Rapport DeepLearning

Multilayer perceptron [20-01-23] :

Dans cette première partie, nous avons utilisé un modèle de Multilayer perceptron (MLP) pour résoudre un problème de classification de reconnaissance de l'accent. Les données proviennent d'une étude sur la reconnaissance de l'accent menée à l'Institute of Technology de Rochester en 2015. Il s'agit de mots anglais simples lus par des locuteurs de six pays différents pour la détection et la reconnaissance de l'accent.
Notre jeu de données est un tableau de 329 lignes et 13 colonnes: une colonne Langage qui est notre Target ainsi que 12 variables de X1-X12 qui représentent des coefficients de fréquence obtenue en appliquant la méthode (MFCC) au signal.

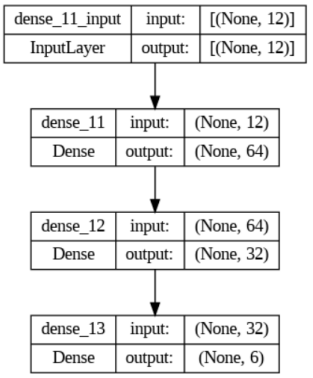
Nous avons procéder de la manière suivante :

Chargement des données :
Les données sont chargées à partir d'un fichier CSV en utilisant la bibliothèque pandas. Les données sont ensuite divisées en entrées (X) et cibles (y).

Encodage des étiquettes cibles :
Les étiquettes cibles, les noms des langues sont encodées en valeurs numériques en utilisant la classe Label Encoder de scikit-learn.

Division des données en ensemble d'entraînement et de test :
Les données sont divisées aléatoirement en un ensemble d'entraînement et un ensemble de test en utilisant la fonction train_test_split de scikit-learn.

Construction du modèle :
Le modèle est construit en utilisant la classe Sequential de Tensorflow. Il comporte 3 couches : une couche d'entrée (input layer), une couche cachée (hidden layer) et une couche de sortie (output layer). La première et la deuxième couches utilisent l'activation "relu" (rectified linear unit) qui est une fonction d'activation couramment utilisée dans les couches cachées d'un réseau de neurones pour introduire de la non-linéarité dans le modèle. La dernière couche utilise l'activation "softmax" qui est une fonction d'activation couramment utilisée dans les couches de sort.



Compilation du modèle : Le modèle est compilé en spécifiant l'optimiseur (adam), la fonction de perte (sparse_categorical_crossentropy) et les métriques (accuracy) à utiliser.

Entraînement du modèle : nous avons entraîné notre modèle sur l'ensemble d'entraînement en utilisant la méthode fit de Keras et en utilisant une boucle pour essayer différentes valeurs d'époques allant de 1 à 60. Nous avons comparé les accuracy obtenues pour chaque valeur d'époques et avons obtenu la meilleure accuracy de 0.81818 à l'époque n°16.

Résultat

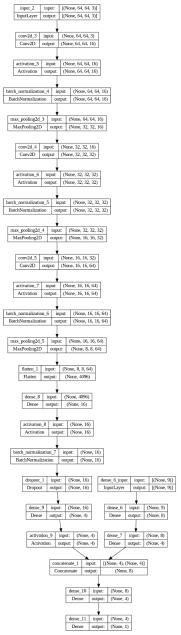
```
Epoch 59/59
9/9 [==============================] - 0s 4ms/step - loss: 9.4461e-07 - accuracy: 1.0000
3/3 [==============================] - 0s 5ms/step - loss: 1.2288 - accuracy: 0.8182
Best accuracy: 0.8181818127632141 at epochs: 16
3/3 [==============================] - 0s 5ms/step - loss: 1.2288 - accuracy: 0.8182
```

