Projet

- Le projet est à réaliser par groupes de 2 ou 3 et à rendre pour le **Mercredi 2 Avril 23h59**
- Des questions portant sur ce projet seront susceptibles d'être posées au DS.
- Une part importante de la notation repose sur l'expérimentation de vos algorithmes et d'une démarche d'**interprétations** et de **visualisation** des résultats obtenus. Il est donc intéressant de proposer des explications des comportements observés lorsqu'on fait varier un paramètre ou qu'on change de réseau et de mettre en place des visualisations pour l'illustrer.
- Il est interdit de recopier ou d'utiliser du code produit par quelqu'un d'autre. Toute fraude avérée sera sanctionnée par une note divisée par 2.

1 Réseau dense général

Dans cette partie, on considère le réseau dense à p couches intermédiaires décrit dans la définition 4.1 du polycopié sur les réseaux de neurones denses (attention, il manque les biais dans la version papier, cf correction sur Moodle).
La topologie du réseau est décrite par une liste :

```
dimensions = [D, C1, C2,...,Cp,K]
```

où :

- D décrit la dimension des données ($D = 2$ dans le cadre des jeux de données de points du plan)
- C_i décrit les dimensions des données intermédiaires.
- K décrit la dimension de la prédiction du résultat. Elle est de $K = 1$ dans le cadre d'un problème à deux classes.

Le fichier `Enonce_exercice_1.py` fournit la lecture des données d'entrainement `nuage_exercice_1` et `nuage_test_exercice_1`.

1. Étant donnée une liste `dimensions`, implémenter le réseau de neurones dense associé.
2. Déterminer le nombre de paramètres présents dans chacun des réseaux donnés par les dimensions suivantes, c'est-à-dire le nombre de scalaires présents dans toutes les matrices de W (on négligera les biais b).
   - `[2, 3, 3, 3, 1]`
   - `[2, 7, 7, 7, 1]`
   - `[2, 15, 15, 1]`
   - `[2, 3, 15, 15, 1]`

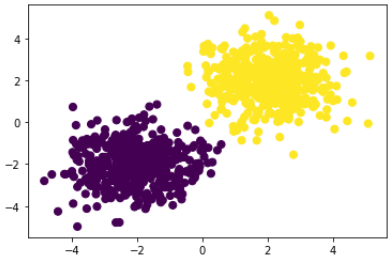
- `[2, 15, 15, 3, 1]`
- `[2, 40, 1]`
- `[2, 20, 20, 1]`
- `[2,5,4,4,4,4,1]`

3. Construire les réseaux précédents et les entrainer sur les données `(X_train, T_train)`.
4. Commenter les résultats obtenus. On pourra notamment regarder :
   - L'évolution de l'erreur d'entropie.
   - Le taux de précision maximal obtenu sur les données de test.
   - La difficulté à obtenir un pas de descente faisant converger le réseau.
   - L'influence de la répartition de dimensions (par exemple `[2,5,8,1]` ou `[2,8,5,1]`.

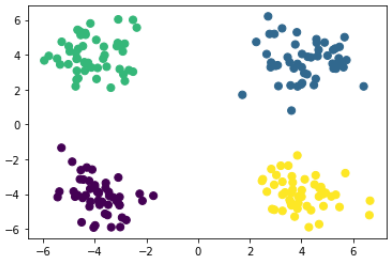
   Proposer des explications aux résultats obtenus.

2 Problème de classification à $K \geq 3$ classes

Dans cette partie, on s'intéresse au problème de classification pour un problème à $K \geq 3$ classes.
Jusqu'à présent, on cherchait à distinguer entre deux classes :



On cherche maintenant à classifier des points répartis en un nombre $K \geq 3$ de classes :



Pour cela, on va changer les matrices de données T et de résultats Y, C.
Ces matrices seront désormais de dimension (N, K).

