École de technologie supérieure
Département des enseignements généraux
Local B-2500 514-396-8938
Site internet: `http://www.etsmtl.ca/`

MTI881

ASPECTS MATHÉMATIQUES DE L'APPRENTISSAGE PROFOND

NOTES DE COURS

RÉDIGÉES PAR
GUILLAUME ROY-FORTIN

MARS 2025

Table des matières

Avant-propos

Le deep learning n'a plus besoin de présentation : ses nombreuses prouesses sont largement documentées et célébrées. Mais sait-on vraiment ce qui rend ces réseaux profonds aussi habiles ? Pour plusieurs, l'apprentissage profond demeure une mystérieuse boîte noire qui produit des résultats étonnants en opérant toutefois selon des modalités opaques.

Or, une partie de la théorie derrière les exploits de ces réseaux de neurones se trouve au confluent de l'algèbre linéaire, des statistiques, du calcul différentiel et de la théorie de la probabilité, pour ne nommer que ceux-là. Autrement dit, les mathématiques peuvent éclairer un peu certains aspects de cette boîte noire. C'est ce que propose d'explorer le cours MTI881.

Ce cours intitulé *Aspects mathématiques de l'apprentissage profond* a été donné de nouveau à l'été 2022 et 2023. Il est destiné aux étudiants destiné aux étudiants gradués de génie logiciel et technologies de l'information. Le présent document contient les notes de cours pertinentes à la matière vue en classe. Il s'agit d'une première itération encore très incomplète à plein d'égards. Nous prions donc le lecteur d'être tolérant envers l'auteur !

Les inévitables erreurs et coquilles peuvent être signalées à l'auteur par courriel à l'adresse suivante : guillaume.roy-fortin@etsmtl.ca.

Guillaume Roy-Fortin,
Professeur enseignant à l'ÉTS.

Chapitre 1

Rappels d'algèbre linéaire et de calcul différentiel

Dans ce chapitre, nous allons brièvement survoler certains éléments essentiels qui seront nécessaires pour la compréhension des chapitres ultérieurs. Ce survol sera bref et il est donc recommandé au lecteur qui n'est pas familier avec ces notions de consulter l'une des nombreuses références disponibles sur l'algèbre linéaire ou le calcul différentiel avant de se plonger dans le reste du texte.

1.1 Espaces vectoriels euclidiens

1.1.1 Vecteurs et opérations sur les vecteurs

Définition 1.1 : Vecteur

Un **vecteur u** $\in \mathbb{R}^n$ est un $n-$tuple de nombres réels $u_1, u_2, ..., u_n$ que l'on peut représenter comme un vecteur colonne

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}$$

ou encore comme un vecteur rangée

$$\mathbf{u}^T = \begin{pmatrix} u_1 & u_2 & ... & u_n \end{pmatrix}.$$

Dans l'espace euclidien à n dimensions, on associe de manière unique le vecteur

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ ... \\ u_n \end{pmatrix}$$

à *n'importe quel* trajet équivalent à celui qui part de l'origine **0** et se terminant au point de coordonnées $(u_1, u_2, ..., u_n)$.

Exemple 1.1

Lorsque n est de basse dimension (2 ou 3) il est utile de représenter graphiquement les vecteurs. Par exemple, pour $\mathbf{u} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \in \mathbb{R}^2$, on a

Similairement, on peut représenter $\mathbf{u} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \in \mathbb{R}^3$ par

On présente maintenant les opérations de base que l'on peut exécuter sur les vecteurs.

Définition 1.2 : Opérations vectorielles

Soit $\mathbf{u} = \begin{pmatrix} u_1 \\ \dots \\ u_n \end{pmatrix}$ et $\mathbf{v} = \begin{pmatrix} v_1 \\ \dots \\ v_n \end{pmatrix}$ deux vecteurs de $\mathbb{R}^n$.

- L'**addition vectorielle** permet d'additioner deux vecteurs pour en créer un troisième :

$$\mathbf{u} + \mathbf{v} = \begin{pmatrix} u_1 + v_1 \\ \dots \\ u_n + v_n \end{pmatrix}.$$

- Soit $k \in \mathbb{R}$. On peut contracter, dilater (et inverser si $k < 0$) un vecteur $\mathbf{u}$ via la **multiplication scalaire**, qui est définie par

$$k\mathbf{u} = \begin{pmatrix} ku_1 \\ \dots \\ ku_n \end{pmatrix}.$$

Dans le cas de la multiplication par un scalaire, les cas $|k| > 1$ et $|k| < 1$ correspondent respectivement à une dilatation et à une contraction du vecteur original. Si $k < 0$, l'orientation du vecteur est inversée, c'est-à-dire qu'on inverse les points de départ et d'arrivée du trajet qu'il représente.

Exemple 1.2

Voici une interprétation géométrique des opérations que l'on vient de définir sur les vecteurs.

- Addition vectorielle :
- Multiplication scalaire :

Définition 1.3 : Produit scalaire

Soit $\mathbf{u}, \mathbf{b}$ deux vecteurs de $\mathbb{R}^n$. Le **produit scalaire** entre $\mathbf{u}$ et $\mathbf{v}$ est

$$(u, v) = \sum_{i=1}^{n} u_i v_i = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n.$$

Il importe de remarquer que le résultat du produit scalaire est un nombre que l'on peut interpréter comme une mesure de similarité entre deux vecteurs. On verra dans quelques instants de quelle manière exactement cette mesure de similarité peut être interprétée. Notons finalement que le produit scalaire est parfois appelé produit intérieur ou simplement par son nom anglais *« dot product »*.

Définition 1.4 : Norme

La **norme** d'un vecteur $\mathbf{u} \in \mathbb{R}^n$, notée $||\mathbf{u}||$, est calculée par

$$||\mathbf{u}|| = \sqrt{(u, u)} = \sqrt{u_1^2 + u_2^2 + \ldots + u_n^2}.$$

C'est un nombre non-négatif qui correspond à la longueur (ou magnitude) du vecteur $\mathbf{u}$.

Remarquons par ailleurs que dans le cas $n = 2$, la formule pour calculer la norme d'un vecteur n'est rien de plus que le théorème de Pythagore.

Exemple 1.3

Soit $\mathbf{u} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \in \mathbb{R}^2$. Alors,

$$(u, u) = 1^2 + 2^2 = 5$$

et

$$||\mathbf{u}|| = \sqrt{(u, u)} = \sqrt{5}.$$

Le prochain résultat fournit une interprétation géométrique du produit scalaire.

Théorème 1.1

Soit deux vecteurs $\mathbf{u}, \mathbf{v}$ et θ l'angle aigu entre eux. Alors,

$$(\mathbf{u}, \mathbf{v}) = ||\mathbf{u}|| \cdot ||\mathbf{v}|| \cdot \cos\theta.$$

Exemple 1.4

Soit $\mathbf{u} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ et $\mathbf{v} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Alors, $(u, v) = 2 \cdot 1 + 1 \cdot 0 = 2$, $||\mathbf{u}|| = \sqrt{5}$ et $||\mathbf{v}|| = 1$, de sorte que l'angle θ entre les deux vecteurs est donné par

$$\theta = \arccos\left(\frac{2}{\sqrt{5}}\right),$$

ce que l'on peut observer géométriquement

En fait, (u, v) nous fournit la longueur signée de la projection de $\mathbf{u}$ sur $\mathbf{v}$.

Un corollaire du théorème précédent est le critère suivant pour déterminer si deux vecteur sont perpendiculaires.

Théorème 1.2

Deux vecteurs $\mathbf{u}, \mathbf{v}$ sont orthogonaux (ou perpendiculaires) si et seulement

$$(\mathbf{u}, \mathbf{v}) = 0,$$

ce que l'on notera parfois par $\mathbf{u} \perp \mathbf{v}$.

Exemple 1.5

Soit les vecteurs $\mathbf{u} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$ et $\mathbf{v} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$. Alors $(\mathbf{u}, \mathbf{v}) = -1 \cdot 4 + 2 \cdot 2 = 0$. Or,

$$0 = (\mathbf{u}, \mathbf{v}) = ||\mathbf{u}|| \cdot ||\mathbf{v}|| \cos\theta,$$

d'où on tire directement que $\cos\theta = 0$ et donc que $\theta = \frac{\pi}{2}$ et $\mathbf{u} \perp \mathbf{v}$.

Théorème 1.3 : Inégalité de Cauchy-Schwarz

Soit $\mathbf{u}, \mathbf{v}$ des vecteurs de $\mathbb{R}^n$. Alors,

$$|(\mathbf{u}, \mathbf{v})| \leq ||\mathbf{u}|| \cdot ||\mathbf{v}||,$$

avec égalité si et seulement si $\mathbf{u}$ et $\mathbf{v}$ sont colinéaires, c'est-à-dire s'il existe un scalaire k tel que $\mathbf{u} = k\mathbf{v}$.

1.1.2 Combinaison linéaire, indépendance et base

Définition 1.5 : Combinaison linéaire

Soit $\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_k\}$ une collection de k vecteurs de $\mathbb{R}^n$. Une **combinaison linéaire** de ces vecteurs est une expression de la forme

$$\sum_{i=1}^{k} \alpha_i \mathbf{u}_i = \alpha_1 \mathbf{u}_1 + \alpha_2 \mathbf{u}_2 + ... + \alpha_k \mathbf{u}_k,$$

où les nombres α_i sont des réels appelés **coefficients** de la combinaison linéaire.

Exemple 1.6

Soit $\mathbf{u}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ et $\mathbf{u}_2 = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$. Alors,

$$\mathbf{v} = 3\mathbf{u}_1 + 2\mathbf{u}_2 = 3\begin{pmatrix} 1 \\ 1 \end{pmatrix} + 2\begin{pmatrix} -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

est une combinaison linéaire de $\{\mathbf{u}_1, \mathbf{u}_2\}$. Le portrait géométrique est le suivant :

L'exemple suivant présente une combinaison linéaire avec 3 vecteurs particuliers qui forment ce que l'on appelle la *base canonique* de $\mathbb{R}^3$.

Exemple 1.7

Soit les vecteurs $\mathbf{i} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \mathbf{j} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ et $\mathbf{k} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ dans l'espace euclidien de dimension 3. Alors, pour tout $\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$ de $\mathbb{R}^3$, on peut écrire

$$\mathbf{u} = u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k}.$$

On dit donc que l'ensemble $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ forme une **base** de $\mathbb{R}^3$.

Définition 1.6 : Base

Un ensemble de n vecteurs $\mathfrak{B} = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n\}$ forme une **base** de $\mathbb{R}^n$ si, pour tout $v \in \mathbb{R}^n$, il existe des nombres $\alpha_i \in \mathbb{R}$, $i = 1, 2, ..., n$ tels que

$$\begin{aligned} v &= \sum_{i=1}^{n} \alpha_i \mathbf{u}_i \\ &= \alpha_1 \mathbf{u}_1 + \alpha_2 \mathbf{u}_2 + ... + \alpha_n \mathbf{u}_n. \end{aligned}$$

Autrement dit, cela veut dire que l'ensemble

$$\text{sp}\{\mathbf{u}_1, ..., \mathbf{u}_n\}$$

de toutes les combinaisons linéaires des vecteurs de $\mathfrak{B} = \{\mathbf{u}_1, ..., \mathbf{u}_n\}$ engendre tout $\mathbb{R}^n$.

Exemple 1.8

On affirme (sans preuve) que $\mathfrak{B} = \left\{\begin{pmatrix}1\\2\end{pmatrix}, \begin{pmatrix}-1\\1\end{pmatrix}\right\}$ est une base de $\mathbb{R}^2$. Soit par exemple $\mathbf{v} = \begin{pmatrix}-1\\7\end{pmatrix}$. Alors,

$$\mathbf{v} = 2\begin{pmatrix}1\\2\end{pmatrix} + 3\begin{pmatrix}-1\\1\end{pmatrix}.$$

Les coefficients de la combinaison linéaire permettant d'engendrer $\mathbf{v}$ dans la base $\mathfrak{B}$ sont $\alpha_1 = 2$ et $\alpha_2 = 3$.

Lorsqu'on travaille dans une base fixée $\mathfrak{B} = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n\}$ et que

$$\mathbf{v} = \sum_{i=1}^{n} \alpha_i \mathbf{u}_i,$$

on dit que les nombres α_i sont les **coordonnées** de $\mathbf{v}$ dans la base $\mathfrak{B}$ et on écrit alors

$$[v]_{\mathfrak{B}} = \begin{pmatrix}\alpha_1\\...\\\alpha_n\end{pmatrix}.$$

Exemple 1.9

En reprenant la base $\mathfrak{B} = \left\{\begin{pmatrix}1\\2\end{pmatrix}, \begin{pmatrix}-1\\1\end{pmatrix}\right\}$ de l'exemple 1.8 et toujours avec $\mathbf{v} = \begin{pmatrix}-1\\7\end{pmatrix}$, on a $\mathbf{v} = 2\mathbf{u}_1 + 3\mathbf{u}_2$ et donc

$$[v]_{\mathfrak{B}} = \begin{pmatrix}2\\3\end{pmatrix}.$$

1.2 Calcul matriciel et transformations linéaires

Définition 1.7 : Transformation linéaire

Une **transformation linéaire** est une application

$$T : \mathbb{R}^n \to \mathbb{R}^n$$

telle que, pour tout $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ et $\alpha, \beta \in \mathbb{R}$, on a :

$$T(\alpha\mathbf{u} + \beta\mathbf{v} = \alpha T(\mathbf{u}) + \beta T(\mathbf{v}).$$

Toute transformation linéaire peut être uniquement représentée par une matrice et inversement toute matrice représente une transformation linéaire.

Définition 1.8 : Matrice

Une **matrice** A est un tableau de nombres (réels ou complexes). Dans le cas réel on écrira

$$A \in \mathbb{R}^{m\times n},$$

où m est le nombre de lignes et n le nombre de colonnes. On note par a_{ij} l'élément situé dans i-ème ligne et la j-ème colonne.

La forme générale d'une matrice $A \in \mathbb{R}^{m\times n}$ est donc

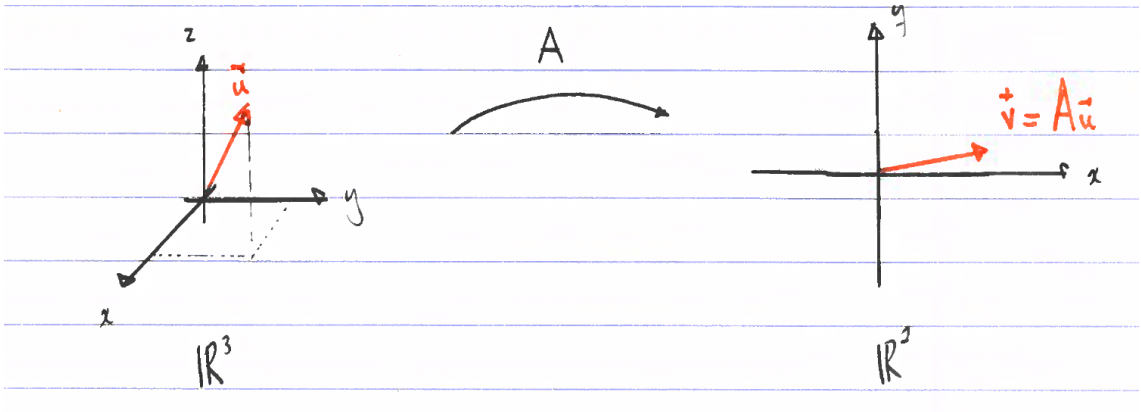
$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Exemple 1.10

Soit $A \in \times$ la matrice donnée par

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 3 & 1 \end{pmatrix}$$

Cette matrice représente une application $A : \mathbb{R}^3 \to \mathbb{R}^2$:



Définition 1.9 : Transposée

La **transposée** A^T d'une matrice $A \in \mathbb{R}^{m\times n}$ est la matrice $A^T \in \times$ telle que $(A^T)_{ij} = a_{ji}$.

Exemple 1.11

Soit $A = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 3 & 1 \end{pmatrix}$ une matrice 2×3. Alors, la transposée de A est la matrice 3×2 donnée par

$$A^T = \begin{pmatrix} 1 & 2 \\ -1 & 3 \\ 0 & 1 \end{pmatrix}.$$

On remarque par exemple que $(A)_{12} = a_{12} = (A^T)_{21} = -1$.

Remarquons au passage qu'un vecteur $\mathbf{v} \in \mathbb{R}^n$ peut-être vu comme une matrice

$$\vec{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}; \quad \vec{v}^T = \begin{pmatrix} v_1 & \dots & v_n \end{pmatrix}$$

où $\mathbf{v} \in \mathbb{R}^{n \times 1}$ est un vecteur colonne dont la transposée $\mathbf{v}^T \in \mathbb{R}^{1 \times n}$ est un vecteur ligne.

