Deep Learning (I-ILIA-029) -- Travaux Pratiques

TP 3 : Deep Learning pour la prédiction de prix de maisons

Introduction : L'objectif de ce TP est de développer des modèles ***Deep Learning*** pour la prédiction de prix de maisons avec une nouvelle base de données qui est plus consistante que celle du TP1 en termes d'observations et caractéristiques (*Features*) **(Voire Annexe 1)**. En effet, la BD vue au TP1 reste moins complexe et des modèles *Machine Learning* étaient largement suffisants pour trouver une solution. Par ailleurs, la BD du TP2 nécessite l'emploi de modèles Deep Learning appliquant une non linéarité pour résoudre de le problème avec une efficacité optimale. Ce TP2 vous permettra également de prendre la main avec les outils suivants :

- **Pytorch Lightning** : un framework intuitif conçu pour simplifier la création, l'entraînement et évaluation des modèles *Deep Learning* sous Pytorch. Vous pouvez consulter cette vidéo d'introduction pour en savoir plus.
- **Wandb** : plateforme permettant d'enregistrer/suivre en temps réel l'évolution des modèles, sans devoir gérer les fichiers logs (courbes d'entrainement, résultats, etc.) en local

Les modèles développés durant ce TP seront évalués avec les mesure **MAE** et **R^2**

- **MAE (erreur absolue moyenne)** : moyenne des écarts absolus entre les prédictions et les valeurs réelles. **Plus la valeur MAE est proche de 0, mieux c'est.**

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Sachant que **yi** est la variable cible, et **xi** la valeur de prédiction.

- **R^2 score (coefficient de détermination)** : capacité du modèle à expliquer la variance des données. Son intervalle de variation va de **-∞** à **1.** Plus **R^2** est proche de **1**, **mieux c'est**.
  - **R^2 = 1 :** le modèle est parfait pour la prédiction
  - **R^2 = 0 :** le modèle ne fonctionne pas mieux que la simple moyenne des données.
  - **R^2 < 0 :** le modèle est moins performant qu'une prédiction constante ou moyenne.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

Sachant que $\mathbf{y_i}$ est la variable cible, et $\hat{\mathbf{y}}$ la valeur de prédiction.

I. Partie I : Préparation des données :

- **Instruction 0 :** télécharger et lancer le code de démarrage « **Input_TP3_IA_2025.ipynb** »
- **Instruction 1 :** installer les librairies nécessaires pour l'analyse et visualisation des données
- **Instruction 2 :** à l'extérieur du notebook créer un compte Wandb (voici le lien : https://wandb.ai/login)
  a. Cliquer sur l'icône de votre profil utilisateur dans le coin supérieur droit.
  b. Sélectionner **« user settings »**, puis aller à la section **« API Keys »**.
  c. Cliquez sur **« Reval »** et copier la clé API affichée
- **Instruction 3 :** coller cette clé dans le code de la cellule

```
import wandb
wandb.login(key="????????????????????????????????????????")
```

- **Instruction 4 :** importer les librairies nécessaires pour la partie Deep Learning

➢ **Question 1 :** télécharger la base de données accessible via ce **lien**

➢ **Question 2 :** afficher la dimension et quelques échantillons de la BD en utilisant ***shape*** **et** ***head***

- **Instruction 5 :** analyser et exécuter la fonction « ***preprocess_data*** » permettant d'ajouter une colonne « **age** » calculée à partir de « **year built** » et « **sales year** » pour estimer l'age de la maison

➢ **Question 3 :** appliquer cette fonction pour préparer et prétraiter vos données.

- **Instruction 6 :** diviser la base de données en train (**80%**), validation (**10%**) et test (**10%**) en utilisant train_test_split. Lire la documentation.
- **Instruction 7 :** normaliser les données en utilisant la fonction à analyser « ***normalize_data*** ».
- **Instruction 8 :** analyse et exécuter le code de la classe « ***TabularDataset*** » permettant de **générer des couples (entrée, sortie)** sous forme de **tensors PyTorch**, ce qui est indispensable pour entraîner les modèles Deep Learning sous Pytorch.
- **Instruction 9 :** à l'aide de la classe « ***TabularDataset*** », créer des données sous forme de tensors pour les trois parties : train, validation et validation. Ensuite utiliser les « **Dataloader** » pour charger les données sous formes de batch (batch size = 128).