Aperçu du code :

```python
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.callbacks import TensorBoard
import datetime

# Charger les données
data = pd.read_csv("data_set.csv")
X = data.drop(["language"], axis=1)
y = data["language"]

# Encoder les étiquettes cibles
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# Spliter les données en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Construire le modèle
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, input_dim=12, activation="relu"))
model.add(tf.keras.layers.Dense(32, activation="relu"))
model.add(tf.keras.layers.Dense(6, activation="softmax"))

# Compiler le modèle
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

# Enregistrer les données de suivi dans un répertoire de journal pour TensorBoard
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)

# Entraîner le modèle
best_acc = 0
best_epochs = 0
for epochs in range(1, 60):
    model.fit(X_train, y_train, epochs=epochs, batch_size=32)
```

```
    test_loss, test_acc = model.evaluate(X_test, y_test)
    if test_acc > best_acc:
        best_acc = test_acc
        best_epochs = epochs
print("Best accuracy:", best_acc, "at epochs:", best_epochs)


# Dessiner le modèle
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```

Fusion : *[23-01-2023]*



Le script proposé pour appliquer la méthode de fusion charge des données d'attributs de maison à partir d'un fichier CSV. Il effectue un prétraitement sur les données en enlevant les codes postaux ayant moins de 25 points de données, en effectuant une normalisation min-max sur les caractéristiques continues et une codification binaire sur les caractéristiques catégorielles. Le code charge également les images de maison à l'aide de la bibliothèque OpenCV, les redimensionne en 32x32 et les combine en une seule image de sortie de 64x64x3. Le résultat de ce code est les données d'attributs de maison prétraitées et les images de maison combinées.

Le travail effectué en deux partie :

L'omission d'une variable :
En exécutant la prédiction sur les données de test en gardant toute les variables, nous obtenons le résultat suivant :

```
[INFO] predicting house prices...
3/3 [==============================] - 0s 4ms/step
[INFO] avg. house price: $533,388.27, std house price: $493,403.08
[INFO] mean: 71.34%, std: 22.19%
```

Voici le résultat obtenu après la suppression de la variable "area" du modèle :