Définition 1.10 : Multiplication matricielle

La **multiplication matricielle** de matrices A, B est définie si A possède le même nombre de *colonnes* que B a de *lignes*. Ainsi, si A est $m \times n$ et B est $n \times p$, alors $C = AB$ est de dimension $m \times p$:

$$\begin{pmatrix} & \\ & \end{pmatrix} \cdot \begin{pmatrix} & \\ & \end{pmatrix} = \begin{pmatrix} & \\ & \end{pmatrix}$$

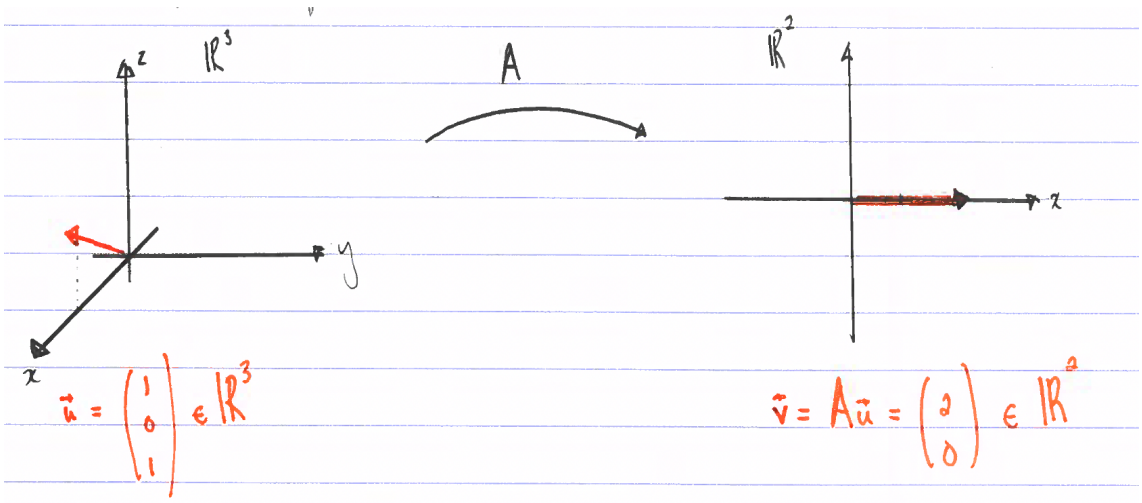
Le produit est défini par $C_{ij} = \sum_k A_{ik} B_{kj}$.

Exemple 1.12

Soit $A = \begin{pmatrix} 2 & 3 & 0 \\ -1 & 4 & 1 \end{pmatrix}$ et $\mathbf{u} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$. Alors,

$$A\mathbf{u} = \begin{pmatrix} 2 & 3 & 0 \\ -1 & 4 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \cdot 1 + 3 \cdot 0 + 0 \cdot 2 \\ -1 \cdot 1 + 4 \cdot 0 + 1 \cdot 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

Encore une fois, il faut voir A comme une transformation linéaire de $\mathbb{R}^3$ vers $\mathbb{R}^2$:



Théorème 1.4 : Propriétés de la multiplication matricielle

Soit A, B et C des matrices dont les dimensions sont telles que tout les produits mentionnés plus bas sont définis. Alors,

1. $A(B+C) = AB + AC$ (Distributivité)
2. $A(BC) = (AB)C$ (Associativité)
3. $(AB)^T = B^T A^T$

Attention : la multiplication matricielle n'est *pas* commutative ! C'est-à-dire que, en général, $AB \neq BA$.

On remarque que cela nous permet de réécrire le produit scalaire $(\mathbf{u}, \mathbf{v})$ de deux vecteurs via

$$\mathbf{u}^T\mathbf{v} = \begin{pmatrix} u_1 & \dots & u_n \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = u_1 v_1 + \dots + u_n v_n.$$

Cette notation sera couramment employée dans le reste du texte.

Définition 1.11 : La matrice identité

La **matrice identité** I_n est la matrice $n \times n$ qui possède des 1 sur la diagonale et 0 partout ailleurs. Elle satisfait

$$I_n\mathbf{u} = \mathbf{u}, \quad \forall \mathbf{u} \in \mathbb{R}^n.$$

Par exemple, la matrice identité dans $\mathbb{R}^3$ est donnée par

$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Définition 1.12 : Matrice inverse

La matrice *inverse* d'une matrice carrée A de dimension n est la matrice A^{-1} qui satisfait

$$AA^{-1} = A^{-1}A = I_n.$$

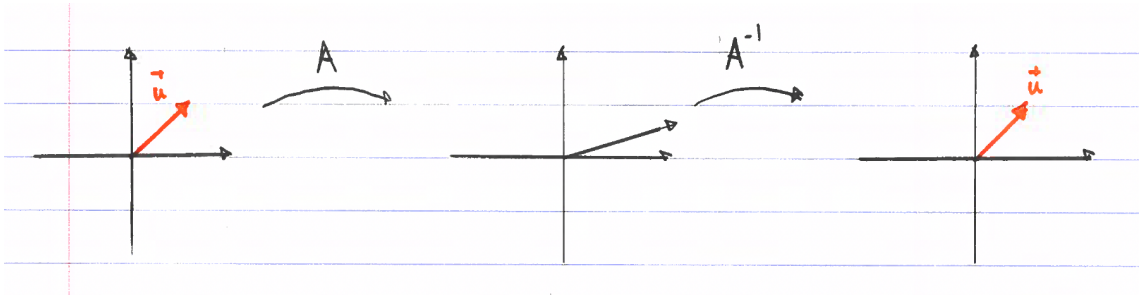
Il est très important de remarquer que ce ne sont pas toutes les matrices carrées qui sont inversibles. L'étude de l'inversibilité d'une matrice est un sujet délicat qui est typiquement au coeur d'un premier cours consacré exclusivement à l'algèbre linéaire.

Exemple 1.13

Soit $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. Alors, A^{-1} existe et on a $A^{-1} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$ En effet, on calcule aisément que

$$AA^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I_2.$$

Géométriquement, on a



Ici, $\mathbf{u} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $A\mathbf{u} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ $A^{-1}\begin{pmatrix} 2 \\ 1 \end{pmatrix} = \mathbf{u}$.

Définition 1.13 : Transformation affine

Une **transformation affine** $A : \mathbb{R}^n \to \mathbb{R}^m$ est une transformation de la forme

$$A\mathbf{u} = W\mathbf{u} + \mathbf{b},$$

où W est une matrice $m \times n$ et $b \in \mathbb{R}^m$.

Exemple 1.14

Montrer géométriquement l'effet de la transformation affine $A : \mathbb{R}^3 \to \mathbb{R}^2$ définie par $A\mathbf{u} = W\mathbf{u} + \mathbf{b}$ avec

$$W = \begin{pmatrix} 1 & 1 & -2 \\ 0 & -1 & 0 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

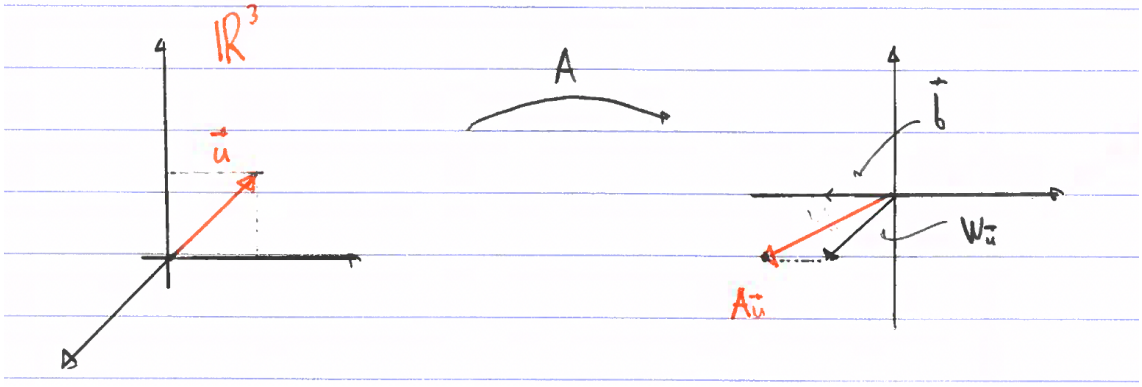
sur le vecteur $\mathbf{u} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$.

Solution :

On calcule directement $W\mathbf{u} = \begin{pmatrix} 1 & 1 & -2 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, de telle sorte que

$$A\mathbf{u} = W\mathbf{u} + \mathbf{b} = \begin{pmatrix} -1 \\ -1 \end{pmatrix} + \begin{pmatrix} -1 \\ 0 \end{pmatrix} = \begin{pmatrix} -2 \\ 1 \end{pmatrix}.$$

Géométriquement, on a



1.3 Vecteurs propres et théorème spectral

Définition 1.14 : Vecteur et valeur propre

Soit A une matrice carrée de dimension n. Alors, un **vecteur propre** de A est un vecteur $\mathbf{u}$ qui satisfait l'équation

$$A\mathbf{u} = \lambda\mathbf{u}.$$

Dans ce cas, on dit que le nombre λ est la **valeur propre** associée au vecteur propre $\mathbf{u}$.

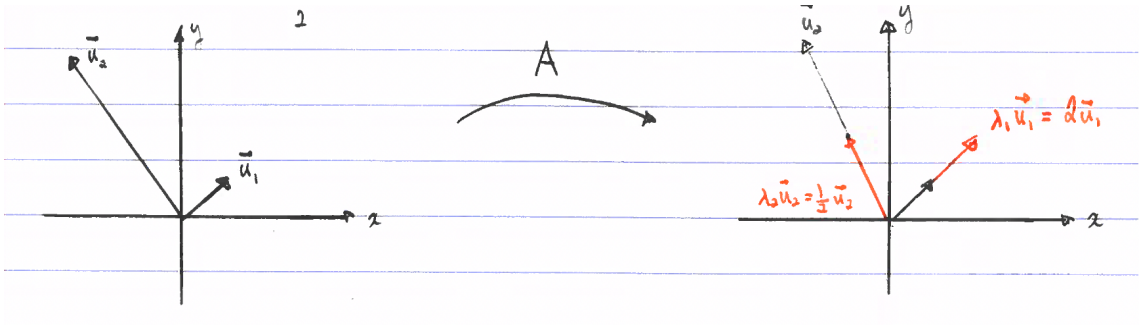
Exemple 1.15

Soit $\begin{pmatrix} 0 & 1 \\ -2 & 3 \end{pmatrix}$ et $\mathbf{u} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$. Alors,

$$A\mathbf{u} = \begin{pmatrix} 0 & 1 \\ -2 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix} = (-1)\mathbf{u}.$$

Donc, $\mathbf{u}$ est un vecteur propre de A associé à la valeur propre $\lambda = -1$.

Comment interpréter géométriquement les vecteurs et valeurs propres d'une matrice A? Supposons que $\mathbf{u}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ et $\mathbf{u}_2 = \begin{pmatrix} -2 \\ 4 \end{pmatrix}$ sont des vecteurs propres de valeurs propres $\lambda_1 = 2$ et $\lambda_2 = \frac{1}{2}$ respectivement. On a alors



Ainsi, une matrice A agit sur ses vecteurs propres en les dilatant ou en les contractant (et en inversant leur sens si $\lambda < 0$), mais sans modifier leur direction.

Définition 1.15 : Matrices orthogonale et symétrique

- Une matrice **orthogonale** est une matrice telle que
$$A^T = A^{-1}.$$
- Une matrice **symétrique** est une matrice telle que
$$A = A^T.$$

On remarque que ces deux définitions impliquent que, forcément, seule une matrice carrée peut être orthogonale ou symétrique.

Théorème 1.5 : Théorème spectral

Soit S une matrice symétrique de dimension n.

- Alors, S admet une base orthonormale $\mathfrak{B} = \{\mathbf{u}_1, ..., \mathbf{u}_n\}$ de vecteurs propres de S.
- De plus, si on pose $Q = \begin{pmatrix}\mathbf{u}_1 & \mathbf{u}_2 & ... & \mathbf{u}_n\end{pmatrix}$, alors Q est une matrice orthogonale et on a
$$D = Q^{-1}AQ,$$
avec $D = \text{diag}\{\lambda_i\}$, c'est-à-dire
$$D = \begin{pmatrix}\lambda_1 & 0 & 0 & \\ 0 & \lambda_2 & 0 & \\ & & \ddots & \\ & & & \lambda_n\end{pmatrix}.$$

On rappelle qu'une base orthonormale (ONB) est une base $\{\mathbf{u}_1, ..., \mathbf{u}_n\}$ de $\mathbb{R}^n$ telle que $\mathbf{u}_i \perp \mathbf{u}_j$ dès que $i \neq j$ et $||\mathbf{u}_i|| = 1$, $i = 1, ..., n$.

Exemple 1.16

Soit $A = \begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}$. Vérifier que A est symétrique et qu'elle admet une base orthonormale de vecteurs propres.

Solution :

On a $A^T = \begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix} = A$ et donc A est symétrique.

Avec un logiciel de calcul symbolique, on trouve 2 vecteurs propres

$$\mathbf{u}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \text{ et } \mathbf{u}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

de valeurs propres respectives $\lambda_1 = 3$ et $\lambda_2 = -1$. On calcule $(\mathbf{u}_1, \mathbf{u}_2) = (-1)\cdot 1 + 1 \cdot 1 = 0$ et donc $\mathbf{u}_1$ et $\mathbf{u}_2$ sont bien orthogonaux. Soit maintenant

$$\mathbf{v}_1 = \frac{\mathbf{u}_1}{||\mathbf{u}_1||} \text{ et } \mathbf{v}_2 = \frac{\mathbf{u}_2}{||\mathbf{u}_2||}$$

la normalisation de ces vecteurs propres. Alors, $\{\mathbf{v}_1, \mathbf{v}_2\}$ est une base orthonormale de vecteurs propres de A.

1.4 Calcul différentiel et optimisation

On révise désormais très brièvement certains thèmes propres au calcul différentiel qui nous permettront plus tard d'optimiser des fonctions de plusieurs variables. On commence d'abord avec le cas le plus simple en supposant que f et g sont deux fonctions dérivables d'une variable. Alors, la *règle de la chaîne* permet de dériver la composition

$$(f \circ g)'(x) = f'(g(x))g'(x).$$

En utilisant la notation de Leibniz, on a plutôt

$$\frac{d}{dx}(f \circ g) = \frac{df}{dg}\frac{dg}{dx}.$$

Exemple 1.17

Soit $y(x) = 3x + 4$ et $f(y) = y^3$. Alors,

$$\begin{aligned}(f \circ y)'(x) &= \big[f(y(x))\big]' = f'(y)y'(x) \\ &= 3y^2(3x+4)' \\ &= 3(3x+4)^2 \cdot 3 = 9(3x+4)^2.\end{aligned}$$

Évidemment, si on avait directement eu $h(x) = (3x+4)^3$, on aurait calculé

$$h'(x) = 3(3x+4)^2 \cdot 3 = 9(3x+4)^2.$$

Le résultat suivant permet d'étendre cette règle à la composition de fonctions de 2 variables.

Théorème 1.6 : Règle de la chaîne, 2 variables indépendantes

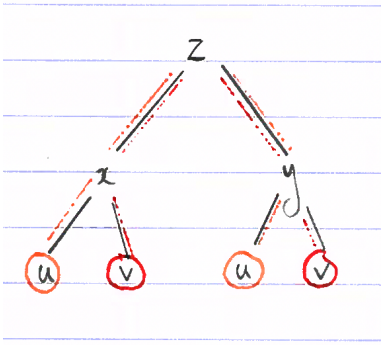
Supposons que $x(u,v)$ et $y(u,v)$ sont 2 fonctions différentiables de u,v et que $z(x,y)$ est une fonction différentiable de x et y. Alors,

$$z(u,v) = z(x(u,v), y(u,v))$$

est une fonction différentiable de u,v et

$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u} \text{ et } \frac{\partial z}{\partial v} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial v}.$$

Ces règles se visualisent par l'arbre suivant



Exemple 1.18

Soit $x(u,v) = u^2 + v^2$ et $y(u,v) = 3u - 2v$ et $z(x,y) = e^{2x+y}$. Alors, on calcule

$$\frac{\partial x}{\partial u} = 2u,\ \frac{\partial x}{\partial v} = 2v,\ \frac{\partial y}{\partial u} = 3,\ \frac{\partial y}{\partial v} = -2.$$

De plus,

$$\frac{\partial z}{\partial x} = 2e^{2x+y} \text{ et } \frac{\partial z}{\partial y} = e^{2x+y}.$$

Donc,

$$\begin{aligned}\frac{\partial z}{\partial u} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u} &= 2e^{2x+y}\cdot(2u) + e^{2x+y}\cdot 3 \\ &= e^{2x+y}(4u+3) \\ &= e^{2u^2+2v^2+3u-2v}(4u+3).\end{aligned}$$

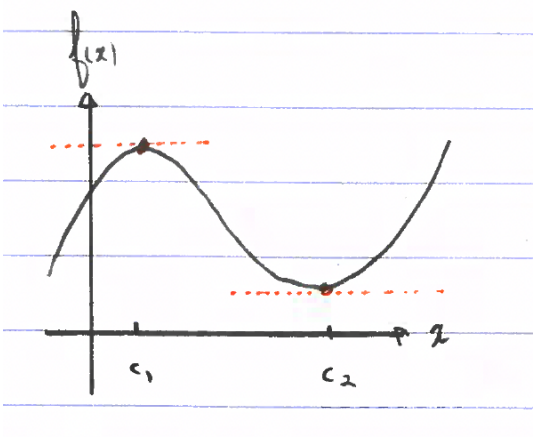
Un calcul similaire permet d'obtenir $\dfrac{\partial z}{\partial v}$.

