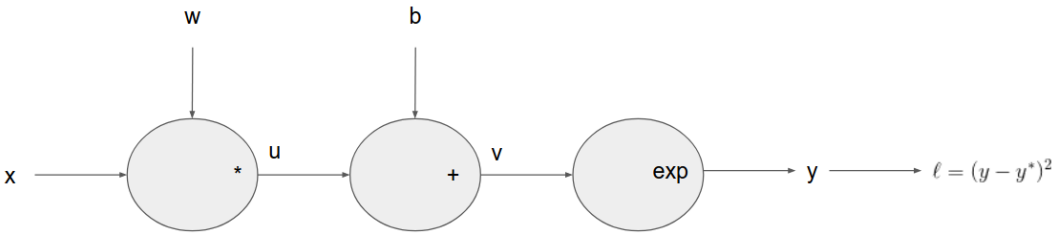
Éviter de recalculer deux fois la même chose (mémoïsation)

A complete example (Wikipedia)



Notebook: compute gradients

FORWARD PASS



$u = wx$
$v = u + b$
$y = e^v$
$\ell = (y - y^*)^2$

Question (piège ?)

Pourquoi A* c'est de l'IA ?

Question (piège ?)

Pourquoi A* c'est de l'IA ?

Est-ce que l'algorithme de Dijkstra c'est de l'IA aussi ?

Parce que les deux sont Best-first search

Algorithme Recherche du meilleur en premier

Mettre la *source* dans la file
tant que la file n'est pas vide **faire**
  Extraire le nœud u ayant la priorité $f(u)$ minimale
  si u est la *cible* **alors renvoyer** dist, prec
  pour tout voisin v du nœud u **faire**
    candidat $\leftarrow$ dist(u) + poids arête w_{uv}
    si c'est un meilleur candidat i.e. candidat $<$ dist(voisin) **alors**
      dist(voisin v) $\leftarrow$ candidat
      prec(voisin v) $\leftarrow$ nœud u
      Ajouter v à la file avec priorité $f(v)$

Valeurs de priorité $f(u)$

- Dijkstra : parcourir par plus courte distance(source, nœud u)
- *Greedy best-first search* : $h(u)$ distance à vol d'oiseau à l'arrivée
- A* : par $dist(u) + h(u)$ croissant.
- Prim (arbre couvrant) : distance à l'arbre en cours

Principe d'optimalité de Bellman

> *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

Équation de Bellman

Soit un état s, on choisit une action a qui nous rapporte une récompense $R(s, a)$ et nous met dans un état s'. V indique la récompense moyenne obtenue si l'on agit de façon optimale.

$$V(s) = \max_a R(s, a) + \gamma V(s')$$

Ça vous rappelle quelque chose ?

Principe d'optimalité de Bellman

> *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

Équation de Bellman

Soit un état s, on choisit une action a qui nous rapporte une récompense $R(s, a)$ et nous met dans un état s'. V indique la récompense moyenne obtenue si l'on agit de façon optimale.

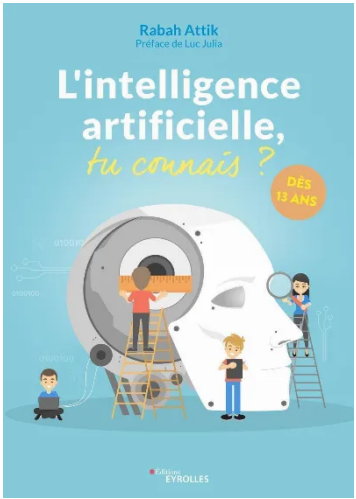
$$V(s) = \max_a R(s, a) + \gamma V(s')$$

