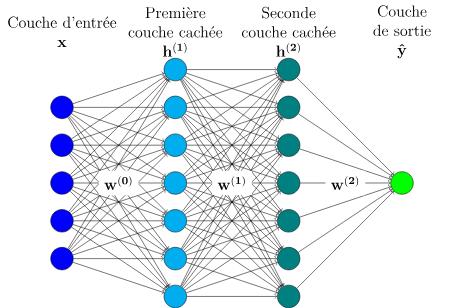
# TP6 : Multilayer perceptron pour prédire le prix médian

Ce TP vise à prédire le prix médian d’une maison à partir de ses caractéristiques. Il est conçu pour introduire les concepts de base du deep learning sans trop de complexité. En utilisant le réseau de neurones artificiels (RNA), plus précisément un réseau dense multi-couches (multilayer perceptron MLP).



**Objectifs**

* Comprendre les étapes de base de la création d'un modèle de prédiction avec un réseau de neurones.
* Apprendre à utiliser TensorFlow/Keras pour construire et entraîner un modèle.
* Apprendre à évaluer la performance d'un modèle de régression.

**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 concepts fondamentaux de la régression linéaire.
* Une compréhension des termes de base en machine learning.
* Comprendre le rôle de la normalisation ou la standardisation des données.

**Données**

* Ce jeu de données fournit des caractéristiques telles que la criminalité, le nombre de pièces, l'âge des maisons, etc., et le prix médian des maisons.



* **CRIM:** Taux de criminalité par habitant.
* **ZN:** Proportion de terrains résidentiels zonés pour des lots de plus de 25 000 mètres carrés.
* **INDUS:** Proportion de surface d'activité commerciale par ville.
* **CHAS:** Variable binaire (0 ou 1) représentant si la zone est frontalière à Charles River (1) ou non (0).
* **NOX:** Concentration d'oxydes d'azote (ppm).
* **RM:** Nombre moyen de pièces par maison.
* **AGE:** Proportion de maisons construites avant 1940.
* **DIS:** Distances pondérées aux cinq centres d'emplois de Boston.
* **RAD:** Indice d'accessibilité aux autoroutes.
* **TAX:** Taux d'imposition sur les valeurs foncières par $10 000
* **LSTAT:** Pourcentage de la population de faible statut socio-économique.
* **MEDV:** Prix médian des maisons en milliers de dollars.

**Parties**

1. **Chargement des données (Importation et préparation) :**
   * Importer les librairies nécessaires (pandas, numpy, sklearn, keras, pyplot).
   * Charger le dataset « TP4\_dataset.csv ».
   * Créer un DataFrame pour faciliter la manipulation des données.
2. **Préparation des données :**
   * Séparer les données en variables indépendantes (X, caractéristiques des maisons) et la variable dépendante (y, prix des maisons).
   * Standardiser les données en utilisant StandardScaler.
   * Diviser les données en ensembles d'entraînement, de validation et de test.
3. **Construction du modèle (MLP) :**
   * Définir le réseau de neurones à l'aide de keras.Sequential(). Le modèle est composé de couches denses (Dense) avec des fonctions d'activation ReLU. La couche de sortie a une seule unité car nous prédisons un seul nombre (le prix de la maison).
4. **Compilation du modèle :**
   * Utiliser la fonction model.compile() pour paramétrer le modèle avec les paramètres:
     1. optimizer='adam': adam est un algorithme qui ajuste les "paramètres" (les poids) du réseau pour trouver la meilleure façon de prédire le prix des maisons.
     2. loss='mse': C'est la mesure de l'erreur du modèle. Le modèle essaie de minimiser cette erreur pendant l'entraînement. mse signifie "erreur quadratique moyenne" (Mean Squared Error). Plus la valeur de loss est basse, meilleur est le modèle.
     3. metrics=['mae']: C'est un indicateur supplémentaire qui mesure l'erreur du modèle. mae signifie "erreur absolue moyenne".
5. **Entraînement du modèle :**
   * model.fit() entraîne le modèle sur les données d'entraînement en utilisant les paramètres spécifiés.
     1. epochs=100: Spécifie le nombre de fois que le modèle va passer sur l'ensemble des données d'entraînement.
     2. batch\_size=32: Détermine la taille des sous-ensembles des données d'entraînement sur lesquelles le modèle est entraîné à chaque itération.
     3. validation\_data=(X\_val, y\_val): Pour éviter le sur-apprentissage. Il fournit des données (X\_val, y\_val) séparées de l'entraînement sur lesquelles le modèle est évalué après chaque époque.
     4. verbose=1: Affiche les progrès de l'entraînement à chaque époque, notamment la perte et la précision.
6. **Évaluation du modèle sur les données de test :**
   * Evaluer le modèle sur les données de test en calculant le MAE. Cela donne une estimation de la performance du modèle sur des données qu'il n'a pas vues pendant l'entraînement (utilisation de model.evaluate() ).
7. **Prédictions et calcul de métriques additionnelles :**
   * Appliquer la fonction de prédiction sur l’ensemble de test model.predict()
   * Calculer La MSE (mean\_squared\_error()) et le R-squared (r2\_score) en se basant sur le résultat de model.predict() et les données de test (y\_test).
8. **Interprétation des résultats :**
   * Analyser le niveau de votre modèle avec R-squared :
     1. Si le R-squared est supérieur à 80%, alors le modèle a un bon niveau de prédiction.
     2. Si le R-squared est supérieur à 60%, alors le modèle a un moyen niveau de prédiction.
     3. Si non, alors le modèle a un faible niveau de prédiction
   * Analyser le niveau de votre modèle avec MSE :
     1. Si le MSE est inférieure à 20, alors le modèle ne génère pas trop d’erreurs.
     2. Si non, alors le modèle génère trop d’erreurs.
9. **(Bonus) Visualisation de la courbe d'apprentissage :**
   * Dans deux subplot afficher la perte d’entrainement et la perte de validation en MSE (loss) et MAE.

**Travail à rendre :**

Appliquer le modèle MLP sur la dataset IRIS.