

Chapitre 4

Classification et apprentissage

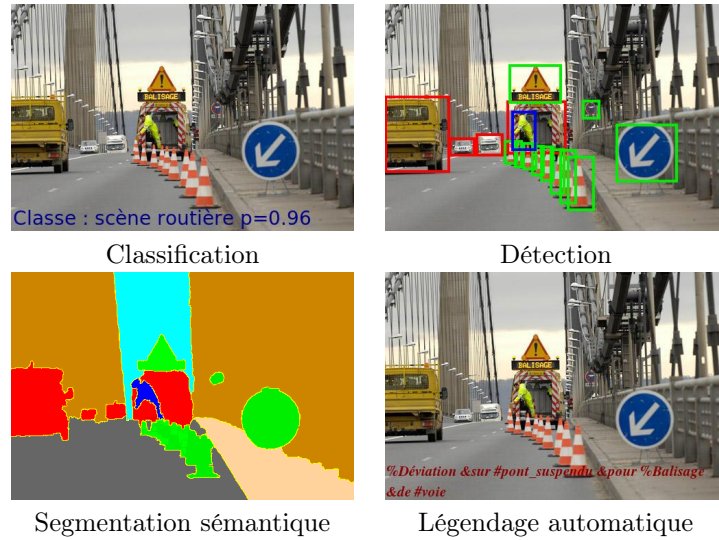


FIGURE 4.1 – Quelques tâches typiques en vision par ordinateur

4.1 Introduction

Dans les chapitres précédents, nous avons vu comment améliorer, simplifier le contenu d'une image et en extraire des caractéristiques locales pertinentes. Mais dans les différentes tâches de la vision par ordinateur (voir Fig. 4.1), la *décision* ne peut pas reposer sur une approche purement impérative, car la distance entre la représentation numérique d'une image et son interprétation est beaucoup trop grande (le fameux *fossé sémantique*). C'est pourquoi on va chercher à construire des modèles fondés sur une connaissance préalable, acquise par l'expérience (l'*entraînement*) et introduite dans la mémoire du système *avant* (apprentissage hors-ligne) et/ou *pendant* (apprentissage en ligne) sa mise en fonction. L'entraînement est réalisée à partir de données *exemplaires* pour lesquelles la décision attendue peut être fournie (apprentissage *supervisé*) ou absente (apprentissage *non supervisé*).

L'apprentissage peut être modélisé comme la construction d'une fonction \mathbf{F} d'un espace d'observation \mathcal{X} , contenant les données $\{X_i\}$ extraites des images, vers un espace de décision \mathcal{Y} , dans lequel on notera Y_i la décision attribuée à X_i . Lorsque l'espace de décision est discret et dépourvu de métrique, on parle de *Classification*. S'il est continu et structuré, on parle alors plutôt de *Régression*.

Les méthodes d'apprentissage prennent en général la forme de l'optimisation d'une fonction de coût \mathcal{L} calculée sur un ensemble d'entraînement \mathcal{T} :

- Si les décisions attendues $\{V_i\}$ sont connues (apprentissage supervisé) : $\arg \min_{\mathbf{F}} \sum_{i \in \mathcal{T}} \mathcal{L}(\mathbf{F}(X_i), V_i)$
- Sinon (apprentissage non supervisé) : $\arg \min_{\mathbf{F}} \sum_{i \in \mathcal{T}} \mathcal{L}(\mathbf{F}(X_i))$

4.2 Réduction de dimension et regroupement

Une préoccupation constante dans l'apprentissage à partir d'images, est de réduire la dimension des données d'observations X_i afin de (1) Réduire la complexité de nos modèles et algorithmes, et (2) Limiter la redondance entre les données. Dans cette section nous allons considérer deux approches classiques très utilisées en reconnaissance d'images :

- **Analyse en Composantes Principales** : Approche algébrique de *réduction de la dimension* aux directions de *variance maximale*.
- **Regroupement** (*Clustering*) : Approche métrique de *quantification du nombre de caractéristiques* en les limitant à un petit nombre de représentants.

4.2.1 Analyse en Composantes Principales

L'analyse en composantes principales (ACP) est une technique statistique ancienne [17] qui vise à réduire la dimension de vecteurs de \mathbb{R}^d en passant de la base canonique à une autre base, dite des composantes principales, dans laquelle on pourrait réduire le nombre de dimensions à un petit nombre $k \ll d$, en limitant la perte d'information.

On considère un ensemble de n vecteurs d'observation $\{X_i\}$ de \mathbb{R}^d dans la base canonique (Fig. 4.2(a)). L'ACP peut s'exprimer sous la forme itérative suivante :

On cherche un repère monodimensionnel (A, \mathbf{u}) , où A est un point de \mathbb{R}^d et \mathbf{u} un vecteur unitaire de \mathbb{R}^d , tel que l'expression des vecteurs $\{X_i\}$ sous forme scalaire par leurs coordonnées dans (A, \mathbf{u}) minimise l'erreur de reconstruction :

$$(\hat{A}, \hat{\mathbf{u}}) = \arg \min_{(A, \mathbf{u})} \sum_{i=1}^n \varepsilon_i^2 \quad (4.1)$$

$$\text{avec } \varepsilon_i^2 = \|X_i - (A + u_i \mathbf{u})\|^2 \quad (4.2)$$

$$= \|X_i - (A + \mathbf{u}^t X_i \mathbf{u})\|^2 \quad (4.3)$$

$$= \|(X_i - A) - \mathbf{u}^t X_i \mathbf{u}\|^2 \quad (4.4)$$

Voir Fig. 4.2(b). Or d'après Pythagore (voir Fig. 4.2(c)) :

$$\|X_i - A\|^2 = \varepsilon_i^2 + \|\mathbf{u}^t X_i \mathbf{u}\|^2 \quad (4.5)$$

$$\text{Et donc } \varepsilon_i^2 = \|X_i - A\|^2 - \|\mathbf{u}^t X_i \mathbf{u}\|^2 \quad (4.6)$$

$$= \|X_i - A\|^2 - (\mathbf{u}^t X_i)^2 \|\mathbf{u}\|^2 \quad (4.7)$$

$$= \|X_i - A\|^2 - (\mathbf{u}^t X_i)^2 \quad (4.8)$$

$$\text{Ainsi, } \min_{(A, \mathbf{u})} \sum_i \varepsilon_i^2 = \min_{(A, \mathbf{u})} \sum_i [\|X_i - A\|^2 - (\mathbf{u}^t X_i)^2] \quad (4.9)$$

$$= \min_A \sum_i \|X_i - A\|^2 - \max_{\mathbf{u}} \sum_i (\mathbf{u}^t X_i)^2 \quad (4.10)$$

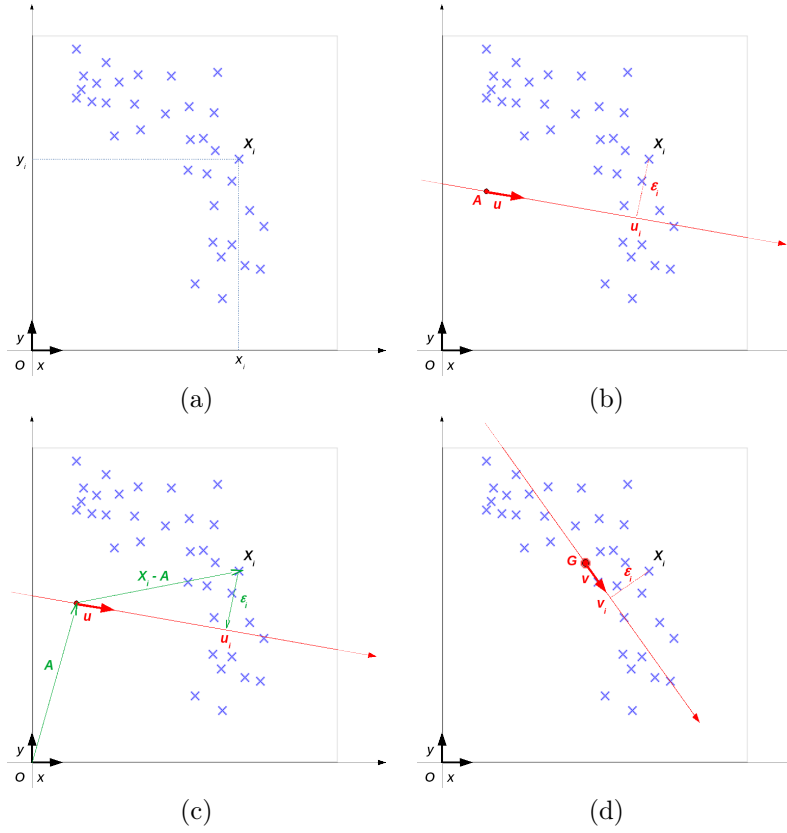


FIGURE 4.2 – ACP : $G = \arg \min_A \sum_i \|X_i - A\|^2$ et $\mathbf{v} = \arg \max_{\mathbf{u}} \sum_i (\mathbf{u}^t X_i)^2$.

L'origine du repère optimal, $\arg \min_A \sum_i \|X_i - A\|^2$, est donc le point qui minimise la distance moyenne aux $\{X_i\}$, c'est-à-dire leur centre de gravité $G = \frac{1}{n} \sum_i X_i$. Et le vecteur de coordonnée optimale $\arg \max_{\mathbf{u}} \sum_i (\mathbf{u}^t X_i)^2$ est celui qui maximise la variance des coordonnées, c'est donc le vecteur \mathbf{v} associé à la plus grande valeur propre de la matrice de covariance des $\{X_i\}$ (voir Fig. 4.2(d)).