- La matrice des données T sera telle que :

$$T[i,j] = \begin{cases} 1 \text{ si le point } X[i] \text{ est dans la classe } \mathcal{C}_j \\ 0 \text{ sinon} \end{cases}$$

- La matrice de prédiction de classes C sera telle que :

$$C[i,j] = \begin{cases} 1 \text{ si la classe la plus probable pour le point } X[i] \text{ est } \mathcal{C}_j \\ 0 \text{ sinon} \end{cases}$$

- La matrice de prédiction Y sera telle que :

$$Y[i,j] = \mathbb{P}(X[i] \in \mathcal{C}_j) = \text{ probabilité estimée que X[i] soit dans la classe } \mathcal{C}_j$$

Pour la régression logistique, la formule de prédiction devient :

$$Y = softmax\left(XW + b\right)$$

avec :

$$softmax(x_1, x_2, ..., x_n) = \frac{1}{\sum_{k=1}^{K} e_{x_k}} \left(e^{x_1}, e^{x_2}, ..., e^{x_K}\right)$$

Pour que les produits matriciels soient cohérents, il est nécessaire d'adapter les dimensions des paramètres :

- $W \in M_{D,K}(\mathbb{R})$
- $b \in M_{1,K}(\mathbb{R})$

Dans le cas d'un réseau dense, les couches intermédiaires ne sont pas modifiées. La dernières couche est modifiée comme celle de la régression logistique.
La fonction d'erreur d'entropie devient :

$$J = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln(y_{nk})$$

Pour l'étape d'entrainement, les gradients restent inchangés à l'exception de celui de b qui prend en compte la nouvelle dimension de b :

$$\begin{cases} \delta_{p+1} = Y - T \\ \overrightarrow{grad}_W(J) = {}^tX \cdot (Y - T) \\ \forall j \in [\![1, K]\!], \overrightarrow{grad}_b(J)_{1,j} = \sum_{n=1}^{N} (Y_{n,j} - T_{n,j}) \end{cases}$$

Télécharger le fichier `Enonce_partie_2.py` qui contient les fonctions pour lire les jeux de données.
Adaptation de la régression logistique :

1. Expliquer pourquoi minimiser la fonction J permet d'avoir un réseau qui réalise une bonne classification des points de données.

2. Les données de classes contenues dans `T_train` sont des matrices de dimension $(N, 1)$ telles que $T[i] = j$ où j est la classe. Écrire une fonction `convertit(T)` qui prend en argument une matrice T de ce type et qui renvoie une matrice de $M_{N,K}(\mathbb{R})$ comme décrite dans l'énoncé.

3. Écrire une fonction `softmax(A)` qui prend en argument une matrice $A \in M_{n,p}(\mathbb{R})$ et qui renvoie une matrice B telle que la i-ème ligne de B soit le $softmax$ de la i-ème ligne de A.

4. Écrire une fonction `predit_proba(X,W,b)` qui renvoie la prédiction Y.

5. Expliquer comment obtenir la matrice C de prédiction de classes à partir de la matrice Y. Écrire une fonction `predit_classe(Y)` qui renvoie la matrice C.

6. Appliquer l'algorithme de régression logistique au jeu de données `probleme_4_classes`. Commenter les résultats obtenus.

7. Appliquer l'algorithme de régression logistique au jeu de données `probleme_5_classes`. Commenter les résultats obtenus.

8. Pour le problème à 5 classes, tracer sur un même graphique les droites de séparations de chaque classe et les nuages de points de données. Commenter le résultat obtenu.

Adaptation des réseaux de neurones denses :

1. Visualiser le jeu de données `probleme_6_classes`.

2. Construire un réseau de neurones dense qui classifie de manière satisfaisante ce problème.

3 Méthodes de traitement d'images

L'objectif de cette partie est de définir et implémenter des techniques de traitements d'images. Nous établirons le lien avec les réseaux de neurones en ouverture.

Exercice 1 (Pooling)
Le *pooling* est une méthode de traitement d'images permettant de réduire la taille de l'image tout en conservant ses caractéristiques principales. Une couche de *pooling* prend en entrée une matrice `X` de taille (D_x, D_y) et un couple d'entiers (r_x, r_y) et renvoie une matrice `Y` de taille $\left(\frac{D_x}{r_x}, \frac{D_y}{r_y}\right)$.