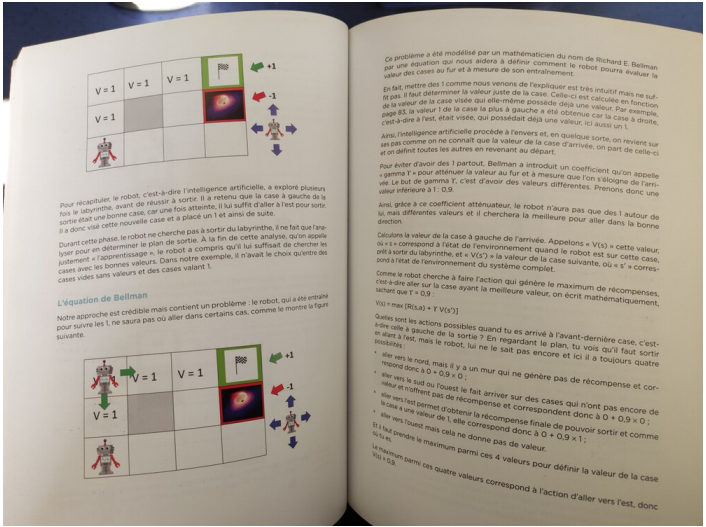
Ça vous rappelle quelque chose ?

Bellman-Ford

$$V(u) = \max_v -w_{u,v} + V(v) \quad V = -d$$

Trouvé dans une médiathèque





Pour récapituler, le robot, c'est-à-dire l'intelligence artificielle, a exploré plusieurs fois le labyrinthe, avant de réussir à sortir. Il a retenu que la case à gauche de la sortie était une bonne case, car une fois atteinte, il lui suffit d'aller à l'est pour sortir. Il a donc visé cette nouvelle case et a placé un 1 et ainsi de suite.

Durant cette phase, le robot ne cherche pas à sortir du labyrinthe, il ne fait que l'analyser pour en déterminer le plan de sortie. À la fin de cette analyse, qu'on appelle justement « l'apprentissage », le robot a compris qu'il lui suffisait de chercher les cases avec les bonnes valeurs. Dans notre exemple, il n'avait le choix qu'entre des cases vides sans valeurs et des cases valant 1.

L'équation de Bellman

Notre approche est crédible mais contient un problème : le robot, qui a été entraîné pour suivre les 1, ne saura pas où aller dans certains cas, comme le montre la figure suivante.

Ce problème a été modélisé par un mathématicien du nom de Richard E. Bellman par une équation qui nous aidera à définir comment le robot pourra évaluer la valeur des cases au fur et à mesure de son entraînement.

En fait, mettre des 1 comme nous venons de l'expliquer est très intuitif mais ne suffit pas. Il faut déterminer la valeur juste de la case. Celle-ci est calculée en fonction de la valeur de la case visée qui elle-même possède déjà une valeur. Par exemple, page 83, la valeur 1 de la case la plus à gauche a été obtenue car la case à droite, c'est-à-dire à l'est, était visée, qui possédait déjà une valeur, ici aussi un 1.

Ainsi, l'intelligence artificielle procède à l'envers et, en quelque sorte, on revient sur ses pas comme on ne connaît que la valeur de la case d'arrivée, on part de celle-ci et on définit toutes les autres en revenant au départ.

Pour éviter d'avoir des 1 partout, Bellman a introduit un coefficient qu'on appelle « gamma Γ » pour atténuer la valeur au fur et à mesure que l'on s'éloigne de l'arrivée. Le but de gamma Γ, c'est d'avoir des valeurs différentes. Prenons donc une valeur inférieure à 1 : 0,9.

Ainsi, grâce à ce coefficient atténuateur, le robot n'aura pas que des 1 autour de lui, mais différentes valeurs et il cherchera la meilleure pour aller dans la bonne direction.

Calculons la valeur de la case à gauche de l'arrivée. Appelons « V(s) » cette valeur, où « s » correspond à l'état de l'environnement quand le robot est sur cette case, prêt à sortir du labyrinthe, et « V(s') » la valeur de la case suivante, où « s' » correspond à l'état de l'environnement du système complet.