On itère ensuite le procédé en cherchant dans \mathbf{v}^\perp , les $d-1$ vecteurs unitaires orthogonaux à \mathbf{v} , celui qui minimise l'erreur résiduelle :

$$\mathbf{w} = \arg \min_{\mathbf{u} \in \mathbf{v}^\perp} \sum_{i=1}^n \|X_i - (G + v_i \mathbf{v} + u_i \mathbf{u})\|^2 \quad (4.11)$$

On trouve ainsi que \mathbf{w} est le vecteur associé à la deuxième plus grande valeur propre de la matrice de covariance des $\{X_i\}$. On continue avec $(\mathbf{v}, \mathbf{w})^\perp$, et ainsi de suite.

On peut donc décrire l'algorithme de calcul de l'ACP comme suit : Notons \mathbf{X} la matrice $(d \times n)$ composée de l'ensemble des n vecteurs d'observation $\{X_i\}$

rangés en colonnes :

$$\mathbf{X} = \begin{pmatrix} X_1^1 & X_2^1 & \dots & X_n^1 \\ \vdots & \vdots & \ddots & \vdots \\ X_1^d & X_2^d & \dots & X_n^d \end{pmatrix}, \text{ et } G = \frac{1}{n} \sum_{i=1}^n X_i \quad (4.12)$$

La matrice ($d \times d$) de covariance des observations s'écrit :

$$\mathbf{C} = \frac{1}{n} \mathbf{X}\mathbf{X}^t - GG^t \quad (4.13)$$

ACP : Algorithme

- Calcul de la matrice de covariance $\mathbf{C} = \frac{1}{n} \mathbf{X}\mathbf{X}^t - GG^t$
- Calcul et tri des valeurs propres $(\lambda_1, \dots, \lambda_d)$, telles que $\lambda_1 > \dots > \lambda_d$, et de leurs vecteurs propres associés $(\mathbf{u}_1, \dots, \mathbf{u}_d)$.
- Les k ($k \ll d$) premiers vecteurs propres $\{\mathbf{u}_1, \dots, \mathbf{u}_k\}$ sont les *composantes principales*.

Les composantes principales fournissent donc les dimensions théoriques de l'espace d'observation les mieux adaptées à la répartition des données dans \mathbb{R}^d . La matrice de corrélation calculée dans l'espace des composantes principales étant par construction diagonale, ces composantes sont décorrélées (mais pas indépendantes!). Il est tentant de chercher à expliquer les différentes composantes principales selon telle ou telle caractéristique des données. Une telle "explicabilité" est néanmoins en général hasardeuse. Voyons maintenant deux exemples particulièrement éclairants en analyse d'images.

Un premier exemple parlant est la méthode proposée par Turk et Pentland pour la reconnaissance de visages [46]. L'application pour la reconnaissance est hors de propos ici ; il s'agit seulement d'illustrer le cas où les vecteurs d'observations sont des imageries 32×32 de visages. Voir Fig ; 4.3(a) : les vecteurs $\{X_i\}$ sont donc de dimension 1024, et la matrice de corrélation de taille 1024×1024 .

La figure 4.3(b) montre le visage "moyen" G qui est la moyenne des $\{X_i\}$, et le repère 3d montre les 3 axes principaux correspondant aux vecteurs propres $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ de la matrice de covariance \mathbf{C} associées aux 3 plus grandes valeurs propres $(\lambda_1, \lambda_2, \lambda_3)$. En examinant, sur chacun des 3 axes, les visages "extrêmes" reconstruits à partir de G le long de cet axe (typiquement $G \pm \sqrt{\lambda_i} \mathbf{u}_i$), on peut interpréter chaque dimension comme l'expression d'un caractère particulier du visage présentant une variation importante (et donc un fort potentiel de discrimination pour un algorithme de classification).

Un autre exemple important est lié au travaux d'Hyvriinen *et al* [20] sur les statistiques d'images naturelles. Dans cette étude, les auteurs ont extraits aléatoirement une grande quantité d'imageries 32×32 d'une collection de photographies noir et blanc prises dans des environnements naturels. La figure 4.4(a) montre les 256 premières composantes principales correspondantes. On peut faire les remarques suivantes :

- L'expérience est très largement répétable, au sens que les 100 premières composantes varient très peu d'un tirage aléatoire à un autre.
- On observe une inflexion sur le graphe montrant la variation du logarithme de la variance en fonction de la composante (i.e. $\log(\lambda_i)$ en fonc-



FIGURE 4.3 – Exemple d'ACP calculée sur des imagerie 32×32 de visages pour une application de reconnaissance de visages [46]. (a) Quelques échantillons des observations $\{X_i\}$ utilisées pour calculer la matrice de covariance. (b) Le visage moyen G (au centre), les 3 composantes principales ($\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$) (en haut à gauche), et des visages reconstruits "extrêmes" sur chacun des 3 axes principaux.

tion de i , voir figure 4.4(b)), au niveau de $i = 100$, ce qui corrobore la remarque précédente.

- On peut noter une grande ressemblance entre les premières composantes et les premières dérivées d'une gaussienne 2d (voir chapitre "Caractéristiques multi-échelles"), ce qui fournit une première justification statistique de l'utilisation des dérivées partielles en tant que caractéristiques *ad hoc*. Nous en rencontrerons d'autres par la suite.

4.2.2 Regroupement par K-moyennes

Les algorithmes de regroupement (*clustering*) visent aussi à réduire le volume de l'espace d'observation \mathcal{X} , mais au lieu de réduire la dimension des vecteurs comme l'ACP, ils réduisent le *nombre de vecteurs* à un petit nombre de représentants (dits *prototypes*), chaque prototype étant associé à un groupe (cluster).

Ainsi, l'algorithme des K-moyennes (K-means) [29] réalise une partition \mathcal{C} d'un ensemble de n observations $\{X_i\} \subset \mathbb{R}^d$ en K classes C_k ($1 \leq k \leq K$).

Notons $\Pi = \{\pi_k\}$ un ensemble de K prototypes choisis dans l'espace d'observation et représentant chacun une classe. Soit δ une distance dans \mathbb{R}^d . L'objectif est de minimiser la fonction de coût suivante :

$$J_{\mathcal{C}}^{\Pi} = \sum_{k=1}^K \sum_{X_i \in C_k} \delta(X_i, \pi_k)^2 \quad (4.14)$$

Cela revient donc à minimiser la moyenne (par cluster) des distances moyennes entre un prototype et les éléments du cluster qu'il représente. On ne connaît pas d'algorithme permettant de résoudre cette minimisation en temps polynomial, mais on connaît une solution calculable en temps multilinéaire (respectivement $\mathcal{O}(ndK)$ et $\mathcal{O}(nd)$) dans les deux cas suivants :

- Si l'ensemble des prototypes Π est fixé ; dans ce cas :

$$C_k = \{X_i; \forall j \neq k, \delta(X_i, \pi_k) \leq \delta(X_i, \pi_j)\} \quad (4.15)$$

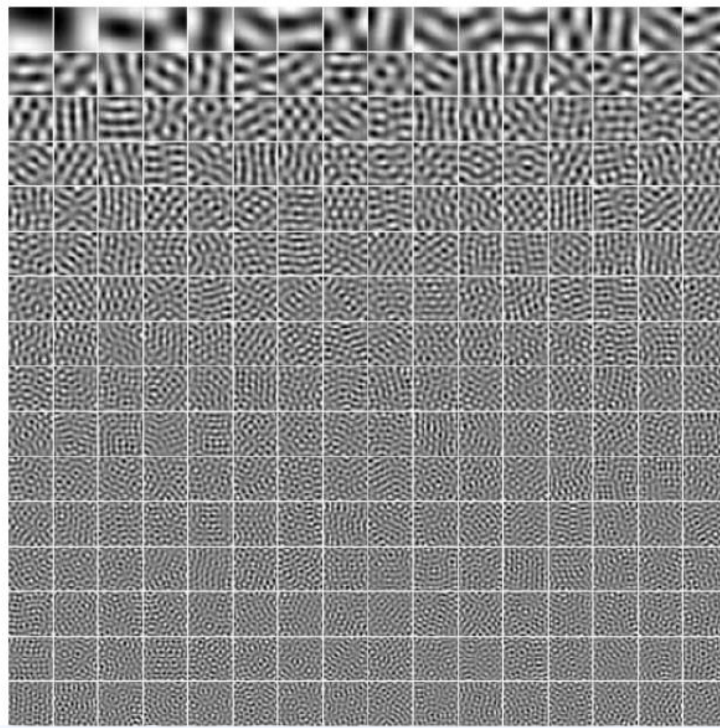
- Si la partition en classes \mathcal{C} est fixée ; dans ce cas :

$$\pi_k = \frac{1}{|C_k|} \sum_{X_i \in C_k} X_i \quad (4.16)$$

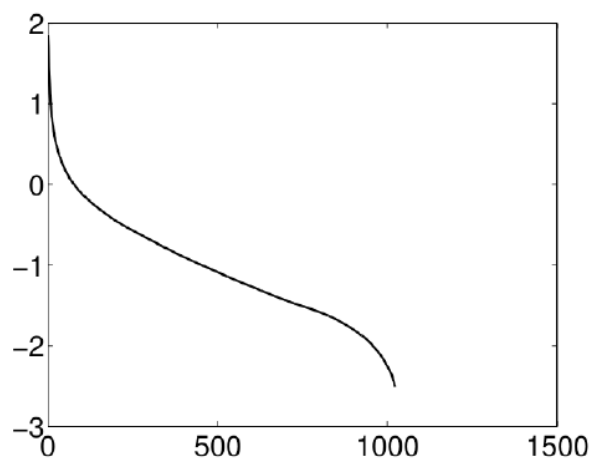
L'algorithme des K-moyennes est donc un algorithme itératif qui converge vers un minimum local de $J_{\mathcal{C}}^{\Pi}$, en ajustant alternativement \mathcal{C} et Π :

