TP99-MNIST-GAN

September 18, 2019

1 GAN on MNIST

We will still use the same MNIST dataset with each example shaped as `(28,28,1)` array. But this time no need to split into train/valid dataset.

[1]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras

import matplotlib.pyplot as plt
%matplotlib inline
import datetime as dt

from pathlib import Path

from sklearn.datasets import fetch_openml
from sklearn import preprocessing

import io
import scipy

data_home = '/tmp/scikit_learn_data/'
datafile = '/tmp/mnist.npz'

datapath = Path(datafile)
if not(datapath.exists()):
    print("Data File not found... downloading it")
    Xmnist, ymnist = fetch_openml('mnist_784',
                                  version=1,
                                  return_X_y=True,
                                  data_home=data_home)
    np.savez(datapath.as_posix(),
             X=np.array(Xmnist, dtype='u8'),
             y=np.array(ymnist, dtype='u8'))
    print("Data File downloaded and saved")
    del Xmnist, ymnist

print("Data File found... loading it into memory")
```

```
data = np.load(datapath.as_posix())
Xmnist = data['X']/255.
ymnist = keras.utils.to_categorical(data['y'])
print("Data File loaded")

Xtrain, Ytrain = Xmnist[:60000], ymnist[:60000]
Xtest, Ytest = Xmnist[-10000:], ymnist[-10000:]

Xtrain = Xtrain.reshape((Xtrain.shape[0], 28, 28, 1))
Xtest = Xtest.reshape((Xtest.shape[0], 28, 28, 1))
```

```
Data File found... loading it into memory
Data File loaded
```

1.1 Training GAN in KERAS

GAN combined two networks a `generator` and a `discriminator`. Nevertheless, - while the generator is learnt the weights of the discrimnitor need to be fixed, - and while the discrimnitor is learnt the weights of the generator need also to be fixed. That's why there is no easy way to use the `fit` method proposed by keras. You had to write yourself the big training loop with the `train_on_batch` method.

Here is an example code to show you how to combine the generator and the discriminator:

```
generator = [...] # create your own generator
generator.build(input_shape=[...])
# no need to compile the generator as it will not be learnt by itself
generator.summary()

discriminator = [...] # create your own discriminator
discriminator.build(input_shape=[...])
discriminator.compile(loss='binary_crossentropy', optimizer='adam')
discriminator.summary()

# we fixed discriminator weights
# must be call after discriminator compilation
# see definition in the next cell
set_trainable(discriminator, False)

# Combined the generator and the discriminator into a gan model
# here the discriminator weights are fixed
gan = keras.Sequential(name='gan')
gan.add(generator)
gan.add(discriminator)
gan.build(input_shape=[...])
gan.compile(loss='binary_crossentropy', optimizer='adam')
gan.summary()
```

Now at each batch iteration: - the discriminator should be trained with fake examples labeled 0 and real examples labeled 1. - the gan should be trained with only fake examples labeled 1.

You can produce fake example using the 'predict' method of the generator.

```
for epochs
    for batch
        #[...]
        Xfake_batch = generator.predict(some_noise)
        #[...]
        # Xdiscriminator_batch contains both fake and real examples
        # respectively labelled 0 and 1
        discriminator.train_on_batch(Xdiscriminator_batch, Ydiscriminator_batch)
        # Xgan_batch contains only fake examples
        # labelled 1
        gan.train_on_batch(Xgan_batch, Ygan_batch)
```

In order to help you, the following function are given: - `set_trainable` to change the trainable state of a whole model - `batch_iterator` an iterator producing slice indices to split a dataset into batches

See examples bellow.

[2]:
```
def set_trainable(model, trainable=True):
    model.trainable = trainable
    for l in model.layers:
       l.trainable = trainable

def batch_iterator(nx, batch_size, shuffle=True):
# nx : bumber of examples in the set
# batch_size: the desired batch size
# shuffle: whether to shuffle examples or not
# It works even if nx is not divisible by batch_size
    idx = np.arange(nx)
    if shuffle:
        np.random.shuffle(idx)
    n_batchs = int(np.floor(nx/batch_size))
    if n_batchs * batch_size < nx:
        n_batchs += 1
    start = 0
    for batch in range(n_batchs):
        end = min(start + batch_size, nx)
        aSlice = idx[start:end]
        start = end
        yield aSlice, batch, n_batchs
# yield the current slice, the current batch index, and the total number of␣
 ↪batches

print("Batch iterator examples")
```

```
for aSlice, batch, n_batch in batch_iterator(100,10,shuffle=False):
    print(aSlice, batch, n_batch)
print("")
for aSlice, batch, n_batch in batch_iterator(100,10):
    print(aSlice, batch, n_batch)
print("")
for aSlice, batch, n_batch in batch_iterator(100,15,shuffle=False):
    print(aSlice, batch, n_batch)
```

```
Batch iterator examples
[0 1 2 3 4 5 6 7 8 9] 0 10
[10 11 12 13 14 15 16 17 18 19] 1 10
[20 21 22 23 24 25 26 27 28 29] 2 10
[30 31 32 33 34 35 36 37 38 39] 3 10
[40 41 42 43 44 45 46 47 48 49] 4 10
[50 51 52 53 54 55 56 57 58 59] 5 10
[60 61 62 63 64 65 66 67 68 69] 6 10
[70 71 72 73 74 75 76 77 78 79] 7 10
[80 81 82 83 84 85 86 87 88 89] 8 10
[90 91 92 93 94 95 96 97 98 99] 9 10

[29  4 13 17  7  0 16  5 52 54] 0 10
[ 1 46 36 58 98 83 14  3 24 93] 1 10
[45 61 35 56 53 65 47 51 75 32] 2 10
[76 27 69 64 15 11 86 28 44 60] 3 10
[48 80 95 59 94 25 39 70 87 88] 4 10
[96  2 40 63 38 33 79 50 92 18] 5 10
[77 21 30 74 12 62 19 68 22  6] 6 10
[10  8 85 66 71 99 20 57 73 78] 7 10
[49 90  9 31 89 67 84 42 97 34] 8 10
[41 26 72 81 82 23 91 43 55 37] 9 10

[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14] 0 7
[15 16 17 18 19 20 21 22 23 24 25 26 27 28 29] 1 7
[30 31 32 33 34 35 36 37 38 39 40 41 42 43 44] 2 7
[45 46 47 48 49 50 51 52 53 54 55 56 57 58 59] 3 7
[60 61 62 63 64 65 66 67 68 69 70 71 72 73 74] 4 7
[75 76 77 78 79 80 81 82 83 84 85 86 87 88 89] 5 7
[90 91 92 93 94 95 96 97 98 99] 6 7
```

1.2 Exercice

1) Build a GAN to produce MNIST like images.
2) Build a Pac-GAN to produce more diversity (2 images at the input of the discriminator).
3) Build a Condition GAN to produce MNIST like images conditioned to the class label.