```
[INFO] predicting house prices...
3/3 [==============================] - 0s 4ms/step
[INFO] avg. house price: $533,388.27, std house price: $493,403.08
[INFO] mean: 24.34%, std: 25.14%
```

En comparant ce résultat à celui où toutes les variables ont été utilisées, nous pouvons voir que la suppression de la variable "area" du modèle a entraîné une diminution importante de la précision des prévisions du modèle.
La moyenne de la différence absolue en pourcentage est passée de 71,34% à 24,28%. Cela suggère que la variable "surface" joue un rôle important dans la prédiction des prix des maisons, et son omission du modèle a considérablement affecté les performances du modèle.

Séparation de la variable catégorielle des variables numériques

Le script proposé fusionne les variables catégorielles et les variables numériques en une seule variable. L' idée derrière cette partie c'est de séparer ces deux types de variables et les faire passer par deux réseaux séparés qui se rejoignent avec le troisième alimenté par les images. Ceci peut être bénéfique et permettra à chaque réseau de se concentrer sur la tâche spécifique pour laquelle il est le plus adapté, en utilisant des techniques de traitement appropriées pour chaque type de données. Cela peut également améliorer les performances globales du modèle en optimisant les performances de chaque sous-réseau.
Pour cela nous avons été amené à diviser la fonction `process house attributes` en deux sous fonctions :

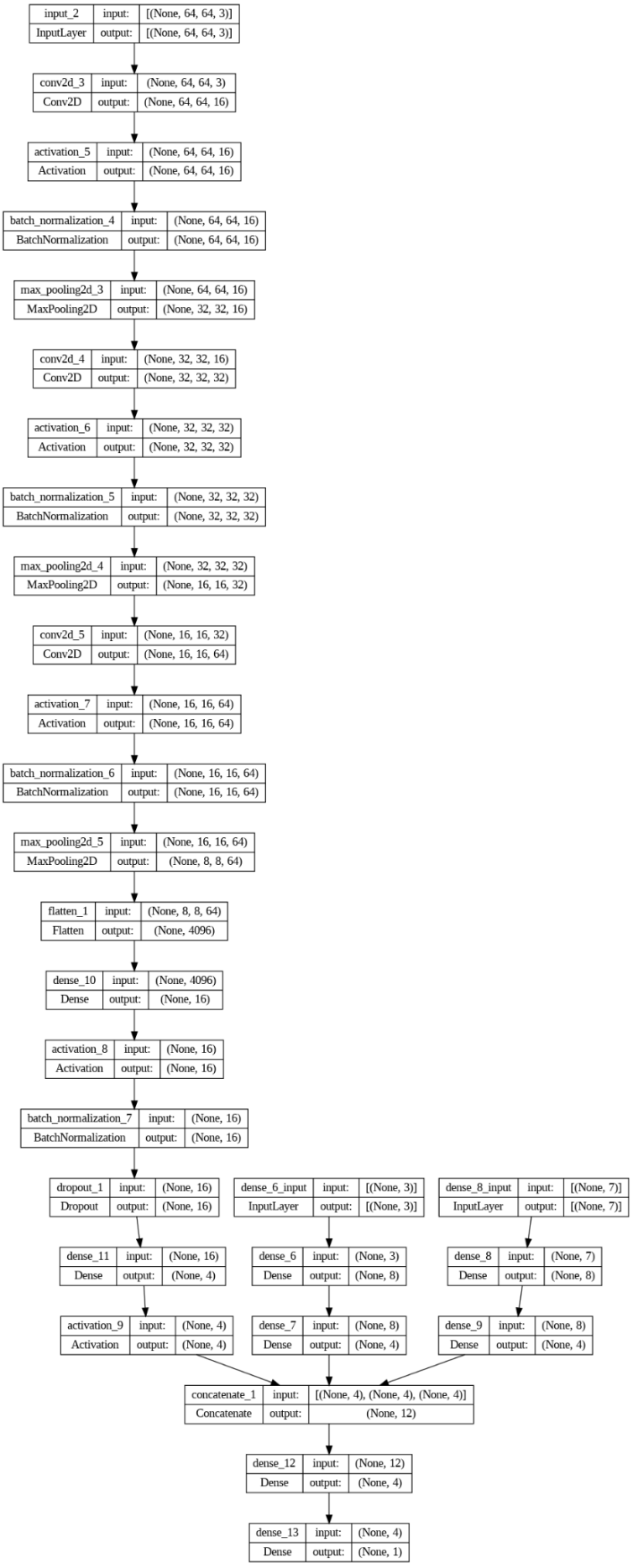
- `process house attributes categorical :`

pour les variables catégorielles ce qui effectuera un redimensionnement MinMax sur les caractéristiques continues et renverra les caractéristiques continues redimensionnées en tant que données d'entraînement et de test.

- `process house_attributes_continous:`

pour encoder les données catégorielles du code postal en utilisant l'encodage one-hot et renverra les caractéristiques catégorielles encodées en tant que données d'entraînement et de test.

```
def process_house_attributes_continous(df, train, test):
  continuous = ["bedrooms", "bathrooms", "area"]
  cs = MinMaxScaler()
  trainContinuous = cs.fit_transform(train[continuous])
  testContinuous = css.transform(test[continuous])
  trainX = np.hstack([trainContinuous])
  testX = np.hstack([testContinuous])
  return (trainX, testX)

def process_house_attributes_categorical(df, train, test):
  zipBinarizer = LabelBinarizer().fit(df["zipcode"])
  trainCategorical = zipBinarizer.transform(train["zipcode"])
  testCategorical = zipBinarizer.transform(test["zipcode"])

