# M2 MIAGE IDA – Deep Learning
## TD 1 – Réseaux de neurones

**Objectifs :**

- Comprendre le fonctionnement interne des réseaux de neurones en les implémentant en Python
- Utiliser SKLearn pour construire un réseau de neurones (rappel)
- Utiliser Keras pour construire un réseau de neurones

## Exercice 1 – Perceptron Multicouche avec SciKitLearn (Rappel)

L'objectif de cet exercice est de vous rappeler comment implémenter des MLP en SciKitLearn. Cette bibliothèque est souvent suffisante pour une telle architecture. Les exercice suivant utiliseront Keras nécessaire au DeepLearning

1) Charger le dataset IRIS. Utilisez un pairplot (bibliothèque seaborn) pour visualiser et analyser les données.
2) Séparez les données en 4 matrices : x_train, x_test, y_train, y_test. Les données d'entrée sont sur les 4 premières colonnes et les données de sortie sont sur la dernière colonne. Le rapport train/test doit être de 75/25.
3) En utilisant SKLearn, construire un classificateur logistique sur l'ensemble de données d'entraînement. Évaluer le modèle généré sur l'ensemble de données de test et affichez le score de prédiction. Que pensez-vous de ce score ?
4) Transformez les données pour les rendre utilisables par un réseau de neurones : la sortie doit être une matrice Nxn (n étant le nombre de classes du problème et N le nombre d'observations) au lieu d'un vecteur Nx1. Chaque colonne doit contenir un 1 si l'observation est de cette classe et zéro sinon. Regarder la fonction `to_categorical`.
5) Toujours en utilisant SKLearn, créer un perceptron pour classifier le jeu de données IRIS. Le réseau doit contenir une couche cachée de 16 neurones. Vous pouvez modifier les hyper-paramètres (solveur, fonction d'activation, taux d'apprentissage, etc.) du réseau comme vous le souhaitez.
6) Entraînez le réseau sur l'ensemble de données d'entraînement et affichez le score sur les ensembles de données d'entraînement et de test.

## Exercice 2 – Perceptron Multicouche avec Keras

L'objectif de cet exercice est de prendre en main la bibliothèque Keras en recodant l'exercice suivant avec ce nouvel outil. Pour cela, nous repartons des données organisées pour les réseaux de neurones (sortie de `to_categorical`) du dataset IRIS.

1) À l'aide de Keras, créez un perceptron construit comme suit :
    a. Une couche dense de 16 neurones connectée à une couche d'entrée de 4 neurones. La fonction d'activation de cette couche doit être une sigmoïde.
    b. Une couche de sortie de 3 neurones avec une fonction d'activation softmax.

2) Utiliser la fonction de perte `categorical_crossentropy` (perte logarithmique), l'optimiseur Adam et activer le précalcul de la métrique de précision.
3) Entraîner le réseau pendant 100 epochs.
4) Comparer la précision de la classification obtenue avec la régression logistique de l'exercice 2.

## Exercice 3 – Adapter un réseau à un problème donné

L'objectif est de construire un perceptron multicouche en modifiant son architecture pour l'adapter à un problème donné.

1) Charger le dataset MNist et visualiser les données chargées. Affichez les premières images du jeu de données. Que contient ce dataset et quel est son objectif ?
2) Modifier les données pour :
    a. Linéariser les données en deux dimensions.
    b. Normaliser les données
    c. Transformer le vecteur de classes pour qu'il soit utilisable par un réseau de neurones.
3) Entraîner l'architecture de l'exercice précédent à ce nouveau problème. Attention à la taille des couches d'entrée et de sortie !
4) Entraîner le réseau de quelques epochs avec une taille de batch de 128. Observez le score du réseau. Que pensez-vous de ce résultat ?

## Exercice 4 – Créer des réseaux plus profonds

L'objectif est de créer un perceptron multicouche sur une tâche plus complexe de classification d'images.

1. Charger, afficher et formater le jeu de données Cifar10.
2. Adapter le perceptron de l'exercice précédent. Entraîner, évaluer et afficher les erreurs de classification. Essayez de modifier doucement le réseau (pas de nouvelle couche mais une couche cachée plus grande, des fonctions d'activation différentes, etc.) et évaluer ces différents réseaux. Que pensez-vous de la prédiction que vous obtenez ?
3. Construire le perceptron multicouche suivant :
    a. Première couche cachée de 512 neurones densément connectés à la couche d'entrée, fonction d'activation relu, dropout de 0,2 et normalisation par lots.
    b. Deuxième couche cachée de 512 neurones densément connectés à la couche d'entrée, fonction d'activation Relu, normalisation par lot et dropout de 0,2.
4. Exécuter 50 epochs d'apprentissage et évaluer le réseau obtenu. Que pensez-vous du score de prédiction ?
5. Enregistrer votre modèle sur votre disque dur et calculer la précision et la perte pour chaque epoch d'entraînement.
6. Pensez-vous pouvoir faire mieux ? Si oui, construisez l'architecture !

## Exercice 5 – Réseau de neurones

L'objectif de cet exercice est d'implémenter un réseau de neurones simple en Python pour en comprendre le fonctionnement. Le code produit doit permettre de construire un perceptron multicouche avec une fonction d'activation relu pour les couches cachées et une fonction sigmoid sur la couche de sortie. Le réseau doit pouvoir apprendre les poids synaptiques sur un dataset d'entrainement en utilisant la loi de Hebb (algorithme d'apprentissage le plus simple). Une fois l'entrainement réalisé, le réseau doit pouvoir prédire une valeur pour un nouveau sample. Pour simplifier, on se place dans le cadre de la classification.

1) Construire l'architecture générale du réseau. Créer les classes :
    a. `Neuron` contenant une valeur, l'ensemble des synapses entrant et sortant, une fonction d'agrégation et une fonction d'activation
    b. `Synapse` contenant le poids synaptique et le neurone entrant et le sortant
    c. `MultiLayerPerceptron` contenant une liste de `Neuron` organisés en couche et les paramètres nécessaires à l'apprentissage
2) Implémenter les classes ci-dessus pour que la classe `MultiLayerPerceptron` (seule visible pour l'utilisateur final) puisse :
    a. `__init__` : Créer un perceptron à partir d'un tableau représentant le nombre de neurones à créer dans chaque couche. Par exemple si on passe [3, 5, 3, 2], il faut construire un réseau contenant 1 couche d'entrée contenant 3 neurones, 2 couches cachées contenant respectivement 5 et 3 neurones et une couche de sortie contenant 2 neurones.
    b. `randomize_weights` : Initialiser aléatoirement tous les poids du réseau entre [-1, 1].
    c. `predict` Prédire un résultat à partir un exemple d'entrainement (1 ligne du dataset).
    d. `update_weights` : Mettre à jours les poids synaptiques en utilisant l'algorithme de Hebb vu en cours.
    e. `train` : Entrainer un réseau sur un dataset donné.
3) Proposer un exemple d'utilisation de votre réseau soit sur un dataset connu (IRIS, MNist, etc.) soit sur un dataset fait à la main (fonction OR par exemple)