Chaque case $Y[i,j]$ de Y est associée à une sous-matrice de taille (r_x, r_y) de X. Les associations se font successivement en partant de gauche à droite et de bas en haut.

- X est une matrice de taille $(4,4)$.
- $(r_x, r_y) = (2,2)$.
- Y est une matrice de taille $\left(\frac{4}{2}, \frac{4}{2}\right) = (2,2)$.

Il existe plusieurs choix de *pooling* :

- Le *pooling max* où la valeur de $Y[i,j]$ est le maximum des valeurs associées.
  Si $Y[i,j]$ est associé aux valeurs $[10, 25, 36, 13]$, alors $Y[i,j] = 36$.

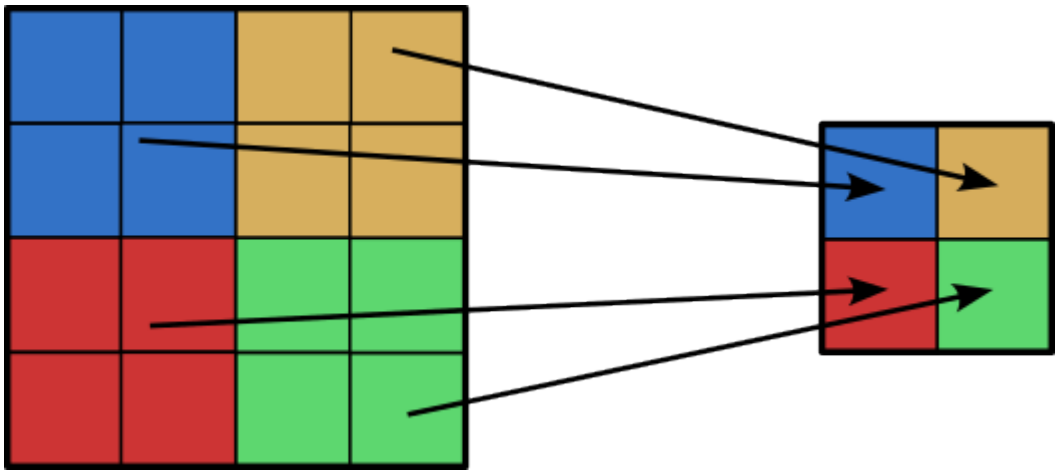


Figure 1: Pooling d'une matrice $(4, 4)$ en un matrice $(2, 2)$

- Le *pooling moyen* où la valeur de $Y[i, j]$ est la moyenne des valeurs associées.
  Si $Y[i, j]$ est associé aux valeurs $[10, 25, 36, 13]$, alors $Y[i, j] = 21$.

- Le *pooling median* où la valeur de $Y[i, j]$ est la médiane des valeurs associées, c'est-à-dire la valeur située au milieu (supérieur si il y a un nombre pair de valeurs) lorsqu'on les trie par ordre croissant.
  Si $Y[i, j]$ est associé aux valeurs $[10, 25, 36, 13]$, alors on peut les trier par ordre croissant.

  $$10 \leq 13 \leq 25 \leq 36$$

  La valeur médiane est alors $Y[i, j] = 25$.
  La médiane des valeurs $1 \leq 10 \leq 21 \leq 32 \leq 45$ est 21.

1. Implémenter une fonction `pooling_max(X,ratio_x,ratio_y)` qui prend en argument une matrice X de taille (D_x, D_y) et des ratios (r_x, r_y) et qui renvoie la matrice Y de taille $\left(\frac{D_x}{r_x}, \frac{D_y}{r_y}\right)$ définie par le *pooling max*.

2. Implémenter des fonctions `pooling_moy` et `pooling_median`.

3. Appliquer ces fonctions à l'image X de telle manière à obtenir des images X_{max}, X_{moy} et X_{median} de taille $(120, 107)$. Dans la suite de l'énoncé, on appliquera nos fonctions à ces nouvelles images, car elles seraient beaucoup plus longues sur l'image originale X.