1.4.1 Descente de gradient

Supposons que vous êtes en forêt dans un paysage montagneux et que vous ne voyez pas autour de vous en raison de la densité des arbres. Quel est le meilleur moyen de retrouver le base de la montagne ?

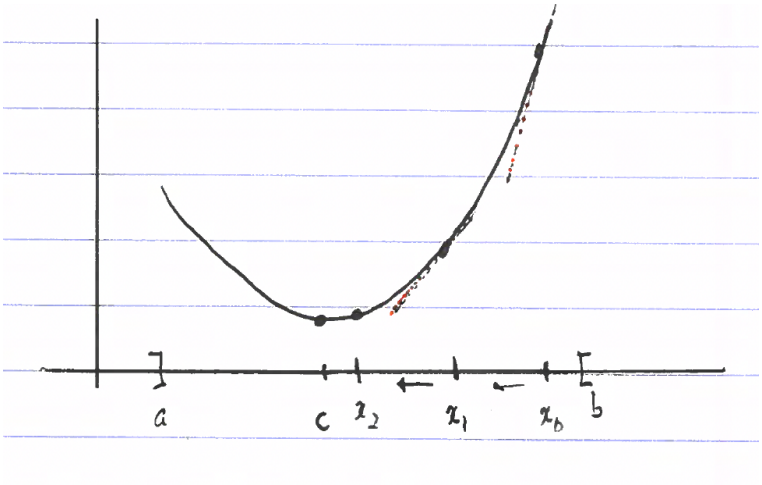
Vous pouvez examiner localement la topographie autour de vous et décider de faire un petit pas dans la direction immédiate qui vous fera le plus diminuer votre altitude. En répétant suffisamment longtemps ce petit manège, vous devriez arriver à retrouver la vallée. La descente de gradient est une technique mathématique qui fait exactement la même chose lorsqu'on vise à trouver le minimum d'une fonction.

On rappelle d'abord le fait suivant: si $f :]a,b[\to \mathbb{R}$ est une fonction dérivable et que f admet un extremum (minimum ou maximum) local en $c \in]a,b[$, alors $f'(c) = 0$.



Ici, on a $f'(c_1) = f'(c_2) = 0$.

Pour approcher numériquement un tel extremum local, on emploie donc la technique itérative de descente du gradient.



On fixe d'abord un *pas* $\eta > 0$ et on initialise un processus itératif en un point quelconque $x_0 \in]a,b[$.

1. $x_1 = x_0 + \eta f'(x_0)$, et on remarque que l'on se déplace vers la gauche si $f'(x_0) < 0$ et vers la droite si $f'(x_0) > 0$.

2. $x_2 = x_1 + \eta f'(x_1)$ et ainsi de suite.

Il est important ici de remarquer que cet algorithme, bien que souvent fonctionnel, est très naïf: un tas de complications sont possibles. Que se passe-t-il, par exemple, si on tombe sur un maximum local tel que $f'(c) = 0$? Est-ce que la procédure est certaine de converger vers le minimum recherché? Quand est-on suffisamment « proche » pour cesser les itérations? Ces questions sont importantes et complexes!

Ces idées se généralisent naturellement aux fonctions de plusieurs variables.

Définition 1.16 : Vecteur gradient

Soit $f : \mathscr{D} \subset \mathbb{R}^n \to \mathbb{R}$ une fonction différentiable. Alors, le **gradient** de f est un champ vectoriel $\mathscr{D} \subset \mathbb{R}^n \to \mathbb{R}^n$ défini par

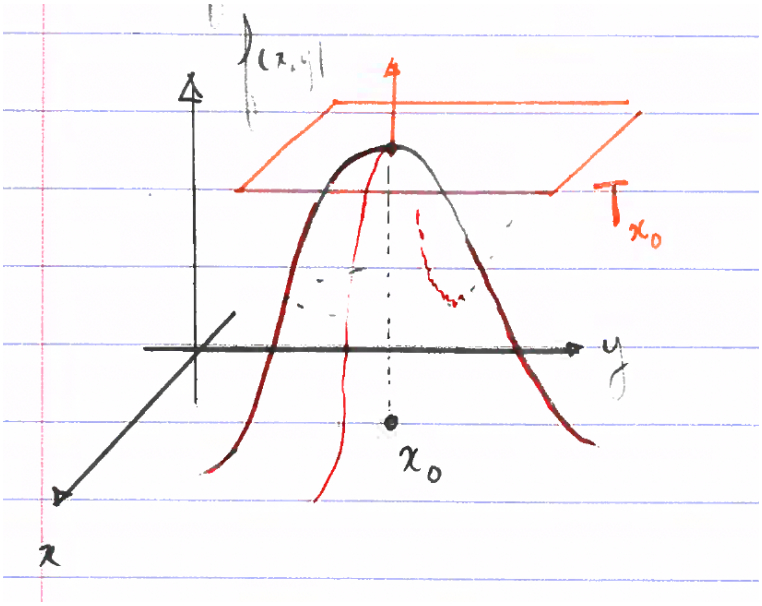
$$\nabla f(x_1, ..., x_n) = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & ... & \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Autrement dit, le vecteur gradient d'une fonction est un vecteur dont les composantes en un point donné sont les dérivées partielles de cette fonction évaluées à ce point.

Théorème 1.7

Soit $f : \mathscr{D} \subset \mathbb{R}^n \to \mathbb{R}$ une fonction différentiable. Si f admet un extremum local en $x_0 \in \mathscr{D}$, alors $\nabla f(x_0) = \mathbf{0}$.

Cela veut dire que si, lors d'une promenade dans un paysage montagneux, on se retrouve au sommet d'une montagne ou dans le creux d'une vallée, alors le *plan tangent* à la surface au point où on se situe est horizontal.



Toute l'idée de descente du gradient repose sur le fait suivant: le vecteur gradient $\nabla f(x_0)$ pointe dans la direction d'accroissement *maximal* de f en $x_0 \in \mathscr{D}$. Similairement, $-\nabla f(x_0)$ pointe dans la direction de la plus grande décroissance de f en x_0.

Exemple 1.19

Soit $f(x,y) = x^2 + y^2$. Alors,

$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} 2x & 2y \end{pmatrix}.$$

Ainsi, en $x_0 = (1,1)$, on a $\nabla f(1,1) = (2,2)$.

Chapitre 2

Réseaux de neurones profonds

Dans ce chapitre, nous allons introduire les réseaux de neurones profonds, en définir les différentes composantes pour finalement indiquer comment ceux-ci peuvent *apprendre*. Ceci se fera à l'aide d'une tâche classique qui est désormais considérée comme le « *Hello World* » de l'apprentissage machine, soit la reconnaissance automatisée de chiffres manuscrits.

2.1 Principes généraux

Considérons donc une tâche classique qui se fait de manière automatique pour les humains : la reconnaissance de chiffres manuscrits.

Chiffres manuscrits tirés de la base de données MNIST.

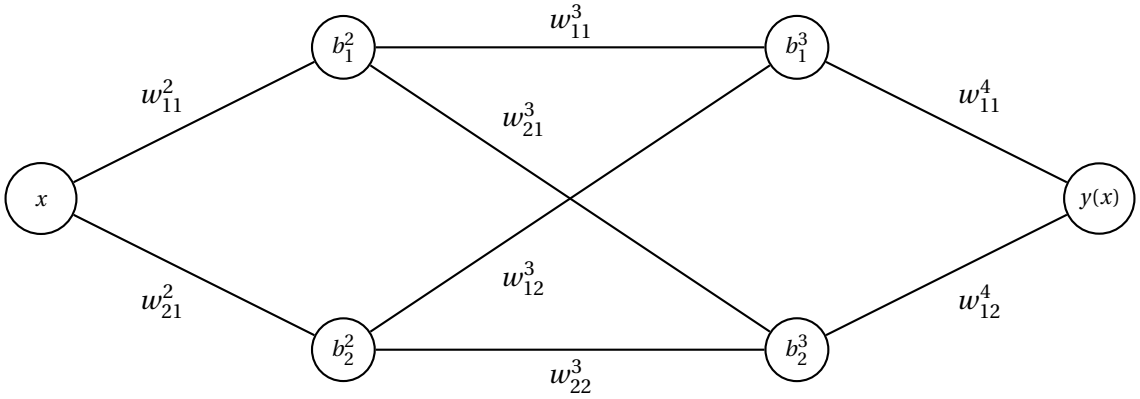
Comment apprendre à reconnaître un « 4 » manuscrit, par exemple, et à le classe comme tel ? C'est assurément quelque chose de très difficile à effectuer directement. On ne pourrait pas espérer avoir beaucoup de succès en décrivant les propriétés figuratives qui composent la représentation d'un chiffre : « *le chiffre 4 est composé d'un long trait, de trois petits, parfois reliés au sommet, parfois pas...* ». Bonne chance !

Pour aborder autrement ce problème, on considère un *réseau de neurones*, qui n'est rien de plus qu'un graphe dans lequel circule de l'information.

Définition 2.1 : Réseau de neurones

En général, un **réseau de neurones profond** (RNP) est un graphe, dont les sommets sont appelés des **neurones** et les arêtes des **connexions**. Les neurones sont disposés en plusieurs **couches** et on imagine l'information circulant à partir de la couche d'entrée vers la couche de sortie.

Il est utile d'imaginer chaque neurone comme une ampoule qui peut être plus ou moins allumée, l'intensité de la lumière étant formellement appelée **activation**.



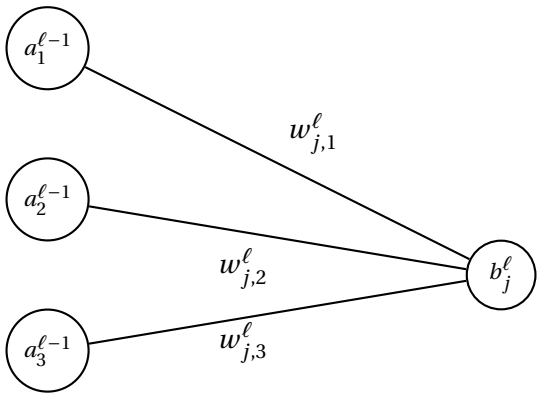
- Les *couches* du réseau sont notées $\ell = 1, ..., L$.
- On note $\{d_1, d_2, ..., d_L\}$ la *largeur* ou *dimension* de chaque couche.
- Au j-ème neurone de la couche ℓ, on associe un *biais* b^ℓ_j.
- À chaque connexion on associe un *poids* w. En particulier, w^ℓ_{jk} est le poids qui relie le k-ème neurone de la couche $\ell - 1$ avec le j-ème neurone de la couche ℓ.
- On note a^ℓ_j l'*activation* ou la *puissance* du signal du j-ème neurone de la couche ℓ.

De manière générale, les indices en super script dénoteront la couche et ceux en sous script la position du neurone dans une couche donnée.

Exemple 2.1
Le réseau de l'exemple ci-haut contient 4 couches : $\ell = 1,2,3,4$ de dimensions respectives $d_i = 1,2,2,1$.

Définition 2.2 : Préactivation

On considère le vecteur $\mathbf{w}_j^\ell$ de tous les poids entrant dans le j-ème neurone de la couche ℓ ainsi que le vecteur $\mathbf{a}^{\ell-1}$ contenant toutes les activations des neurones de la couche $\ell-1$.



Alors, la **préactivation** du j-ème neurone de la couche ℓ est définie par

$$z_j^\ell = \mathbf{w}_j^\ell \cdot \mathbf{a}^{\ell-1} + b_j^\ell.$$

Il est utile de s'éloigner un peu des formules et de la notation qui peut être lourde pour bien comprendre ce en quoi consiste la préactivation : il s'agit simplement d'une moyenne pondérée des activations de la couche précédente. Chaque neurone de la couche précédente contribue à la préactivation d'un neurone donné de la couche subséquente en lui transmettant sa propre activation, par le biais d'une connexion qui peut être plus ou moins forte.

Définition 2.3 : Activation

L'**activation** du j-ème neurone de la couche ℓ est calculée par

$$a_j^\ell = \sigma(z_j^\ell),$$

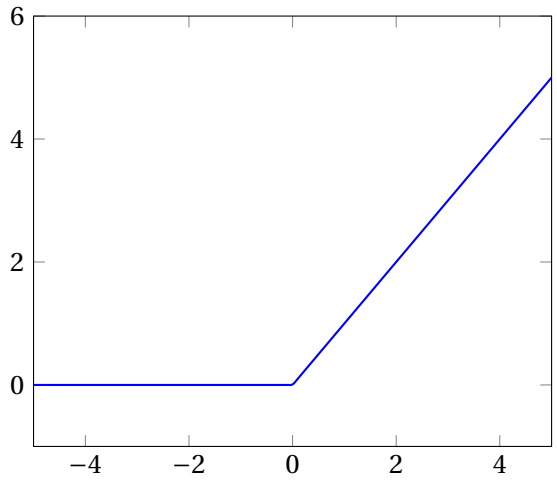
où $\sigma : \mathbb{R} \to \mathbb{R}$ est une *fonction d'activation* fixée et z_j^ℓ est la préactivation du neurone considéré.

Ainsi, l'activation d'un neurone est simplement l'image de sa préactivation par une fonction d'activation fixée.

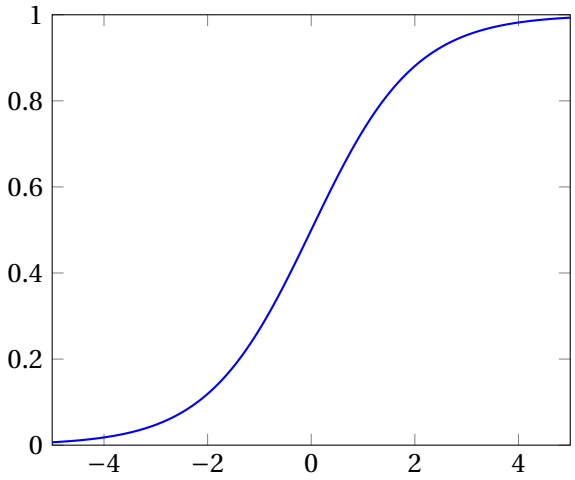
Exemple 2.2

Il existe théoriquement une infinité de fonctions d'activation possibles. Voici deux des plus populaires, que l'on utilisera à plusieurs reprises dans le reste du texte.

- La fonction **ReLU** (*Rectified Linear Unit*), définie par $\sigma(z) = \max\{0, z\}$.



- La fonction **sigmoïde**, donnée par $\sigma(z) = \dfrac{1}{1+e^{-z}}$.



L'objectif des fonctions d'activation est d'introduire une non-linéarité dans le réseau, pour donc élargir la classe des fonctions qu'il est (théoriquement) possible d'approximer. Le modèle complet d'un réseau de neurones est donc une fonction

$$\mathscr{F} : \mathbb{R}^n \to \mathbb{R}^m,$$

avec $n = d_1$ et $m = d_L$.

Il est possible de synthétiser le mécanisme de propagation avant de l'information dans un réseau de neurones en adoptant un point de vue vectoriel. On dénote

- $W^\ell = (w^\ell_{jk})$, la matrice de tous les poids allant de la couche $(\ell - 1)$ à la couche ℓ.

- b^ℓ, le vecteur contenant tous les biais des neurones de la couche ℓ.
- z^ℓ, le vecteur contenant les préactivations des neurones de la couche ℓ.
- Enfin, a^ℓ, le vecteur des activations des neurones de la couche ℓ.

Avant l'activation, la propagation avant de l'information de la couche $(\ell-1)$ à la couche ℓ est une transformation affine

$$T_\ell : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell},$$

donnée par

$$z_\ell = T_\ell(a^{(\ell-1)} = W^\ell a^{\ell-1} + b^\ell.$$

Ainsi, la fonction $\mathscr{F} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_L}$ que réalise le réseau peut être ainsi décomposée

$$\mathscr{F} = \mathrm{Act}_L \circ T_L \circ \mathrm{Act}_{L-1} \circ T_{L-1} \circ \ldots \circ \mathrm{Act}_1 \circ T_1,$$

où Act$=\sigma$ sont les activations vectorielles et T_i sont les transformations affines vues un peu plus haut.

On revient maintenant à notre objectif principal, soit la tâche de construire un réseau capable de classer des chiffres manuscrits. Avec la terminologie que l'on vient de développer, on veut que le réseau représente une fonction $\mathscr{F}$ qui soit capable d'approximer correctement une fonction dite *parfaite* ou *idéale*, typiquement notée f^*, la fonction qui associe parfaitement chaque image d'un chiffre manuscrit à la catégorie (entre 0 et 9) appropriée. Mais comment faire apprendre à un réseau ? C'est en modifiant ses réglages, ses boutons ses *paramètres* que l'on pourra lui permettre de s'ajuster et de graduellement se rapprocher de la fonction idéale f^*.

Définition 2.4 : Paramètres

Les *paramètres* d'un réseau de neurones sont l'ensemble des poids w et biais b qui le composent. On peut voir chaque paramètre comme un petit bouton qu'il faut constamment ajuster et régler pour obtenir la configuration que l'on recherche pour accomplir la tâche visée.

Enfin, on distingue deux types d'apprentissage principaux pour les réseaux de neurones.