K-means : Algorithme

- **Initialisation** : On choisit un ensemble de K prototypes $\Pi = \{\pi_k\}$.
- **Répéter** jusqu'à convergence :
 1. Mise à jour de \mathcal{C} : $C_k = \{X_i; \forall j \neq k, \delta(X_i, \pi_k) \leq \delta(X_i, \pi_j)\}$
 2. Mise à jour de Π : $\pi_k = \frac{1}{|C_k|} \sum_{X_i \in C_k} X_i$



(a)



(b)

FIGURE 4.4 – Extrait de [20] : (a) les 256 premières composantes principales calculées sur des imagerie 32×32 extraites d'images naturelles. (b) Graphe montrant la variation de la log-variance de la première à la dernière composante principale : $f : i \mapsto \log(\lambda_i)$

- **Complexité** : Si l'algorithme met s itérations pour converger (ou si s est fixé comme nombre d'itérations maximum, la complexité totale du K-means est donc de $\mathcal{O}(ndKs)$.
- **Initialisation** : Comme tous les algorithmes itératifs, le K-means est extrêmement sensible au choix des K prototypes initiaux. Les solutions basiques (K tirages aléatoires dans \mathbb{R}^d ou parmi les $\{X_i\}$, affectation aléatoire d'un parmi les K clusters à tous les $\{X_i\}, \dots$) sont couramment utilisées mais il existe de nombreuses autres techniques et heuristiques [7] dont les efficacités restent très dépendantes de l'application visée.
- **Choix de K** : La nécessité de fixer K comme paramètre d'entrée du K-means est une contrainte qui peut s'avérer rédhibitoire pour certaines applications (typiquement, petit nombre de classes inconnu à l'avance). Il existe là aussi des techniques permettant de déterminer un nombre optimal de clusters, qui consistent à répéter l'algorithme en faisant varier K et en choisissant celui qui maximise la qualité de la séparation, typiquement mesurée par combinaisons de distances intra-clusters, inter-prototypes, etc., voir [41]. Noter que dans ce cas la complexité devient $\mathcal{O}(ndK_{\max}^2s)$, où K_{\max} est le plus grand nombre de clusters testé.

L'algorithme K-Means possède de nombreuses applications en vision par ordinateur, par exemple pour créer un dictionnaire de caractéristiques afin de catégoriser des environnements visuels par la fréquence relative des différentes caractéristiques [8]. En analyse d'images, il est utile en tant qu'outil de quantification vectorielle pour réduire le nombre de valeurs différentes d'une population de vecteurs en tenant compte de leur répartition, par exemple pour quantifier les couleurs (voir Fig. 4.5).

4.3 Apprentissage bayésien

4.3.1 Généralités

L'inférence bayésienne est un outil de base utile dans de nombreux domaines et qui permet de fournir des méthodes simples et bien fondées pour la classification et la segmentation supervisée d'images. Elle consiste à estimer des probabilités d'événements hypothétiques (fondant un modèle de classification) à partir de statistiques sur des événements observés (fournis par la supervision).

Les données d'observation X , ainsi que les classes c sont donc modélisées comme des vecteurs (resp. variables) aléatoires. On s'intéresse aux probabilités suivantes :

- $P(X)$ et $P(c)$ les probabilités *a priori*, respectivement d'observer le vecteur X , ou de rencontrer un vecteur d'observation de classe c .
- $P(X, c)$ la probabilité *jointe*, d'observer la donnée X de classe c .
- $P(X/c)$ la *vraisemblance* conditionnelle de la donnée X , c'est-à-dire la probabilité d'observer X sachant qu'on a affaire à un vecteur de classe c .
- $P(c/X)$ la probabilité *a posteriori* de la classe c , sachant qu'on a observé le vecteur X .

L'*apprentissage bayésien* consiste à construire de façon empirique les lois de ces probabilités à partir de statistiques réalisées sur un ensemble de données d'apprentissage étiquetées $\{X_i, c(X_i)\}$.

A partir de ce modèle, l'*inférence bayésienne* consiste à associer à une donnée



FIGURE 4.5 – Quantification des couleurs par K-Means (à droite), comparée à une quantification uniforme des composantes (R, G, B) (à gauche) pour le même nombre de couleurs.

X inconnue sa classe c^* . Elle est classiquement réalisée sous l'une des deux formes suivantes :

- le critère du **Maximum A Posteriori** (MAP) :

$$c_{\text{MAP}}^*(X) = \arg \max_c P(c/X) \quad (4.17)$$

- le critère du **Maximum de vraisemblance** (ML) :

$$c_{\text{ML}}^*(X) = \arg \max_c P(X/c) \quad (4.18)$$

Bien que ces deux critères semblent sémantiquement très différents de par leurs définitions mentionnées plus haut, ils sont en fait liés via le *Théorème de Bayes*. En effet, la construction du modèle consiste à estimer la loi jointe $P(X, c)$ à partir des n données d'apprentissage annotées $\{X_i, \mathbf{c}(X_i)\}$:

En effet, on a $P(X, c) = P(c).P(X/c)$, avec :

- $P(c)$ la probabilité d'occurrence de la classe c , qui est parfois fixée arbitrairement, et parfois estimée par la fréquence de la classe dans les données d'apprentissage : $P(c) \simeq \frac{1}{n} |\{X_i; \mathbf{c}(X_i) = c\}|$
- $P(X/c)$ est estimée sous la forme d'une distribution empirique des données associées à la classe c : $P(X/c) \simeq H_c(X)$

où $|S|$ représente le cardinal de l'ensemble S et $H_c(X)$ un histogramme multidimensionnel normalisé des vecteurs $\{X_i; \mathbf{c}(X_i) = c\}$

Or, on a également $P(X, c) = P(X).P(c/X)$, d'où le Théorème de Bayes :

$$P(c/X) = \frac{P(X/c).P(c)}{P(X)} \quad (4.19)$$

Et comme le dénominateur ne dépend pas de c , le critère du MAP devient :

$$c_{\text{MAP}}^*(X) = \arg \max_c P(X/c).P(c) \quad (4.20)$$

$$= \arg \max_c (\log P(X/c) + \log P(c)) \quad (4.21)$$

Finalement seule la pondération par la loi *a priori* distingue le critère du MAP de celui du ML.

4.3.2 Modèles gaussiens

L'expression générale de l'estimation de la loi $P(X/c)$ par un histogramme multidimensionnel $H_c(X)$ pose en pratique de gros problèmes combinatoires lorsque les vecteurs $X \in \mathbb{R}^d$ sont de grande dimension, et sur le choix de la quantification à adopter pour chaque composante, particulièrement lorsque les distributions sont fortement non uniformes.

Pour cette raison, on utilise souvent un modèle probabiliste simplifié de la distribution de $P(X/c)$, comme la gaussienne multivariée, qui permet de réduire le nombre de paramètres estimés à $\frac{d(d+3)}{2}$ par classe :

$$P(X/c) = f_c(X) = \frac{1}{\sqrt{(2\pi)^d \det \mathbf{C}_c}} \exp \left(-\frac{1}{2} (X - G_c)^t \mathbf{C}_c^{-1} (X - G_c) \right) \quad (4.22)$$

où $G_c \in \mathbb{R}^d$ (d paramètres) et $\mathbf{C}_c \in \mathbb{R}^{d \times d}$ ($\frac{d(d+1)}{2}$ paramètres) sont respectivement le vecteur moyen et la matrice de covariance des données d'apprentissage étiquetées à la classe c .

