Generative Adversarial Networks

MVA, Cours Delires

March 7, 2019

The goal of this TP is to explore Generative Adversarial Networks (GANs). Your job is to fill in the parts indicated with "**FILL IN CODE**" and to answer the questions in this document. If you are using the local machines, in order to use the Keras package, you must first load an anaconda environment with these packages. To do this, execute the following commands in a terminal :

```
export PATH=/cal/softs/anaconda/anaconda3/bin:$PATH # Enable anaconda
source activate ~anewson/.conda/envs/keras_env #activate the environment with keras etc
```

If you have your own machine, you can install you own environment. You will need the following packages :

- keras
- matplotlib
- numpy

We will be using the "mnist" dataset. This is a set of 60,000 greyscale images, of size 28×28. The images are of handwritten digits. This simple dataset is easy to manipulate and experiment with, since the images are not too large, so training does not take too long. Feel free to try this code on other datasets to see how they work.

1 GANs

In the first part of the TP we will implement a simple GAN, with an MLP-type architecture. Since GAN training can take some time, there are two functionalities already coded to save ("save_gan_model") and load ("load_gan_model") a model. The gan is saved in a .pickle file. You need to specify the model with the path, name with the extension (for example "mnist_gan_model.pickle") The code saves the current model every "sample_interval" steps. If you want to load a certain model, just

As in the previous TP, We are going to implement our architectures with the Keras package. Here is an example of the definition of a simple one-layer neural network :

For the definitions of the necessary architectures, you can use the following Keras layers and operations :

- Flatten
- Dense
- Activation
- LeakyReLU
- Reshape
- Conv2D
- Conv2DTranspose

Please look at the online Keras documentation[1] for more information about these functions. There are other APIs which exist in Keras (such as the Sequential() API, you may use these if you like).

The GAN loss function is the following :

$$\min_G \max_D \mathbb{E}_{x \in p_{data}} \left[\log D(x)\right] + \mathbb{E}_{z \in p_z} \left[\log \left(1 - D(G(z))\right)\right], \tag{1}$$

where G is the generator, D is the discriminator, z is the latent code, which follows a normal distribution, wand

1.1 GAN architecture

We are going to train the GAN on the mnist dataset. Your task is to implement the following architecture, with thegenerator and the discriminator as follows :

- Generator (input : a random vector in the latent space)
  1. Dense layer, to size 256
  2. Leaky ReLU ($\alpha = 0.2$)
  3. Dense layer, output size 512
  4. LeakyReLU(alpha=0.2)(y)
  5. Dense layer, output size 784
  6. Tanh activation
  7. Reshape ($28 \times 28 \times 1$)
- Discriminator (input : an image)
  1. Flatten()
  2. Dense, to size 512
  3. Leaky ReLU ($\alpha = 0.2$)
  4. Dense, to size 256
  5. Leaky ReLU ($\alpha = 0.2$)
  6. Dense, to size 1
  7. Sigmoid activation

The training algorithm then consists of the following steps :

Algorithm 1 GAN training algorithm

Data: X : training image dataset
N : batch size
k : number of inner loop iterations
n : number of optimisation iterations
for $i = 0$ to $n - 1$ **do**
  for $i = 0$ to $k - 1$ **do**
    $x \leftarrow$ random_batch(X,N)
    $z \leftarrow$ random_normal_vectors(N)
    Train discriminator on real images x
    Train discriminator on fake images $G(z)$
  end for
  $z \leftarrow$ random_normal_vectors(N)
  Train generator on fake images $G(z)$
end for

[1] https://keras.io/

Questions:

1. Use the "tp_mva_gan.py' code to implement the GAN with the MLP architecture, with $d = 32$
2. Implement the discriminator and generator losses. In order to implement the discriminator loss simply, rewrite it in terms of the binary cross-entropy and a correctly-chosen label (either 0 or 1). Note : you *cannot* do this with the generator
3. Carry out training on the mnist dataset