Définition 2.5 : Types d'apprentissage

- L'apprentissage **supervisé** lorsqu'on on utilise des données d'entraînement étiquetées.
- L'apprentissage **non-supervisé** dans le cas contraire, c'est-à-dire lorsque les données d'entraînement utilisées ne sont pas étiquetées.

Dans le cas qui nous occupe, la base de données de chiffres manuscrits MNIST étant étiquetée, nous sommes dans un contexte d'apprentissage supervisé.

2.2 Apprendre via la descente de gradient

Notre but est donc de construire un réseau de neurones approprié à notre tâche et de le faire apprendre, c'est-à-dire de trouver les valeurs des paramètres w, b qui lui permettront d'avoir les meilleures

performances pour la tâche de classification de chiffres manuscrits. Pour être capable de mesurer les performances du réseau, nous utiliserons la fonction dite de **coût** ou **d'erreur** suivante

$$C(w,b) = \frac{1}{2n}\sum_x ||\mathbf{y}(x) - \mathbf{a}^L||^2,$$

où x est une donnée d'entraînement sous forme vectorielle, **y(x)** est la sortie désirée associée à l'entrée x et n correspond au nombre total de données d'entraînement. Intuitivement, la fonction de coût compare la différence entre la sortie attendue et la sortie obtenue pour chaque donnée d'entrée x et en fait une moyenne.

Enfin, il est important de remarquer que la fonction de coût dépend des paramètres. En effet, la performance du réseau sera plus ou moins bonne selon le choix des poids et des biais que l'on aura effectué. Est-ce la seule fonction de coût possible ? Évidemment, la réponse est non. Nous aurions par exemple pu utiliser

$$\tilde{C}(w,b) := \#\text{d'images correctement classées},$$

ce qui est conceptuellement correct, dans la mesure où cette fonction donne également une mesure de la performance du réseau. Or, cette fonction n'est pas *lisse* et donc non différentiable. L'avantage de C telle que définie plus haut est qu'il s'agit d'une fonction de classe C^∞ avec laquelle on peut utiliser la puissance des techniques issues du calcul différentiel.

On veut trouver le point (w,b) qui va *minimiser* la fonction de coût C et on peut donc utiliser la technique de descente du gradient.

$$\begin{cases} w_k \mapsto w'_k = w_k - \eta\frac{\partial C}{\partial w_k}, \\ \\ b_j \mapsto b'_j = b_j - \eta\frac{\partial C}{\partial b_j}. \end{cases} \tag{2.1}$$

Puisque $C = \frac{1}{2n}\sum_x C_x$, avec $C_x = \frac{1}{2}||\mathbf{y}(x) - \mathbf{a}^L||^2$, calculer le gradient ∇C exige de balayer *toutes* les données d'entraînement. Pour pallier ce problème, on introduit le concept suivant.

Définition 2.6 : Descente stochastique de gradient

La **descente stochastique de gradient** (que l'on abrégera souvent par *SGD*) est une technique pour minimiser une fonction de coût $C(w,b)$ et qui se décline en trois étapes :

1. On choisit *aléatoirement* une **mini-batch** de m données d'entraînement

$$x_1, x_2, ..., x_m.$$

2. On met à jour les arguments de la fonction de coût, soit les paramètres w, b via

$$\begin{cases} w_k \mapsto w'_k = w_k - \eta\frac{1}{m}\sum_{j=1}^m \frac{\partial C_{x_j}}{\partial w_k}, \\ \\ b_\ell \mapsto b'_\ell = b_\ell - \eta\frac{1}{m}\sum_{j=1}^m \frac{\partial C_{x_j}}{\partial b_\ell}. \end{cases} \tag{2.2}$$

2.3 Rétropropagation

Le descente de gradient exige de calculer *toutes* les dérivées partielles $\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}$ pour chacun des paramètres w, b qui constituent le réseau de neurones. Une approche naïve consiste à estimer chacune de celle-ci par

$$\frac{\partial C}{\partial w} \approx \frac{C(w+\epsilon) - C(w)}{\epsilon}$$

pour un choix suffisamment petit de ϵ. Or, cette manière de procéder exige de calculer la quantité $C(w+\epsilon)$ pour chaque paramètre! Si on a N de ceux-ci, il faudra alors faire N traversées du réseau pour compléter une seule itération de la descente de gradient! On comprend aisément que lorsque N devient grand, cette approche devient impossible à implémenter en pratique.C'est la technique de *rétropropagation* qui nous permettra de calculer ces dérivées partielles de manière plus efficace. Le traitement que nous proposons ici est basé sur celui dans [Nielsen].

L'idée générale est d'introduire une petite perturbation Δ_j^ℓ dans la préactivation: $z_j^\ell \mapsto z_j^\ell + \Delta_j^\ell$. Un développement de Taylor du premier degré donne maintenant

$$C(z_j^\ell + \Delta_j^\ell) - C(z_j^\ell) \approx \frac{\partial C}{\partial z_j^\ell}\Delta_j^\ell.$$

Puisque $C \geq 0$ et qu'on veut minimiser C, on veut donc que $\frac{\partial C}{\partial z_j^\ell}\Delta_j^\ell << 0$. Deux cas de figure sont ainsi possibles. D'abord, si $\frac{\partial C}{\partial z_j^\ell}$ est gros, on choisit Δ_j^ℓ de signe opposé. Dans le cas contraire où $\frac{\partial C}{\partial z_j^\ell}$ est petit, on ne peut rien faire et ce neurone est donc déjà, dans un certain sens, configuré de manière optimale. C'est cette interprétation de la taille de la dérivée $\frac{\partial C}{\partial z_j^\ell}$ qui nous incite à présenter la définition suivante.

Définition 2.7 : Erreur d'un neurone

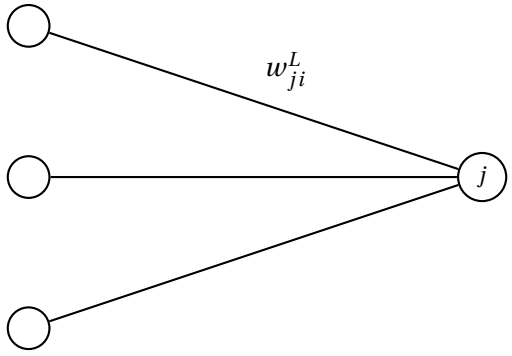
L'**erreur** δ_j^ℓ d'un neurone est

$$\delta_j^\ell = \frac{\partial C}{\partial z_j^\ell}.$$

De manière vectorielle, on notera parfois les erreurs de la couche ℓ par

$$\delta^\ell = \begin{pmatrix} \delta_1^\ell \\ \vdots \\ \delta_{d_\ell}^\ell \end{pmatrix}.$$

On considère d'abord les dernières couches $(L-1)$ et L d'un réseau.



On rappelle ici que $a^L_j = \sigma(z^L_j)$. De plus, la préactivation z^L_j est calculée par

$$z^L_j = \left(\sum_k w^L_{kj} \cdot a^{L-1}_k\right) + b^L_j.$$

Ainsi, en vertu de la règle de la chaîne, on a

$$\frac{\partial C}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j),$$

que l'on peut réécrire pour obtenir la première équation de la rétropropagation.

Théorème 2.1 : Première équation de la rétropropagation (RP1)

$$\delta^L_j = \frac{\partial C}{\partial a^L_j} \cdot \sigma'(z^L_j).$$

De plus, une autre application de la règle de la chaîne nous donne immédiatement

$$\delta^\ell_j = \frac{\partial C}{\partial z^\ell_j} = \sum_k \frac{\partial C}{\partial z^{\ell+1}_k}\frac{\partial z^{\ell+1}_k}{\partial z^\ell_j} = \sum_k \frac{\partial z^{\ell+1}_k}{\partial z^\ell_j}\delta^{\ell+1}_k.$$

On sait que

$$z^{\ell+1}_k = \sum_i w^{\ell+1}_{ki} a^\ell_i + b^{\ell+1}_k,$$

d'où

$$\frac{\partial z^{\ell+1}_k}{\partial z^\ell_j} = w^{\ell+1}_{kj}\sigma'(z^\ell_j).$$

Ceci nous permet d'obtenir une seconde équation de la rétropropagation.

Théorème 2.2 : Seconde équation de la rétropropagation (RP2)

$$\delta^\ell_j = \sum_k w^{\ell+1}_{kj} \cdot \delta^{\ell+1}_k \cdot \sigma'(z^\ell_j).$$

L'usage combiné des deux premières équations de la rétropropagation nous permet de propager vers l'arrière les erreurs δ_j^ℓ d'une couche à l'autre en débutant par la dernière. Or, notre objectif initial est de connaître le gradient de la fonction de coût. Il nous reste donc à trouver une manière d'exprimer les dérivées partielles $\frac{\partial C}{\partial w}$, $\frac{\partial C}{\partial b}$ en fonction des erreurs δ_j^ℓ, ce que feront nos deux dernières équations de la rétropropagation.

Une autre utilisation directe de la règle de la chaîne nous permet de calculer

$$\frac{\partial C}{\partial b_j^\ell} = \sum_k \frac{\partial C}{\partial z_k^\ell} \cdot \frac{\partial z_k^\ell}{\partial b_j^\ell} = \frac{\partial C}{\partial z_j^\ell} = \delta_j^\ell.$$

Théorème 2.3 : Troisième équation de la rétropropagation (RP3)

$$\frac{\partial C}{\partial b_j^\ell} = \delta_j^\ell.$$

Finalement, une dernière utilisation de la règle de la chaîne donne

$$\frac{\partial C}{\partial w_{jk}^\ell} = \sum_i \frac{\partial C}{\partial z_i^\ell} \cdot \frac{\partial z_i^\ell}{\partial w_{jk}^\ell} = \frac{\partial C}{\partial z_j^\ell} \cdot \frac{\partial z_j^\ell}{\partial w_{jk}^\ell} = \delta_j^\ell \cdot a_k^{\ell-1},$$

où l'on a utilisé le fait que $\frac{\partial z_i^\ell}{\partial w_{jk}^\ell}$ vaut 0 dès que $i \neq j$.

Théorème 2.4 : Quatrième équation de la rétropropagation (RP4)

$$\frac{\partial C}{\partial w_{jk}}^{\ell} = \delta_j^\ell \cdot a_k^{\ell+1}.$$

Chapitre 3

Problèmes d'apprentissage

Les idées de base permettant l'apprentissage d'un réseau de neurones présentées au chapitre précédent se heurtent souvent en pratique à des obstacles qui neutralisent le bon fonctionnement du réseau. Dans ce chapitre, nous allons explorer certains de ces obstacles et considérer différentes approches pour les contourner.

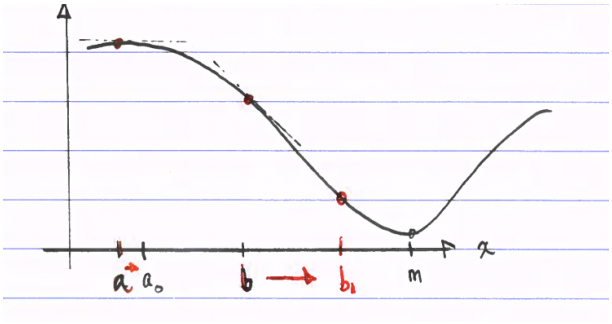
3.1 Problèmes d'apprentissage avec la fonction de coût quadratique et la sigmoïde

On rappelle que la fonction de coût *quadratique* introduite au chapitre précédent est définie par

$$C = C(w,b) = \frac{1}{2n}\sum_x ||y(x) - a^L||^2.$$

De plus, on utilisera la fonction d'activation sigmoïde $\sigma : \mathbb{R} \to \mathbb{R}$ donnée par $\sigma(x) = \frac{1}{1+e^{-x}}$. On remarque aisément que σ tend respectivement vers $1,0$ lorsque x tend vers $\pm\infty$.

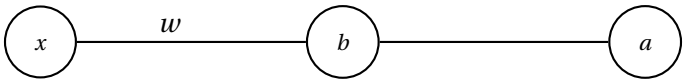
Revenons maintenant à notre approche de descente de gradient en une dimension. On considère la fonction $f(x)$ suivante avec le taux d'apprentissage $\eta = 1$.



On observe les faits suivants :

- Lorsque la dérivée $f'(a)$ s'approche de zéro, la convergence vers m est très lente.
- Au contraire, quand $|f'(a)|$ est « grand », la converge vers m est plus rapide.

Considérons un réseau de neurones très simple $\mathscr{F} : \mathbb{R} \to \mathbb{R}$, donné par



Ici, $z = wx + b$ et $a = \sigma(z)$, avec σ la fonction sigmoïde. Notre objectif est de faire apprendre à ce réseau l'application

$$1 \mapsto 0.$$

Autrement dit, on voudrait qu'il apprenne à fournir la sortie $a = 0$ lorsqu'on lui fournit $x = 1$ comme entrée. Pour voir un peu comment le réseau va apprendre, fixons d'abord $w = 0.6$ et $b = 0.9$. Alors,

$$\mathscr{F}(x) = \sigma(0.6x + 0.9).$$

et on calcule aisément que $\mathscr{F}(1) \approx 0.82$. Pour le moment, donc, ce réseau fait assez mal son travail. C'est la fonction de coût quadratique qui permet de quantifier cette inefficacité

$$C_1 = \frac{1}{2}(a^L - y)^2 = \frac{1}{2}(0.82 - 0)^2 \approx 0.334.$$

Est-ce que la descente de gradient s'effectuera efficacement? Pour y voir, calculons d'abord la dérivée partielle suivante

$$\frac{\partial C}{\partial w} = \frac{\partial}{\partial w}\left(\frac{1}{2}(y(x) - \sigma(wx + b)\right)^2 = (y(x) - \sigma(z)) \cdot \sigma'(z) \cdot z'.$$

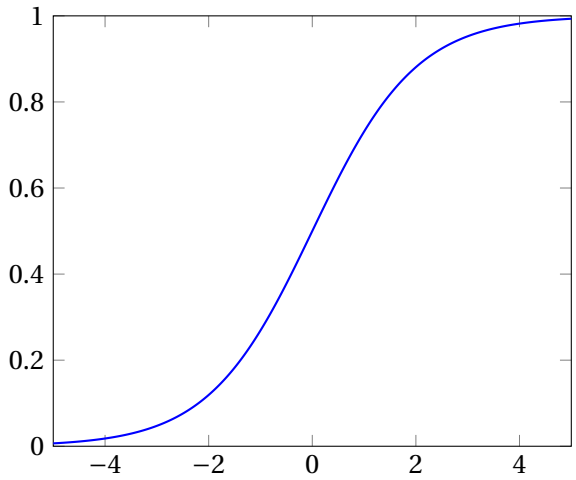
On a donc

$$\frac{\partial C}{\partial w} = (a - y) \cdot x \cdot \sigma'(z).$$

En remplaçant avec $x = 1$ et $y = 0$ et via un calcul similaire, on obtient finalement

$$\frac{\partial C}{\partial w} = a\sigma'(z), \quad \frac{\partial C}{\partial b} = a\sigma'(z).$$

Or, rappelons-nous du graphe de la fonction sigmoïde



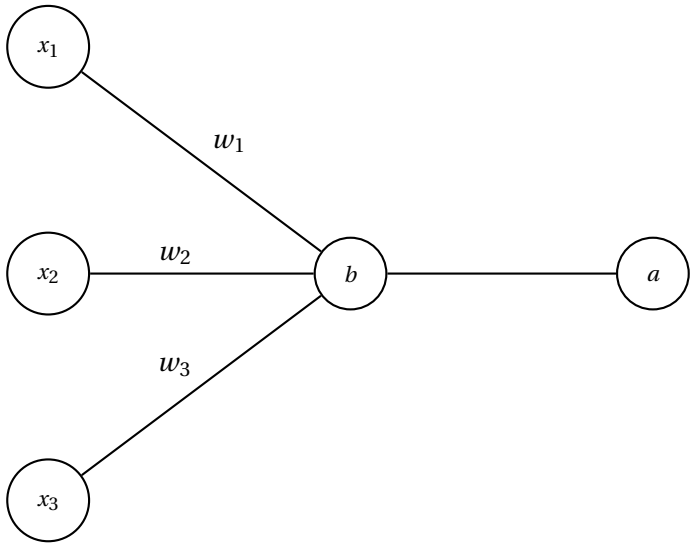
Ainsi, quand $a = \sigma(z)$ est grand, i.e. C est grand, alors $\sigma'(z)$ est petit et donc *une grande erreur implique ici un apprentissage lent*! Pourtant, on voudrait bien le contraire, car c'est lorsque l'erreur est grande que l'apprentissage devrait être efficace, rapide.

Exemple 3.1
Calculons ici explicitement la dérivée de la fonction sigmoïde.

$$\sigma'(z) = (-1)(1+e^{-z})^{-2} \cdot (1+e^{-z})' = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{e^{-z}}{1+e^{-z}}\sigma(z).$$

3.2 Entropie croisée

L'objectif de cette section est de résoudre le problème rencontré précédemment en introduisant une nouvelle fonction de coût qui sera telle que l'apprentissage est efficace lorsque l'erreur est grande. On considère maintenant le réseau suivant.



La préactivation est donc calculée par

$$z = \left(\sum_{i=1}^{3} w_i x_i\right) + b$$

et $a = \sigma(z)$, où σ est toujours la sigmoïde.