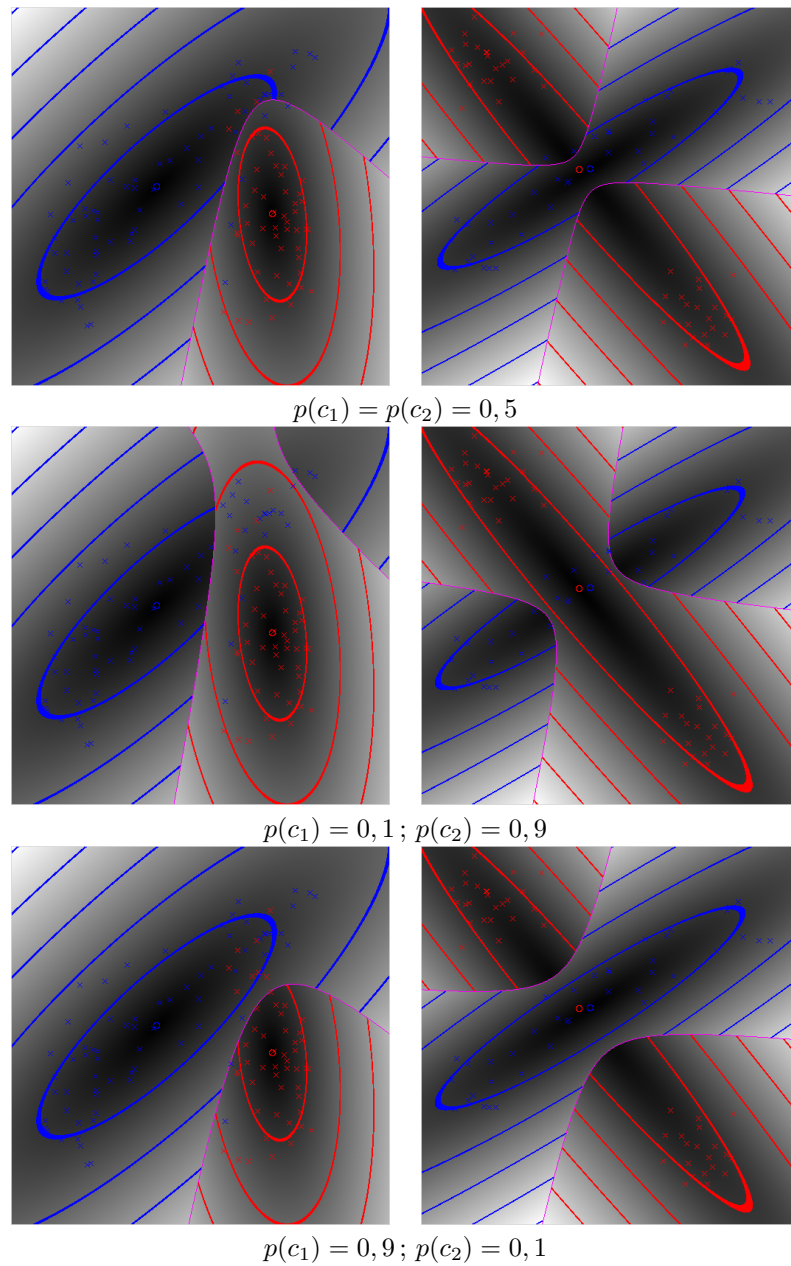


FIGURE 4.6 – Partition de l'espace 2d en 2 classes fondée le critère bayésien du Maximum a Posteriori, pour un modèle gaussien entraîné par 2 nuages de points associées à la classe c_1 (en bleu) et c_2 (en rouge). Chaque colonne représente un cas différent. La partition est rendue visible par la couleur des lignes de niveau et par la frontière en magenta. Chaque ligne correspond à des valeurs différentes d'a priori de classes. Haut : classes équiprobables (équivalent au critère du Maximum de Vraisemblance). Milieu : classe c_1 moins probable. Bas : classe c_1 plus probable.

La figure 4.6 montre deux exemples de classification bayésienne binaire de l'espace 2d en utilisant un modèle gaussien entraîné sur 2 populations : les points bleus, associés à la classe c_1 , et les points rouges, associés à la classe c_2 . Les paramètres gaussiens de chaque nuage apparaissent sous la forme de (1) le centre de gravité, représenté par le rond bleu ou rouge, et (2) la matrice de covariance, représentée par les lignes de niveaux bleues ou rouges de la carte de distance de Mahalanobis, définie par :

$$\mathcal{M}_{(G_c, \mathbf{C}_c)}(X) = \sqrt{(X - G_c)^t \mathbf{C}_c^{-1} (X - G_c)} \quad (4.23)$$

La couleur des lignes de niveaux détermine donc la classification au sens du Maximum a Posteriori, de chaque point de l'espace selon la couleur de la région qu'il occupe. On peut voir sur la figure que la classification fondée sur le modèle gaussien peut séparer convenablement les deux nuages de points, même si leur distribution n'est pas gaussienne, mais aussi que la séparation dépend beaucoup de la probabilité a priori des classes, ce qui souligne les limites du critère de Maximum de Vraisemblance.

Les modèles gaussiens multivariés présentent l'avantage de pouvoir estimer les distributions $P(X/c)$ efficacement et de coder les modèles avec un petit nombre de paramètres : $\frac{pd(d+3)}{2}$, où p est le nombre de classes et d la dimension de l'espace d'observation. Néanmoins ils souffrent d'une limitation majeure : ils font l'hypothèse d'une distribution gaussienne unimodale des observations de chaque classe, ce qui est rarement le cas en pratique.

4.3.3 Modèles naïfs

Les modèles naïfs permettent de contourner cette limitation, mais au pris d'une hypothèse tout aussi forte : celle de l'indépendance des différentes composantes de X conditionnellement à la classe c :

$$P(X/c) = P(x_1, \dots, x_d/c) = \prod_{j=1}^d P(x_j/c) \quad (4.24)$$

ce qui conduit au critère du MAP suivant :

$$c_{\text{MAP}}^*(X) = \arg \max_c \left(\log P(c) + \sum_{j=1}^d \log P(x_j/c) \right) \quad (4.25)$$

En pratique les modèles bayésiens naïfs peuvent souvent fonctionner mieux qu'un modèle multivarié unimodal, dans la mesure où ils permettent d'estimer des distributions certes scalaires, mais potentiellement aussi complexes qu'on veut.

En particulier, alors qu'il était pratiquement exclu d'estimer les lois de $P(X/c)$ par des histogrammes multidimensionnels $H_c(X)$, il devient beaucoup plus aisé d'estimer individuellement les distributions scalaires $P(x_j/c) \simeq H_c(x_j)$, aboutissant à une taille de modèle de pdq où q est le nombre de valeurs (bins) quantifiées pour les histogrammes (contre une taille de pd^d pour le modèle général multivarié).

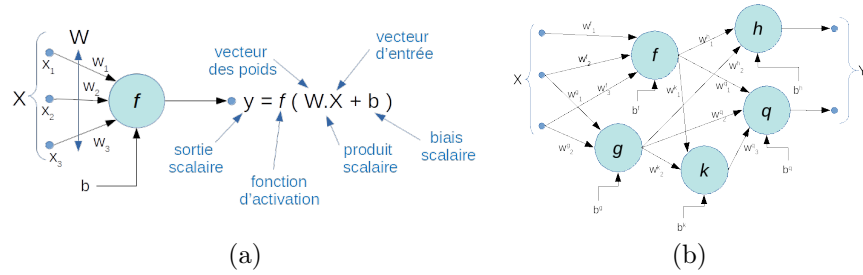


FIGURE 4.7 – (a) Le neurone formel. (b) Un réseau de neurones formels.

Certains modèles vont encore plus loin dans la réduction en considérant des modèles bayésiens naïfs gaussiens, où chaque composante est supposée suivre une distribution gaussienne :

$$P(x_j/c) = f_c(x_j) = \frac{1}{\sigma_j \sqrt{2\pi}} \exp\left(-\frac{1}{2}(x_j - \mu_j)^2\right) \quad (4.26)$$

où μ_j et σ_j sont la moyenne et l'écart-type des composantes x_j pour les données d'observation associées à la classe c , ce qui réduit encore la taille du modèle à $2pd$.

4.4 Autres techniques supervisées

Coming soon...

4.4.1 k-plus-proches-voisins

4.4.2 Séparateur à Vaste Marge

4.4.3 Forêt aléatoire

4.5 Réseaux de neurones

L'origine des réseaux de neurones artificiels est très ancienne ; elle remonte - au moins - au premier modèle de neurone formel proposé par Mc Culloch et Pitts [33] dans les années 40. Voir Figure 4.7(a). Un réseau de neurones formels est alors simplement un graphe orienté dont les nœuds sont formés par les neurones (auxquels on associe une fonction scalaire dite d'activation) et les arêtes par les axones (connexions entrantes ou sortantes des neurones nantis d'un poids). Un tel réseau forme ainsi une fonction dont l'entrée X est le vecteur formé par les sources du graphe (nœuds ayant un degré d'entrée nul) et la sortie Y le vecteur formé par les puits du graphe (nœuds ayant un degré de sortie nul). Voir Figure 4.7(b).

Les réseaux de neurones artificiels ont depuis lors constitué une approche très attrayante pour l'analyse d'images de par leur extrême flexibilité, la sortie Y pouvant représenter aussi bien une décision scalaire (classification) que la régression d'un ensemble arbitraire de paramètres (détection, suivi, estimation

de pose ou de déplacement), ou d'une image elle-même (traitement d'images!). L'entrée X quant à elle peut représenter aussi bien des caractéristiques extraites d'une image, que l'image elle-même (analyse de bout-en-bout, ou end-to-end).

Néanmoins pour exploiter ce potentiel, il fallait pouvoir adapter automatiquement les paramètres d'un réseau (le poids de ses connexions) aux problèmes et aux données visés, ce qui est l'objet de la procédure d'entraînement. Or ce n'est que dans les années 80 par la diffusion de l'algorithme de rétro-propagation du gradient [39], qu'une telle procédure a été rendue possible sur des réseaux suffisamment complexes pour être utilisables sur des images. Et ce n'est qu'au début des années 2010 que les réseaux de neurones dits profonds ont véritablement commencé à déferler dans le monde de l'analyse d'images [24].

4.5.1 Généralités

Ainsi dans un réseau de neurones tel que celui de la figure 4.7(b), l'*architecture* définie par le graphe et les fonctions d'activation est statique et fixée *a priori*, tandis que les *paramètres* définies par l'ensemble des poids W et des biais b sont adaptables et modélés par l'apprentissage, ou entraînement.

