Apprentissage Profond (I-ILIA-029) – Travaux Pratiques

TP 04 : MLP et CNN pour la classification d'images

- **Partie I : réseau de neurones à une seule couche (MLP1) :**

Après avoir manipulé les différents outils et méthodes d'extraction de caractéristiques d'images, vous pouvez développer votre classifieur d'images. Nous vous proposons donc de travailler sur la classification d'images de chiffres manuscrits provenant de la base de données MNIST. Le but est de reconnaître la valeur du chiffre à partir de son image.

Figure 1: images de la base de données MNIST

- **Partie 0 : préparation des données et définition des paramètres d'entraînement :**

Nous vous proposons de télécharger le code de démarrage depuis Moodle «**TP4_I-ILIA-029_2025_Input.ipynb** » avant de le compléter en suivant les étapes suivantes :

- **Question 1 :** vérifier que le GPU est bien sélectionné ;
- **Instruction 1 :** installer la librairie Pytorch Lightning ;
- **Instruction 2 :** connecter votre notebook à la plateforme WanDB ;
- **Instruction 3 :** importer les librairies nécessaires ;
- **Instruction 4 :** définir les paramètres d'entrainement :
  - Nombre de classes : **10**
  - Batch size : **64**
  - Nombre d'époques : **20**
  - Accélérateur : **GPU**
- **Instruction 5 :** télécharger la base de données MNIST ;
- **Instruction 6 :** diviser les données et créer les tenseurs/dataloaders ;
- **Question 2 :** inspecter et afficher le format de vos données d'entrainement, validation et test ;
- **Instruction 7 :** afficher les dix premières images avec désignation de leurs labels ;
- **Question 3 :** analyser attentivement la classe « ***MNISTModel*** » avant de la compléter ;

▪ Partie 1 : développement d'un modèle MLP à une seule couche :

- **Instruction 8 :** définir l'architecture neuronale du premier modèle à 1 seule couche (Fig. 1) ;

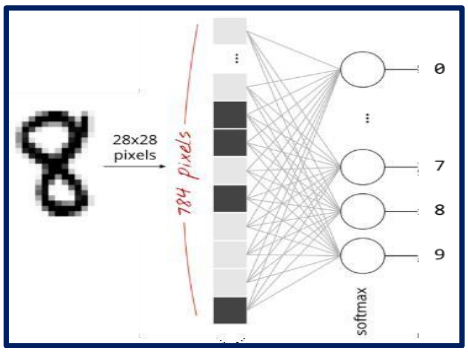

Figure 1 : réseau de neurones à une seule couche

Note 1: la fonction ***view*** permet de redimensionner l'image sous forme de vecteur « Flatten ». (Fig. 1).

- **Instruction 9 :** visualiser l'architecture de votre premier modèle avec la fonction ***summary()*** ;

➢ **Question 4 :** lancer l'entrainement avec ces paramètres :

- Optimiseur : SGD
- Pas d'apprentissage : lr=0.01

➢ **Question 4b :** vérifier et interpréter les résultats d'entraînement de votre 1er modèle MLP via Wandb ;

➢ **Question 5 :** évaluer votre premier modèle avec les données de test ;

➢ **Question 5b :** relancer l'entrainement avec plus d'époques et évaluer vos résultats ;

▪ Partie 2 : développement d'un modèle MLP à plusieurs couches avec Sigmoïd :

En vue d'améliorer les performances du modèle ci-dessus, implémenter un nouveau modèle à plusieurs couches cachées [1] selon l'architecture suivante :

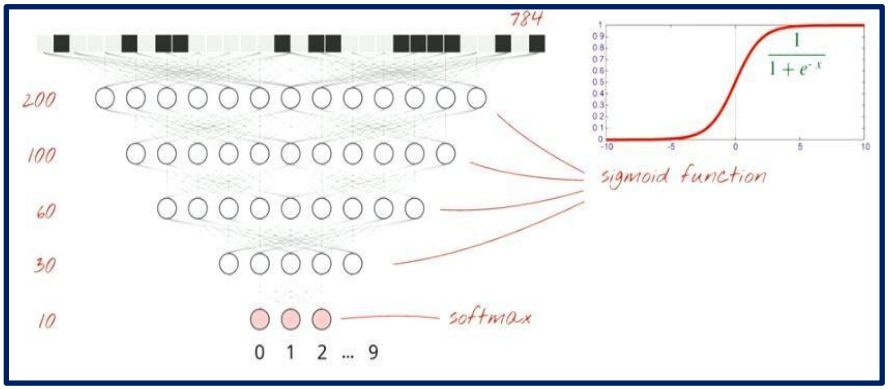

Figure 2: architecture du 2ème modèle MLP

- **Question 6 :** définir l'architecture neuronale à 5 couches avec Sigmoïd illustrée dans la Fig. 2 ;
- **Question 7 :** visualiser l'architecture de votre 2ème modèle avec la fonction summary()