Définition 3.1 : Entropie croisée

La fonction de coût d'**entropie-croisée** est définie par

$$C = -\frac{1}{n}\sum_x y\ln(a) + (1-y)\ln(1-a),$$

où

- x : données d'entrées,
- n : nombre de données d'entrées,
- a : la sortie produite par le réseau pour la une donnée d'entrée (i.e. $a = a(x)$),
- y : la sortie désirée pour x.

Exemple 3.2

Montrons que la fonction de coût d'entropie-croisée est non-négative. On commence par remarquer que puisque $0 < a < 1$, on a aussi $0 < 1-a < 1$ et donc que les quantités $\ln(a)$ et $\ln(1-a)$ sont négatives. De plus, y et $1-y$ sont positifs et donc les termes $y\ln(a)$ et $(1-y)\ln(1-a)$ sont négatifs et $C > 0$.

Voyons voir si cette fonction d'entropie-croisée une bonne fonction d'erreur. On suppose d'abord que $a \approx 0$ et que $y = 0$. En se concentrant sur une seule donnée x, on a $n = 1$ et on remarque que

$$\begin{aligned} C &= -\frac{1}{n}\sum_x y\ln(a) + (1-y)\ln(a) \\ &= -y\ln(a) - (1-y)\ln(1-a) \approx 0. \end{aligned}$$

Un raisonnement similaire tient également lorsque $a \approx 1$ et $y = 1$. Ainsi, lorsque le réseau produit une réponse proche de celle qui est désirée, l'erreur est minime. Supposons maintenant que le réseau fait très mal son travail, par exemple $a \to 0^+$ et $y = 1$. On a alors

$$C = -y\ln(a) - (1-y)\ln(1-a) \to \infty.$$

Donc, si $a \approx y$, le coût (ou l'erreur) est bas et il est élevé lorsque a diffère substantiellement de y, ce qui rend raisonnable l'usage de l'entropie-croisée comme fonction d'erreur. Ceci dit, est-ce que cette fonction est meilleure que la fonction d'erreur quadratique qui causait le problème d'apprentissage de la section précédente? Pour y voir plus clair, on pose $y = y(x)$, $a = \sigma(z)$ et on calcule les dérivées partielles suivantes

$$\begin{aligned} \frac{\partial C}{\partial w_j} &= \frac{\partial}{\partial w_j}\left(-\frac{1}{n}\sum_x y\ln(a) + (1-y)\ln(1-a)\right) \\ &= -\frac{1}{n}\sum_x y\frac{1}{\sigma(z)}\sigma'(z)x_j + (1-y)\frac{1}{1-\sigma(z)}(-\sigma'(z))x_j \\ &= -\frac{1}{n}\sum_x \frac{y(1-\sigma(z))\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))} - \frac{(1-y)\sigma(z)\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))} \\ &= -\frac{1}{n}\sum_x \left(\frac{y - y\sigma(z) - \sigma(z) + y\sigma(z)}{\sigma(z)(1-\sigma(z))}\right)\sigma'(z)x_j \\ &= \frac{1}{n}\sum_x \frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))}(\sigma(z) - y). \end{aligned}$$

De plus, on sait que, si $\sigma(z) = \dfrac{1}{1+e^{-z}}$, alors $\sigma'(z) = \dfrac{e^{-z}}{1+e^{-z}}$. Or,

$$(1-\sigma(z)) = 1 - \frac{1}{1+e^{-z}} = \frac{1+e^{-z}-1}{1+e^{-z}} = \frac{e^{-z}}{1+e^{-z}}.$$

On obtient ainsi la relation

$$\sigma'(z) = (1-\sigma(z))\sigma(z).$$

et on a donc finalement

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z) - y).$$

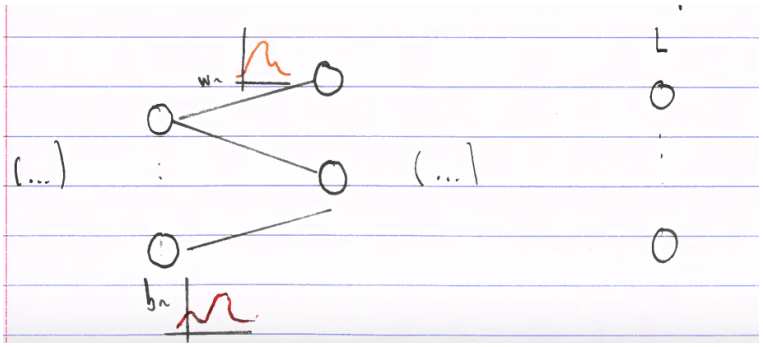
Le taux d'apprentissage du poids w_j est donc contrôlé par la quantité $(\sigma(z) - y)$. Ainsi, plus large est l'erreur, plus vite on apprend, ce qui est une propriété désirable!En particulier, on évite également le problème d'apprentissage causé par $\sigma'(z)$ décrit dans la section précédente.

3.3 Initialisation et problèmes d'apprentissage : théorèmes de Hanin

On considère un réseau ReLU, c'est-à-dire un réseau où la fonction d'activation est $\sigma(z) = \max(0, z)$ avec des couches de largeur $d_j; j = 1, 2, ..., L$ et dont les poids et les biais sont initialisés *aléatoirement*

$$\begin{cases} w_{jk}^{\ell} \sim W_{\ell}, \\ b_j^{\ell} \sim B_{\ell}, \end{cases} \tag{3.1}$$

où W_ℓ et B_ℓ sont des densités de probabilité. On remarque au passage que si les paramètres w, b sont aléatoires, alors le réseau lui-même devient une fonction aléatoire.



Les paramètres d'un réseau sont échantillonés à partir de distributions aléatoires.

On notera aussi $a^{\ell} = \begin{pmatrix} a_1^{\ell} \\ \vdots \\ a_{d_\ell}^{\ell} \end{pmatrix}$ le vecteur des activations de la couche ℓ.

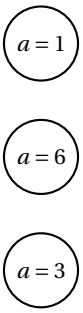
Définition 3.2 : Activation moyenne

L'**activation moyenne** (normalisée) M_ℓ de la couche ℓ est donnée par

$$M_{\ell} = \frac{1}{d_{\ell}} ||a^{\ell}||^2.$$

Exemple 3.3

On considère la seconde couche d'un réseau de neurones suivante.



Elle correspond au vecteur d'activation $a^2 = \begin{pmatrix} 1 \\ 6 \\ 3 \end{pmatrix}$, avec $d_2 = 3$ et $||a^2||^2 = (1^2 + 6^2 + 3^2) = 46$. Donc,

l'activation moyenne de la couche 2 de ce réseau est

$$M_2 = \frac{1}{d_2}||a^2||^2 = \frac{1}{3}\cdot 46 \approx 15,3.$$

La question qui nous occupera pour le reste de ce chapitre est la suivante: que se passe-t-il avec l'apprentissage du réseau si M_L devient de plus en plus petit (ou grand) au fur et à mesure que L augmente?

Définition 3.3 : Échec d'apprentissage du 1er type (EA1)

On définit l'**échec d'apprentissage du 1er type** par

L'activation moyenne M_L de la dernière couche augmente ou diminue exponentiellement avec la profondeur L du réseau.

On suppose ici que l'entrée $x \in \mathbb{R}^{d_1}$ est fixée. La motivation derrière cette dernière définition est la suivante: si on peut trouver une manière efficace d'éviter (EA1), on devrait alors pouvoir entraîner un réseau profond aussi facilement qu'un réseau peu profond.

Théorème 3.1 : (EA1) Hanin, Rolnick (2018)

L'espérance $\mathbb{E}(M_L)$ de l'activation moyenne de la dernière couche est égale à la norme $||x||$ de l'entrée si

i. W_ℓ et B_ℓ sont des distributions symétriques.

ii. Les variables aléatoires w^ℓ_{jk} et b^ℓ_j sont indépendantes.

iii. Les variances des distributions W_ℓ et B_ℓ satisfont

$$\text{Var}(W_\ell) = \text{Var}(B_\ell) = \frac{2}{\text{fan-in}} = \frac{2}{d_{\ell-1}}.$$

On appellera donc le *fan-in* d'un poids ou d'un biais associé à une couche ℓ la taille $d_{\ell-1}$ de la couche précédente.Notons au passage que si ces variances sont plus élevées, alors $\mathbb{E}(M_L)$ croîtra exponentiellement avec L tandis que c'est la diminution qui sera exponentielle dans le cas où les variances sont inférieures au seuil crucial de $\frac{2}{d_{\ell-1}}$. On rappelle également que deux variables aléatoires X et Y sont *indépendantes* si

$$P(X = x \text{ et } Y = y) = P(X = x)\cdot P(Y = y)$$

pour tout choix de réalisations x et y. Finalement, on calcule la variance d'une variable aléatoire X par

$$\text{Var}(X) = \mathbb{E}\left[(X - \mathbb{E}(X))^2\right] = \mathbb{E}(X^2) - (\mathbb{E}(X))^2.$$

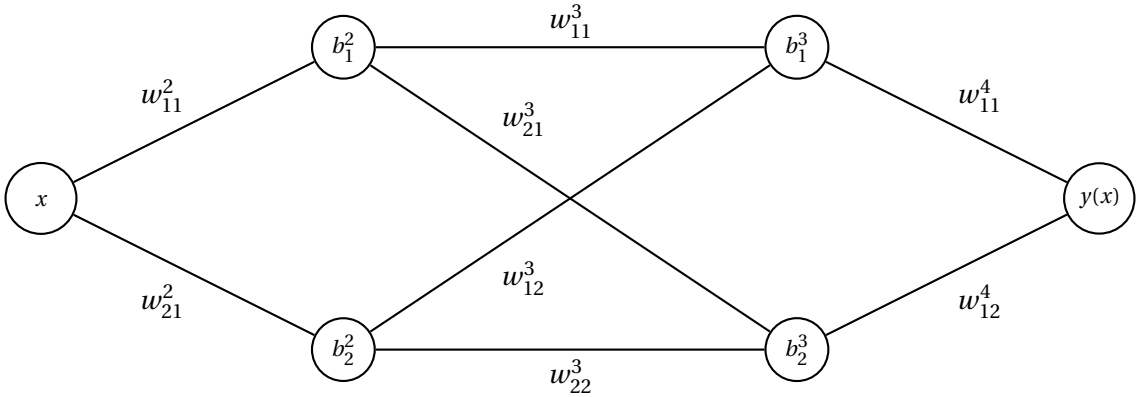
Chapitre 4

Universalité et puissance expressive

On sait qu'un réseau de neurones est une fonction $\mathscr{F} : \mathbb{R}_1^d \to \mathbb{R}^{d_L}$. Est-il raisonnable d'espérer que celui-ci puisse approximer une fonction modèle f ? Autrement dit, quelle est la puissance expressive théorique d'un réseau de neurones ? Ce sont les questions auxquelles nous allons nous consacrer dans ce chapitre.

4.1 La question de l'universalité

On considère un réseau de neurones comme celui-ci :



De manière générale, un réseau de neurones est une fonction $f : \mathbb{R}^m \to \mathbb{R}^n$, avec $m = d_1$ et $n = d_L$, où les nombres $\{d_1, d_2, ..., d_L\}$ donnent la *largeur* de chaque couche. Une interrogation naturelle est celle de l'*universalité* de ces réseaux de neurones :

« Étant donnée une fonction idéale ou naturelle f^*,
existe-t-il un réseau de neurones f tel que $f \approx f^*$? »

4.2 Le théorème de Cybenko

Définition 4.1 : Espace de fonctions continues

Soit $X \subset \mathbb{R}^n$. Alors, $C^0(X)$ est l'espace contenant les fonctions

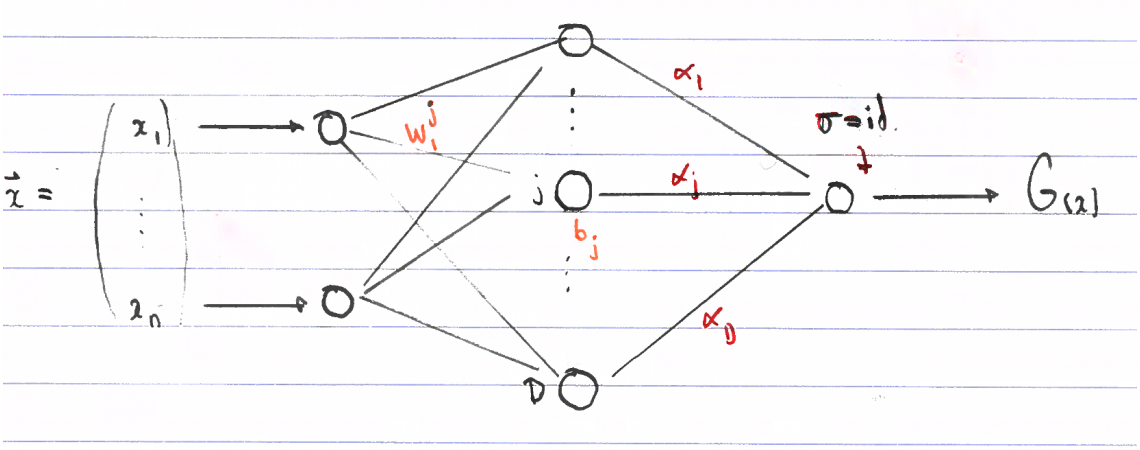
$$f : X \to \mathbb{R}$$

qui sont partout continues.

On considère maintenant l'ensemble

$$S = \{G(x) = \sum_{j=1}^{D} \alpha_j \sigma(w^j \cdot x + b_j) : w^j \in \mathbb{R}^n;\ \alpha_j, \beta_j \in \mathbb{R}\},$$

où $x \in \mathbb{R}^n$. Les éléments $G(x) \in S$ correspondent aux réseaux



Ici, les w^j sont les vecteurs contenant tous les poids qui entrent dans le $j-$ème neurone de la couche profonde. Donc, les $G(x)$ sont des réseaux de

$$\mathbb{R}^n \to \mathbb{R}$$

avec une seule couche cachée de largeur (ou dimension) D et sans activation à la sortie.

Définition 4.2 : Hypercube unité

L'**hypercube unité** I^n de dimension n est l'ensemble

$$I^n = [0,1]^n \subset \mathbb{R}^n.$$

On généralise désormais la classe de fonctions d'activation qui ressemblent à la sigmoïde employée dans les chapitres précédents.

Définition 4.3 : Activations sigmoïdales

Une fonction d'activation

$$\sigma : \mathbb{R} \to \mathbb{R}$$

est **sigmoïdale** si

$$\sigma(x) \to \begin{cases} 0 & \text{si } x \to -\infty, \\ 1 & \text{si } x \to \infty. \end{cases}$$

Exemple 4.1

On se rappelle la définition de l'activation sigmoïde

$$\sigma(z) = \frac{1}{1+e^{-z}}.$$

Alors, $\lim_{z\to-\infty} \sigma(z) = 0$ et $\lim_{z\to\infty} \sigma(z) = 1$. Donc, σ est sigmoïdale.

Le théorème suivant fournit une réponse claire à la question de l'universalité posée au début de ce chapitre.

Théorème 4.1 : Cybenko (1989)

Soit σ une fonction d'activation sigmoïdale continue. Alors, l'ensemble S est *dense* dans $C^0(I^n)$. Autrement dit, pour tout $f^* \in C^0(I^n)$ et pour tout $\epsilon > 0$, il existe

$$G(x) = \sum_{j=1}^{D} \alpha_j \sigma(w^j \cdot x + b_j)$$

tel que

$$\sup_{x\in I^n} |G(x) - f^*(x)| < \epsilon.$$

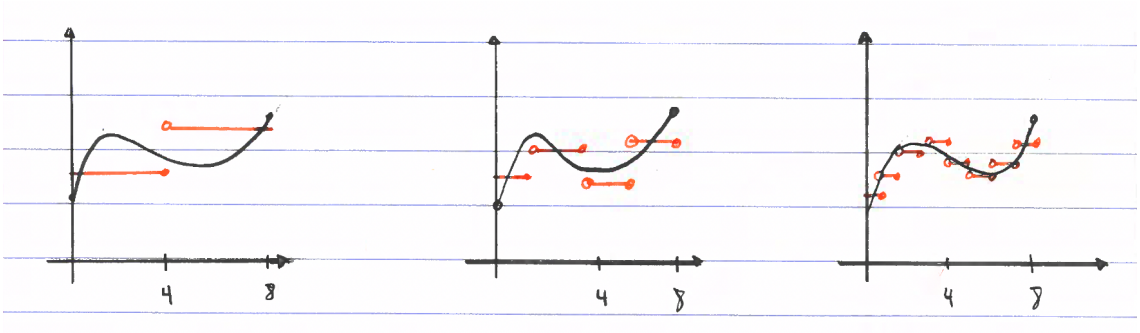
La quantité

$$||G - f||_{L^\infty} = \sup_{x\in I^n} |G(x) - f^*(x)|$$

mesure l'erreur ponctuelle d'approximation de f^* par G en $x \in I^n$. On peut ainsi voir $\epsilon > 0$ comme le seuil de tolérance à l'erreur de notre approximation. C'est donc dire que l'on peut approximer n'importe quelle fonction continue f^* avec un seuil de précision ϵ arbitrairement petit, par un réseau qui contient une seule couche cachée. C'est ce que l'on veut dire par *universalité* des réseaux de neurones.