2 Conditional GAN

The conditional GAN is a GAN which uses an additional input : the label of the class of image we want to generate (such as the number 0-9 in mnist, or a dog in CIFAR). This uses a modified loss function, with a class c :

$$\min_G \max_D \mathbb{E}_{x \in p_{data}} \left[\log D(x|c)\right] + \mathbb{E}_{z \in p_z} \left[\log \left(1 - D(G(z|c))\right)\right] \tag{2}$$

Questions :

1. Use the 'tp_mva_cgan.py' code to implement the Conditional GAN with the MLP architecture, with mnist. For this, you must modify the code in build_generator and build_discriminator. Some tips for this :
   - You can concatenate two tensors with : output_tensor = Concatenate()([a, b]), where a and b are two tensors
   - To create a model which takes two inputs, do : model_out = Model(inputs=[a_in, b_in], outputs = [c_out])

3 Using GPUs for more complex datasets

NOTE : This section of the TP is *not* to hand in. This is just to give you the possibility to use a GPU, in particular for your projects.

You now have a code which can generate any sort of image (hopefully, given enough training). We can try it out with the CIFAR10 dataset (more complex data, of 10 classes : airplane, bird etc.), however it will require some more computing power !! Therefore, we will use some machines which are in another lab room, C51. These have graphics processing units (GPUs), with which we can carry out parallel computations.

The easiest way to do this is to access the GPU remotely through the jupyter notebook. We will start the jupyter notebook remotely, and at the same time create a link from a port on your local machine to the same port on the remote machine. This will allow you to create a python notebook on a browser locally, but have the code executed on a GPU. Otherwise, you can also just do an ssh to the machine and call the code remotely (execute python my_code.py).

Now, there are only 8 machines available, so you will have to share them. The names of the machines are :

- "c51-10", "c51-11','c51-13","c51-14","c51-15","c51-17", "c51-18","c51-19".

You should choose one of these machines, such that there are not more than three accesses at the same time on the same machine.

Furthermore, you need to choose a port number to which you will link the local jupyter notebook. Choose, for example 8888, 8889, 8890, 8891 etc. **For each machine, you can have at most one access** to any given port. If two people try to access the same port, you will not be able to.

In the TP folder, there are two scripts which you need to modify. Firstly "jupyter_distance.sh" :

```
export PORT=8888
export REMOTEMACHINE=c51-11
ssh $REMOTEMACHINE << EOF
 export PATH=/cal/softs/anaconda/anaconda3/bin:$PATH
 source activate /cal/homes/ladjal/.conda/envs/tf_C51
```

```
 jupyter notebook --no-browser --port=$PORT
EOF
```

You need to modify the PORT and REMOTEMACHINE variables to your chosen value. Secondly, "pipe_script.sh" :

```
export PORT=8888
export REMOTEMACHINE=@c51-12
ssh -N -L localhost:$PORT:localhost:$PORT $REMOTEMACHINE
```

Again, you need to modify the PORT and REMOTEMACHINE variables. **It is very important to make sure that these variables have the same values in the two scripts**.

You should execute these two scripts **in two different terminals** (this is important, because once they are executed, they do not give you back control).

Now, once you have done this, open an internet browser, and navigate to to location specified by the first terminal (for example, `http://localhost:8888/?token=6526f4487b306493bac3140eba4388118ce98028bc89f9d5`). You can then create a new jupyter notebook and copy/paste your code.

3.1 Killing the process listening on the port

Note : If you have exited from the remote machine, without killing the process associated with your port, you will not be able to use the port again, unless you kill that process. You can execute the following command in a terminal on the remote machine to find the pid of the process :

```
lsof -n -i4TCP:8888
```

If you have used the port 8888, for example.This will give you an output like the following example :

```
COMMAND   PID  USER FD  TYPE DEVICE SIZE/OFF NODE NAME
jupyter-n 147726 anewson 7u IPv4 661396 0t0 TCP 127.0.0.1:8888 (LISTEN)
```

To kill the process, just execute on the remote machine :

```
kill 147726
```