Machine Learning III

Practice II: Group F

Deep Generative Adversarial Nets

Group F:

Agathe de Drouas, Francisco Fortes, Andrea Fusetti, Milan Goetz, Eleanor Manley, Tamara Amer Nayef Qassem, Luisa Runge

1. Generating MNIST data

1. Explain the high-level idea of Generative Adversarial Nets.

Generative adversarial net is an algorithm architecture that uses two neural networks, the networks work simultaneously, in contrast to each other. In order to create new, synthesized data that can pass for real data.
The generator, the first neural network, generates new data instances and the second neural network, the discriminator classifies this data as real which means belonging to the training set or as false which means created by the generator..
In other words, the goal of the discriminator is to authentify the real data instances from the training set as real. In contrast, the goal of the generator is to generate data instances that will be authentified as real by the discriminator even though they are fake.

Process:

The generator takes random numbers and returns a new data instance and then the discriminator is fed with fake data and real data from the training set.
The discriminator will classify the data instances as boolean, 1 meaning real and 0 meaning false.

Both the generator and discriminator are trained together. The discriminator is updated to get better at discriminating real and fake samples in the next round, while the generator is updated based on how well, or not, the generated samples fooled the discriminator.

The tasks of these two neurals networks are opposed as their objectives are opposed. Generator tries to fool the discriminator while the discriminator tries to predict the generator. They are playing a zero-sum game. When the discriminator successfully identifies a real and fake image it is rewarded whereas the generator is penalized with updates to model parameters. And the opposite happens when the generator fools the discriminator.

GAN solves the problem of generating data when you don't have enough to begin with and requires no human supervision which makes it very efficient.

2. Both the generator and discriminator are convolutional neural nets. Which are the inputs and expected outputs of both of them before and after training?

Before the training, the input to the generator is a series of latent samples (series of 100 randomly generated numbers).
Once trained it transforms the input (noise) into some data instances (output). In our case a 28*28 handwritten digit image. (Same size as the MNIST images). After, the generator also uses the discriminator's classification feedback as input in order to improve the generation of new instances that the discriminator will finally classify as real.
On the other hand, the discriminator receives as an input the generator output and data instances from the training set.
We train the discriminator using the supervised learning to tell whether an image is real or not. The output will be positive value '1' for real images and value '0' for fake images.

	Input	Output
Generator	Random input (noise) and discriminator feedback	Transforms input (noise) into data instances
Discriminator	Generator output and real data instances	Boolean classification: 1 = Real 0 = False

Both the generator and discriminator are convolutional neural nets and they are both trained together.

3. The core functions are train and its sub function train_step. Explain step by step what they are doing.

The functions train_step and train in the notebook are used for training the generator and discriminator.

Train_step:

In the first line the noise generated as a vector of random normal numbers is taken as input from the generator in order to generate the fake images.
Thereafter the real images and the fake generated images are given as input to the discriminator models, which is used to classify which ones are real and which are fake.
The next line computes the generator loss. It returns how well the generator has been in 'tricking' or not the discriminator, using cross entropy loss function.
Next, the discriminator loss is calculated. It needs to take into account both the loss from real images and from fake ones, as it takes both as input.
The gradients are then used to optimize the generator and the discriminator.

Train:

For each epoch:
The train function performs the train_step function on images of each batch which produces an image for the GIF.
The model is saved as a checkpoint after every 15 epochs.
It also prints and saves the created image and the time that it took to create it.

2. Generate CIFAR10-images

Refer to notebook: 02_dcgan_cifar10_GroupF.ipynb

1) TODO 1. Complete the code for the Generator model.

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(8*8*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((8, 8, 256)))
    assert model.output_shape == (None, 8, 8, 256) # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 8, 8, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 16, 16, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 32, 32, 3)

    return model
```

2) TODO 2. Complete the code for the Discriminator model.

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                    input_shape=[32, 32, 3]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

            model.add(layers.Conv2D(128,    (5,    5),    strides=(2,    2),
padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

For the final GIF, refer to: dcgan_CIFAR10_GroupF.gif

3. STAND ON THE SHOULDERS OF GIANTS

1. Explain what a Conditional Generative Adversarial network is.

When using GAN we have no control on the output. This is why we can use the conditional GAN, where the generator and the discriminator models are conditioned on the class label (or different type of data). In our case, for example, the different labels are the hand writing digits generated.

Conditional Generative Adversarial Networks allow GANs to be extended as conditional models where both the generator and discriminator are conditioned on some additional information y. The information, y, is fed to the generator joint to the prior noise, pz(z), whose combination is flexible due to the adversarial nature of the process.

Regarding the discriminator, the data x and the information y are given, as the new mapping function G(z|y) (z given y), and a new discriminative model D(x|y) creates the outputs. The conditional min max value function, minG maxD V(D,G), replaces the previous functions (G(z) and D(x)) with the conditional ones (G(z|y) and D(x|y)):

minmaxV(D,G) = Ex~pdata(x)[logD(x)]+Ez~pz(z)[log(1−D(G(z)))].

2. **Create your own Conditional Generative Adversarial Network to generate conditioned samples in the Fashion Mnist dataset.**

Refer to notebook: 03_cdcgan_fashion_mnist_GroupF.ipynb

```
# make your own generator model
def make_generator_model(n_classes):
 inputs = tf.keras.Input(shape=(1,))
 lat = tf.keras.Input(shape=(100,))

 li = layers.Embedding(n_classes, 50)(inputs)
 li = layers.Dense(7*7, use_bias=False)(li)
 li = layers.Reshape((7, 7, 1))(li)
  g = layers.Dense(128*7*7, use_bias=False)(lat)
 g = layers.LeakyReLU()(g)
 g = layers.Reshape((7, 7, 128))(g)
  merge = layers.Concatenate()([g, li])
  g = layers.Conv2DTranspose(128, (4,4), strides=(2,2),
padding='same')(merge)
 g = layers.LeakyReLU()(g)
  g = layers.Conv2DTranspose(64, (4,4), strides=(2,2),
padding='same')(g)
 g = layers.LeakyReLU(alpha=0.2)(g)

 outputs = layers.Conv2D(1, (7,7), activation='tanh',
padding='same')(g)
 model = tf.keras.Model(inputs=[lat, inputs], outputs=outputs)
  return model
```

```
#make your own discriminator model
def make_discriminator_model(n_classes):
 labels = tf.keras.Input(shape=(1,))
 images = tf.keras.Input(shape=(28, 28, 1))
  li = layers.Embedding(n_classes, 50)(labels)
 li = layers.Dense(28*28)(li)
 li = layers.Reshape((28, 28, 1))(li)
  merge = layers.Concatenate()([images, li])
  fe = layers.Conv2D(128, (3,3), strides=(2,2), padding='same')(merge)
 fe = layers.LeakyReLU()(fe)
  fe = layers.Conv2D(128, (3,3), strides=(2,2), padding='same')(fe)
 fe = layers.LeakyReLU()(fe)
  fe = layers.Flatten()(fe)
 fe = layers.Dropout(0.3)(fe)
  outputs = layers.Dense(1, activation='tanh')(fe)
 model = tf.keras.Model(inputs=[images, labels], outputs=outputs)
  return model
```

For the final GIF, refer to: dcgan_FashionMnist_GroupF.gif