Introduction à l'apprentissage profond
TP2
Safia Boudra & Itheri Yahiaoui

Utilisation de modèle Keras avec scikit-learn

On peut utiliser un modèle Keras avec scikit-learn[1] via les fonctions `KerasClassifier()` pour un problème de classification et `KerasRegressor()` pour un problème de régression linéaire. Dans la suite de ce TP on va de pencher sur l'utilisation de `KerasClassifier()`.

Une utilisation typique suit l'exemple suivant :

```
from keras.wrappers.scikit_learn import KerasClassifier

# Fonction pour creer le modele Keras
def define_model():
    # definir le modele
    # compiler le modele
    return model

# utiliser le modele avec KerasClassifier
model = KerasClassifier(build_fn=define_model)
```

Le modèle Keras est assigné à l'attribut `build_fn` de la fonction `KerasClassifier()`.

[1] https://keras.io/scikit-learn-api/

Evaluation de modèle Keras avec la validation croisée

L'évaluation du modèle avec la validation croisée consiste à définir les *kfold* avec la fonction `StratifiedKFold()` et « boucler » sur chaque fold afin d'évaluer le modèle pour chaque partition des données.

```
from sklearn.model_selection import StratifiedKFold
import numpy as np
# determiner un nbr de seed pour reproduire l'execution
seed = 7
np.random.seed(seed)
# 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
cvscores = []
for train, test in kfold.split(X, Y):
    model = Sequential()
    model.add(Dense(12, input_dim=8, kernel_initializer='uniform',
activation='relu'))
    model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(X[train], Y[train], epochs=150, batch_size=10, verbose=0)
    scores = model.evaluate(X[test], Y[test], verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
    cvscores.append(scores[1] * 100)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

Dans l'exemple de ce TP2, on va utiliser la fonction `cross_val_score() :`

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

# charger la base de données

# recuperer la matrice de donnees dans X et les labels des classes dans Y

# Fonction pour creer le modele Keras
def define_model():
    # definir le modele
    # compiler le modele
    return model

# utiliser le modele avec KerasClassifier
model = KerasClassifier(build_fn=define_model, epochs=150, batch_size=10, ver-
bose=0)
# validation croisee avec 10-fold
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
# appliquer la validation croisee sur le modele
results = cross_val_score(model, X, Y, cv=kfold)
# afficher les moyenne des scores sur les 10-fold
print(results.mean())
```

Exercices

1. Ecrire le code python avec keras et sckit-learn d'un réseau de neurone pour une classification binaire sur la base ***diabetes***[2] en utilisant la validation croisée.

2. Ecrire le code python avec Keras et sckit-learn d'un réseau de neurone pour une classification multi-classe sur la base ***seeds***[3] en utilisant la validation croisée.

[2] https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv
[3] https://archive.ics.uci.edu/ml/datasets/seeds

Evaluation des paramètres du modèle Keras avec une grille de recherche (Grid Search)

Dans cette partie, on se penche sur l'évaluation des paramètres du modèle Keras avec une grille de recherche afin de trouver la combinaison optimale qui optimise des performances du réseau de neurones.

L'optimisation avec une grille de recherche est définie via la classe `GridSearchCV` de scikit-learn, où un objet de cette classe reçoit les trois attributs suivants :

`estimator`	Modèle Keras avec scikit-learn
`param_grid`	Un `dict` avec les noms de paramètres (string) comme 'key', et une liste des valeurs à évaluer.
`cv`	Utilization de validation croisée.
`n_jobs`	-1 Pour paralleliser la grille de recherche

Par la suite, les attributs `best_score_` et `best_params_` parmettent de retrouver la valeur optimale du paramètre évalué avec le meilleur scores associé.

```
from sklearn.model_selection import GridSearchCV
# exemple d'evaluation du nombre optimal d'epoch
grid_param = dict(epochs=[10,20,30])
# application d'une grille de recherche avec une validation croisée de 3-fold
grid = GridSearchCV(estimator=model, param_grid=grid_param, n_jobs=-1, cv=3)
# appliquer sur la base de données
grid_result = grid.fit(X, Y)
# afficher les results
print("Best: %f with %s" % (grid_result.best_score_, grid_result.best_params_))
```

En partant du principe de la grille de recherche on va pouvoir évaluer les paramètres suivants d'un réseau de neurones :

1. Méthode d'initialisation des poids du réseau.
2. Les différentes fonctions d'activation.
3. Nombre de neurones dans la couche cachée.
4. Algorithme d'optimisation pour le calcul du gradient stochastique.
5. Taux d'apprentissage et momentum.
6. Taille du batch et nombre d'epoch.