Comme le robot cherche à faire l'action qui génère le maximum de récompenses, c'est-à-dire aller sur la case ayant la meilleure valeur, on écrit mathématiquement, sachant que Γ = 0,9 :

$$V(s) = \max_{a}(R(s,a) + \Gamma V(s'))$$

Quelles sont les actions possibles quand tu es arrivé à l'avant-dernière case, c'est-à-dire celle à gauche de la sortie ? En regardant le plan, tu vois qu'il faut sortir en allant à l'est, mais le robot, lui ne le sait pas encore et ici il a toujours quatre possibilités :

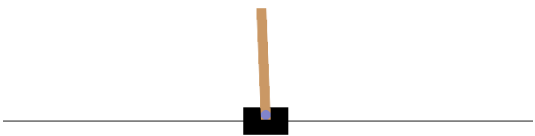
- aller vers le nord, mais il y a un mur qui ne génère pas de récompense et correspond donc à 0 + 0,9 × 0 ;
- aller vers le sud ou l'ouest le fait arriver sur des cases qui n'ont pas encore de valeur et n'offrent pas de récompense et correspondent donc à 0 + 0,9 × 0 ;
- aller vers l'est permet d'obtenir la récompense finale de pouvoir sortir et comme la case a une valeur de 1, elle correspond donc à 0 + 0,9 × 1 ;
- aller vers l'ouest mais cela ne donne pas de valeur.

Et il faut prendre le maximum parmi ces 4 valeurs pour définir la valeur de la case où tu es.

Le maximum parmi ces quatre valeurs correspond à l'action d'aller vers l'est, donc 1 + 0,9.

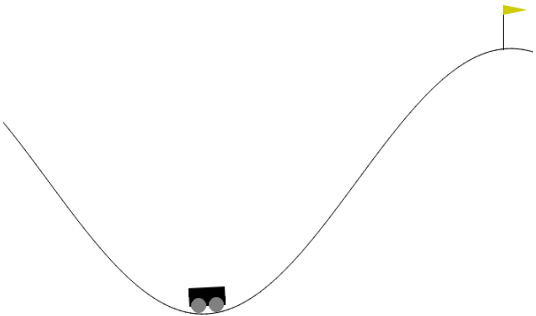
Exemples d'environnements d'apprentissage par renforcement

CartPole-v1



$S = (x, \dot{x}, \theta, \dot{\theta}) \in \mathbb{R}^4$
$A \in \{\leftarrow, \rightarrow\}$
$R = 1$

MountainCar-v2



$S = (x, \dot{x}) \in \mathbb{R}^2$
$A \in \{\leftarrow, 0, \rightarrow\}$
$R = -1$

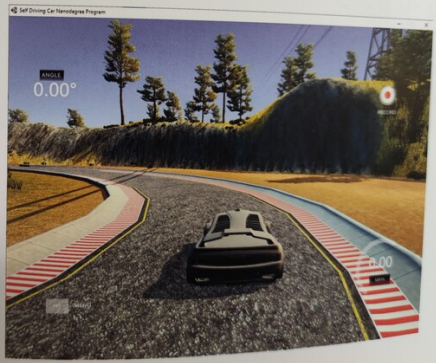
Dans le même bouquin, quelques pages plus loin

Code une voiture autonome avec Python

Dans cet exercice, tu vas jouer en ligne et entraîner une voiture autonome. Comment vas-tu procéder ? Avant le début de la course sur le circuit, tu pourras activer un enregistrement qui récupérera les multiples images de la vidéo de ton jeu qui serviront à générer un modèle basé sur un réseau de neurones à convolution. Te rappelles-tu des réseaux spécialisés pour la reconnaissance d'images ?

Nous nous appuierons sur les outils développés par NVIDIA, une société américaine spécialisée dans la fabrication de cartes graphiques qui développe depuis 2017 de nouvelles cartes pour l'intelligence artificielle.

Les voitures autonomes sont l'un des domaines de recherche et d'activité les plus dynamiques. Ce qui semblait relever de la



Gold mine problem (Bellman, 1952)

Two gold mines, Anaconda (amount x) and Bonanza (amount y).

