Ecole National Polytechnique ENP **Spécialité : DSIA** **Enseignant : Oussama ARKI**	**2°Année du 2°Cycle** **Module : Apprentissage Automatique(AA)** **Année 2022-2023**

TP 05 : Tensorflow et RN

(Shoe vs Sandal vs Boot Image Classification)

Introduction

Un tenseur est un tableau à plusieurs dimensions, qui généralise la notion de matrice et de vecteur et permet de faire les calculs dans les réseaux de neurones.

Qu'est-ce qu'un tenseur?

Un tenseur est un tableau de nombres à plusieurs dimensions. Voici des exemples de tenseurs:
• un vecteur *V* est un tenseur de dimension 1 (c'est un tableau à une seule dimension),
• une matrice *M* est un tenseur de dimension 2 (c'est un tableau à deux dimensions),
• un 3-tenseur *T* est un tableau à 3 dimensions.

Tensorflow

TensorFlow est un outil open source d'apprentissage automatique développé par Google. Le code source a été ouvert le 9 novembre 2015 par Google et publié sous licence Apache. Elle implémente des méthodes d'apprentissage automatique basées sur le principe des réseaux de neurones profonds (deep learning). Une API Python est disponible.
La puissance de TensorFlow est de faire appel à la différenciation automatique pour calculer le gradient.
Nous pouvons l'exploiter directement dans un programme rédigé en Python ou bien passer par **Keras**
Une de ses caractéristiques est d'être adapté à des architectures matérielles dédiées au calcul **(GPU).**

À quoi sert un GPU ?

Le but d'un GPU est d'abord de gérer et d'optimiser tous les rendus graphiques. Avoir une unité de calcul réservée aux graphismes permet de libérer de la puissance pour le CPU. C'est notamment une pièce essentielle de tout PC dédié aux jeux vidéo. Aujourd'hui, sa puissance et sa flexibilité d'utilisation lui permettent d'avoir d'autres utilités. Certains logiciels spécifiques utilisent d'ailleurs la carte graphique en priorité. Le GPU est notamment utilisé pour :

- l'animation 3D ;
- les jeux vidéo (2D, 3D, VR) ;
- le montage vidéo ;
- la création d'effets visuels pour le cinéma ;
- **l'apprentissage automatique (*machine learning*) ;**
- l'intelligence artificielle ;
- le minage de cryptomonnaies.

Le calcul par le GPU, c'est quoi ?

Le calcul par le GPU permet de paralléliser les tâches et d'offrir un maximum de performances dans de nombreuses applications : le GPU accélère les portions de code les plus lourdes en ressources de calcul, le reste de l'application restant affecté au CPU. Les applications des utilisateurs s'exécutent ainsi bien plus rapidement.

Keras (https://keras.io/)
Keras est une librairie Python qui encapsule l'accès aux fonctions proposées par plusieurs librairies de machine learning, en particulier Tensorflow. De fait, Keras n'implémente pas nativement les méthodes. Elle sert d'interface avec Tensorflow simplement.

Pourquoi une couche supplémentaire ?

Parce qu'elle nous facilite grandement l'utilisation de Tensorflow en proposant des fonctions et procédures relativement simples à mettre en œuvre. L'accès aux fonctionnalités de Tensorflow devenant transparentes

Le module keras a été élaboré pour pouvoir utiliser Tensorflow plus simplement

Installer TensorFlow avec pip (via Anaconda Prompt (miniconda3))
La page du tutorial : https://www.tensorflow.org/install/pip#system-install

Packages TensorFlow 2 disponibles

- **tensorflow** : dernière version stable pour les processeurs et les GPU *(Ubuntu et Windows)*
- **tf-nightly** : version de développement *(instable)*. Les packages pour Ubuntu et Windows incluent la compatibilité avec les GPU.

Créer un environnement virtuel (recommandé)

Les environnements virtuels Python permettent d'isoler l'installation d'un package du système.

```
python -m venv --system-site-packages .\venv
```

Activez l'environnement virtuel :

```
.\venv\Scripts\activate
```

Installez des packages dans un environnement virtuel sans modifier la configuration du système hôte. Pour ce faire, commencez par mettre à niveau `pip` :

```
pip install --upgrade pip
```

Pour quitter l'environnement virtuel :

```
deactivate  # don't exit until you're done using TensorFlow
```

Installer le package pip TensorFlow :
Deux options pour l'installation :
1-Installation-dans-lenvironnement-virtuel :

```
pip install --upgrade tensorflow
```

2-Installation-dans-le-système-dexploitation

```
pip3 install --user --upgrade tensorflow  # install in $HOME
```

Vérifiez l'installation :
python -c "import tensorflow as tf;print(tf.reduce_sum(tf.random.normal([1000, 1000])))"

Comment créer un réseau de neurones (perceptron multi-couches) avec keras ?

