Deep learning - Final project report

Generative Adversarial Networks and Cycle-GAN

VU Thi Van Anh

École nationale de la statistique et de l'administration économique
ENSAE ParisTech
université PARIS-SACLAY

This assignment gives a brief bibliographical review in both papers of the DC-GAN and the Wasserstein GAN, and then summarizes a more advanced architecture that combines two GANs the CycleGAN. The second part of this paper to describe in short the implementation of CycleGAN and result in using this GAN in translating images from the MNIST dataset to the USPS dataset.

1 Brief theoretical summary

1.1 DC-GAN

This section introduces briefly the model called DC-GAN (Deep Convolutional GAN) proposed by Radford et al. [1]. In stead of max-pooling or fully connected layers, the original DC-GAN architecture composes four convolutional layers for the discriminator and four fractionally-strided convolutions layers for generator.

The generator maps the noise variable $z \in R^{100}$, a length-100 vector, to a RGB image with width and height to be 64×64. The basic block of the generator contains a transposed convolution layer followed by the batch normalization and ReLU activation. The transposed convolution layer here uses a 4×4 kernel, 1×1 strides and zero padding, so a input size of 1×1 will give an output with its width and height increased by 3 respectively.
The generator consists of four basic blocks that increase input's both width and height from 1 to 32. At last, a transposed convolution layer is used to generate the output, further doubles the width and height to match the desired 64×64 shape, and reduces the channel size to 3 . The tanh activation function is applied to project output values into the $(-1, 1)$ range.

The discriminator is a normal convolutional network network except that it uses a leaky ReLU as its activation function. Given $\alpha \in [0, 1]$, its definition is:

$$\text{Leaky ReLU}(x) = \begin{cases} x \text{ if } x > 0 \\ \alpha x \text{ otherwise} \end{cases}$$

Obviously, it is normal ReLU if $\alpha = 0$, and an identity function if $\alpha = 0$. For $\alpha \in (0, 1)$, leaky ReLU is a nonlinear function that give a non-zero output for a negative input. It aims to fix the "dying ReLU" problem that a neuron might always output a negative value and helps the gradients flow easier through the architecture.

The basic block of the discriminator is a convolution layer followed by a batch normalization layer and a leaky ReLU activation. The hyperparameters of the convolution layer are similar to the transpose convolution layer in the generator block.