- Anaconda: probability p_1 to collect $r_1 x$ $\quad$ $1 - p_1$ to break the machine forever
- Bonanza: probability q_1 to collect $r_2 y$ $\quad$ $1 - q_1$ to break the machine forever

Parameters of the problem: $p_1, q_1, r_1, r_2 \in [0, 1]$, $x, y \in \mathbb{R}^+$.

What is the optimal action, that maximizes the amount of extracted gold?

Gold mine problem (Bellman, 1952)

Two gold mines, Anaconda (amount x) and Bonanza (amount y).

- Anaconda: probability p_1 to collect $r_1 x$ $\quad$ $1 - p_1$ to break the machine forever
- Bonanza: probability q_1 to collect $r_2 y$ $\quad$ $1 - q_1$ to break the machine forever

Parameters of the problem: $p_1, q_1, r_1, r_2 \in [0, 1]$, $x, y \in \mathbb{R}^+$.

What is the optimal action, that maximizes the amount of extracted gold?

Let $f(x, y)$ be the maximum expected gold extracted.

Solution to the gold mine problem

$$f(x,y) = \max \left\{ \begin{array}{l} p_1(r_1 x + f((1-r_1)x, y) \\ q_1(r_2 y + f(x, (1-r_2)y) \end{array} \right\}.$$

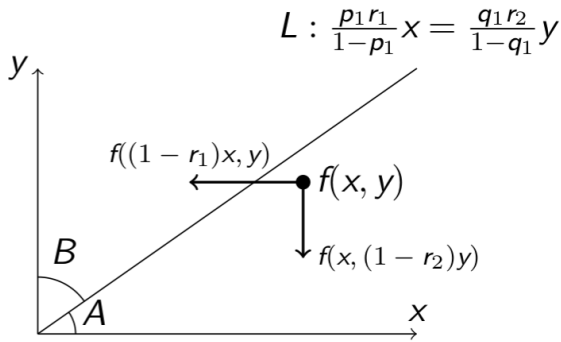
Solution

Si $\dfrac{p_1 r_1}{1-p_1} x > \dfrac{q_1 r_2}{1-q_1} y$,

Choisir A

Sinon

Choisir B.

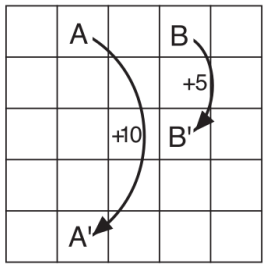


Policy evaluation on Gridworld (not a DAG)

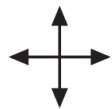
$$S \in \{(i,j)\}_{1 \le i,j \le 5}$$

$$A \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$$

$$R = \begin{cases} 10 \text{ if leaving from A} \\ 5 \text{ if leaving from B} \\ -1 \text{ if hitting a wall} \\ 0 \text{ otherwise.} \end{cases}$$