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLP2                                     [1, 10]                   --
├─Linear: 1-1                            [1, 200]                  157,000
├─Linear: 1-2                            [1, 100]                  20,100
├─Linear: 1-3                            [1, 60]                   6,060
├─Linear: 1-4                            [1, 30]                   1,830
├─Linear: 1-5                            [1, 10]                   310
==========================================================================================
Total params: 185,300
Trainable params: 185,300
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.19
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.74
Estimated Total Size (MB): 0.75
==========================================================================================
```

- **Question 8 :** lancer l'entrainement avec ces paramètres :
  - Optimiseur : SGD
  - Pas d'apprentissage : lr=0.8
  - Epochs = 10
  - Model name = « ***mlp_deep_sigmoid*** »
- **Question 8b :** vérifier et interpréter les résultats d'entraînement de votre 2ème modèle MLP via Wandb ;
- **Question 9 :** évaluer votre deuxième modèle avec les données de test ;
- **Question 9b :** relancer l'entrainement avec plus d'époques et évaluer vos résultats ;

Partie 3 : développement d'un modèle MLP à plusieurs couches avec Relu :

- **Question 10 :** définir la même architecture (Fig.2) avec l'activation **Relu** pour les 4 premières couches
- **Question 11 :** visualiser l'architecture de votre 3ème modèle avec la fonction summary()
- **Question 12 :** lancer l'entrainement avec ces paramètres :
  - Optimiseur : SGD
  - Pas d'apprentissage : lr=0.0057
  - Model name = « ***mlp_deep_relu*** »
- **Question 12b :** vérifier et interpréter les résultats d'entraînement de votre 2ème modèle MLP via Wandb ;
- **Question 13 :** évaluer votre deuxième modèle avec les données de test ;
- **Question 13b :** relancer l'entraînement avec plus d'époques et évaluer vos résultats. Que constatez-vous?

Partie 4 : développement d'un modèle MLP à plusieurs couches avec Relu et Dropout :

- **Question 14 :** définir une nouvelle architecture avec les paramètres suivants :
  - Garder les 5 couches avec la fonction d'activation « Relu » pour les 4 premières ;
  - Appliquer le **Dropout** pour les **3 premières couches** avec un taux de **20%, 15% et 10%** respectivement.
- **Question 15 :** visualiser l'architecture de votre 4ème modèle avec la fonction summary()

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLP4                                     [1, 10]                   --
├─Linear: 1-1                            [1, 200]                  157,000
├─Dropout: 1-2                           [1, 200]                  --
├─Linear: 1-3                            [1, 100]                  20,100
├─Dropout: 1-4                           [1, 100]                  --
├─Linear: 1-5                            [1, 60]                   6,060
├─Dropout: 1-6                           [1, 60]                   --
├─Linear: 1-7                            [1, 30]                   1,830
├─Linear: 1-8                            [1, 10]                   310
==========================================================================================
Total params: 185,300
Trainable params: 185,300
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.19
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.74
Estimated Total Size (MB): 0.75
==========================================================================================
```

- **Question 16 :** lancer l'entrainement avec ces paramètres :
  - Optimiseur : SGD
  - Pas d'apprentissage : lr=0.01
  - Model name = « ***mlp_deep_relu_dropout*** »
- **Question 16b :** vérifier et interpréter les résultats d'entraînement de votre 2ème modèle MLP via Wandb ;
- **Question 17 :** évaluer votre deuxième modèle avec les données de test ;
- **Question 17b :** relancer l'entraînement avec plus d'époques et évaluer vos résultats. Que constatez-vous?