1-Importation

On commence par l'importation des packages :

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
```

Architecture du réseau. Nous importons les classes **Sequential** et **Dense** pour définir notre modèle et son architecture.

Sequential parce que les couches de neurones vont être ajoutées séquentiellement.

La classe Sequential est une structure, initialement vide, qui permet de définir un empilement de couches de neurones

Dense : parce que tous les neurones de couche précédente seront connectés à tous les neurones de la couche suivante.

2-Instanciation du modèle

On commence par créer une instance de notre modèle de réseau de neurones

```
modele_RN=tf.keras.models.Sequential() #un reseau de neuron sequentiel
```

3-l'ajout des couches

Apres avoir créer le modèle, on peut ajouter les couches les unes après les autres

Exemple : pour ajouter une couche de 32 neurones et la fonction sigmoïde comme activation

```
modele_RN.add(tf.keras.layers.Dense(32,activation='sigmoid'))
```

Exemple : pour ajouter une couche de sortie de 10 neurones avec softmax comme activation

```
modele_RN.add(tf.keras.layers.Dense(10,activation='softmax'))
```

remarque : il faut préciser la forme de l'entrée (input shape) de notre réseau de neurones.

```
modele_RN.predict(x_train[0:1]) # pour ce faire on utilise la prédiction sur un exemple !!
```

4-la compilation

Apres avoir définir le réseau, on passe à la phase de compilation

```
modele_RN.compile(
loss="sparse_categorical_crossentropy", #la fonction de perte à optimiser (cas classification)
optimizer="sgd", # l'algo d'optimisation utilisée (alternative de descente de gradient)
metrics=["accuracy"] # la métrique utilisée pour mesurer la qualité de la modélisation
)
```

Available optimizers : https://keras.io/api/optimizers/

Losses : https://keras.io/api/losses/

Metrics : https://keras.io/api/metrics/

Remarque :

modele_RN.summary() : cette fonction affiche un résumé sur le réseau

5-Entrainement du réseau

history=modele_RN.fit(x_train,y_train,epochs=50,batch_size=512,validation_split=0.2)

epochs est le nombre maximum d'itérations ;

batch_size correspond aux nombre d'observations que l'on fait passer avant de remettre à jour les poids synaptiques.

validation_split : la validation set utilisée pour tester l'Accuracy du model après chaque époque

Une fois l'apprentissage finalisé, nous pouvons afficher les poids estimés :

modele_RN.get_weights()

Remarque : la variable *history* contient l'historique de l'apprentissage

Exemple : obtenir l'évolution de la perte et de l'Accuracy pour train-set et validation-set

```
loss_curve=history.history["loss"]
accuracy_curve=history.history["accuracy"]
loss_val_curve=history.history["val_loss"]
accuracy_val_curve=history.history["val_accuracy"]
```

Un code pour visualiser le resultat avec matplotlib

```
plt.plot(loss_curve,label="Train")
plt.plot(loss_val_curve,label="Val")
plt.title('loss')
plt.legend()
plt.show()
plt.plot(accuracy_curve,label="Train")
plt.title('accuracy')
plt.plot(accuracy_val_curve,label="Val")
plt.legend()
plt.show()
```

6-Evaluation du modèle

Pour la prédiction :

```
predictions=modele_RN.predict(x_test)
```

Remarque : dans le cas d'un réseau avec softmax comme fonction d'activation pour la couche de sortie, pour chaque observation du test, le réseau associe un vecteur qui contient autant de résultats que le nombre de neurones dans la couche de sortie, donc pour avoir la classe prédite on utilise la fonction argmax() pour avoir l'indice de la probabilité maximale (classe):

```
predictions_class=[]
for i in range (0,taille_test):
    predictions_class.append(predictions[i].argmax())
```

Maintenant on peut évaluer notre model

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

Exemple :

```
matrice_confusion=confusion_matrix(y_test,predictions_class)
sns.heatmap(matrice_confusion, annot=True, fmt='g', cbar=False, cmap='BuPu')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('RN Confusion Matrix')
plt.show()
```

TP05 : Shoe vs Sandal vs Boot Image Dataset Classification

Objectif :

Créer un réseau de neurones, qui permet de classifier les images de Shoe vs Sandal vs Boot Image Dataset en 3 classes ['Boot', 'Sandal', 'Shoe']

Contexte

Cet ensemble de données d'images Shoe vs Sandal vs Boot contient 15 000 images de chaussures, de sandales et de bottes. 5000 images pour chaque catégorie. Les images ont une résolution de 136x102 pixels dans le modèle de couleur RVB.

Il y a trois classes :.

- Shoe
- Sandal
- Boot

Cet ensemble de données est idéal pour effectuer une classification multiclasse avec des réseaux de neurones profonds comme les **CNN**.