TPautoencoder

February 11, 2021

1 TP autoencoder

Diane LINGRAND

diane.lingrand@univ-cotedazur.fr

[ ]:
```
import tensorflow
print(tensorflow.__version__)
import tensorflow.keras
```

[ ]:
```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D,␣
 ↪Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.datasets import mnist

from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
import glob
#!pip install tqdm
from tqdm import tqdm
import random
from matplotlib import pyplot as plt
```

2 Part I: playing with the MNIST dataset

[ ]:
```
# loading the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# preprocessings
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
xTrain = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
xTest = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

size1 = len(xTrain[0])
size = len(x_train[0])
```

```
print(size,size1)
```

2.1 a first simple autoencoder

[ ]:
```
inputImage = Input(shape=(size1,))
encoded = Dense(size1, activation='relu')(inputImage)
decoded = Dense(size1, activation='sigmoid')(encoded)

autoencoder = Model(inputImage, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.summary()
```

[ ]:
```
### learning the autoencoder
from tensorflow.keras.callbacks import EarlyStopping
ourCallback = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=10,␣
↪verbose=0, mode='auto', baseline=None, restore_best_weights=True)

autoencoder.fit(xTrain, xTrain, epochs=100, batch_size=128, validation_split=0.
↪2, shuffle=True,callbacks=[ourCallback])
```

[ ]:
```
# scores
print("train score = ", autoencoder.evaluate(xTrain,xTrain))
print("test score = ", autoencoder.evaluate(xTest,xTest))
```

[ ]:
```
xTrainPredicted = autoencoder.predict(xTrain)
```

[ ]:
```
# let's look at the images
index = random.randint(0,len(xTrain)-1)
print("image number: ", index)
im1 = xTrain[index]
im1 = im1.reshape((size,size))
im2 = xTrainPredicted[index]
print(im2.shape)
im2 = im2.reshape((size,size))
print(im2.shape)
fig = plt.figure()
ax = fig.add_subplot(1, 2, 1)
plt.imshow(im1, cmap='gray')
ax = fig.add_subplot(1, 2, 2)
plt.imshow(im2, cmap='gray')
plt.show()
```

[ ]:
```
#more images (could also be done on the test dataset)
index = random.randint(0,len(xTrain)-1)
print("image number: ", index)
```

```
fig = plt.figure(figsize=(20, 8))
for i in range(1,11):
    index = random.randint(0,len(xTrain)-1)
    im1 = xTrain[index]
    im1 = im1.reshape((size,size))
    im2 = xTrainPredicted[index]
    im2 = im2.reshape((size,size))
    ax = fig.add_subplot(4, 10, i)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    plt.imshow(im1, cmap='gray')
    ax = fig.add_subplot(4, 10, i+10)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    plt.imshow(im2, cmap='gray')

    index = random.randint(0,len(xTrain)-1)
    im1 = xTrain[index]
    im1 = im1.reshape((size,size))
    im2 = xTrainPredicted[index]
    im2 = im2.reshape((size,size))
    ax = fig.add_subplot(4, 10, i+20)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    plt.imshow(im1, cmap='gray')
    ax = fig.add_subplot(4, 10, i+30)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    plt.imshow(im2, cmap='gray')
plt.show()
```

2.2 a more complex autoencoder

[ ]:

```
inputImage = Input(shape=(size1,))
encoded = Dense(500, activation='relu')(inputImage)
encoded = Dense(250, activation='relu')(encoded)
encoded = Dense(28, activation='relu')(encoded)
decoded = Dense(250, activation='relu')(encoded)
decoded = Dense(500, activation='relu')(decoded)
decoded = Dense(size1, activation='sigmoid')(decoded)

autoencoder = Model(inputImage, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.summary()
```

2.3 a convolutional autoencoder

[ ]:
```
inputImage = Input(shape=(size1,))
encoded = layers.Conv2D(32, (3, 3), activation='relu',␣
↪padding='same')(inputImage)
encoded = layers.MaxPooling2D((2, 2), padding='same')(encoded)
encoded = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
encoded = layers.MaxPooling2D((2, 2), padding='same')(encoded)

decoded = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
decoded = layers.UpSampling2D((2, 2))(decoded)
decoded = layers.Conv2D(32, (3, 3), activation='relu',padding='same')(decoded)
```

What is the dimension of the bottleneck ? Learn, test, and observe the differences.