Dans le cadre d'un apprentissage supervisé, le réseau de neurones est entraîné à partir d'un ensemble d'apprentissage $\{X_i, V_i\}$ où les X_i sont les données d'entraînement et les V_i les sorties attendues (supervision). On considère alors une fonction de coût (loss) $\mathcal{L}(Y, V) \geq 0$, qui mesure la différence (erreur) entre une sortie calculée Y et une sortie attendue V . L'objectif de l'entraînement est de minimiser l'erreur globale sur l'ensemble d'apprentissage :

$$\min_{\theta} \sum_i \mathcal{L}(\mathcal{O}_{\theta}(X_i), V_i),$$

où $\mathcal{O}_{\theta}(X)$ est la sortie calculée par le réseau de paramètres θ sur l'entrée X .

La procédure d'entraînement est réalisée par une succession de passes avant (forward) et de passes arrière (backward) sur les données d'apprentissage.

- Dans une passe avant, une donnée X est soumise au réseau et on compare la sortie produite Y à la sortie attendue V en utilisant la fonction d'erreur $\mathcal{L}(Y, V)$.
- Dans une passe arrière, l'erreur commise $\mathcal{L}(Y, V)$ est rétro-propagée à l'ensemble des neurones, et les poids des connexions sont ajustés en fonction de leur contribution à l'erreur commise :

$$w_{ij} \leftarrow w_{ij} - \varepsilon \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

où ε est le taux d'apprentissage (learning rate).

La passe avant consiste donc simplement à appliquer, comme on le ferait en inférence, la fonction correspondant à l'état courant du réseau, et à calculer l'erreur. Dans l'algorithme ci-dessous, on suppose pour simplifier l'écriture que toutes les fonctions d'activation sont égales à g :

Propagation avant $Y \leftarrow \text{ForwardProp}(W, X)$

- Pour tous les neurones i de la couche d'entrée $s_i = x_i$.
- Pour toutes les couches l suivantes :
 - Pour tous les neurones j de la couche l :

$$s_j = g \left(\sum_k w_{kj} s_k + b_j \right).$$

- Pour tous les neurones j de la couche de sortie $y_j = s_j$.

Pour la rétro-propagation de l'erreur, on notera que la contribution à l'erreur des poids de connexion $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ n'est calculable directement que pour les connexions de la couche de sortie. Pour calculer les neurones des couches précédentes, il faut faire appel aux règles de dérivation des fonctions composées pour calculer récursivement leurs contributions à l'erreur de prédiction. Précisons d'abord nos notations / conventions :

- w_{ij} est le poids de la connexion du neurone i vers le neurone j .
- s_i est la valeur de sortie du neurone i .
- On supposera pour simplifier toutes les fonctions d'activation égales à g .
- $a_j = \sum_i w_{ij} s_i + b_j$ est la valeur d'activation du neurone j . Sa sortie est donc $s_j = g(a_j)$.

L'équation de mise à jour s'écrit donc :

$$w_{ij} \leftarrow w_{ij} - \varepsilon \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

En introduisant a_j , la valeur d'activation du neurone j , on a :

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_j} \times \frac{\partial a_j}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_j} \times s_i$$

Puisque $a_j = \sum_i w_{ij} s_i + b_j$. D'autre part on a :

$$\frac{\partial \mathcal{L}}{\partial a_j} = \frac{\partial \mathcal{L}}{\partial s_j} \times \frac{\partial s_j}{\partial a_j} = \frac{\partial \mathcal{L}}{\partial s_j} \times g'(a_j)$$

Puisque $s_j = g(a_j)$.

Enfin, en développant l'impact de la sortie du neurone j sur toutes les entrées k de la couche qui suit celle de j :

$$\frac{\partial \mathcal{L}}{\partial s_j} = \sum_k \frac{\partial \mathcal{L}}{\partial a_k} \frac{\partial a_k}{\partial s_j} = \sum_k \frac{\partial \mathcal{L}}{\partial a_k} w_{jk}$$

Puisque $a_k = \sum_j w_{jk} s_j + b_k$. En conclusion, il suffit de savoir calculer le gradient de l'erreur sur la dernière couche pour le calculer sur toutes les couches précédentes par récursivité. Or pour la dernière couche le calcul est immédiat.

Supposons pour simplifier une fonction d'erreur quadratique :

$$\mathcal{L}(Y, V) = \frac{1}{2} \|Y - V\|^2.$$

Pour la dernière couche on a $s_j = y_j$ et donc : $\mathcal{L} = \frac{1}{2} \sum_j (s_j - v_j)^2$, on a donc :

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial a_j} = \frac{\partial \mathcal{L}}{\partial s_j} \frac{\partial s_j}{\partial a_j} = (s_j - v_j) \times g'(a_j) \\ \frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = (s_j - v_j) \times g'(a_j) \times s_i \end{cases}$$

Il faut également mettre à jour les biais :

$$b_j \leftarrow b_j - \varepsilon \frac{\partial \mathcal{L}}{\partial b_j}$$

Il suffit alors de remarquer que :

$$\frac{\partial \mathcal{L}}{\partial b_j} = \frac{\partial \mathcal{L}}{\partial a_j} \times \frac{\partial a_j}{\partial b_j} = \frac{\partial \mathcal{L}}{\partial a_j} \text{ car } a_j = \sum_i w_{ij} s_i + b_j$$

Donc sur la dernière couche :

$$\frac{\partial \mathcal{L}}{\partial b_j} = (s_j - v_j) \times g'(a_j)$$

Finalement, l'algorithme de la passe arrière s'écrit (Pour une fonction d'erreur \mathcal{L} quadratique) :

Rétropropagation du gradient $W \leftarrow \text{BackProp}(W, Y, V)$

- Pour tous les neurones j de la couche de sortie :
 - calculer l'erreur $\Delta_j = (s_j - v_j) \times g'(a_j)$
- Pour toutes les couches l précédentes :
 - Pour tous les neurones i de la couche l :
 - calculer l'erreur $\Delta_i = \left(\sum_k \Delta_k w_{ik} \right) \times g'(a_i)$
- Pour toutes les connexions (i, j) du réseau :
 - mettre à jour le poids $w_{ij} \leftarrow w_{ij} - \varepsilon \times s_i \times \Delta_j$
 - mettre à jour le biais $b_j \leftarrow b_j - \varepsilon \times \Delta_j$

4.5.2 Architectures convolutionnelles

Les images (et par extension les vidéos) sont des données de grande dimension, et qui plus est de tailles arbitraires et pas toujours prédictibles, dépendant de la résolution des images et de la durée typiques des objets et événements d'intérêt. L'application à de telles données d'un réseau de neurones à architecture fixe pose donc des problèmes pratiques et combinatoires majeures. Or, comme on l'a vu dans les chapitres précédents, les algorithmes d'analyse d'images reposent très largement sur des opérations locales et invariantes par translation.

Le réseau de neurones convolutionnel (CNN) est de ce point de vue un cadre très naturel quand on songe à l'application à des données nanties d'une topologie régulière, qu'elle soit 1d (son), 2d (image), 3d ou plus (image 3d, vidéo,...). Dans un CNN, les couches convolutionnelles sont *a priori* indépendantes de la taille des images. En effet, un même neurone est appliqué à toutes les positions

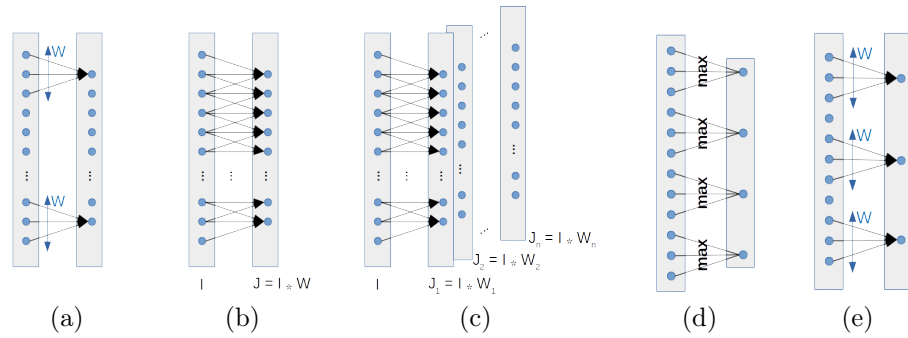


FIGURE 4.8 – (a) CNN : un même neurone est appliqué aux différentes parties de la donnée. (b) il en résulte une opération de convolution. (c) plusieurs neurones forment donc un banc de filtres de convolution. (d) pooling. (e) stride.

(spatiales, temporelles ou de canaux pour une image multi-composantes). Voir Figure 4.8(a). En faisant abstraction du biais et de la fonction d'activation, il en résulte donc l'application du produit scalaire du vecteur de poids du neurone par le voisinage de chaque pixel, de façon invariante par translation, c'est-à-dire une convolution, voir Figure 4.8(b). Par extension, l'ensemble des neurones d'une couche forment un banc de filtres de convolution, voir Figure 4.8(c).

Dans un CNN, on retrouve, au même titre que les approches multi-échelles classiques en analyse d'images, des mécanismes de réduction des données d'une couche à l'autre, qui peuvent prendre la forme d'aggrégation spatiale (local pooling, voir Figure 4.8(d)), ou de stride, qui consiste à appliquer un pas d'échantillonnage supérieur à celui des données dans l'application de la convolution, voir Figure 4.8(e). Attention, dans ce dernier cas, on s'expose à un risque d'aliasing qui ne sera pas sans influence sur la qualité de la représentation (voir Chapitre 1).