Gridworld

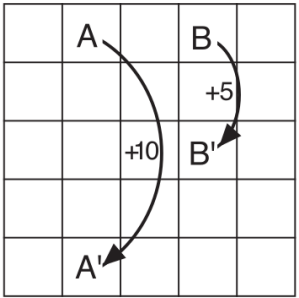


Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

$\pi_{\text{uniform}} = (1/4, 1/4, 1/4, 1/4)$ $v_{\pi_{\text{uniform}}}$

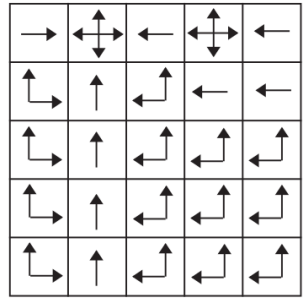
Policy improvement on Gridworld



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

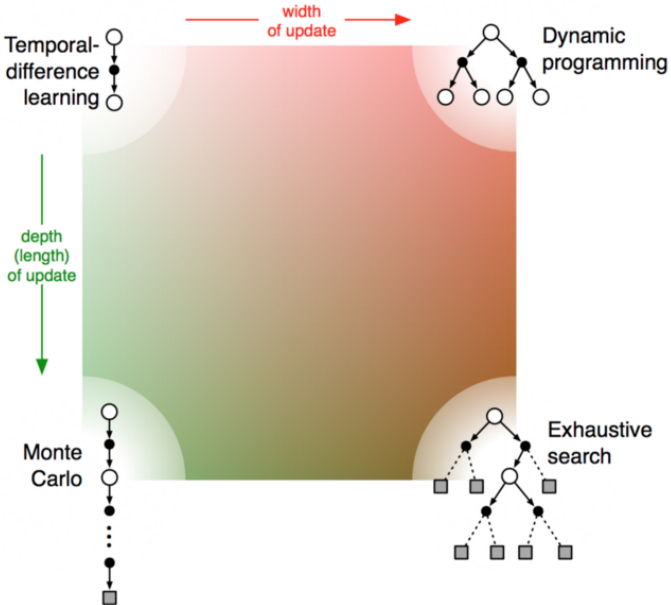
La programmation dynamique de Richard Bellman (1920-1984)

1952 : Richard Bellman publie *Programmation dynamique*

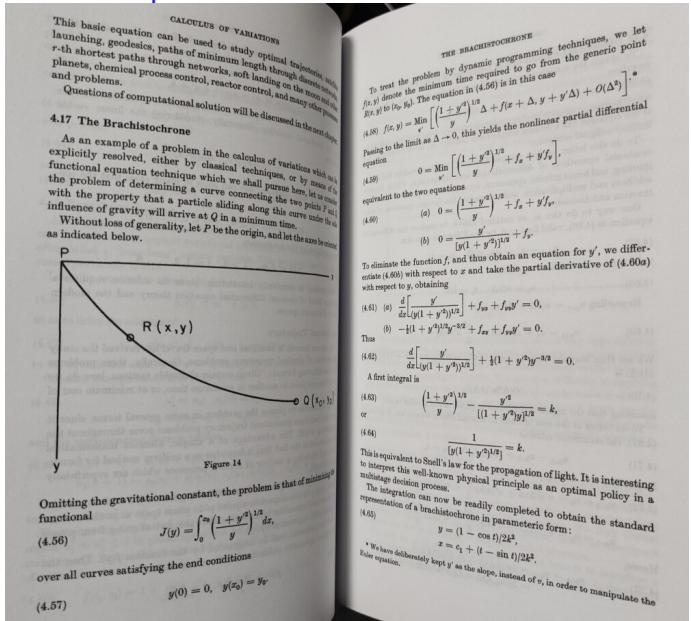
1952 : Autocode, le premier langage de programmation

Il avait la présence d'esprit de résoudre des problèmes (de maths appliquées) par prises de décision optimales successives (*multistage decision processes*), de façon constructive

La programmation dynamique a mené à l'apprentissage par renforcement



Trouvé dans la bibliothèque du CIRM



CALCULUS OF VARIATIONS

This basic equation can be used to study optimal trajectories, including launching, geodesics, paths of minimum length through discrete networks, r-th shortest paths through networks, soft landing on the moon and planets, chemical process control, reactor control, and many other problems.

Questions of computational solution will be discussed in the next chapter.

4.17 The Brachistochrone

As an example of a problem in the calculus of variations which can be explicitly resolved, either by classical techniques, or by means of a functional equation technique which we shall pursue here, let us consider the problem of determining a curve connecting the two points P and Q, with the property that a particle sliding along this curve under the influence of gravity will arrive at Q in a minimum time.

Without loss of generality, let P be the origin, and let the axes be oriented as indicated below.