  Note :

  - **Dataloader :** un générateur qui nécessite une base de données pour créer des batch de données, utilise le parallélisme. Lire la documentation.
  - **Dataset :** sert à charger, transformer les données une par une via la fonction **__getitem__**. Cela rend le code plus modulable. Lire la documentation

- **Instruction 10 :** exécuter les fonctions de calcul des métriques d'évaluation de modèles **MAE** et $\mathbf{R^2}$.

II. Partie II : Développement du premier réseau de neurones à une seule couche :

- **Instruction 11 :** exécuter et analyser attentivement la classe « **BaseModel »,** qui hérite de « **Lightning Module »** , qui permet de structurer l'entraînement, la validation et le test du modèle avec un calcul automatisé des métriques.

- **Instruction 12 :** définir l'architecture neuronale de votre premier modèle à **01 seule couche** sans fonction d'activation.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLPModelSingleLayer                      [1, 1]                    --
├─Linear: 1-1                            [1, 1]                    20
==========================================================================================
Total params: 20
Trainable params: 20
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.00
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
==========================================================================================
```

Figure 1 : résumé de l'architecture du modèle à une seule couche

- **Instruction 13 :** visualiser l'architecture de votre modèle à l'aide de la fonction « ***summary()*** »

➢ **Question 4 :** Lancer l'entraînement de votre modèle après avoir définit les paramètres suivants **:**

a. Fonction de perte : **nn.MSELoss**

b. Modèle : instancier votre modèle avec **« 20 epochs »** et un pas d'apprentissage : **lr = 0.0120**

c. Trainer : boucle d'entrainement, implémentée par Torch Lightning. Lire la documentation

d. Lancer l'entrainement en utilisant la fonction « **fit** » avec les données d'entrainement et validation. Lire la documentation

Note : Adam est une variante de SGD permettant d'ajuster le taux d'apprentissage automatiquement et d'assurer une direction plus stable à l'apprentissage « ***momentum*** ». Plus de détail : voir ce **lien**.

➢ **Question 5 :** vérifier et interpréter les résultats de votre entraînement via Wandb.

➢ **Question 6 :** évaluer le modèle avec les données de test en utilisant la fonction « ***trainer.test*** »

➢ **Question 7 :** Calculer les prédictions des données de test avec la fonction « ***trainer.predict*** »

➢ **Question 8 :** analyser et exécuter la fonction « **plot_model_predictions »**

➢ **Question 9 :** appliquer la fonction « **plot_model_predictions »** aux données de test. Que constatez-vous ?

III. Partie III : Développement du 2ème réseau de neurones profond à 2 couches :

➢ **Question 10 :** créer un second modèle basé sur la structure suivante :

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLPModel1                                [1, 1]                    --
├─Linear: 1-1                            [1, 64]                   1,280
├─Linear: 1-2                            [1, 1]                    65
==========================================================================================
Total params: 1,345
Trainable params: 1,345
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.00
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.01
Estimated Total Size (MB): 0.01
==========================================================================================
```

Figure 2 : résumé de l'architecture du réseau de neurones à deux couches

➢ **Question 11 :**

- lancer l'entrainement avec « **100 epochs** », « **lr= 0.0022** », fonction activation « **ReLU** » pour la couche 1 et nom de modèle **« Neural_2 »**. Analyser les résultats sur wandb
- évaluer le modèle avec les données de test
- visualiser le résultat des prédictions avec la fonction « **plot_model_predictions »**
- Que constatez-vous ?

IV. Partie IV : Développement du 3ème réseau de neurones profond à 4 couches :

➢ **Question 12 :** créer un troisième modèle encore plus profond suivant la structure suivante :