Un réseau de neurones *totalemment convolutionnel* possède la bonne propriété de pouvoir s'appliquer à une image de n'importe quelle taille. La taille du vecteur de sortie ne dépend dans ce cas que de la taille de l'entrée. Un tel CNN ne s'applique donc que pour des tâches de régression fournissant une image en sortie. Pour une tâche de classification ou d'estimation de paramètres de tailles fixes, la partie finale, ou décisionnelle, du réseau prend en général la forme d'un ensemble de couches entièrement connectées (i.e. pour chaque neurone d'une couche de sortie, la taille du vecteur de poids coïncide avec celle de la donnée d'entrée). Dans ce cas, l'architecture du réseau doit contenir un mécanisme de passage de la partie convolutionnelle (opérations locales) à la partie entièrement connectée (générant des opérations globales). Ce mécanisme peut prendre des formes diverses :

- *Flattening* : la sortie de la dernière couche convolutionnelle est «aplatie» sous la forme d'un vecteur ; l'entrée de la première couche entièrement connectée est de la taille de la dernière sortie de la partie convolutionnelle.
- *Global pooling, order 1* : pour chaque neurone de la dernière couche convolutionnelle, on calcule une statistique unique (comme la valeur moyenne) de l'ensemble des données spatiales ; la taille de la première couche entièrement connectée est égale au nombre de neurones (ou canaux) de la

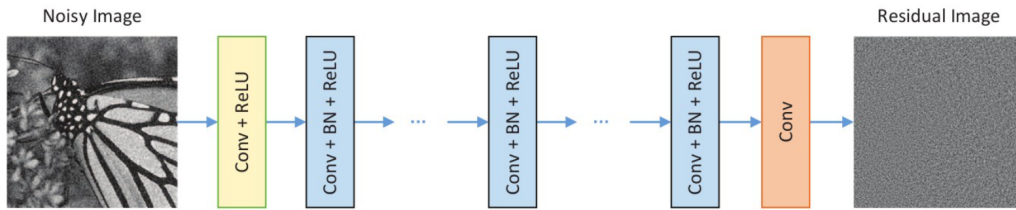


FIGURE 4.9 – Réseau entièrement convolutionnel pour le débruitage généralisé d’images [53].

dernière couche convolutionnelle.

- *Global pooling, order 2* : il s’agit de la matrice de covariance, calculée sur l’ensemble des données spatiales, des valeurs de sortie des neurones de la dernière couche convolutionnelle ; la taille de la donnée produite est donc égale au carré du nombre de neurones (ou canaux) de la dernière couche convolutionnelle.

On peut noter que les deux dernières solutions, celles de global pooling, offrent la possibilité en pratique d’appliquer le réseau à des images de n’importe quelle taille. En effet, contrairement au flattening où la taille de la couche d’entrée dépend de la taille des données, pour le global pooling elle ne dépend que du nombre de canaux de la dernière couche convolutionnelle. Pour autant, il serait faux de penser que la résolution des images d’entrée est indifférente à la représentation qui en est faite dans la dernière couche convolutionnelle, qu’on interprète comme une projection dans un espace de caractéristiques apprises, ou espace latent (embedding).

En effet, la composition des opérateurs de convolution et de local pooling dans les couches successives produisent à la fois une diminution de la résolution des données, et une *augmentation* de l’échelle de représentation locale, la sortie d’un neurone d’une couche profonde dépendant finalement d’un support spatial de l’image d’entrée d’autant plus important que la couche est profonde. Ce support spatial associé à chaque neurone est appelé *champ récepteur*. Par exemple, dans [53], les auteurs proposent un réseau entièrement convolutionnel pour le débruitage généralisé d’images (voir Figure 4.9). Ce réseau comporte 20 couches convolutionnelles, sans spatial pooling (donc pas de réduction de résolution spatiale), et chaque neurone de chaque couche réalise une convolution 3×3 . Par associativité de la convolution, les neurones de la couche numéro k possèdent un champ récepteur de taille $(2k + 1) \times (2k + 1)$. La dernière couche de ce réseau opère donc sur un champ récepteur de taille 41, qui forme la portée spatiale (ou échelle) finale de la représentation, et donc le niveau de contexte (ou de sémantique) que le réseau est capable d’appréhender dans ses données d’entraînement.

Les réseaux dédiés à la classification d’images ne sont pas, eux, être totalement convolutionnels, car ils produisent une sortie de taille fixe, dépendant non de la taille de l’image mais du nombre de classes. C’est le cas d’AlexNet [24] réseau entraîné sur la base ImageNet-1K [11] comprenant 1,2 million d’images d’apprentissage appartenant à 1000 classes différentes. L’architecture d’AlexNet est illustrée sur la figure 4.10. On peut voir qu’elle se divise en une première

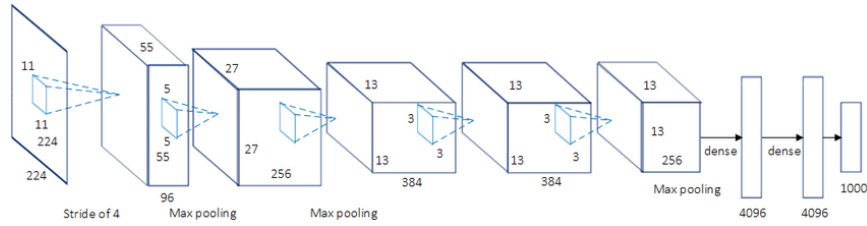


FIGURE 4.10 – Réseau AlexNet pour la classification d’images en 1000 classes [24].

partie à base de convolutions, correspondant à l’extraction de caractéristiques (features) apprises, et une seconde partie faite de couches entièrement connectées, correspondant à la partie classification, qui se termine par un vecteur de taille 1000 qui, une fois normalisé, est interprété comme une distribution de probabilités sur les différentes classes.

Contrairement aux réseaux entièrement convolutionnels, ces architectures ne peuvent pas directement être appliquées à des images de différentes tailles, car le nombre de neurones de la première couche dense doit coïncider avec la taille de sortie de la dernière couche convolutionnelle.

Dû à leur intrinsèque complexité (650 000 neurones représentant plus de 60 millions de poids dans AlexNet), tenter d’expliquer le fonctionnement d’un réseau profond entraîné par l’analyse de ses neurones est pour le moins hasardeux. Il est néanmoins intéressant d’observer les poids des neurones de la première couche, qu’on interprêtera comme une projection dans un espace de caractéristiques obtenu par un banc de filtre de convolution. A cet égard, l’observation des 96 neurones de la première couche d’AlexNet (voir Figure 4.11) permet de confirmer l’importance fondamentale des dérivées partielles évoquée dans la section 3.3, dans la mesure où les poids des neurones (qui forment des noyaux de convolution «couleur» de tailles $11 \times 11 \times 3$) ressemblent souvent à des noyaux dérivateurs d’ordre 1, 2, voire supérieur. On peut aussi noter qu’une majorité de ces filtres sont agnostiques à la couleur, la valeur des poids associés aux trois composantes étant la plupart du temps identiques (noyaux en niveaux de gris).

4.5.3 Transformers

On a vu dans la section précédente que les architectures convolutionnelles font interagir, au niveau de chaque couche, des éléments des données d’entrée formant de petits voisinages, liés à la taille réduite des filtres de convolution utilisés. Pour élargir la portée spatiale du calcul, il faut donc empiler les couches de convolution, de façon à, par associativité de la convolution, agrandir le champ récepteur des neurones, qui augmente linéairement avec la profondeur.

Les couches d’auto-attention (transformers) permettent de dépasser cette limitation de localité spatiale en faisant interagir - en une seule couche - des éléments potentiellement très éloignés dans les données d’entrée. Et de la même façon que les CNN ont intégré la convolution comme primitive de base pour la généralisation des opérations locales par des noyaux appris, les couches d’auto-attention généralisent les opérations non locales en apprenant à la fois des fonc-

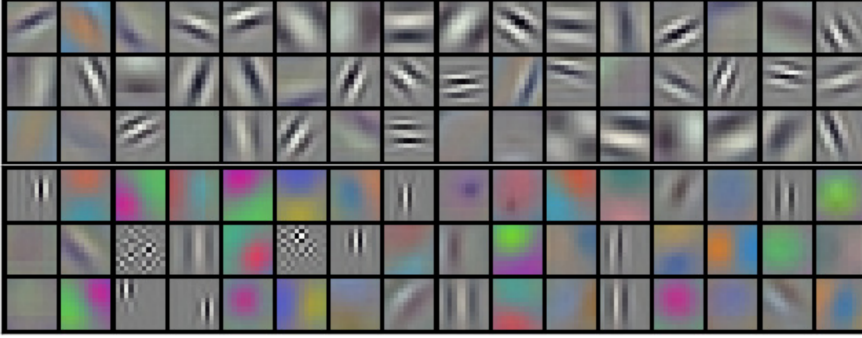


FIGURE 4.11 – Poids des 96 neurones de la première couche convolutionnelle d'AlexNet [24].

tions de «regroupement» (quels pixels doivent interagir) et les pondérations associées.

Les transformers ont fait leur apparition pour l'analyse du langage naturel, dans le but de dépasser la portée temporelle limitée liée à la causalité des architectures à bases de neurones récurrents classiquement utilisés, afin que des mots potentiellement très éloignés dans une phrase puissent être associés dans un calcul [47]. Ils sont devenus également très populaires en Vision par Ordinateur, en particulier grâce au succès du Vision Transformer (ViT), qui est rapidement devenu un standard utilisé dans de nombreuses tâches de vision [12].