2.4 separate encoding and decoding

[ ]:
```
# encoder part
inputImage = Input(shape=(size1,))
encoded = Dense(500, activation='relu')(inputImage)
encoded = Dense(250, activation='relu')(encoded)
encoded = Dense(28, activation='relu')(encoded)
# decoder part
decoded = Dense(250, activation='relu')(encoded)
decoded = Dense(500, activation='relu')(decoded)
decoded = Dense(size1, activation='sigmoid')(decoded)

# autoencoder model that will be learned
autoencoder = Model(inputImage, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(xTrain, xTrain, epochs=100, batch_size=128, validation_split=0.
↪2, shuffle=True,callbacks=[ourCallback])
autoencoder.summary()

#encoder model: weigths are already learned
encoderModel = Model(inputImage, encoded)
encoderModel.summary()
# decoder model (adapt this code to your network): weigths are already learned
inputLatent = Input(shape=(28,))
decoder = autoencoder.layers[-3](inputLatent)
decoder = autoencoder.layers[-2](decoder)
decoder = autoencoder.layers[-1](decoder)
decoderModel = Model(inputLatent,decoder)
decoderModel.summary()
```

[ ]:
```
#For a single data:
index = random.randint(0,len(xTest)-2)
```

```
print("image number: ", index)
im = xTest[index]

# encoding of an image
latentData = encoderModel.predict(np.array([im,]))

print("shape = ",latentData.shape)

decodedData = decoderModel.predict(latentData)
print("shape2 = ", decodedData.shape)
```

[ ]:
```
#display
fig = plt.figure()
ax = fig.add_subplot(1,2,1)
plt.imshow(im.reshape(28,28), cmap='gray')
ax = fig.add_subplot(1,2,2)

plt.imshow(decodedData[0].reshape(28,28), cmap='gray')
plt.show()
```

3 Part II: playing with the animals10 dataset

[ ]:
```
# download the dataset
from google.colab import drive
drive.mount('/content/drive/')
datasetRoot='/content/drive/My Drive/DLS2020/raw-img/'
```

[ ]:
```
# or work localy using already downloaded dataset
# modify to your local directory
datasetRoot='/home/lingrand/Ens/MachineLearning/animals/raw-img/'
```

[ ]:
```
classes = ['mucca', 'elefante', 'gatto', 'cavallo', 'scoiattolo', 'ragno',␣
↪'pecora', 'farfalla', 'gallina', 'cane']
#training data

rootTrain = datasetRoot+'train/'
size = 66 #112 #224 #try different image size reductions
size2 = size*size
nbImages = 0
for cl in classes:
    nbImages += len(glob.glob(rootTrain+cl+'/*'))
xTrain = xTrain = np.empty(shape=(nbImages,size2))
print("total number of images: ",len(xTrain))
i = 0
for cl in tqdm(classes):
    listImages = glob.glob(rootTrain+cl+'/*')
```

```
    print("class: ", cl, " : ", len(listImages))
    for pathImg in tqdm(listImages):
        img = image.load_img(pathImg, target_size=(size,␣
↪size),color_mode='grayscale')
        im = image.img_to_array(img)
        im = np.reshape(im,size2)
        im /= 255.0
        xTrain[i, :] = im
        i += 1
print(xTrain.shape)
```

[ ]:
```
print(size2)
```

3.1 Let us build an autoencoder

[ ]:
```
inputImage = Input(shape=(size2,))
encoded = Dense(4000, activation='relu')(inputImage)
encoded = Dense(2000, activation='relu')(encoded)

decoded = Dense(4000, activation='relu')(encoded)
decoded = Dense(size2, activation='sigmoid')(decoded)

autoencoder = Model(inputImage, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
print(autoencoder.summary())
```

[ ]:
```
from tensorflow.keras.callbacks import EarlyStopping
ourCallback = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=10,␣
↪verbose=0, mode='auto', baseline=None, restore_best_weights=True)
```

[ ]:
```
#learning
autoencoder.fit(xTrain, xTrain, epochs=100,batch_size=256, validation_split=0.
↪2, shuffle=True,callbacks=[ourCallback])
```

[ ]:
```
print("score = ", autoencoder.evaluate(xTrain,xTrain))
```

[ ]:
```
# predictions
xTrainPredicted = autoencoder.predict(xTrain)
```

[ ]:
```
# let's look at the differences

index = random.randint(0,len(xTrain)-1)
print("image number: ", index)
im1 = xTrain[index]
im1 = im1.reshape((size,size))
im2 = xTrainPredicted[index]
```

```
print(im2.shape)
im2 = im2.reshape((size,size))
print(im2.shape)
fig = plt.figure()
ax = fig.add_subplot(1, 2, 1)
plt.imshow(im1, cmap='gray')
ax = fig.add_subplot(1, 2, 2)
plt.imshow(im2, cmap='gray')
plt.show()
```

Try also to plot more images.