Figure 14

Omitting the gravitational constant, the problem is that of minimizing the functional

$$J(y) = \int_0^{x_0} \left(\frac{1+y'^2}{y}\right)^{1/2} dx, \tag{4.56}$$

over all curves satisfying the end conditions

$$y(0) = 0, \quad y(x_0) = y_0. \tag{4.57}$$

THE BRACHISTOCHRONE

To treat the problem by dynamic programming techniques, we let $f(x, y)$ denote the minimum time required to go from the generic point (x, y) to (x_0, y_0). The equation in (4.56) is in this case

$$f(x, y) = \min_{y'} \left[\left(\frac{1+y'^2}{y}\right)^{1/2} \Delta + f(x+\Delta, y+y'\Delta) + O(\Delta^2)\right].^* \tag{4.58}$$

Passing to the limit as $\Delta \to 0$, this yields the nonlinear partial differential equation

$$0 = \min_{y'} \left[\left(\frac{1+y'^2}{y}\right)^{1/2} + f_x + y' f_y\right], \tag{4.59}$$

equivalent to the two equations

$$(a) \quad 0 = \left(\frac{1+y'^2}{y}\right)^{1/2} + f_x + y' f_y,$$

$$(b) \quad 0 = \frac{y'}{[y(1+y'^2)]^{1/2}} + f_y. \tag{4.60}$$

To eliminate the function f, and thus obtain an equation for y', we differentiate (4.60) with respect to x and take the partial derivative of (4.60a) with respect to y, obtaining

$$(a) \quad \frac{d}{dx}\left[\frac{y'}{[y(1+y'^2)]^{1/2}}\right] + f_{xy} + f_{yy} y' = 0,$$

$$(b) \quad -\tfrac{1}{2}(1+y'^2)^{1/2} y'^{-3/2} + f_{xy} + f_{yy} y' = 0. \tag{4.61}$$

Thus

$$\frac{d}{dx}\left[\frac{y'}{[y(1+y'^2)]^{1/2}}\right] + \tfrac{1}{2}(1+y'^2) y'^{-3/2} = 0. \tag{4.62}$$

A first integral is

$$\left(\frac{1+y'^2}{y}\right)^{1/2} - \frac{y'^2}{[(1+y'^2)y]^{1/2}} = k, \tag{4.63}$$

or

$$\frac{1}{[y(1+y'^2)^{1/2}]} = k. \tag{4.64}$$

This is equivalent to Snell's law for the propagation of light. It is interesting to interpret this well-known physical principle as an optimal policy in a multistage decision process.

The integration can now be readily completed to obtain the standard representation of a brachistochrone in parametric form:

$$y = (1 - \cos t)/2k^2,$$
$$x = c_1 + (t - \sin t)/2k^2. \tag{4.65}$$

* We have deliberately kept y' as the slope, instead of v, in order to manipulate the Euler equation.

Les grands modèles de langage (LLM) : quels objectifs ?

1. Transformer : prédire le mot suivant

```
Transformers / are / a / new / machine / [learning]
Transformers / are / a / new / machine / learning / [architecture]
```

2. Données de démonstration d'experts :

```
Query: put the first letters in uppercase in "optimizing human learning"
Answer: Optimizing Human Learning
```

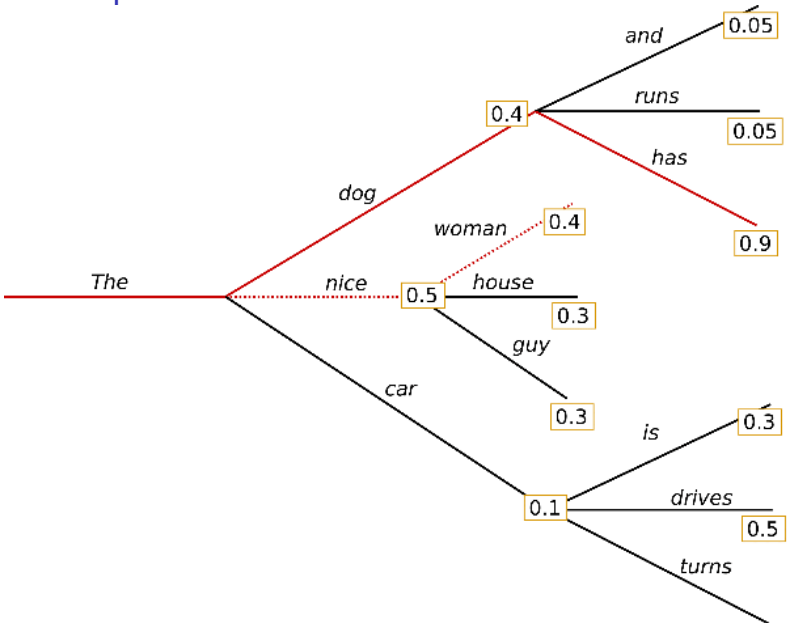
3. Données de comparaison d'experts :

```
Query: write a poem
Answer 1: Roses are red
Answer 2: Once upon a time, a prince in a castle
```