  trainX = np.hstack([trainCategorical])
  testX = np.hstack([testCategorical])
  return (trainX, testX)
```


input_2 | input: | [(None, 64, 64, 3)]
InputLayer | output: | [(None, 64, 64, 3)]
conv2d_3 | input: | (None, 64, 64, 3)
Conv2D | output: | (None, 64, 64, 16)
activation_5 | input: | (None, 64, 64, 16)
Activation | output: | (None, 64, 64, 16)
batch_normalization_4 | input: | (None, 64, 64, 16)
BatchNormalization | output: | (None, 64, 64, 16)
max_pooling2d_3 | input: | (None, 64, 64, 16)
MaxPooling2D | output: | (None, 32, 32, 16)
conv2d_4 | input: | (None, 32, 32, 16)
Conv2D | output: | (None, 32, 32, 32)
activation_6 | input: | (None, 32, 32, 32)
Activation | output: | (None, 32, 32, 32)
batch_normalization_5 | input: | (None, 32, 32, 32)
BatchNormalization | output: | (None, 32, 32, 32)
max_pooling2d_4 | input: | (None, 32, 32, 32)
MaxPooling2D | output: | (None, 16, 16, 32)
conv2d_5 | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 64)
activation_7 | input: | (None, 16, 16, 64)
Activation | output: | (None, 16, 16, 64)
batch_normalization_6 | input: | (None, 16, 16, 64)
BatchNormalization | output: | (None, 16, 16, 64)
max_pooling2d_5 | input: | (None, 16, 16, 64)
MaxPooling2D | output: | (None, 8, 8, 64)
flatten_1 | input: | (None, 8, 8, 64)
Flatten | output: | (None, 4096)
dense_10 | input: | (None, 4096)
Dense | output: | (None, 16)
activation_8 | input: | (None, 16)
Activation | output: | (None, 16)
batch_normalization_7 | input: | (None, 16)
BatchNormalization | output: | (None, 16)
dropout_1 | input: | (None, 16)
Dropout | output: | (None, 16)
dense_6_input | input: | [(None, 3)]
InputLayer | output: | [(None, 3)]
dense_8_input | input: | [(None, 7)]
InputLayer | output: | [(None, 7)]
dense_11 | input: | (None, 16)
Dense | output: | (None, 4)
dense_6 | input: | (None, 3)
Dense | output: | (None, 8)
dense_8 | input: | (None, 7)
Dense | output: | (None, 8)
activation_9 | input: | (None, 4)
Activation | output: | (None, 4)
dense_7 | input: | (None, 8)
Dense | output: | (None, 4)
dense_9 | input: | (None, 8)
Dense | output: | (None, 4)
concatenate_1 | input: | [(None, 4), (None, 4), (None, 4)]
Concatenate | output: | (None, 12)
dense_12 | input: | (None, 12)
Dense | output: | (None, 4)
dense_13 | input: | (None, 4)
Dense | output: | (None, 1)

Après l'exécution du script, nous remarquons une amélioration importante dans la précision des prévisions du modèle. Cela peut nous conduire à dire que la séparation des données continues et catégorielles ait amélioré la performance de notre modèle. Par contre nous savons très bien que la précision n'est qu'un indicateur de performance. il est donc important de considérer d'autres indicateurs et de tester sur des données supplémentaires pour s'assurer que les résultats obtenus sont généralisables.

Suite à la séparation

```
[INFO] predicting house prices...
3/3 [==============================] - 0s 41ms/step
[INFO] avg. house price: $533,388.27, std house price: $493,403.08
[INFO] mean: 65.82%, std: 81.18%
```

Transfer_learning : [23-01-2023]

Variational AE [20-01-23] :

Afin de répondre aux instructions, nous avons procéder comme suit :

Modification de la fonction de reconstruction :

nous avons modifier la fonction de reconstruction error loss pour utiliser la mean squared error (MSE) en remplaçant la ligne suivante :

```
def getloss():
  ''' we define all the required loss functions, one for each model output
  return a dictionnary with key-values : model_output_name:loss function '''
  def loss_cod(y_true,y_pred):
    z_mean , z_log_var = tf.split(y_pred,num_or_size_splits=2, axis=-1)
    kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
    kl_loss = K.sum(kl_loss, axis=-1)
    kl_loss *= -0.5
    return kl_loss
  def loss_rec(y_true,y_pred):
    recons_loss = tf.reduce_mean(tf.square(y_true - y_pred),axis=-1)
    recons_loss = tf.reduce_mean(recons_loss)
    return recons_loss*784
  loss_dict = {'enc_out':loss_cod,'dec_out':loss_rec}
  return loss_dict
```

par celle-ci :

recons_loss = tf.reduce_mean(tf.keras.losses.mean_squared_error(y_true, y_pred))

Modification de la fonction de reconstruction : retruquer un truc ahhahah