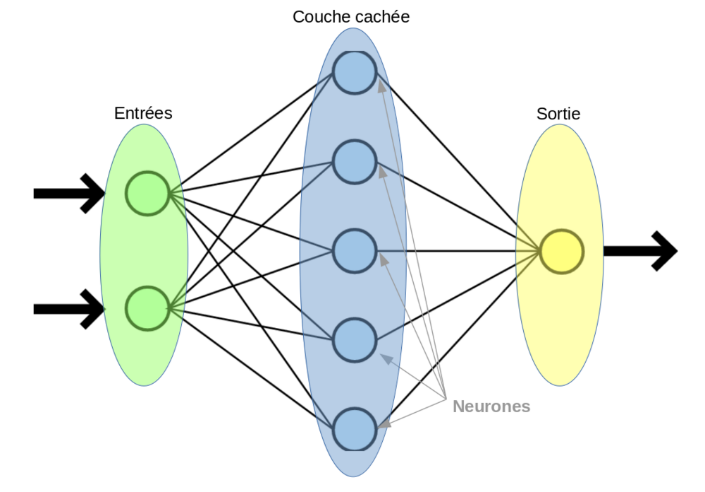
# TP4 : Classification IRIS avec un Réseau de Neurones

Ce TP est conçu pour vous offrir une compréhension pratique et concrète des étapes nécessaires pour développer un modèle de deep learning simple et efficace. Ce modèle est le Réseau de Neurones Artificiels



**Objectifs**

* Apprendre à construire un modèle de réseau de neurones pour la classification.
* Comprendre l'utilisation des fonctions d'activation ReLU et softmax.
* Pratiquer la visualisation des résultats de classification.

**Matériel**

* Un ordinateur avec Python 3.x installé.
* Les bibliothèques Python installées : Pandas, numpy, scikit-learn, keras (tensorflow), matplotlib.

**Prérequis**

* Une connaissance de base de Python.
* Une compréhension des termes de base en machine learning.
* Une compréhension des concepts fondamentaux sur les réseaux de neurones.

**Données**

* Le dataset Iris contient des informations sur les fleurs d'iris, avec les caractéristiques suivantes :
  + **sepal\_length**: Longueur du sépale (cm)
  + **sepal\_width**: Largeur du sépale (cm)
  + **petal\_length**: Longueur du pétale (cm)
  + **petal\_width**: Largeur du pétale (cm)
  + **species**: Espèce de l'iris (setosa, versicolor, virginica)
* Vous pouvez l’importer directement depuis la libraire scikit-learn, load\_iris() va retourner l’ensemble des données IRIS :

from sklearn.datasets import load\_iris

**Parties**

1. **Charger les Bibliothèques**
   1. [layers, models] depuis keras
   2. [train\_test\_split] depuis sklearn
   3. [pyplot] depuis matplotlib
2. **Charger l'ensemble de données Iris**
   1. Charger les données à l’aide de la fonction load\_iris()
   2. Séparer les données X (iris.data) et les étiquettes X (iris.target)
3. **Diviser les données en ensembles d'entraînement et de test**
   1. Diviser les données en ensembles d'entraînement (80%) et de test (20%) à l’aide de la fonction train\_test\_split()
4. **Construire le modèle de réseau de neurones**
   1. Construire le modèle Séquentiel (model.Sequential) avec :
      1. Une couche dense (layers.Dense) avec :
         1. 10 neurones : pour capturer des relations non linéaires simples
         2. Fonction d'activation ReLU : pour permettre de résoudre des problèmes non linéaires en activant seulement les neurones avec des valeurs positives et en mettant à zéro les valeurs négatives.
         3. Inshape=(4,) : pour capturer les 4 caractéristiques en entrée
      2. Une couche dense de sortie (layers.Dense)
         1. 3 neurones : pour les 3 classes de sortie
         2. Fonction d’activation softmax : pour avoir les probabilités pour chaque classe
5. **Compiler le modèle**
   1. Compiler le modèle à l’aide de la fonction model.compile() avec :
      1. L’optimizer=’adam’ :
      2. loss='sparse\_categorical\_crossentropy': Perte utilisée pour les étiquettes non one-hot encodées.
      3. metrics=['accuracy']: Utilise l'exactitude pour mesurer les performances du modèle.
6. **Entraîner le modèle**
   1. Entrainer le modèle à l’aide de la fonction model.fit() avec :
      1. Les données d’entrainement data (X)
      2. Les données d’entrainement labels (y)
      3. Entrainer le modèle pendant 50 époques
7. **Évaluer le modèle**
   1. Evaluer les performances du modèle à l’aide de la fonction model.evaluate() avec :
      1. Les données de test data (X)
      2. Les données de test labels (y)
   2. Afficher l’accuracy retourné
8. **Faire des prédictions et visualiser les résultats**
   1. Récupérer les prédictions (pred\_y) sur les données de test data (X) à l’aide de la fonction model.predit()
   2. Créer une figure avec des sous graphes
   3. Chaque sous graphe contient trois barres qui représente les prédictions pour chaque classe.
   4. Chaque sous graphe a pour titre la classe réel et la classe prédite

**Travail à rendre :**

Appliquer le modèle précédemment vu sur la dataset de diabètes pour prédire la maladie de diabète .