4.3 Heuristique de l'universalité

On présente dans cette section les principales idées derrière une preuve d'un théorème d'universalité comme celui de Cybenko. Le principe général est d'approximer une fonction compliquée par des objets plus simples. On considère



Ces fonctions « escaliers » sont constantes par morceaux et on les appelle des fonctions *simples*.

Définition 4.4 : Fonctions indicatrices, simples

- Soit $(a,b) \in \mathbb{R}$ un intervalle. La **fonction indicatrice** de l'intervalle (a,b) est définie par

$$\chi_{(a,b)} = \begin{cases} 1 & \text{si } x \in (a,b), \\ 0 & \text{sinon.} \end{cases} \tag{4.1}$$

- Une fonction **simple** est une fonction qui peut s'écrire sous la forme

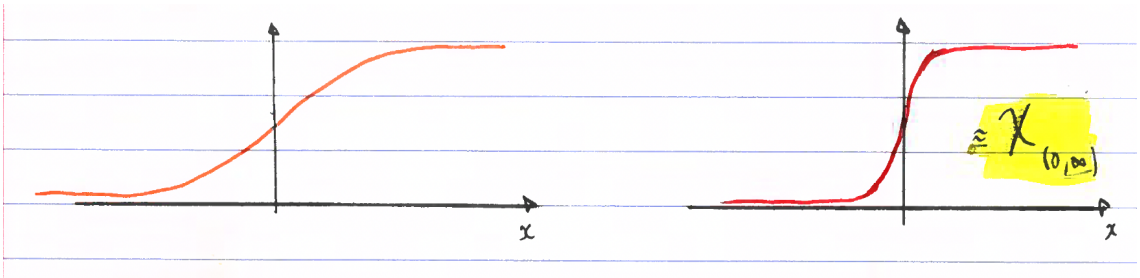
$$s(x) = \sum_{i=1}^{n} \alpha_i \chi_{(a,b)}.$$

Il s'avère qu'en prenant suffisamment de morceaux, c'est-à-dire en laissant n devenir assez grand dans la définition précédente, on peut estimer avec une précision arbitrairement grande les fonctions continues par les fonctions simples.

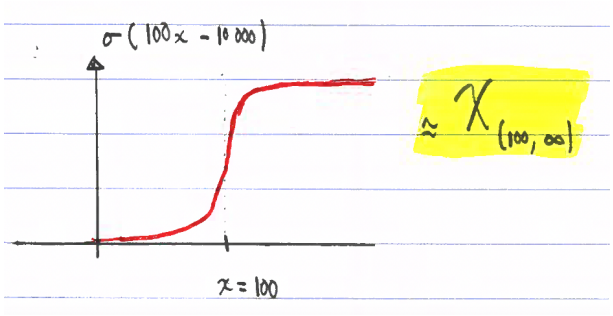
Théorème 4.2 : Densité des fonctions simples

Les fonctions simples sont denses dans $L^p(I^n) \subseteq C^0(I^n)$.

Il s'agit maintenant de voir si les réseaux $G(x)$ du théorème de Cybenko peuvent produire de telles fonctions simples. Pour le reste de cette section, on fixe $\sigma(x) = \frac{1}{1+e^{-x}}$. On considère les graphes de $\sigma(x)$ et $\sigma(100x)$.



On remarque que le graphe de $\sigma(100x)$ ressemble à celui de la fonction indicatrice de l'intervalle $(0,\infty)$. Ainsi, on peut par exemple considérer $\sigma(100x - 10000)$



De manière générale, pour λ suffisamment grand, on a

$$\sigma(\lambda x + \beta) \approx \chi_{(a,\infty)},$$

avec $a = -\frac{\beta}{\lambda}$. De plus, on remarque que

$$\chi_{(a,b)} = \chi_{(a,\infty)} - \chi_{(b,\infty)}.$$

Donc, les combinaisons linéaires des $\sigma(\lambda x + \beta)$ engendrent les fonctions simples. Or, ces combinaisons linéaires sont précisément les $G(x)$ des réseaux du théorème de Cybenko! En effet, ces combinaisons linéaires sont de la forme

$$G(x) = \sum_{i=1}^{N} \alpha_i \sigma(\lambda x + \beta).$$

Quelques remarques et questions en terminant pour compléter notre discussion sur la puissance expressive.

- Nous n'avons fait aucun usage de la profondeur ici : les réseaux du théorème de Cybenko ont $L = 3$ et $d_2 = N = N(\epsilon)$ grand ! En fait, typiquement on aura

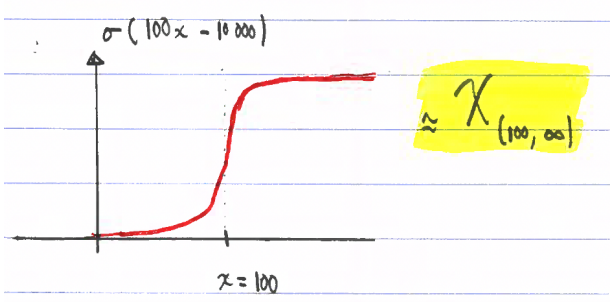
  $$N(\epsilon) \to \infty$$

  quand $\epsilon \to 0$.

- Nos « blocs lego » de construction sont des fonctions simples. Pourrait-on utiliser la profondeur des réseaux de neurones pour construire des « blocs lego » plus flexibles ?

- Est-ce que la profondeur sert vraiment ? Comment, avec un nombre fini de neurones, concevoir le « meilleur » réseau ?

La figure suivante présente deux paradigmes généraux d'architecture pour les réseaux de neurones, soit respectivement les modes *wide and shallow* et *deep and narrow*.



Chapitre 5

Rôle de la profondeur dans la puissance expressive

On a vu dans le chapitre précédent que les réseaux de neurones étaient des approximateurs universels. Or, pour établir cet important constat, aucun usage de la profondeur n'a été. Est-il donc avantageux de disposer des neurones en couches successives plutôt que de les superposer dans une seule couche large? L'objectif de ce chapitre est de clarifier la portée de l'usage de la profondeur dans les réseaux de neurones.

5.1 Une section en dents de scie

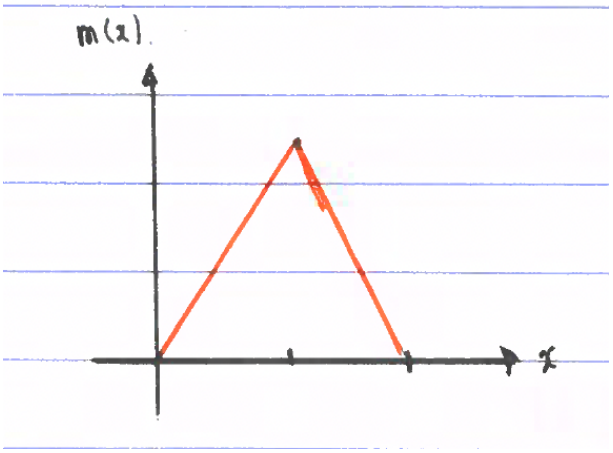
On rappelle le sous-espace vectoriel composé des réseaux de neurones du théorème de Cybenko

$$S = \{G(x) = \sum_{j=1}^{D} \alpha_j \sigma(w^j \cdot x + b_j) : w^j \in \mathbb{R}^n; \alpha_j, \beta_j \in \mathbb{R}\}$$

et où $x \in \mathbb{R}^n$. Nous savons par le théorème de Cybenko que S est dense dans l'espace des fonctions continues. Or, ces réseaux sont très peu profonds: ils ont une seule couche cachée. Est-ce que la profondeur sert vraiment? Pour explorer cette question, on définit l'ensemble $\mathscr{R}(L, D; \sigma)$ comme celui contenant tous les réseaux de neurones de profondeur L, de largeur maximal D et activés par σ. Soit aussi la fonction $m : [0,1] \to \mathbb{R}$ définie par

$$m(x) = \begin{cases} 2x & \text{si } 0 \leq x \leq \frac{1}{2}, \\ 2-2x & \text{si } \frac{1}{2} \leq x \leq 1. \end{cases}$$

Existe-t-il un réseau de neurones capable de calculer exactement la fonction $m(x)$?



On fixe $\sigma = \text{ReLU}$. Alors, on a

$$2\sigma(x) = \begin{cases} 2x & \text{si } 0x \geq 0, \\ 0 & \text{si } x \leq 0. \end{cases}$$

De plus,

$$\sigma\left(x - \frac{1}{2}\right) = \begin{cases} x - \frac{1}{2} & \text{si } x \geq \frac{1}{2}, \\ 0 & \text{sinon.} \end{cases}$$

Aussi,

$$-4\sigma\left(x - \frac{1}{2}\right) = \begin{cases} 2 - 4x - \frac{1}{2} & \text{si } x \geq \frac{1}{2}, \\ 0 & \text{sinon.} \end{cases}$$

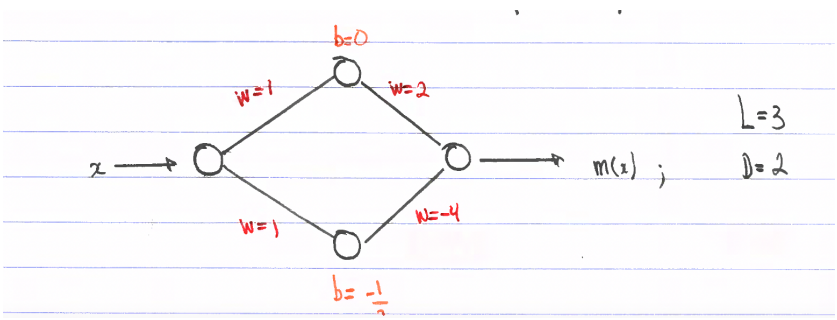
Ainsi,

$$2\sigma(x) - 4\sigma\left(x - \frac{1}{2}\right) = \begin{cases} 0 & \text{si} x \leq 0, \\ 2x & \text{si } 0 \leq x \leq \frac{1}{2}, \\ 2 - 2x & \text{si } x \geq \frac{1}{2}. \end{cases}$$

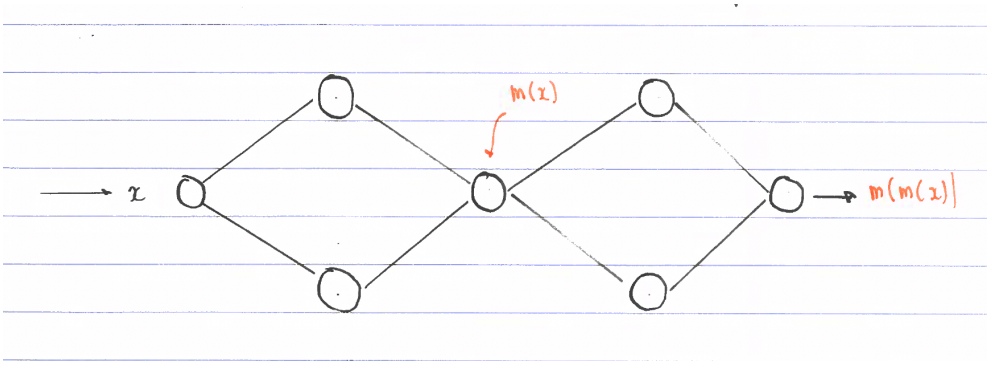
Donc,

$$m(x) = \sigma\left(2\sigma(x) - 4\sigma\left(x - \frac{1}{2}\right)\right),$$

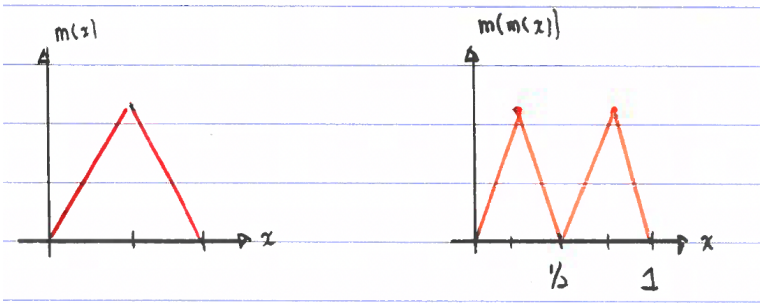
ce qui correspond au réseau suivant.



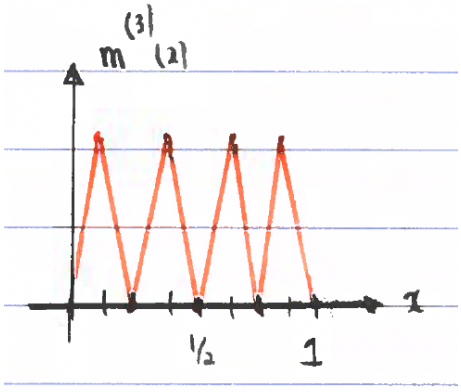
Que se passe-t-il si on rend le réseau plus profond?



Graphiquement, on remarque donc que la composition produit plus de dents de scie.



De manière similaire, on peut aussi considérer la composition $m^{(3)}(x) = m(m(m(x)))$.



La relation entre l'ordre de la composition et le nombre de pics produits sur le graphe est décrite dans le tableau suivant.

n	Nombre de neurones dans le réseau de $m^{(n)}$	Nombre de pics produits
1	$4 = 3+1$	1
2	$7 = 2\cdot 2+1$	2
3	$10 = 3\cdot 2+1$	4
$\vdots$		
n	$3n+1$	2^{n-1}

En général, le réseau associé à la composition d'ordre n, c'est-à-dire à $m^{(n)(x)}$ produit donc 2^{n-1} oscillations avec

$$\begin{cases} L = 2n+1 \text{ couches}, \ D = 2, \\ 3n+1 \text{neurones}, \\ O(n) \text{ paramètres}. \end{cases}$$

Et si on disposait nos $3n+1$ neurones à la manière des réseaux du théorème de Cybenko, soit en largeur sur une seule couche profonde? Autrement dit, considérons plutôt

$$g_n \in \mathscr{R}(L=3, D=3n; \sigma),$$

où

$$g_n(x) = \sum_{j=1}^{3n} \alpha_j \sigma(w^j \cdot x + b_j).$$

Les réseaux g_n utilisent le même nombre de neurones que ceux qui représentent $m^{(n)}(x)$ et il est naturel de se demander laquelle des deux configurations est la plus avantageuse pour la puissance expressive. On veut donc comparer le rôle de la composition à celui de l'additivité.

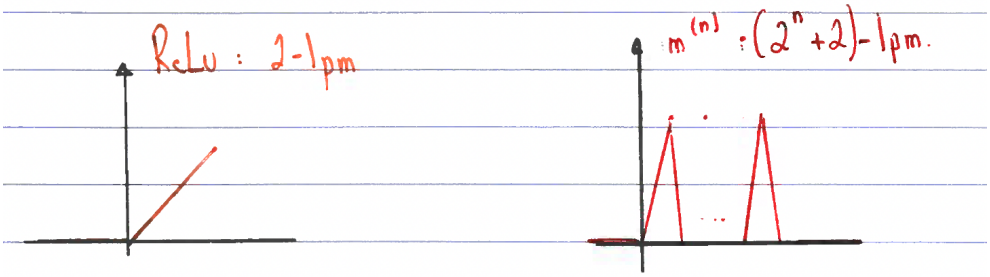
5.2 Additivité vs. composition

Définition 5.1 : Fonction α-linéaire par morceaux.

Une fonction f est α **linéaire par morceaux** (α-lpm) si elle est linéaire par morceaux avec *au plus* α morceaux.

Exemple 5.1

La fonction ReLU est clairement $2-lpm$ tandis que $m^{(n)}(x)$ est $(2^n+2)-lpm$.



Le résultat suivant précise les effets respectifs de l'addition et de la composition sur la linéarité par morceaux.

Théorème 5.1

Soit f une fonction α-lpm et g une fonction β-lpm. Alors,

i. $(f+g)$ est $(a+b)$-lpm.

ii. $(f\circ g)$ est (ab)-lpm.

Comme corollaire, on obtient donc immédiatement les faits suivants :

- $m^{(n)}(x)$ est 2^n-lpm.
- $g\in\mathscr{R}(L=3;D=3n;\sigma=\text{ReLU})$ est $6n$-lpm.

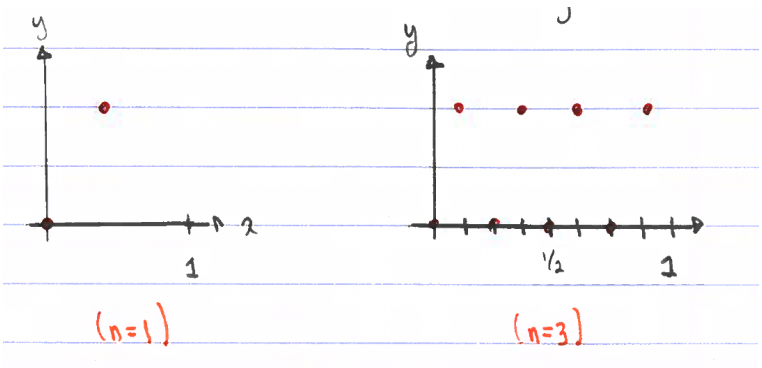
On voit donc ici toute la portée de la composition dans la puissance expressive. En effet, si par exemple $n=100$, alors le réseau profond $m^{(100)}(x)$ peut créer jusqu'à 2^{100} morceaux tandis que celui peu profond à la Cybenko ne peut qu'en créer 600.

