TD1 – Kppv et réseaux de neurones pour la classification d'images

L'objectif de ce TD est d'écrire en langage Python un programme complet de classification d'images. Deux modèles de classification seront successivement développés et expérimentés : les k-plus-proches-voisins et les réseaux de neurones. Le module numpy sera utilisé pour la manipulation des matrices et le module scikit-image[1] pour la manipulation des images.

1. Présentations des données pour les expérimentations

La base d'images qui sera utilisée pour les expérimentations est **CIFAR-10** qui consiste en 60 000 images en couleur de taille 32x32 réparties en 10 classes (avion, voiture, oiseau, chat, ...).
Cette base peut être obtenue à l'adresse *https://www.cs.toronto.edu/~kriz/cifar.html* où sont également données les indications pour lire les données.

2. Système de classification à base de l'algorithme des kppv

L'objectif est de développer un classifieur de type kppv. Sa mise en œuvre doit être optimisée en utilisant une représentation vectorielle des données : seuls des calculs matriciels doivent être réalisés lors de la phase de calcul des distances, et donc cette fonction ne doit pas comporter de boucle (à l'exception de celle portant sur les batchs de test si l'évaluation est réalisée de cette manière). Le module Python numpy devra être utilisé.

2.1 Développement du classifieur

- écrire la fonction *lecture_cifar* prenant en argument le chemin du répertoire contenant les données, et renvoyant une matrice X de taille NxD où N correspond au nombre de données disponibles, et D à la dimension de ces données (nombre de valeurs numériques décrivant les données), ainsi qu'un vecteur Y de taille N dont les valeurs correspondent au code de la classe de la donnée de même indice dans X. X et Y devront être des objets numpy. Bien penser à convertir en float32 les valeurs de X (par défaut entiers non signés sur 1 octet).
- écrire une fonction *decoupage_donnees* de découpage des données en ensemble d'apprentissage / test, prenant en argument les matrices X et Y, et renvoyant les données d'apprentissage et de test : Xapp, Yapp, Xtest, Ytest. Un découpage aléatoire en 80% des données pour le test et 20% des données pour l'apprentissage pourra par exemple être considéré.
- écrire une fonction *kppv_distances* prenant en argument Xtest et Xapp et renvoyant la matrice des distances Dist entre toutes les données de l'ensemble de test par rapport à toutes les données de l'ensemble d'apprentissage. La distance euclidienne l2 sera utilisée pour évaluer la distance entre les données. Le calcul

[1] http://scikit-image.org

doit se faire uniquement à l'aide de manipulation de matrices (aucune boucle). *Indice* : $(a-b)^2 = a^2 + b^2 - 2\,ab$
Remarques importantes : si l'occupation mémoire est trop importante, plusieurs manipulations peuvent être envisagées : utilisation pour les données du type float32 au lieu du type float64 (par défaut), utilisation de proportions différentes entre ensemble d'apprentissage / ensemble de test (90% / 10% par exemple), utilisation de plusieurs batchs pour le calcul des distances des données de test (boucle portant sur les batchs et non sur les données individuelles). A noter que dans un premier temps, les données images brutes sont utilisées et donc la dimension de représentation des données est très grande. Ce problème peut être évité en utilisant des descripteurs images dont la dimension sera plus petite.
De plus, si le temps d'exécution est trop élevé en utilisant la totalité des données, des expérimentations préliminaires sur un sous ensemble peuvent être envisagées, avant de faire l'évaluation complète sur l'ensemble complet des données de test.
- écrire une fonction *kppv_predict* prenant en argument Dist, Yapp et K le nombre de voisins et renvoyant le vecteur Ypred des classes prédites pour les éléments de Xtest.
- écrire un fonction *evaluation_classifieur* prenant en argument Ytest et Ypred et renvoyant le taux de classification (Accuracy).