Où la réponse 2 est votée favorablement par les experts

ChatGPT est donc entraîné pour plaire aux experts et généraliser à de nouvelles tâches

Recherche d'une réponse d'un LLM



La tâche est dans les données d'entrée

Machine de Turing : le programme est dans les données

Machine de Turing universelle : exécuter le *i*-ème programme

François Chollet dit ça beaucoup mieux :

> *My mental model for LLMs is that they work as a repository of vector programs. When prompted, they will fetch the program that your prompt maps to and « execute » it on the input at hand. LLMs are a way to store and operationalize millions of useful mini-programs via passive exposure to human-generated content.*
>
> *If a LLM is like a database of millions of vector programs, then a prompt is like a search query in that database [. . .] this « program database » is continuous and interpolative — it's not a discrete set of programs.*

Sources: https://arcprize.org/blog/oai-o3-pub-breakthrough,
https://simonwillison.net/2023/Oct/25/francois-chollet/

Take home message

Nombreuses applications de l'IA, beaucoup d'impacts sociétaux, essentiel pour faire rêver les jeunes (ou les inquiéter)

On peut regretter que l'informatique et l'IA ne s'entendent pas bien alors que les deux s'intéressent à des algorithmes qui optimisent un objectif

Je mise tout sur les *vector databases* et Bellman pour les réconcilier

Analyse (du second degré)

Le ou la prof de maths fait les fonctions à plusieurs variables, jacobiennes, vraiment à la fin de l'année parce que flemme et de toute façon il n'a pas l'intention de le coder

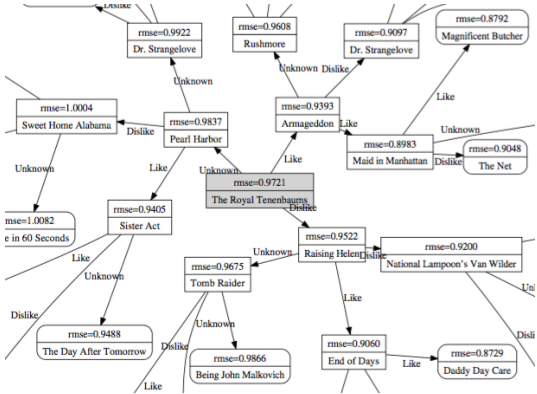
Le ou la prof d'info fait l'IA à la fin parce que « l'IA ça marche pas et de toute façon ça ne tombe jamais aux concours » (pourtant, X 2025) et 1) pf faut installer numpy sur les ordis 2) en SQL, C, OCaml, l'IA c'est infernal

TP 1 – Systèmes de recommandation : KNN, SVD, ALS, SGD

TP 2 – Reconnaissance de motifs OU Gridworld OU Quoridor

TP 3 – Traitement automatique des langues : chaîne de Markov, GPT-based LLM

On pourrait adapter le TP 1 en posant des questions informatives sur les goûts (à la ID3)



Merci pour votre attention

Si vous souhaitez parler de l'environnement de concours NonOS, n'hésitez pas, je ne mords pas

- Abiteboul, Serge et al. (2011). *Web Data Management*. Cambridge University Press. URL : http://webdam.inria.fr/Jorge/files/wdm.pdf.
- 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*. Sous la dir. de C.J. Burges et al. T. 26. Curran Associates, Inc. URL : https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.
- Russell, Stuart et Peter Norvig (2020). *Artificial Intelligence: A Modern Approach*.