Exercice 2 (Convolution et traitement d'images)
Dans cet exercice, nous allons voir quelques méthodes classiques du traitement d'images utilisant la **convolution**. Nous commençons par définir la convolution en dimension 1.

La convolution mathématique de deux fonctions f et g **intégrables sur** $\mathbb{R}$:

$$\forall x \in \mathbb{R}, (f * g)(x) = \int_{\mathbb{R}} f(x-t)g(t)dt$$

Un produit de convolution réalise en quelque sorte une moyenne lissée de f et g. On peut adapter le produit de convolution dans le cas discret fini. Pour une liste X de taille $2N+1$ (indexée de $-N$ à N) et une liste F de taille 2H+1 (indexée de $-H$ à H), on définit le produit de convolution comme suit (on suppose $N > H$) :

$$\forall i \in [\![-N, N]\!], (X * F)_i = \sum_{j=-\infty}^{+\infty} X_{i-j}F_j = \sum_{j=-H}^{H} X_{i-j}F_j$$

Dans le cadre de listes ou `np.array`, il convient de commencer la numérotation à l'indice 0.
On fixe alors :

- $X = [x_0, x_1, ..., x_{N-1}]$ de taille N.
- $F = [f_0, f_2, ..., f_{H-1}]$ de taille H.
- $X * F = Z = [z_0, z_2, ..., z_{N-H}]$.

La formule de convolution 1D devient alors :

$$\forall i \in [\![0, N-H]\!], z_i = \sum_{h=0}^{H-1} (x_{i+H-(h+1)}) \times (f_h) \tag{1}$$

1. Écrire une fonction `convolution1D(X,F)` qui réalise la convolution $1D$ pour la donnée X par le filtre F, c'est-à-dire qui renvoie la liste Z de taille $(N-H+1)$ définie en (1).
2. Implémenter une fonction `cross_correlation1D(X,F)` qui construit la liste Z de taille $(N-H+1)$ définie en (2) par :

$$\forall i \in [\![0, N-H]\!], z_i = \sum_{h=0}^{H-1} (x_{i+h}) \times (f_h) \tag{2}$$

3. La tester sur les filtres donnés dans le fichier `.py` et expliquer le rôle des différents filtres F proposés.
4. Établir un lien entre les fonctions `convolution1D(X,F)` et `cross_correlation1D(X',F')`.

Cette définition de la convolution (resp. cross-corrélation) est celle qui correspond à la définition mathématique. Il en existe cependant d'autres, chacune ayant ses spécificités. Nous allons présenter 2 techniques permettant de modifier la convolution standard afin d'obtenir des résultats de tailles différentes. Ces techniques sont adaptables pour la *cross-corrélation*.