Pour l'analyse d'images, on peut voir les transformers comme une version généralisée et *apprise* des moyennes non locales ou NL-Means [5], présentées dans la section 2.2.3. On se rappelle que les NL-Means eux-mêmes généralisaient l'opération de convolution en l'étendant à des voisinages arbitraires :

$$\mathbf{y}_i = \frac{1}{Z(i)} \sum_{j \in \mathcal{V}(i)} \omega(i, j) \mathbf{x}_j \quad (4.27)$$

où \mathbf{x} et \mathbf{y} sont les images d'entrée et de sortie respectivement, i et j représentent des indices (2d ou 3d) de pixels, $\mathcal{V}(i)$ est un voisinage arbitraire (qui peut représenter toute l'image) du pixel i , $\omega(i, j)$ est le poids (scalaire) du pixel j vis-à-vis du pixel i , et $Z(i)$ est la fonction de normalisation associée au pixel i :

$$Z(i) = \sum_{j \in \mathcal{V}(i)} \omega(i, j)$$

On se rappelle également que les NL-Means cherchaient à débruiter les images, par une extension de l'hypothèse de stationarité mise à défaut dans les images (deux pixels proches spatialement ne sont pas nécessairement proches en valeur), en utilisant des fonctions de poids $\omega(i, j)$ qui, au lieu de ne dépendre que de la distance entre les pixels i et j (comme dans une convolution), dépendent de la ressemblance locale entre i et j dans l'image \mathbf{x} , par exemple :

$$\omega(i, j) = e^{-\frac{d_{\mathbf{x}}(i, j)^2}{h^2}}$$

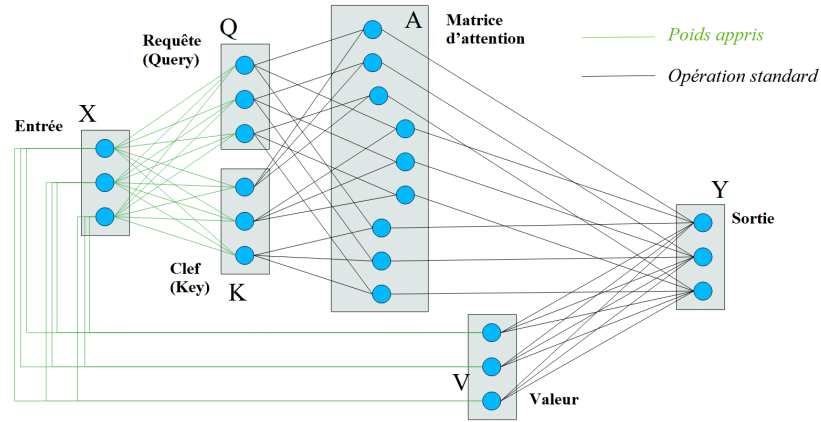


FIGURE 4.12 – Architecture d'un module de Transformer.

où $d_{\mathbf{x}}(i, j)$ est une mesure de similarité locale (par exemple une somme de différences quadratiques entre les voisinages de i et j dans \mathbf{x}), et h une constante agissant sur la distribution des poids.

Dans un transformer, les poids $\omega(i, j)$ vont être appris automatiquement à partir de données exemplaires, et donc pas seulement pour le débruitage, mais pour toute tâche pour laquelle on pourra définir une fonction de perte (loss) pertinente. En effet, dans un transformer, les poids sont assimilés aux composantes d'une matrice A dite *matrice d'attention* :

$$\omega(i, j) \leftrightarrow A_{ij}$$

Cette matrice d'attention est elle-même obtenue par le produit externe de deux vecteurs \mathbf{q} et \mathbf{k} :

$$A = \mathbf{q}\mathbf{k}^t$$

Ces deux vecteurs sont appris à partir du vecteur d'entrée \mathbf{x} qui représente l'image, par projections linéaires réalisées par deux couches entièrement connectées, formalisées par des matrices de poids W_q et W_k respectivement :

$$\mathbf{q} = W_q\mathbf{x} \text{ et } \mathbf{k} = W_k\mathbf{x}$$

Enfin, le vecteur \mathbf{y} , qui représente l'image résultat, est obtenue en multipliant la matrice d'auto-attention A par un vecteur de valeur \mathbf{v} , lequel est lui aussi appris via une couche entièrement connectée formalisée par la matrice de poids W_v :

$$\mathbf{y} = A\mathbf{v} \text{ avec } \mathbf{v} = W_v\mathbf{x}$$

La figure 4.12 synthétise le processus pour des vecteurs (images) à trois composantes (pixels). Le vecteur «requête» \mathbf{q} doit avoir la même taille que la donnée d'entrée \mathbf{x} . Il représente ainsi chaque pixel de l'image qui «requiert l'attention» de tous les autres pixels de l'image \mathbf{x} . Le vecteur «clef» \mathbf{k} a une taille inférieure ou égale à celle de \mathbf{x} . Il représente les «groupes d'attention» qui vont agir d'une façon similaire pour l'obtention du résultat final \mathbf{y} , qui est finalement

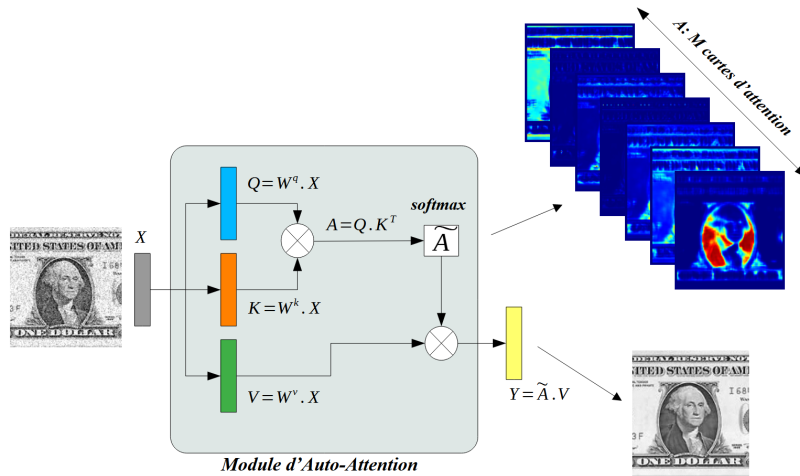


FIGURE 4.13 – Principe du Transformer utilisé en end-to-end : exemple du débruitage

obtenu par une combinaison linéaire des groupes d'attention, pondérés par le vecteur de «valeur» \mathbf{v} , qui doit donc avoir la même taille que le vecteur \mathbf{k} .

En résumé, dans un module d'auto-attention utilisé en end-to-end (voir Figure 4.13), les données d'entrée et de sorties \mathbf{x} et \mathbf{y} sont deux vecteurs (images) de tailles N (où N est le nombre de pixels), le vecteur \mathbf{q} est de taille N , le vecteur \mathbf{k} est de taille M (en général $M < N$), le vecteur \mathbf{v} est de taille M , et donc les matrices de poids W_q , W_k , et W_v sont de tailles $N \times N$, $M \times N$ et $M \times N$ respectivement.

La matrice d'attention A , de taille $N \times M$, est particulièrement intéressante dans la mesure où elle peut permettre d'interpréter les régions dans l'image qui sont pertinentes vis-à-vis d'une certaine tâche ou d'un certain concept. N étant le nombre de pixels et M le nombre de «groupes», on la représente souvent sous la forme d'un ensemble de M «cartes d'attention» de tailles N . Par exemple pour un transformer entraîné à débruiter des images, les cartes d'attention auront tendance à regrouper les pixels dont l'apparence locale est similaire. On peut ainsi retrouver dans la figure 4.13 un comportement proche de la version optimisée des NL-Means dite BM3d [9] qui, au lieu de calculer les poids $\omega(i, j)$ pour tous les N^2 couples de pixels (i, j) , réalise un groupement (clustering) des pixels en M groupes sur des critères de similarité locale, puis obtient le résultat final en réalisant une combinaison linéaire des M groupes.

En pratique, l'application end-to-end de transformers sur des images est déraisonnable, étant donné le nombre de connexions (dimension des 3 matrices de poids) qu'implique les couches entièrement connectées. Les transformers sont donc appliqués en tant que modules sur des données de plus petites tailles, comme des cartes de caractéristiques (feature maps), voir Figure 4.14.

L'architecture de transformer la plus utilisée en vision, ViT [12] est fondée sur une partition régulière de l'image en patches carrés, chacun étant projeté linéairement dans un vecteur de faible dimension par une couche entièrement

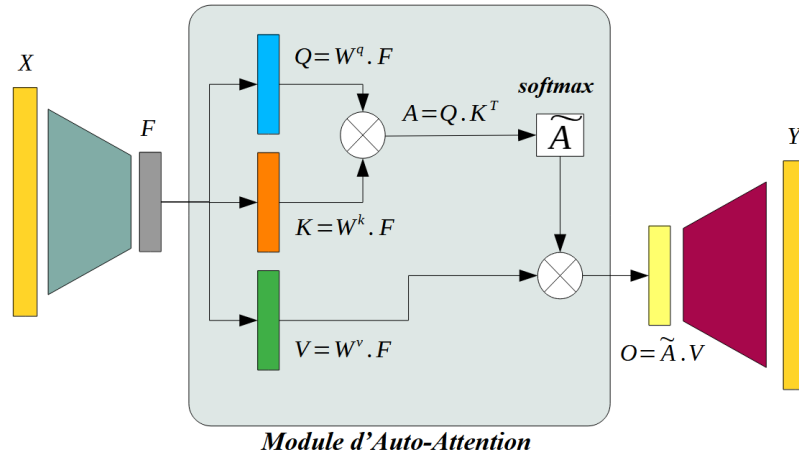


FIGURE 4.14 – Transformer utilisé en tant que module.