5.3 Théorème de séparation de Telgarsky

Le but de cette section est de préciser davantage la différence de la portée des réseaux profonds par rapport à ceux qui ne le sont pas. Pour ce faire, nous utiliserons une tâche de classification qui permettra de mesurer les performances de différents réseaux.

Définition 5.2 : Problème de n-classification

Le **problème de classification de n points**, aussi appelé **problème de n-classification** est le suivant. On considère 2^n points $(x_i, y_i) \in \mathbb{R}^2$, avec $x_i \in [0, 1-2^{-n}]$ et $y_i = 0, 1, 0, 1...$



Pour $f : [0,1] \to \mathbb{R}$, on considère la discrétisation $\tilde{f}$ de l'image de f, définie par

$$\tilde{f}(x) = \begin{cases} 1 & \text{si } f(x) \geq \frac{1}{2}, \\ 0 & \text{si } f(x) < \frac{1}{2}. \end{cases}$$

L'**erreur** $E(f)$ d'une fonction f au test de n-classification est alors donnée par

$$E(f) = \frac{1}{2^n} \sum_i \chi\left\{\tilde{f}(x_i) \neq y_i\right\},$$

où $\chi\left\{\tilde{f}(x_i) \neq y_i\right\}$ vaut 1 quand f a échoué à classer correctement le point x_i et 0 dans le cas contraire.

On peut voir $E(f)$ comme la proportion de points mal classés par f au problème de n-classification. On remarque d'emblée que $E(m^{(n)}(x)) = 0$.

Théorème 5.2 : Telgarsky (2016)

Soit $g \in \mathscr{R}(L, D; \sigma)$ avec $D < 2^{\frac{n-1}{L-1}}$. Alors,

$$E(g) > \frac{1}{6}.$$

Ce résultat tranche de manière définitive le débat entre les réseaux à la Cybenko et ceux disposés en profondeur. En effet, si on force le réseau g à avoir $L = 3$ couches (Cybenko), alors il faut $D \geq 2^{\frac{n-1}{2}}$ pour pouvoir espérer approcher les performances de $m^{(n)}$ au test de n-classification. On dit donc que la profondeur *sépare* le pouvoir expressif des réseaux de neurones.

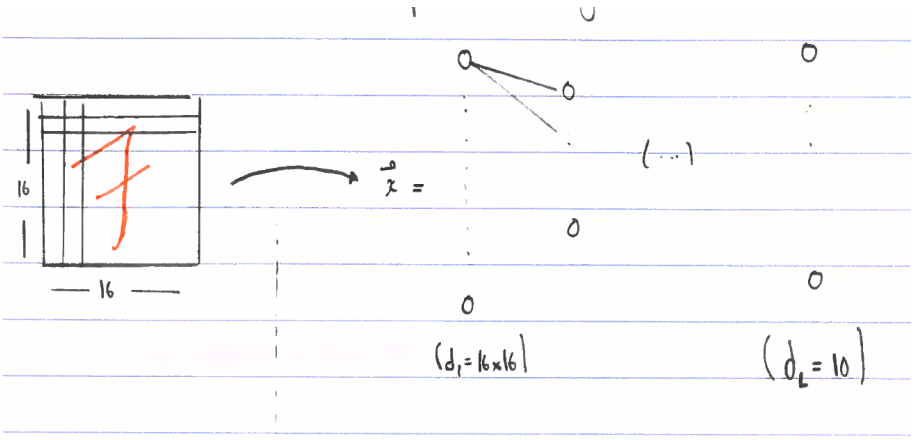
Chapitre 6

Introduction à l'apprentissage profond géométrique

On a vu comment les réseaux de neurones pouvaient apprendre à classer des images représentant des chiffres manuscrits. L'objectif de ce chapitre est de présenter une manière de faire apprendre des réseaux de neurones lorsque les données ne sont pas représentées sous forme de grille.

6.1 Réseaux de convolution et classification d'images

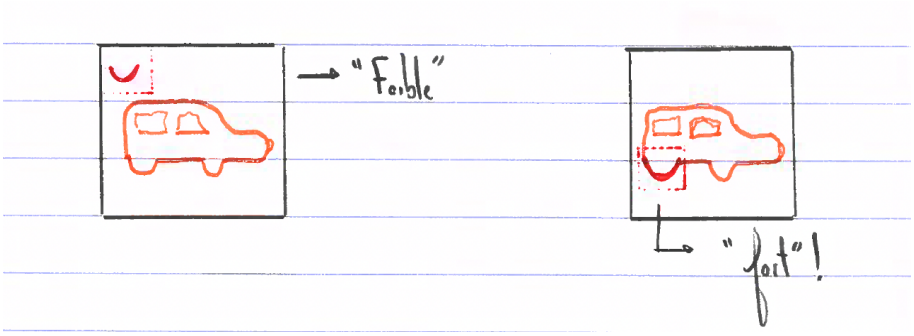
L'approche la plus simple pour la tâche de classification d'images et de prendre le *bitmap* représentant une image et de l'aplatir en vecteur colonne pour le fournir en entrée à un réseau dont la première couche contient autant de neurones que l'image a de pixels. Est-ce la meilleure approche ?



On voudrait construire un réseau qui tient compte de deux aspects de notre tâche :

- Invariance sous translation : « *une voiture est une voiture, peu importe où elle est placée dans l'image.* »
- Localité : « *les pixels proches ont tendance à se ressembler.* »

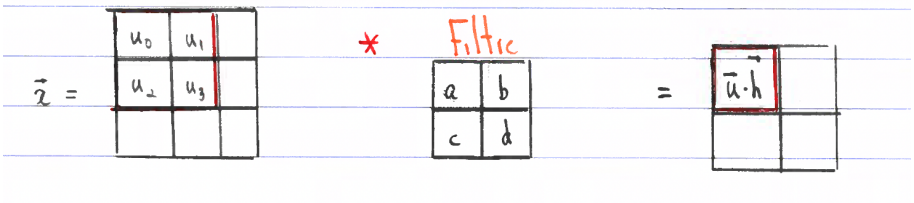
Comment, par exemple, reconnaître une roue ? On utilise un *filtre* ou *noyau* qui va balayer l'image et donner à chaque position un score de ressemblance.



C'est cette idée qui est à la base des réseaux dits de *convolution*. On rappelle d'abord que, puisque

$$\vec{u} \cdot \vec{v} = ||\vec{u}|| \cdot ||\vec{v}|| \cos\theta,$$

le produit scalaire peut être interprété comme une mesure de similarité entre deux vecteurs $\vec{u}$, $\vec{v}$. On dira qu'une *couche convolutionnelle* C_Γ agit de la manière suivante.



Ici, $\vec{u} = (u_0, u_1, u_2, u_3)^T$ est une sous-grille de l'image x et $\vec{h} = (a, b, c, d)^T$ est un filtre. Précisons maintenant certains termes utilisés avec cette méthode.

- *Stride:* incrément de balayage du filtre.
- *Padding:* rajout artificiel de lignes/colonnes au bord pour préserver la dimension.

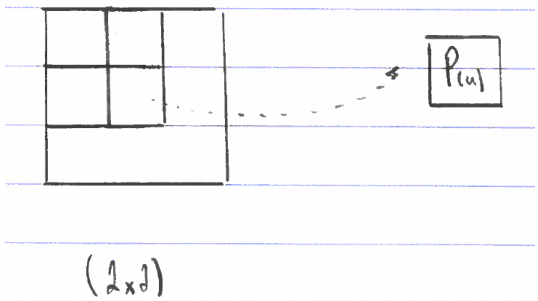
Pourquoi le nom *convolution*? On rappelle que la convolution discrète de deux signaux u et h est définie par

$$(u * h)(n) = \sum_{k=-\infty}^{\infty} u(k)h(n-k) = \sum_{k=0}^{3} u(k)h(n-k),$$

car $u(k) = 0$ dès que $k < 0$ ou $k > 3$. Par exemple,

$$(u * h)(3) = u_0 d + u_1 c + u_2 b + u_3 a$$

et donc la convolution telle que définie plus haut correspond au produit scalaire de u avec une rotation du filtre h. On fera également usage d'une couche dite de *pooling*. Une telle couche P agit pour réduire la dimension de la manière suivante.



Ici, $P(u) = \max u_0, u_1, u_2, u_3$.

Définition 6.1 : Réseau de convolution

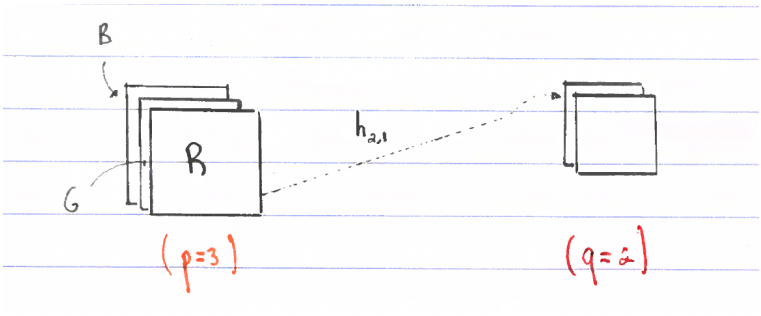
Un **réseau de convolution** U_θ est un réseau de la forme

$$U_\theta = \mathrm{FC} \circ \left(C_{\Gamma^{(*)}} \circ P \circ \ldots \circ P \circ C_{\Gamma^{(1)}}\right),$$

où une couche convolutionnelle $g = C_\Gamma(x)$ agit sur une entrée p-dimensionnelle $x = (x_1, x_2, \ldots, x_p)$ en appliquant une banque de **filtres**

$$\Gamma = \left(h_{\ell,\ell'}\right),$$

avec $\ell = 1, \ldots, q$ et $\ell' = 1, \ldots, p$.



Il est naturel de maintenant se questionner sur la possibilité de définir un tel réseau de convolution lorsque les données d'entrée ne sont pas organisées en grille, comme c'est le cas par exemple avec un graphe ou une variété. Ce sera notre objectif pour la suite de ce chapitre.

6.2 Préliminaires mathématiques

Soit E un espace vectoriel sur $\mathbb{R}$ euclidien et donc muni d'un produit scalaire et soit $\{e_1, e_2, \ldots, e_n\}$ une base orthonormale (ONB) de E. Alors, pour tout $u \in E$, il existe $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ tels que

$$u = \sum_{i=1}^{n} \alpha_i e_i$$

et

$$(u, e_i) = \alpha_i.$$

On rappelle maintenant le théorème spectral vu au chapitre I.

Théorème 6.1 : Théorème spectral

Soit A une matrice symétrique. Alors, il existe une base orthonormale de vecteurs propres de A et on peut écrire

$$D = Q^{-1} A Q$$

où $Q \in O(n)$ est une matrice orthogonale.

Soit maintenant $X \subset \mathbb{R}^n$ compact (i.e. fermé et borné) et

$$L^2(X) = \left\{ f : X \to \mathbb{R} \text{ telles que } \int_X f^2 dV < \infty \right\}$$

l'espace des fonctions de carré intégrable sur X.

Définition 6.2 : Laplacien euclidien

Le **laplacien** est l'opérateur différentiel

$$\Delta : L^2(X) \to L^2(X)$$

défini par

$$\Delta = -\text{div}\,\nabla = \sum_{i=1}^{n} \frac{\partial^2}{\partial x_i^2}.$$

Théorème 6.2 : Hilbert-Schmidt

Il existe une base orthonormale de $L^2(X)$ formée par les fonctions propres ϕ_i du laplacien.

Les fonctions propres satisfont

$$\Delta\phi_i = \lambda_i \phi_i,\ i = 1, 2...$$

et on a alors que pour toute fonction $u \in L^2(X)$,

$$u = \sum_i \alpha_i \phi_i,$$

avec

$$(\phi_i, \phi_j) = \int_X \phi_i \phi_j dV = \delta_{ij}$$

et

$$0 \leq \lambda_1 \leq \lambda_2 \leq ... \nearrow \infty.$$

On rappelle désormais brièvement certains éléments de la théorie de Fourier. On remarque d'abord que

$$\Delta\left(e^{-2\pi i\xi t}\right) = -\frac{\partial^2}{\partial t^2}\left(e^{-2\pi i\xi t}\right) = -(-2\pi i\xi)^2\left(e^{-2\pi i\xi t}\right) = (4\pi^2\xi^2)e^{-2\pi i\xi t}.$$

Ainsi, les fonctions $e^{-2\pi i\xi t}$ sont des fonctions propres du laplacien sur $\mathbb{R}$.

Définition 6.3 : Transformée de Fourier et transformée inverse

- La **transformée de Fourier** d'une fonction f est donnée par

$$\hat{f}(\xi) = \mathscr{F}(f)(\xi) = \int_{\mathbb{R}} f(t) e^{2\pi i\xi t} dt = \left(f, e^{-2\pi i\xi t}\right).$$

- La **transformée de Fourier inverse** d'une fonction $\hat{f}$ est donnée par

$$f(x) = \mathscr{F}^{-1}(\hat{f})(x) = \int_{\mathbb{R}} f(t) e^{-2\pi i\xi t} dt = \left(f, e^{-2\pi i\xi t}\right).$$

On remarque donc qu'on peut définir une théorie de Fourier à partir d'un produit scalaire et des fonctions propres du laplacien. Cette idée nous guidera dans la construction d'une théorie similaire sur des graphes.

Théorème 6.3 : Convolution

Soit f, g deux fonctions. Alors,

$$\mathscr{F}(f * g) = \hat{f} \cdot \hat{g}.$$

Conséquemment,

$$f * h = \mathscr{F}^{-1}\left(\hat{f} \cdot \hat{g}\right).$$

À nouveau, on remarque que l'on peut donc définir la convolution à partir d'une théorie de Fourier.

6.3 Introduction à la théorie spectrale des graphes

On considère un graphe $G = (V, E, W)$ connexe et non-orienté. Ici, V est l'ensemble des sommets, E l'ensemble des arêtes et W est une matrice d'adjacence pondérée.

Définition 6.4 : Fonction sur un graphe

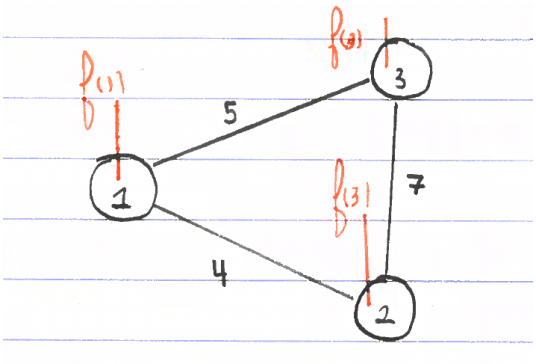
Une fonction sur g est une application

$$f : V \to \mathbb{R},$$

que l'on peut voir comme un vecteur $f \in \mathbb{R}^n$ où $n = |V|$.

Exemple 6.1

On considère le graphe suivant :



Ici, $V = \{1,2,3\}$, $W = \begin{pmatrix} 0 & 4 & 5 \\ 4 & 0 & 7 \\ 5 & 7 & 0 \end{pmatrix}$ et $W = W^T$.

Quand les arêtes n'ont pas de poids, (par exemple si G est le squelette d'une surface tridimensionnelle), on peut considérer un seuil K.

- *Noyaux gaussiens:*

$$W_{i,j} = \begin{cases} \exp\left(\frac{-d^2(i,j)}{2\theta^2}\right) & \text{si } d(i,j) \leq K, \\ 0 & \text{sinon.} \end{cases}$$

- *K plus proches voisins:*

$$W_{i,j} = \begin{cases} 1 & \text{si } d(i,j) \leq K, \\ 0 & \text{sinon.} \end{cases}$$

Il faut maintenant développer une théorie spectrale du laplacien Δ sur G.

Définition 6.5 : Théorie spectrale du laplacien sur un graphe

- *g-gradient:*

$$\nabla : L^2(V) \to L^2(E)$$
$$(\nabla f)_{ij} = \big(f(i) - f(j)\big).$$

- *g-divergence:*

$$\operatorname{div} : L^2(E) \to L^2(V)$$
$$(\operatorname{div} F)_i = \sum_{j:(i,j)\in E} W_{ij} F_{ij}.$$

- *g-laplacien:*

$$\Delta : L^2(V) \to L^2(V); \quad \Delta = -\operatorname{div} \nabla$$
$$(\Delta f)_i = \sum_{j:(i,j)\in E} W_{ij}(f(i) - f(j)).$$

Puisque $L^2(V) = \mathbb{R}^n$, Δ est une matrice que l'on peut écrire

$$\Delta = D - W,$$

où

$$D = \operatorname{diag}\big(\deg(1), ..., \deg(n)\big)$$

et $\deg(i) = \sum_j W_{ij}$. Ainsi, Δ est symétrique et, en vertu du théorème spectral, il existe

$$\Phi = \{\varphi_1, ..., \varphi_n\}$$

une base orthonormale de $\mathbb{R}^n$ telle que

$$\Delta\varphi_i = \lambda_i \varphi_i$$

et

$$0 \leq \lambda_1 \leq ... \leq \lambda_n.$$

Par analogie avec le laplacien usuel, nous sommes désormais en mesure de construire une graphe théorie de Fourier.