Partie 5 : développement d'un CNN avec une seule couche convolutionnelle :

- **Question 18 :** définir une nouvelle architecture avec les paramètres suivants :
  - Une couche convolutionnelle avec 32 filtres de taille (32,32) + Relu pour l'activation
  - Un Maxpooling avec une taille (2,2)
  - Une couche entièrement connectée de 128 neurones + Relu pour l'activation
  - Une couche entièrement connectée de 10 neurones
- **Question 19 :** visualiser l'architecture de votre 4ème modèle avec la fonction summary()

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNNModel                                 [32, 10]                  --
├─Conv2d: 1-1                            [32, 26, 26]              320
├─MaxPool2d: 1-2                         [32, 13, 13]              --
├─Linear: 1-3                            [32, 128]                 21,760
├─Linear: 1-4                            [32, 10]                  1,290
==========================================================================================
Total params: 23,370
Trainable params: 23,370
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 1.00
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.21
Params size (MB): 0.09
Estimated Total Size (MB): 0.31
==========================================================================================
```

- **Question 20 :** lancer l'entrainement avec ces paramètres :
  - Optimiseur : **Adam et** pas d'apprentissage : **par défaut**
  - Model name : **"*cnn_one_conv*"**
  - Epochs : **20**
- **Question 20b :** vérifier et interpréter les résultats d'entraînement de votre 1er modèle CNN via Wandb ;
- **Question 21 :** évaluer votre deuxième modèle avec les données de test ;

Partie 6 : développement d'un CNN avec deux couches convolutionnelles :

- **Question 22 :** définir une nouvelle architecture avec les paramètres suivants :
  - Une couche convolutionnelle avec 32 filtres de taille (32,32) + Relu pour l'activation + Dropout (20%)
  - Une couche convolutionnelle avec 32 filtres de taille (32,32) + Relu pour l'activation + Dropout (20%)
  - Un Maxpooling avec une taille (2,2)
  - Une couche entièrement connectée de 128 neurones + Relu pour l'activation + Dropout (20%)
  - Une couche entièrement connectée de 10 neurones
- **Question 23 :** visualiser l'architecture de votre 4ème modèle avec la fonction summary()
- **Question 24 :** lancer l'entrainement avec ces paramètres :
  - Optimiseur : Adam et pas d'apprentissage par défaut
  - Model name : **"*cnn_two_conv*"**
  - Epochs : **20**
- **Question 24b :** vérifier et interpréter les résultats d'entraînement de votre 2ème modèle MLP via Wandb ;
- **Question 25 :** évaluer votre deuxième modèle avec les données de test ;

Note 2 : en Pytorch, si on utilise leur fonction de perte « **F.cross_entropy() »**, il n'est pas nécessaire de rajouter la fonction d'activation « ***softmax*** » dans le modèle car il est déjà intégré dans son implémentation, voir source.

▪ Partie 7 (Facultatif) : Utilisation d'Optuna pour optimiser les hyperparamètres :

Pour réaliser les parties facultatives, veuillez utiliser ce notebook : « ***Hyperparameters_optimisation.ipynb*** »

Code de démarrage pour la partie facultative :

L'objectif de cette partie est de vous familiariser avec l'outil Optuna, une bibliothèque d'optimisation bayésienne qui permet d'automatiser la recherche des hyperparamètres. Optuna explore de manière optimale l'espace des hyperparamètres pour trouver les meilleures configurations.

- **Question 26 :** installer l'outil ***Optuna ;***
- **Question 27 :** analyser le code qui définit l'espace de recherche ;
- **Question 28 :** compléter le code de la fonction objective et lancer la fonction ;
- **Question 29 :** changer l'espace de recherche avec votre choix ;

• Partie 8 (Facultatif) : Learning rate finder pour optimiser le pas d'apprentissage :

Dans cette partie, l'objectif est d'utiliser une méthode automatique pour trouver un pas d'apprentissage « Learning Rate Finder ». Nous utiliserons un tuner de PyTorch Lightning qui détermine le Learning rate optimal avec la fonction **lr_find()**.

- **Question 30 :** reprendre le code d'entrainement utilisé pour le modèle MLP profond avec dropout MLP3
- **Question 31 :** créer une instance de **pl.tuner.Tuner** qui prend comme paramètre votre trainer
- **Question 32 :** utiliser la fonction lr_find comme suit :

```
tuner = pl.tuner.Tuner(trainer)
lr_finder_results = tuner.lr_find(
model,
train_dataloaders=train_loader,
val_dataloaders=val_loader
)
```

- **Question 33:** relancer l'entraînement et analyser les résultats sur wandb.