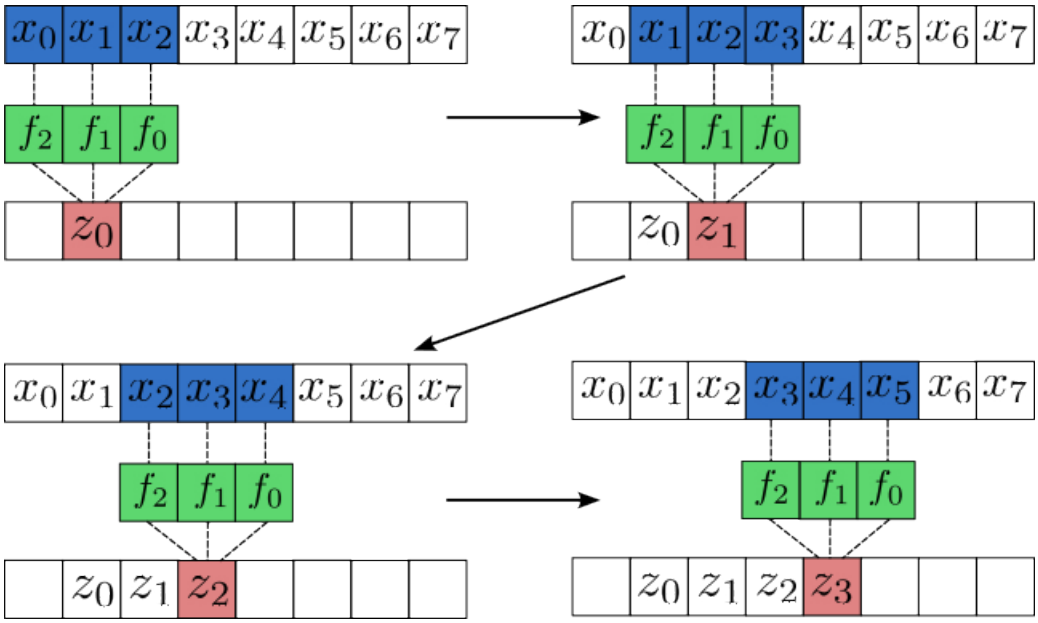


Figure 2: Schéma de la convolution par un filtre de taille 3.

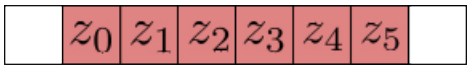


Figure 3: Cases remplies à la fin de la convolution

- *Padding :* Cette méthode consiste à ajouter des 0 en début et en fin du vecteur de données X afin de pouvoir appliquer le filtre F en étant aligné avec les valeurs extrémales de X.

  L'intérêt est d'obtenir une matrice de sortie Z de même taille que la matrice d'entrée X.

$$\forall i \in [\![0, N-1]\!], z_i = \sum_{h=0}^{H-1} \left(x_{i+(H-1)-h-\lfloor \frac{H}{2} \rfloor} \right) \times (f_h)$$

  où les valeurs x_i telles que $i \notin [\![0, N-1]\!]$ sont considérées comme nulles.

- *k-Stride :* Cette méthode consiste à déplacer le filtre de k lors du parcours de la convolution. L'intérêt de cette méthode est de diviser par k la taille du résultat normalement obtenu par la convolution. La convolution classique est donc une 1 stride La formule du résultat est alors :

$$\forall i \in \left[\!\left[0, \frac{N-H}{k}\right]\!\right], z_i = \sum_{h=0}^{H-1} x_{ik+(H-1)-h} f_h$$