Les parties suivantes montrent chaque cas d'optimisation des paramètres d'un réseau de neurones.

Méthode d'initialisation des poids du réseau[4]

```
# fonction pour creer le modele keras
def define_model(init_mode='uniform'):
    # creer le modele
    model = Sequential()
    # passer la valeur de l'argument 'init_mode' à l'attribut 'kernel_initializer'
de Dense
    # compiler le modele
    return model

# charger la base de données

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

# creer le modele
model = KerasClassifier(build_fn=define_model, epochs=100, batch_size=10,
verbose=0)
# parametres de la grille de recherche
init_modes = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal',
'glorot_uniform', 'he_normal',
              'he_uniform']
grid_param = dict(init_mode=init_modes)
grid = GridSearchCV(estimator=model, param_grid=grid_param, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# afficher les resultats
print("Best: %f with %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
# à l'execution
Best: 0.727865 with {'init_mode': 'uniform'}
mean (+/- std) = 0.727865 (0.009744) with: {'init_mode': 'uniform'}
mean (+/- std) = 0.679688 (0.035943) with: {'init_mode': 'lecun_uniform'}
mean (+/- std) = 0.717448 (0.009744) with: {'init_mode': 'normal'}
```

[4] https://keras.io/initializers/

Les différentes fonctions d'activation[5]

```
# fonction pour creer le modele keras
def define_model(activation='relu'):
    # creer le modele
    # passer la valeur de l'argument 'activation' à l'attribut 'activation' de
Dense
    # compiler le modele
    return model

# charger la base de données

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

# creer le modele
model = KerasClassifier(build_fn=define_model, epochs=100, batch_size=10,
verbose=0)
# parametres de la grille de recherche
activations = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid',
'hard_sigmoid', 'linear']
grid_param = dict(activation=activations)
grid = GridSearchCV(estimator=model, param_grid=grid_param, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# afficher les resultats
print("Best: %f with %s" % (grid_result.best_score_, grid_result.best_params_)
```

```
# à l'execution
Best: 0.727865 with {'activation': 'softplus'}
mean (+/- std) = 0.643229 (0.012075) with: {'activation': 'softmax'}
mean (+/- std) = 0.727865 (0.027866) with: {'activation': 'softplus'}
mean (+/- std) = 0.678385 (0.025780) with: {'activation': 'softsign'}
```

[5] https://keras.io/activations/

Nombre de neurones dans les couches cachées.

```
# fonction pour creer le modele keras
def define_model(neurons=1):
    # creer le modele
    # passer la valeur de l'argument 'neurone' au premier attribut de Dense pour
determiner le nbr de neurones
    # dans la couche cachée
    # compiler le modele
    return model

# charger la base de données

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

# creer le modele
model = KerasClassifier(build_fn=define_model, epochs=100, batch_size=10,
verbose=0)
# parametres de la grille de recherche
neurons = [1, 5, 10, 15, 20, 25, 30]
grid_param = dict(neurons=neurons)
grid = GridSearchCV(estimator=model, param_grid=grid_param, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# afficher les resultats
print("Best: %f with %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
# à l'execution
Best: 0.723958 with {'neurons': 5}
mean (+/- std) = 0.701823 (0.020505) with: {'neurons': 1}
mean (+/- std) = 0.723958 (0.026748) with: {'neurons': 5}
mean (+/- std) = 0.713542 (0.020505) with: {'neurons': 10}
```