Définition 6.6 : Théorie de Fourier sur un graphe

- *g-Transformée de Fourier :*

$$\mathscr{F}(f) = \hat{f}(\lambda_k) = (f, \varphi_k) = \sum_{i=1}^{n} f(i)\bar{\varphi_k}(i).$$

En notation matricielle, on obtient

$$\hat{f} = \Phi^T f.$$

- *g-Transformée de Fourier inverse :*

$$\mathscr{F}^{-1}(\hat{f}) = f(i) = \sum_{k=1}^{n} \hat{f}(\lambda_k)\varphi_k(i).$$

En notation matricielle, on obtient

$$f = \Phi \hat{f}.$$

Et nous voilà capable d'enfin définir la convolution sur un graphe.

Définition 6.7 : Convolution sur un graphe

La convolution de deux signaux $f, h \in L^2(V)$ est définie par

$$(f * h) = \mathscr{F}^{-1}(\hat{f} \cdot \hat{g}),$$

c'est-à-dire

$$(f * h)(i) = \sum_{k=1}^{n} \hat{f}(\lambda_k)\hat{h}(\lambda_k)\varphi_k(i),$$

que l'on peut écrire matriciellement par

$$(f * h) = \left(\Phi \cdot \mathrm{diag}(\hat{h}(\lambda_1), ..., \hat{h}(\lambda_n) \cdot \Phi^T\right).$$

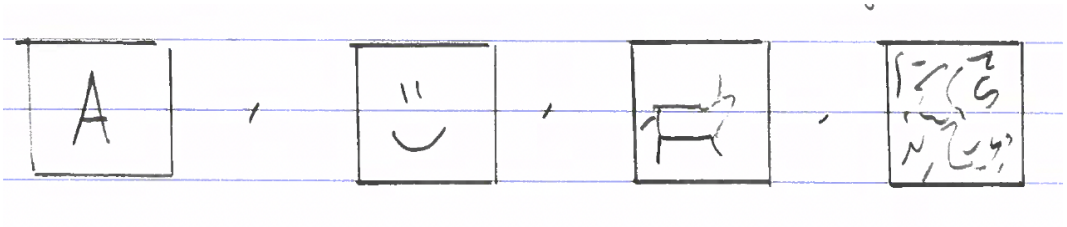
Chapitre 7

Réduction spectrale de la dimension

Les données d'entrées fournies à un réseau de neurones habitent typiquement dans un espace de dimension très élevée. Elles se logent par contre pas de manière uniforme dans cet espace et il est souhaite de vouloir réduire la dimension dans lesquelles elles existent, notamment pour des raisons d'efficacité. Dans ce chapitre, nous verrons différentes techniques pour tenter de réduire la dimension des données d'entrée.

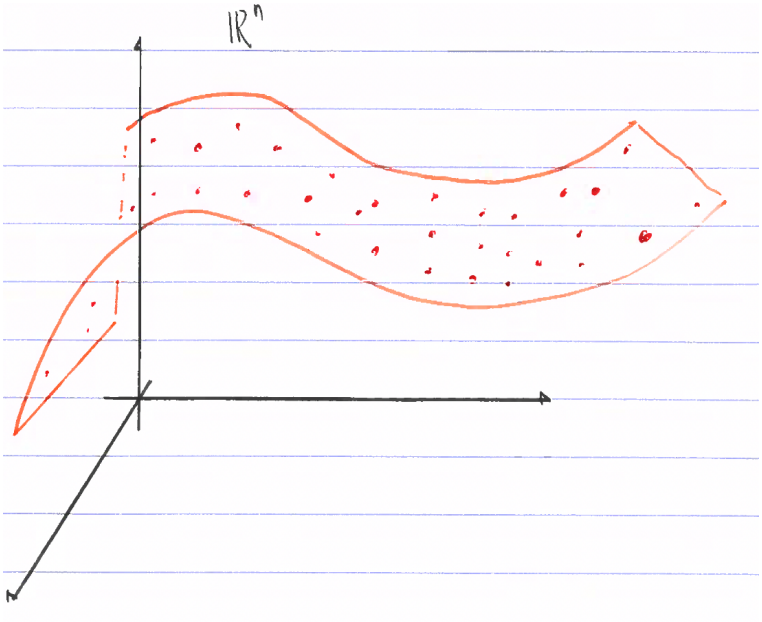
7.1 L'hypothèse de la variété

On considère encore une fois la tâche de classification de chiffres manuscrits de la base de données MNIST. Après aplatissement, une image de cette banque de données est un point $x \in \mathbb{R}^{28\times 28} = \mathbb{R}^{784}$. Est-ce que c'est vraiment l'espace dans lequel vivent ces images ?



L'*hypothèse de la variété* stipule la chose suivante :

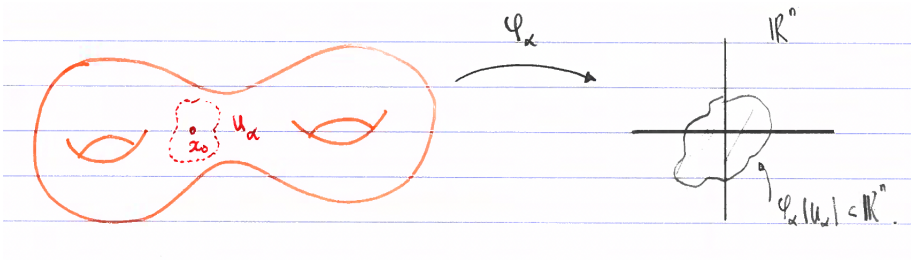
« *Les données de haute dimension sont disposées sur une variété de basse dimension, qui est plongée dans l'espace de haute dimension.* »



7.2 Variétés

Définition 7.1

- Une **variété** M^n de dimension n est un collage lisse de morceaux ressemblant à $\mathbb{R}^n$.



- Les **cartes de coordonnées** sont des applications

$$\varphi_\alpha : U_\alpha \subset M \to \mathbb{R}^n,$$

qui permettent de donner des coordonnées à tout point $p \in M$.

Si les fonctions de transition entre les cartes de coordonnées sont suffisamment régulières, on peut faire du calcul sur ces variétés : dérivées directionnelles, plan tangent etc. Si on équipe M d'une *métrique riemannienne*, on peut alors construire l'opérateur de Laplace-Beltrami

$$\Delta_g : C^\infty(M) \to C^\infty(M),$$

lequel donne lieu à des modes propres, c'est-à-dire des paires de fonctions et valeurs propres qui satisfont l'équation

$$\Delta_g \varphi_\lambda = \lambda \varphi_\lambda.$$

Si M est compacte, alors le spectre $\sigma(\Delta_g)$ est discret et composé d'une suite infinie de valeurs propres

$$0 \leq \lambda_0 \leq \lambda_1 \leq \ldots \nearrow \infty.$$

On veut montrer en quoi les fonctions propres de Δ_g permettent un plongement

$$\begin{aligned} \Pi_k : M &\to \mathbb{R}^k \\ x &\mapsto (\varphi_1(x), \ldots, \varphi_k(x)). \end{aligned}$$

7.3 Analyse en composantes principales

On débute en explorant la *caractérisation variationnelle des valeurs propres.* Soit $A = A^T$ une matrice symétrique de dimension n. Alors, il existe

$$\Phi = \{\varphi_1, \ldots, \varphi_n\}$$

une base orthonormale de vecteurs propres de A. Aussi, pour $u \in \mathbb{R}^n$, on

$$u = \sum_{i=1}^{n} \alpha_i \varphi_i$$

et

$$Au = \sum_{i=1}^{n} \lambda_i \alpha_i \phi_i.$$

Définition 7.2 : Quotient de Rayleigh

Le **quotient de Rayleigh** de $u \in \mathbb{R}^n$ est défini par

$$\mathscr{R}_A(u) = \frac{(Au, u)}{(u, u)} = \frac{u^T Au}{u^T u}.$$

Si $||u|| = 1$, on a donc $u^T u = 1$ et

$$\mathscr{R}_A(u) = (Au, u) = \left(\sum_{i=1}^{n} \lambda_i \alpha_i \phi_i, \sum_{i=1}^{n} \alpha_i \varphi_i \right) = \sum_{i=1}^{n} \lambda_i \alpha_i^2,$$

avec $\sum_{i=1}^{n} \alpha_i^2 = 1$.

Donc,

$$\arg \max_{u, ||u||=1} \mathscr{R}_A(u) = \phi_n,$$

le vecteur propre associé à λ_n et

$$\arg \min_{u, ||u||=1} \mathscr{R}_A(u) = \phi_1,$$

le vecteur propre associé à λ_1.

On considère maintenant un nuage de points $\{x^{(1)}, \ldots, x^{(k)}\} \subset \mathbb{R}^n$. Quelle est la direction (unitaire) $\tilde{w}$ le long de laquelle la variance est maximale? Pour ce faire, on voudrait maximiser

$$\sum_{i=1}^{k} \left(x^{(i)} \cdot w \right)^2 = ||Xw||^2,$$

où X est la matrice dont les k rangées sont les $x^{(i)}$ du nuage de points. On remarque aussi que X^TX est symétrique

$$(X^TX)^T = X^T(X^T)^T = X^TX.$$

On a donc

$$\tilde{w} = \arg\max_{w:||w||=1} w^T(X^TX)w = \arg\max_{w:||w||=1} R_{X^TX}(w)$$

qui est donc le vecteur propre correspondant à la valeur propre la plus élevée de X^TX. On pose $w^{(1)} := \tilde{w}$ que l'on appelle *composante principale* du nuage de points. On peut continuer et trouver

$$w^{(1)}, w^{(2)}, ..., w^{(n)}$$

composantes principales (i.e. vecteurs propres de X^TX) et réduire la dimension via les projections

$$\begin{aligned}\Pi_k :&\mathbb{R}^n \to \mathbb{R}^m\\ &x \mapsto \text{proj}_{\text{sp}\{w^{(1)},...,w^{(n)}\}}x\end{aligned}$$

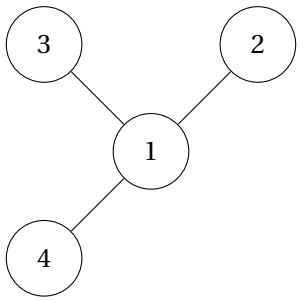
Cela revient à écrire un point $x \in \mathbb{R}^n$ du nuage de points dans la base des vecteurs propres de X^TX et de considérer les m premières coordonnées.

7.4 Réduction spectrale de la dimension

On débute avec un exemple qui va nous permettre d'illustrer la stratégie générale que nous allons employer dans cette section dont l'objectif est d'utiliser la théorie spectrale du laplacien pour définir un plongement de réduction de dimension. On pourra ensuite démontrer que ce plongement est optimal selon une certaine métrique naturelle.

Exemple 7.1

On considère 4 points x_1, x_2, x_3 et x_4 quelconques dans $\mathbb{R}^n$. Avec ces points, nous formons un graphe $\mathcal{G}$. On discutera par la suite des détails de la construction de ce graphe.



On peut ensuite créer les matrices W des poids et $D = \text{diag}\{\deg(1), \deg(2), \deg(3), \deg(4)\}$ des degrés associés à ce graphe. On a donc

$$W = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Le *g-laplacien* Δ sur $\mathscr{G}$ est donné par

$$\Delta = D - W.$$

On considère maintenant le problème aux valeurs propres généralisé suivant

$$\Delta\varphi = \lambda\varphi.$$

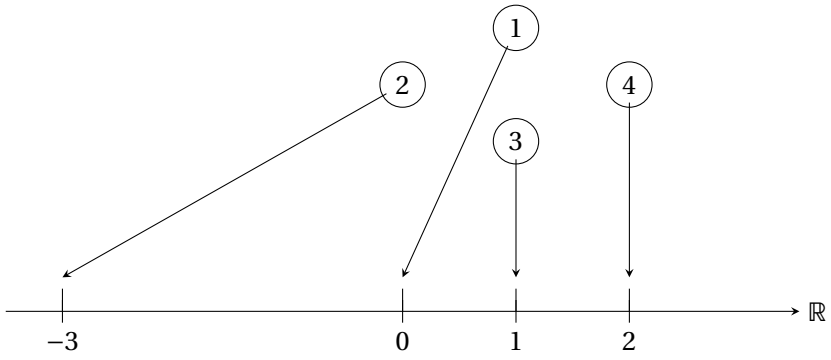
On calcule aisément les valeurs propres

$$\lambda_0 = 0, \lambda_1 = \lambda_2 = 1, \lambda_3 = 2.$$

Si on exclut le premier vecteur propre constant associé à la valeur propre nulle, un vecteur propre associé à la première valeur propre non-nulle λ_1 est

$$\varphi_1 = \begin{pmatrix} 0 \\ -3 \\ 1 \\ 2 \end{pmatrix}.$$

On peut l'utiliser pour opérer une réduction de la dimension de $d = 4$ vers $d = 1$ en associant nos points initiaux $x_1, ..., x_4$ à des points dans $\mathbb{R}$:



La stratégie générale suivra celle de cet exemple, c'est-à-dire que nous allons créer un graphe à partir d'un nuage de points pour ensuite utiliser la théorie spectrale sur ce graphe afin d'envoyer les points vers un espace de dimension inférieur en utilisant les vecteurs propres du laplacien. On peut déjà se demander: pourquoi est-ce pertinent de choisir cette manière de faire?

7.4.1 L'algorithme de Belkin-Niyogi

Soit un nuage de points $x_1, x_2, ..., x_k$ dans $\mathbb{R}^n$. On veut d'abord construire un graphe pondéré $\mathscr{G} = (V, \mathscr{E}, W)$ associé à ce nuage de points. Évidemment, ce graphe aura un sommet par point du nuage et on a donc $V = \{1, 2, ..., k\}$.

I. CHOIX D'UNE FONCTION DISTANCE.

On veut ici connecter par une arête les sommets qui sont « proches. » Deux approches principales possibles :

- n plus proches voisins.
- On fixe $\epsilon > 0$ et on connecte si $||x_i - x_j|| < \epsilon$.

II. ASSIGNATION DES POIDS.

On peut procéder de manière binaire ou alors en utilisant la méthode dite des noyaux de la chaleur.

- $W_{ij} = 1$ si $(i, j) \in \mathscr{E}$
- Noyaux de la chaleur au temps $t > 0$:

$$W_{ij} = exp(-||x_i - x_j||^2 / t).$$

III. CALCUL DES MODES PROPRES.

On considère l'équation aux valeurs propres normalisées

$$\Delta\varphi = \lambda D\varphi.$$

On obtient une suite de valeurs propres

$$0 = \lambda_0 \leq \lambda_1 \leq ..\lambda_{n-1} = 2.$$

Pour $m < n$, on définit la projection

$$\begin{aligned} \Pi_m : \mathbb{R}^n &\to \mathbb{R}^m \\ x_i &\mapsto (\varphi_1(i), ..., \varphi_m(i)). \end{aligned}$$

On exclut la fonction propre $\varphi_0 \equiv cst$.

7.4.2 Optimalité de la projection spectrale

On considère le cas $m = 1$. On a un graphe $G = (V, \mathscr{E}, W)$, avec $|V| = n$. Soit

$$y = (y_1, y_2, ..., y_k)$$

la sortie de la projection spectrale.On pose la fonction de coût

$$C(y) = \frac{1}{2} \sum_{i,j} (y_i - y_j)^2 W_{ij}.$$

On remarque donc qu'il y a une grosse pénalité lorsque W_{ij} est gros, c'est-à-dire que les points initiaux sont proches et que $(y_i - y_j)^2$ est également gros, ce qui correspond au fait que les projections

correspondantes sont éloignées et donc que la transformation n'a pas préservé le voisinage.

On considère maintenant $y^T \Delta y = y^T (D - W) y$. D'une part, on a

$$D = \text{diag}\left(\sum_j W_{1j}, .., W_{nj} \right)$$

et donc

$$Dy = \text{diag}\left(\sum_j y_1 W_{1j}, .., y_n W_{nj} \right).$$

Ainsi

$$y^T D y = \sum_{i,j} y_i^2 W_{ij} = \frac{1}{2} \sum_{i,j} y_i^2 W_{ij} + \frac{1}{2} \sum_{i,j} y_j^2 W_{ij}.$$

D'autre part,

$$Wy = \begin{pmatrix} -w_1- \\ -w_2- \\ \cdots \\ -w_n- \end{pmatrix} y = \begin{pmatrix} \sum_j W_{1j} y_j \\ \sum_j W_{2j} y_j \\ \cdots \\ \sum_j W_{nj} y_j \end{pmatrix},$$

de sorte que

$$y^T W y = \sum_{i,j} y_i y_j W_{ij} = \frac{1}{2} \sum 2 y_i y_j W_{ij}.$$

Donc

$$\mathscr{R}_\Delta(y) = y^T \Delta y = C(y).$$

Ainsi,

$$\arg\min_{||u||=1} C(y) = \varphi_1,$$

soit le premier vecteur propre non-nul de Δ.

Rédigé par Guillaume Roy-Fortin,
2021-2025,
Département des enseignements généraux,
École de technologie supérieure.