1. Implémenter et tester une fonction `convolution1D_padding(X,F)` qui calcule la convolution avec *padding* de X par F.

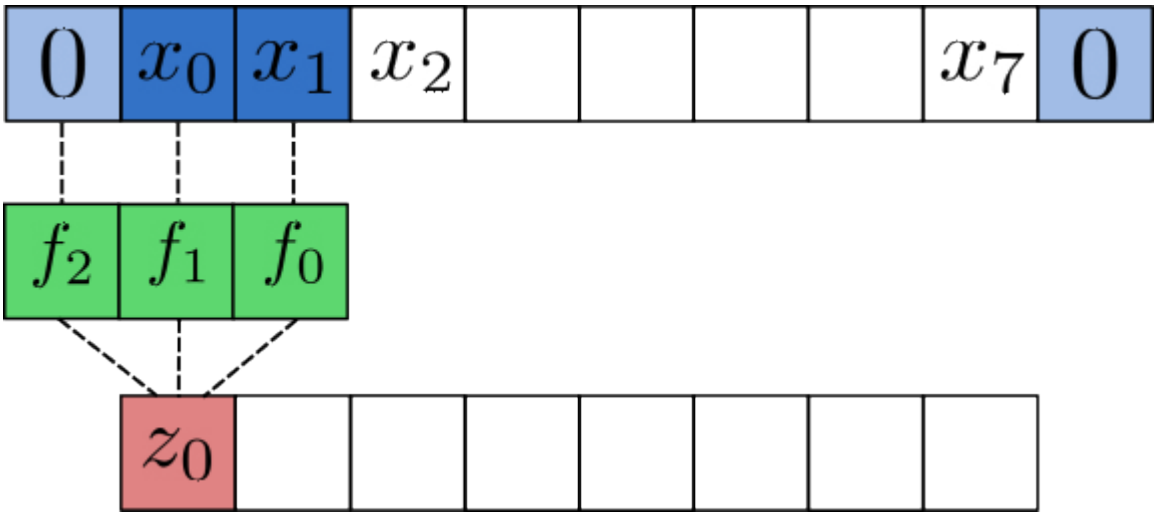


Figure 4: Schéma *padding*

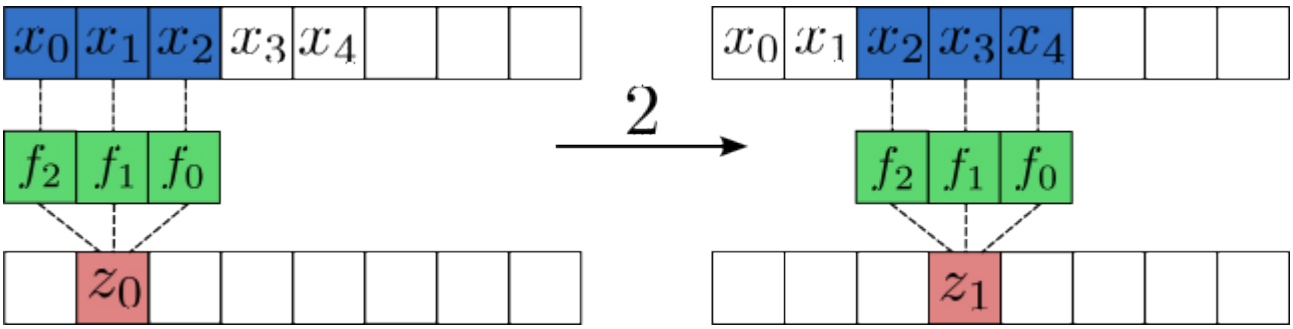


Figure 5: Schéma 2-stride

Figure 6: Cases remplies après la 2-stride ci-dessus

2. Implémenter et tester une fonction `convolution1D_stride(X,F,k)` qui calcule la convolution *k-stride* de X par F.

Nous allons maintenant passer au cas de la convolution en 2D. Nous allons utiliser uniquement la *cross_correlation* car il n'y a pas de $-$ dans les indices manipulés. Comme les deux formules sont équivalentes à une rotation prêt du filtre, ceci n'est pas vraiment restrictif. On verra dans la partie réseaux de neurones pourquoi ce choix n'est pas restrictif.

1. Implémenter une fonction `cross_correlation2D(X,F)` qui prend en argument une matrice X de taille (Dx, Dy) et un filtre F de taille (Hx, Hy) et renvoie la cross_correlation $X * F$ de

taille $(Dx - Hx + 1, Dy - Hy + 1)$ définie par :

$$\forall (i,j), (X * F)_{ij} = \sum_{i'=1}^{Hx} \sum_{j'=1}^{Hy} X_{(i+i')(j+j')} F_{i'j'}$$

2. Implémenter une fonction `applique_filtre(X, F)` qui applique un le filtre `F` à l'image `X`. On affichera le résultat et l'image de départ côte à côte pour observer l'effet obtenu.

3. Appliquer cette fonction aux différents filtres fournis dans l'énoncé et décrire leur effets sur l'image.
   On appliquera le filtre à une matrice `X_pool` obtenue par application du *pooling* à la matrice X originale (qui est trop volumineuse pour des calculs rapides).

4. Il existe des réseaux de neurones, pour lesquels certaines couches intermédiaires sont de la forme :
   $$Z_{(i+1)} = \sigma\left(Z_{(i)} * F_{(i)}\right)$$
   Expliquer l'intérêt que pourraient avoir ce type de réseau en termes de taille de paramètres et d'avantages qu'ils pourraient procurer.