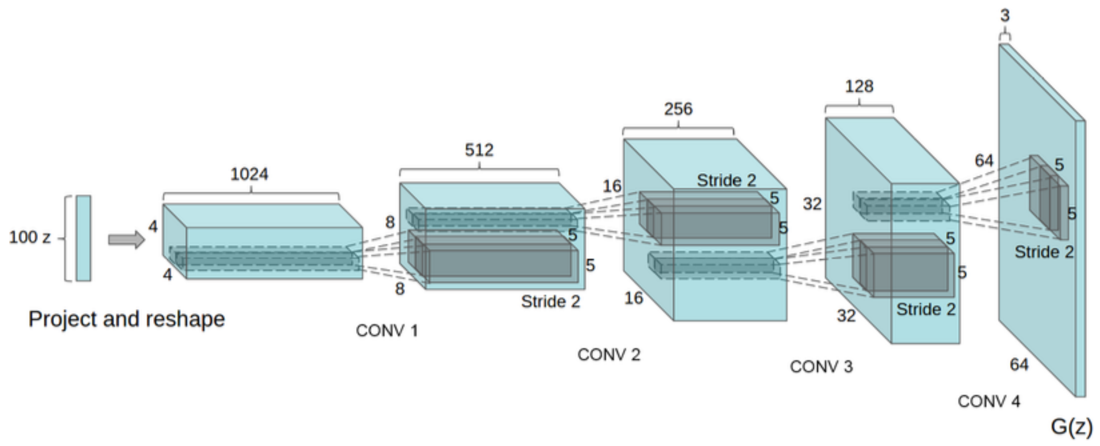

Figure 1: DCGAN generator used for LSUN scene modelling

1.2 Wasserstein-GAN

The paper of Wasserstein GAN proposed by Arjovsky et al, 2017 [2] opens with a lengthy explanation of Wasserstein metric's advantages over other common statistical divergences such as the *Total Variance (TV)* distance, the *Kullback-Leibler (KL)* divergence and the *Jensen-Shannon (JS)* divergence. The first theorem shows that, out of JS, KL and Wasserstein distance, only the last one give the better guarantees in continuity and differentiability which are both necessary in a loss function. Further, the second theorem proves that every distribution that converges under the KL, reverse-KL, TV, and JS divergences also converges under the Wasserstein divergence and a small earth mover distance corresponds to a small difference in distributions. In conclusion, Wasserstein distance is a compelling loss function for generative models. The Earth-Mover (EM) distance or Wasserstein-1:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

where $\prod(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$.

The basic idea behind the Wasserstein GAN is to minimize the Wasserstein distance between the sampling distribution of the data and the distribution of images synthesized using a deep generator.

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \le 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathrm{E}_{x\sim\mathbb{P}_\theta}[f(x))]$$

where the supremum is taken over all 1-Lipschitz functions. So to calculate the Wasserstein distance, we just need to find a 1-Lipschitz function f and we can build a deep network to learn it like other deep learning problem. Indeed, this network is similar to the discriminator **D**, but without sigmoid function and outputs a scalar score rather than a probability. That's a reason why the authors tend to call f_w the critic instead of the discriminator while it's not explicitly to classify inputs as real of fake images.

	Discriminator/Critic	**Generator**
GAN	$\nabla_{\theta_d} \frac{1}{m} \Sigma_{i=1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$	$\nabla_{\theta_d} \frac{1}{m} \Sigma_{i=1}^{m} [log(D(G(z^{(i)})))]$
WGAN	$\nabla_{w} \frac{1}{m} \Sigma_{i=1}^{m} [f(x^{(i)}) - f(G(z^{(i)}))]$	$\nabla_{\theta_d} \frac{1}{m} \Sigma_{i=1}^{m} [f(G(z^{(i)}))]$

While the original GAN [4] showed that in the limit, the maximum of the discriminator objective above is the Jenson-Shannon divergence, on the other hand, in WGAN, it's the Wasserstein distance instead. Obviously, while both traditional GAN and WGAN identify the distributions of real and fake, the traditional GAN discriminator does so in a way that makes gradients vanish over most of the space. In contrast, WGAN critic provides a reasonably nice gradient over all parts of space. Finally, in the experimental section, the authors compared the DCGAN baseline to WGAN, and on the dataset of bedroom, WGAN performs about as well as the Wasserstein loss seems to correlate well with image quality. And even though, the batch normalization is removed in DCGAN, WGAN can still perform.

1.3 Cycle-GAN

The CycleGAN (Zhu et al, 2018 [3]) is a technique that involves the automatic training of image-to-image translation models without paired examples, such as Monet paintings to photos, zebras to horses, etc. Its architecture contains two generators and two discriminators, as shown in Figure 2 below. The two image domains are denoted as X and Y. The generator G takes an image from X as input and generate a realistic image in Y which tricks the discriminator D_X. The generator F, similarly, generates an image in reverse direction and tries to misleads discriminator D_Y

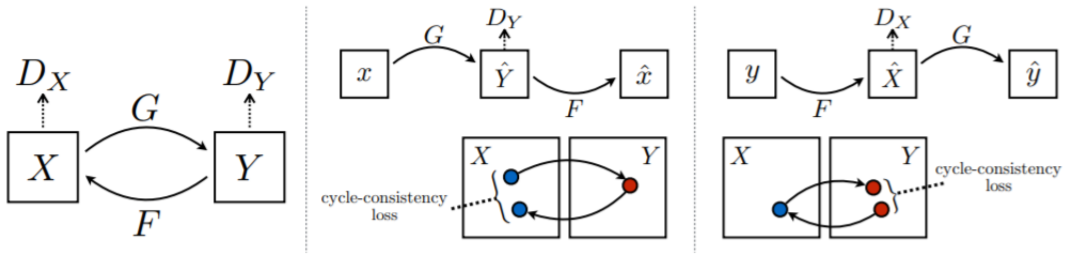

Figure 2: cycleGAN architecture and cycle consistency (Zhu et al, 2018 [3])