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLPModel2                                [1, 1]                    --
├─Linear: 1-1                            [1, 128]                  2,560
├─Linear: 1-2                            [1, 64]                   8,256
├─Linear: 1-3                            [1, 32]                   2,080
├─Linear: 1-4                            [1, 1]                    33
==========================================================================================
Total params: 12,929
Trainable params: 12,929
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.01
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.05
Estimated Total Size (MB): 0.05
==========================================================================================
```

Figure 3 : résumé de l'architecture du réseau de neurones à quatre couches

➢ **Question 13 :**

- lancer l'entrainement avec « **100 epochs** », « **lr= 0.00063** », fonctions d'activations Relu pour les trois premières couches nom de modèle **« Neural_3 »**. et analyser les résultats sur wandb
- évaluer le modèle avec les données de test
- visualiser le résultat des prédictions avec la fonction « **plot_model_predictions »**
- Que constatez-vous ?

V. Partie V : Développement du 4ème réseau de neurones profond à 5 couches :

➢ **Question 14 :** créer un autre modèle avec l'architecture suivante et appliquer le ***Dropout*** en respectant l'ordre définit ci-dessous :

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLPModel3                                [1, 1]                    --
├─Linear: 1-1                            [1, 256]                  5,120
├─Dropout: 1-2                           [1, 256]                  --
├─Linear: 1-3                            [1, 128]                  32,896
├─Dropout: 1-4                           [1, 128]                  --
├─Linear: 1-5                            [1, 64]                   8,256
├─Linear: 1-6                            [1, 32]                   2,080
├─Linear: 1-7                            [1, 1]                    33
==========================================================================================
Total params: 48,385
Trainable params: 48,385
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.05
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.19
Estimated Total Size (MB): 0.20
==========================================================================================
```

Figure 4 : résumé de l'architecture du réseau de neurones à cinq couches avec Dropout

➢ **Question 15 :**

- lancer l'entrainement avec :
  - « **100 epochs** », « **lr= 0.00131** »
  - Fonctions d'activation « **ReLU** » pour les 4 premières couches
  - Dropout de 20% pour les 2 premières couches
- évaluer le modèle avec les données de test
- visualiser le résultat des prédictions avec la fonction « **plot_model_predictions** »
- Que constatez-vous ?

➢ **Question 16 :** Compléter le tableau suivant et tirer vos conclusions de TP

	Train dataset		Validation dataset		Test dataset	
	MAE	**R^2**	**MAE**	**R^2**	**MAE**	**R^2**
Modèle à 1 couche						
Modèle à 2 couches						
Modèle à 4 couches						
Modèle à 6 couches + Dropout						

Tableau 1: Analyse comparative des métriques entre les 4 modèles neuronaux

VI. Annexe 1 :

Dans cette section, nous vous présentons la structure détaillée de la base de données traitée via ce TP. Cette base de données est représentée par **21613 échantillons** de données (lignes) ou chaque donnée (ligne) est représentée par **20 caractéristiques d'entrée** et **une caractéristique de sortie** (Prix de maison)

a. Caractéristiques d'entrée :

- **id** : Identifiant unique pour chaque maison vendue
- **date** : Date de la vente de la maison
- **bedrooms** : Nombre de chambres
- **bathrooms** : Nombre de salles de bain (0.5 représente une pièce avec toilettes mais sans douche)
- **sqft_living** : Superficie intérieure habitable de l'appartement (en pieds carrés)
- **sqft_lot** : Superficie du terrain (en pieds carrés)
- **floors** : Nombre d'étages
- **waterfront** : Variable binaire indiquant si l'appartement donne sur l'eau ou non
- **view** : Indice de 0 à 4 indiquant la qualité de la vue de la propriété
- **condition** : Indice de 1 à 5 évaluant l'état général de l'appartement
- **grade** : Indice de 1 à 13 évaluant la qualité de la construction et du design (1-3 = insuffisant, 7 = moyen, 11-13 = haut de gamme)
- **sqft_above** : Superficie habitable au-dessus du niveau du sol (en pieds carrés)
- **sqft_basement** : Superficie habitable en sous-sol (en pieds carrés)
- **yr_built** : Année de construction initiale de la maison
- **yr_renovated** : Année de la dernière rénovation de la maison
- **zipcode** : Code postal de la maison
- **lat** : Latitude
- **long** : Longitude
- **sqft_living15** : Superficie habitable des 15 maisons voisines les plus proches (en pieds carrés)
- **sqft_lot15** : Superficie des terrains des 15 maisons voisines les plus proches (en pieds carrés)

b. Caractéristiques de sortie :