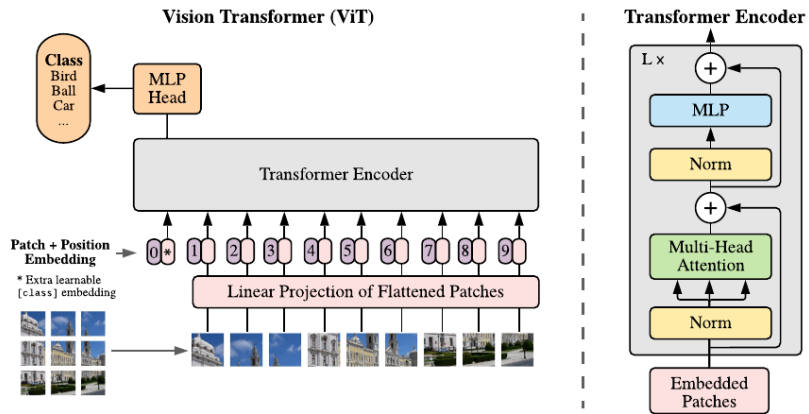
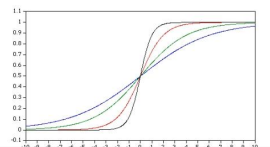
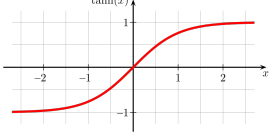
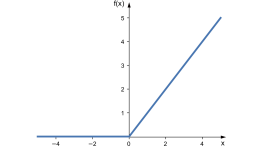


FIGURE 4.15 – Vision Transformer [12].

TABLE 4.1 – Quelques fonctions d'activation classiques

Sigmoide	$f_\lambda(x) = \frac{1}{1+e^{-\lambda x}}$	
	$f'_\lambda(x) = \lambda f_\lambda(x)(1 - f_\lambda(x))$	
TanH	$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$	
	$f'(x) = 1 - f(x)^2$	
ReLU	$f(x) = \frac{x+ x }{2}$	
	$f'(x) = \frac{x+ x }{2x}$	

connectée (perceptron). L'ensemble des vecteurs résultants concaténés forme l'entrée du transformer (voir Figure 4.15). De par l'utilisation de ses couches entièrement connectées, le transformer supprime de fait la relation d'ordre entre les patches, et donc les relations spatiales qui les lient. Or ces relations spatiales jouent un rôle cruciale pour les images. Pour pallier ce défaut, ViT ajoute au vecteur représentatif de chaque patch un encodage de sa position relative dans l'image (position embedding).

On notera enfin que, de la même façon qu'une couche convolutionnelle dans un CNN n'est pas formée d'un seul neurone mais d'une collection de multiples neurones formant un ensemble de canaux de caractéristiques, on aura en général plusieurs transformers différents (multi-head attention), qui vont permettre d'encoder plusieurs concepts pouvant s'avérer utiles et complémentaires pour certaines tâches.

4.5.4 Optimisation

On résume dans cette section les quelques éléments supplémentaires indispensables pour l'entraînement d'un réseau de neurones.

La fonction d'activation du neurone joue un rôle essentiel, celui d'introduire de la non linéarité dans le calcul. Le tableau 4.1 présente les fonctions les plus utilisées en pratique. La fonction sigmoïde, sans doute la plus ancienne, est une approximation dérivable de la fonction seuil de Heaviside. La fonction tangente hyperbolique (TanH) peut être vue comme une version signée de la précédente. La fonction Rectified Linear Unit (ReLU) enfin, est aujourd'hui la plus couramment utilisée dans les réseaux profonds. N'étant pas bornée contrairement aux précédentes, elle permet en effet de limiter l'effet de disparition du gradient lors de la rétropropagation de l'erreur dans les premières couches.

La fonction objective (Loss) a évidemment un rôle déterminant, puisque c'est précisément celle que le processus d'apprentissage doit minimiser. On peut *a priori* définir une variété infinie de fonctions objectives, en respectant certaines contraintes (dérivabilité en particulier), mais on se fonde en général sur des fonctions de base, dont les plus classiques sont présentées dans le tableau 4.2.

TABLE 4.2 – Quelques fonctions objectives (Loss) classiques

Erreur L_2	$\mathcal{L}(Y, V) = \frac{1}{2} \ Y - V\ ^2$
Erreur Logarithmique Moyenne	$\mathcal{L}(Y, V) = \frac{1}{n} \sum_{i=1}^n (\log(1 + V_i) - \log(1 + Y_i))^2$
avec $\forall i, V_i \geq 0, Y_i \geq 0$	
Entropie Croisée	$\mathcal{L}(Y, V) = - \sum_i V_i \log(Y_i)$
avec, $\forall i, 0 \leq V_i \leq 1, 0 \leq Y_i \leq 1$	
Divergence de Kullback-Leibler	$\mathcal{L}(Y, V) = \sum_i V_i \log \frac{V_i}{Y_i}$
avec, $\forall i, 0 \leq V_i \leq 1, 0 \leq Y_i \leq 1$	

Les deux premières fonctions du tableau 4.2 sont plutôt adaptés au cas de la régression (en particulier quand la sortie est une image). La loss L_2 va pénaliser plus fortement les grandes erreurs, tandis que la loss logarithmique ne s'intéresse qu'aux ordres de grandeur. Les deux dernières fonctions sont adaptées au cas de la classification, lorsque le vecteur de prédiction Y encode une distribution des probabilités des différentes classes :

$$Y_i \simeq P(c(X) = i), \text{ avec } \sum_i Y_i = 1$$

On rappelle que lorsque le logarithme est en base 2, l'entropie $E(Y)$ associée à la distribution Y (voir équation 1.25) est le nombre moyen minimum de bits nécessaire pour coder une donnée qui suit la distribution Y , en utilisant un codage optimal tel que le code de Huffman (voir section 1.4). L'entropie croisée $E(X, Y)$ définie dans le tableau 4.2 représente donc le nombre moyen minimum de bits nécessaire pour coder une donnée qui suit la distribution X en utilisant un codage optimal basé sur la distribution Y . Comme ce nombre dépend de la distribution de X , et donc de $E(X)$ on utilise parfois la divergence de Kullback-Leibler, qui correspond au nombre de bits *supplémentaires* nécessaire pour coder une donnée qui suit la distribution X en utilisant un codage optimal basé sur la distribution Y :

$$KL(X, Y) = \sum_i X_i \log \frac{X_i}{Y_i} = \sum_i X_i \log X_i - \sum_i X_i \log Y_i = E(X, Y) - E(X)$$

Dans le cas où V représente la vraie distribution, son entropie est fixe et donc minimiser l'entropie croisée est équivalent à minimiser la divergence KL . D'autre part, on peut noter que dans le cas de la classification, où le vecteur V est défini par $V(i) = 1$ si $i = c$ et $V(i) = 0$ sinon (one-hot class encoding), la minimisation de l'entropie croisée est équivalent au critère bayésien de maximisation de la log-vraisemblance de la donnée X vis-à-vis de la classe i (voir section 4.3). En effet :

$$P(X/i) = \prod_i Y_i^{V_i}$$

Et donc

$$\arg \max_i P(X/i) = \arg \max_i \log P(X/i) \quad (4.28)$$

$$= \arg \max_i \sum_i V_i \log(Y_i) \quad (4.29)$$

$$= \arg \min_i E(V, Y) \quad (4.30)$$

Entraînement - Gradient Stochastique : $W \leftarrow \text{Train}(W, \{X_i, V_i\})$

- Initialisation : $W \leftarrow \text{RANDOM}[] - \eta, +\eta]$
- Répéter jusqu'à convergence (\star) :
 - Pour chaque exemple d'apprentissage k :
 - $Y_k \leftarrow \text{ForwardProp}(W, X_k)$
 - $W \leftarrow \text{BackProp}(W, Y_k, V_k)$

Entraînement - Gradient par lots : $W \leftarrow \text{Train}(W, \{X_i, V_i\})$

- Initialisation : $W \leftarrow \text{RANDOM}[] - \eta, +\eta]$
- Diviser la base d'apprentissage $\mathcal{A} = \{X_i, V_i\}$ en lots \mathcal{A}_b
- Répéter jusqu'à convergence (\star) :
 - Pour chaque lot b :
 - $\mathcal{L}_b \leftarrow \sum_{(X_k, V_k) \in \mathcal{A}_b} \mathcal{L}(\text{ForwardProp}(W, X_k), V_k)$
 - $W \leftarrow \text{BackProp}(W, \mathcal{L}_b)$

Variante (1) du schéma d'optimisation : SGD avec Momentum

$$G_t = \beta G_{t-1} + (1 - \beta) \frac{\partial \mathcal{L}}{\partial w_{ij}}; \beta \in [0, 1]$$

$$w_{ij}^t = w_{ij}^{t-1} - \varepsilon G_t$$

Variante (2) du schéma d'optimisation : Adam

$$G_t = \beta_1 G_{t-1} + (1 - \beta_1) \frac{\partial \mathcal{L}}{\partial w_{ij}}; \beta_1 \in [0, 1]$$

$$K_t = \beta_2 K_{t-1} + (1 - \beta_2) \left(\frac{\partial \mathcal{L}}{\partial w_{ij}} \right)^2; \beta_2 \in [0, 1]$$

$$w_{ij}^t = w_{ij}^{t-1} - \varepsilon \frac{G_t}{\sqrt{K_t + \eta}}$$