3.2 From dense to convolutional layers

Change the Dense layers to Conv2D layers and add Pooling layers.

[ ]:
```
# prepare the data
xTrain = xTrain.reshape(nbImages,size,size)
```

[ ]:
```
print(xTrain.shape)
```

[ ]:
```
# your work is to try different architectures in order to find the best␣
↪reconstruction / reduction of latent representation
#  encoder part
size=66
inputImage = tensorflow.keras.Input(shape=(size, size, 1))
encoded = Conv2D(2, (3, 3), activation='relu', padding='same')(inputImage)
encoded = MaxPooling2D((2, 2), padding='same')(encoded)
#  decoding part
decoded = Conv2D(2, (3, 3), activation='relu', padding='same')(encoded)
decoded = UpSampling2D((2, 2))(decoded)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(decoded)

autoencoder = Model(inputImage, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
print(autoencoder.summary())
```

[ ]:
```
# compile, run, test, ... as in previous section
autoencoder.fit(xTrain, xTrain, epochs=100,batch_size=256, validation_split=0.
↪2, shuffle=True,callbacks=[ourCallback])
```

[ ]:
```
print("score = ", autoencoder.evaluate(xTrain,xTrain))
xTrainPredictedConv = autoencoder.predict(xTrain)
```

[ ]:
```
# let's look at the differences

index = random.randint(0,len(xTrain)-1)
```

```
print("image number: ", index)
im1 = xTrain[index]
im2 = xTrainPredicted[index]
im2 = im2.reshape((size,size))
im3 = xTrainPredictedConv[index]
print(im3.shape)
fig = plt.figure()
ax = fig.add_subplot(1, 3, 1)
plt.imshow(im1, cmap='gray')
ax = fig.add_subplot(1, 3, 2)
plt.imshow(im2, cmap='gray')
ax = fig.add_subplot(1, 3, 3)
plt.imshow(im3[:,:,0], cmap='gray')
plt.show()
```

3.3 Perturbation on the latent representation

In this section, we will perturb the latent representation before decoding. Perturbations will be represented as additive noise. We will also explore the latent space by searching for latent vectors in the neighbourhood of some latent representation of training samples and see how the decoding representation looks like.

[ ]:

```
# encoder model
encoderModel = Model(inputImage, encoded)
# decoder model (adapt this code to your network)
inputLatent = Input(shape=(33,33,2)) # to adapt to your network
decoder = autoencoder.layers[-3](inputLatent)
decoder = autoencoder.layers[-2](decoder)
decoder = autoencoder.layers[-1](decoder)
decoderModel = Model(inputLatent,decoder)
```

[ ]:

```
# ATTENTION: à adapter à votre réseau
index = random.randint(0,len(xTrain)-2)
print("image number: ", index)
im1 = xTrain[index]
jm1 = xTrain[index+1]
# encoding of an image
latentData1 = encoderModel.predict(np.array([im1,]))
latentData2 = encoderModel.predict(np.array([jm1,]))
# perturbation on the latent representation (you have to try different values/
 ↪distributions for epsilon)
epsilon = 0.1
#une possibilité: ajouter du bruit à la représentation latente
#latentData += np.random.uniform(-epsilon, epsilon, size=(1,33,33,16))
print("shape = ",latentData.shape)
# ou bien, prendre la représentation latente de 2 images de meme classe et␣
 ↪faire la moyenne (puis la décoder)
```

```python
# decoding of the latent representation
latentData = 0.5*(latentData1+latentData2)
decodedData = decoderModel.predict(latentData)
print("shape2 = ", decodedData.shape)
#display
fig = plt.figure()
ax = fig.add_subplot(1, 3, 1)
plt.imshow(im1, cmap='gray')
ax = fig.add_subplot(1,3,2)
plt.imshow(jm1, cmap='gray')
ax = fig.add_subplot(1, 3, 3)
plt.imshow(decodedData[0,:,:,0], cmap='gray')
plt.show()
```

[ ]:

```python

```

[ ]:

```python

```