- **price** : Prix de chaque maison vendue

La figure 5 illustre le format de ces données par rapport aux données du TP1

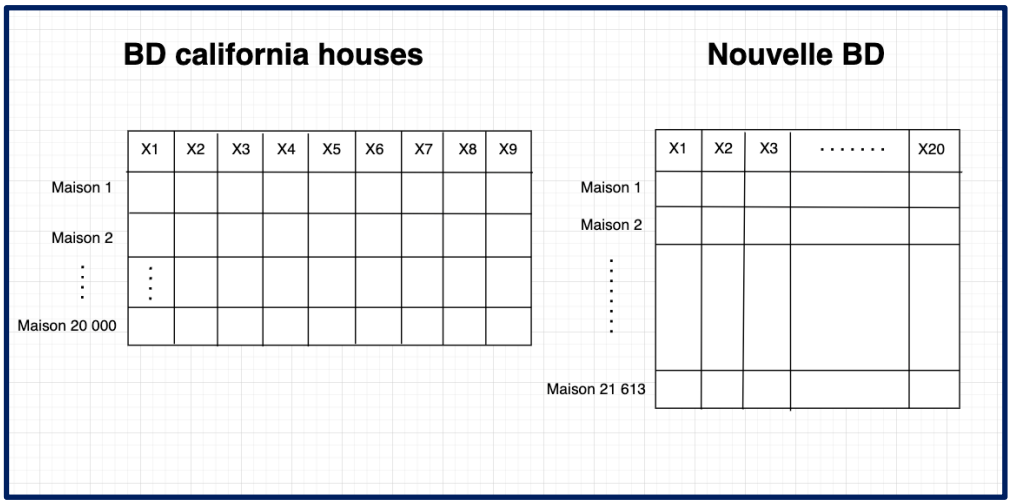

Figure 5: format des données : TP1 et TP2

La figure 6 illustre l'objectif du modèle Deep Learning à développer durant ce TP.

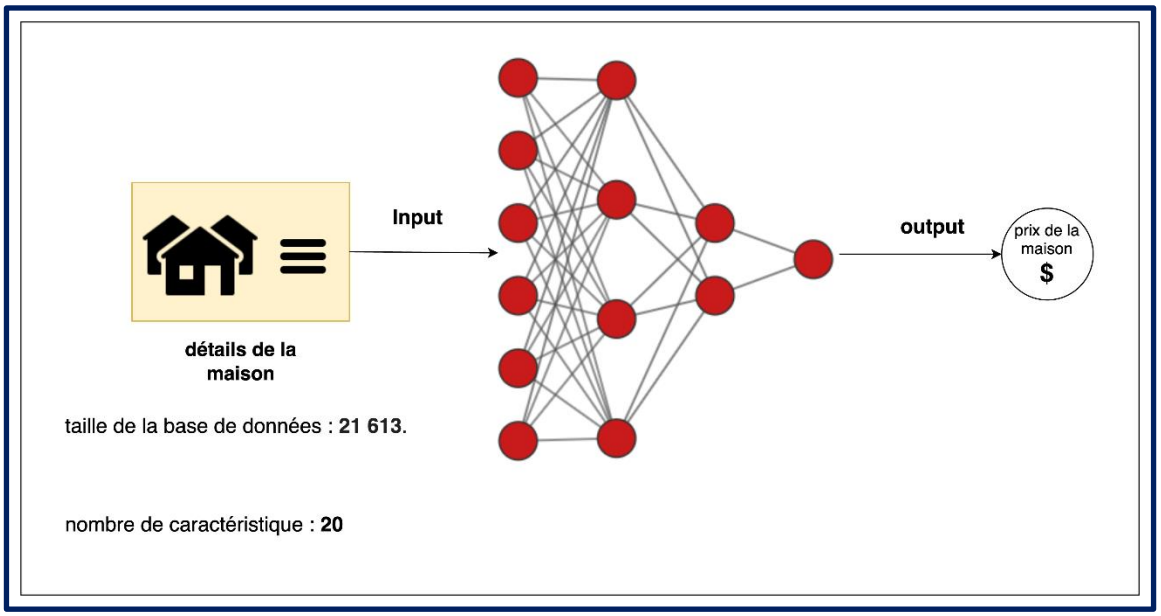

Figure 6: mode de fonctionnement du modèle Deep Learning

Syntaxe de quelques fonctions Torch Lightning :

- **Définir la fonction d'activation Relu :** x = torch.relu(x) (à applique dans le forward)
- **Définir le taux de Dropout :** self.dropout1 = nn.Dropout(0.2) (dans le init)
- **Applique le Dropout à la couche « fcx » :** (dans le forward)

```
x = self.fcx(x)
x = torch.relu(x)
x = self.dropout1(x)
```

Pour aller plus loin :

- Appliquer les modèles *Deep Learning* de ce TP à la base de données vue lors du TP1 (9 *features*)
- Appliquer les modèles *Machine Learning* du TP1 à la base de données vue aujourd'hui (20 features)
- Analyser les résultats expérimentaux et tirer vos conclusions.