2.2 Expérimentations

Des expérimentations devront être réalisées en étudiant les variations suivantes :
- influence du nombre de voisins K sur l'efficacité du classifieur évaluée à l'aide du taux de classification (Accuracy).
- représentation des images par des descripteurs (LBP, HOG, ...) en utilisant le module scikit-image.
- utilisation de la validation croisée à N répertoires (N-fold cross-validation) plutôt qu'un découpage en deux sous-ensembles fixes d'apprentissage et de test.

3. Système de classification à base de réseaux de neurones

3.1 Développement du classifieur

L'objectif ici est de développer un classifieur s'appuyant sur un réseau de neurones de type perceptron (multi-couches).
Les fonctions de lecture et découpage des données réalisées précédemment pourront être réutilisées (éventuellement adaptées) pour ce classifieur.
Ci-dessous, un code Python permettant une passe avant (forward) et le calcul de la fonction de perte de type MSE (Mean Square Error) dans un réseau contenant une couche cachée et utilisant la fonction sigmoïde comme fonction d'activation :

```
import numpy as np

np.random.seed(1) # pour que l'exécution soit déterministe

##########################
# Génération des données #
##########################

# N est le nombre de données d'entrée
# D_in est la dimension des données d'entrée
# D_h le nombre de neurones de la couche cachée
# D_out est la dimension de sortie (nombre de neurones de la couche de sortie)
N, D_in, D_h, D_out = 30, 2, 10, 3
```

```
# Création d'une matrice d'entrée X et de sortie Y avec des valeurs aléatoires
X = np.random.random((N, D_in))
Y = np.random.random((N, D_out))

# Initialisation aléatoire des poids du réseau
W1 = 2 * np.random.random((D_in, D_h)) - 1
b1 = np.zeros((1,D_h))
W2 = 2 * np.random.random((D_h, D_out)) - 1
b2 = np.zeros((1,D_out))

##################################################
# Passe avant : calcul de la sortie prédite Y_pred #
##################################################
I1 = X.dot(W1) + b1 # Potentiel d'entrée de la couche cachée
O1 = 1/(1+np.exp(-I1)) # Sortie de la couche cachée (fonction d'activation de type sigmoïde)
I2 = O1.dot(W2) + b2 # Potentiel d'entrée de la couche de sortie
O2 = 1/(1+np.exp(-I2)) # Sortie de la couche de sortie (fonction d'activation de type
sigmoïde)
Y_pred = O2 # Les valeurs prédites sont les sorties de la couche de sortie

######################################################
# Calcul et affichage de la fonction perte de type MSE #
######################################################
loss = np.square(Y_pred - Y).sum() / 2
print(loss)
```

Compléter ce code pour réaliser la phase de descente du gradient afin de permettre l'apprentissage du réseau.
Adapter ensuite ce réseau pour permettre la classification d'images. Les expérimentations pourront être faites en utilisant la base CIFAR-10.

2.2 Expérimentations

Des expérimentations devront être réalisées en étudiant les variations suivantes :
- faire varier le nombre de neurones sur les couches cachées
- ajout d'une couche cachée
- comparaison des résultats en utilisant une fonction d'activation de type relu et une fonction de perte de type cross-entropy
- ajout d'un terme de régularisation dans la fonction de perte
- faire varier le taux d'apprentissage (learning rate) pour la mise à jour des poids
- utilisation d'un mini-batch pour la descente du gradient
- et comme pour les Kppv, représentation des données par des descripteurs et évaluation par validation croisée.

4. Travail à rendre

Ce travail (kppv+réseaux de neurones) peut être réalisé par binôme, et un compte-rendu numérique devra être produit. Il devra contenir notamment les explications concernant le principe des fonctions que vous avez programmées ainsi que leur code commenté. Les expérimentations que vous aurez réalisées devront être décrites et les résultats également commentés. Le programme complet (ensemble des fichiers .py) devra être joint au fichier pdf au sein d'une archive « .zip » qui devra être déposée sur le site « Moodle» pour le mardi 12 novembre 2019.