Algorithme d'optimisation pour le calcul du gradient stochastique[6]

```
# fonction pour creer le modele keras
def define_model(optimizer='adam'):
    # creer le modele
    # compiler le modele
    # passer la valeur de l'argument 'optimizer' à l'attribut 'optimizer' de
compile() pour
    # la methode de calcul du SGD
    return model

# charger la base de données

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

# creer le modele
model = KerasClassifier(build_fn=define_model, epochs=100, batch_size=10,
verbose=0)
# parametres de la grille de recherche
optimizers = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
grid_param = dict(optimizer=optimizers)
grid = GridSearchCV(estimator=model, param_grid=grid_param, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# afficher les resultats
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
# à l'execution
Best: 0.692708 using {'optimizer': 'Nadam'}
mean (+/- std) = 0.651042 (0.024774) with: {'optimizer': 'SGD'}
mean (+/- std) = 0.648438 (0.061516) with: {'optimizer': 'RMSprop'}
mean (+/- std) = 0.675781 (0.016877) with: {'optimizer': 'Adagrad'}
```

[6] https://keras.io/optimizers/

Taux d'apprentissage et Momentum.

```
from keras.optimizers import SGD

# fonction pour creer le modele keras
def define_model(learn_rate=0.01, momentum=0):
    # creer le modele

    optimizer = SGD(lr=learn_rate, momentum=momentum)
    # passer la valeur de 'optimizer' à l'attribut 'optimizer' de compile() pour
    # la methode de calcul du SGD
    # compiler le modele
    return model

# charger la base de données

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

# creer le modele
model = KerasClassifier(build_fn=define_model, epochs=100, batch_size=10,
verbose=0)
# parametres de la grille de recherche
learn_rates = [0.001, 0.01, 0.1, 0.2, 0.3]
momentums = [0.0, 0.2, 0.4, 0.6, 0.8, 0.9]
grid_param = dict(learn_rate=learn_rates, momentum=momentums)
grid = GridSearchCV(estimator=model, param_grid=grid_param, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# afficher les resultats
print("Best: %f with %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
# à l'execution
Best: 0.677083 with {'learn_rate': 0.001, 'momentum': 0.6}
mean (+/- std) = 0.457031 (0.148335) with: {'learn_rate': 0.001, 'momentum': 0.0}
mean (+/- std) = 0.657552 (0.041626) with: {'learn_rate': 0.001, 'momentum': 0.2}
mean (+/- std) = 0.567708 (0.130741) with: {'learn_rate': 0.001, 'momentum': 0.4}
```

Taille du batch et nombre d'epoch

```
# fonction pour creer le modele keras
def define_model():
    # creer le modele
    # compiler le modele
    return model

# charger la base de données

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

# creer le modele
model = KerasClassifier(build_fn=define_model, verbose=0)
# parametres de la grille de recherche
batch_sizes = [10, 20, 40, 60, 80, 100]
epoch_nbrs = [25, 50, 75, 100]
grid_param = dict(batch_size=batch_sizes, epochs=epoch_nbrs)
grid = GridSearchCV(estimator=model, param_grid=grid_param, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# afficher les resultats
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
# à l'execution
Best: 0.682292 using {'batch_size': 20, 'epochs': 100}
mean (+/- std) = 0.605469 (0.019401) with: {'batch_size': 10, 'epochs': 25}
mean (+/- std) = 0.670573 (0.010253) with: {'batch_size': 10, 'epochs': 50}
mean (+/- std) = 0.673177 (0.003683) with: {'batch_size': 10, 'epochs': 75}
```

Exercices

1. Reprenez les parties de code en les complétant sur la base ***diabetes***[7]
2. Construire vers la fin un réseau de neurone avec l'ensemble des paramètres optimaux obtenus lors de la recherche par grille.
3. Evaluer les paramètres d'un réseau de neurones pour la classification multi-classes sur la base ***seeds***[8].
4. Construire le modèle final sur la base ***seeds*** avec l'ensemble de paramètres optimaux obtenus lors de l'évaluation.

[7] https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv
[8] https://archive.ics.uci.edu/ml/datasets/seeds