Like usual GAN settings, the discriminators encourage generators to output realistic images using the GAN loss:

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y\sim p_{data}(y)}[logD_Y(y)] + \mathbb{E}_{x\sim p_{data}(x)}[logD_Y(G(x))]$$

The notion of cycle consistency in Cycle GAN is proposed in order to train with unpaired data. While in original GAN, the generator becomes better by getting feedback from its perspective discriminator, in the case of CycleGAN, each generator gets an additional feedback from the other generator. This feedback ensures that a generated image is cycle consistent, this means that applying consecutively both generators on an image should yield a similar image ($x \to G(x) \to F(G(x)) \approx x$ and $y \to F(y) \to G(F(y)) \approx y$). The cycle consistency loss is:

$$\mathcal{L}_{cycle}(G, F, X) = \mathbb{E}_{x\sim p_{data}(x)[\|F(G(x))-x\|_1]}$$

Putting the two losses together, the full objective for CycleGAN is:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda\mathcal{L}_{cycle}(G, F, X) + \lambda\mathcal{L}_{cycle}(F, G, Y)$$

2 CycleGAN implementation

Dataset

The main objective of this project in to translate from the MNIST dataset to the USPS dataset. While the dimension inputs of MNIST and USPS are 28×28 and 16×16, respectively, so I decide to use the input dimension of 16×16 for both generator and discriminator. As all data set contains black-white images, random cropping is used here for data augmentaion. More details, all images are resized to dimension of 18×18 and then randomly cropped to 16×16. This ensure to remain information from handwritten digits. Furthermore, all pixels are normalised in range $[-1, 1]$ as Tanh activation function is used in the last layer of Generator like usual GAN.

Generator and Discriminator

Both generator and discriminator are built based on base layers DownSampling and UpSampling (inherited from the idea of U-Net architecture). The DownSampling layers is a set of 3 layers: Convolution2D(fix kernel = 4, strides = 2, padding = 'same') $\rightarrow$ LeakyReLU (alpha = 0.2) $\rightarrow$ InstannceNormalization. Similarly, the UpSampling layers contains 3 layers: Conv2DTranspose(fix kernel = 4, strides = 2, padding = 'same') $\rightarrow$ ReLU $\rightarrow$ InstanceNormalisation. Therefore, we have generator as below.

Generator Layer	**Parameters**	**Output shape**
Images		$16 \times 16 \times 1$
DownSampling 1	Filters = 32	$8 \times 8 \times 32$
DownSampling 2	Filters = 64	$4 \times 4 \times 64$
DownSampling 3	Filters = 128	$2 \times 2 \times 128$
DownSampling 4	Filters = 256	$1 \times 1 \times 256$
UpSampling + concatenate with DownSampling 3's output	Filters = 128	$2 \times 2 \times 128$
UpSampling + concatenate with DownSampling 2's output	Filters = 64	$4 \times 4 \times 64$
UpSampling + concatenate with DownSampling 1's output	Filters = 256	$8 \times 8 \times 32$
Conv2DTranspose	Filter = 1; kernel = 4 strides = 2; padding = 'same' activation = 'tanh'	$16 \times 16 \times 1$

And the discriminator as follow.

Discriminator Layer	**Parameters**	**Output shape**
Images		$16 \times 16 \times 1$
DownSampling 1	Filters = 64	$8 \times 8 \times 64$
DownSampling 2	Filters = 128	$4 \times 4 \times 128$
DownSampling 3	Filters = 256	$2 \times 2 \times 256$
DownSampling 4	Filters = 512	$1 \times 1 \times 512$
Conv2D	Filter = 1 kernel = 4; strides = 2 padding = 'same'	$1 \times 1 \times 1$

Losses

In this project, I use MSE for adversarial loss and discriminator loss. I had tried CrossEntropy however, the speed is quite slow in compared with MSE. Cycle loss and identity losse are MAE as described in the original paper [1]. So, the total generator loss is the sum of adversarial loss, cycle loss and identity loss with the weights among these losses is 1:1:10 respectively.

Optimizers used for both generator and discriminator are Adam with $lr = 2e{-}3, beta\_1 = 0.5, beta\_2 = 0.999$

3 Result and conclusion

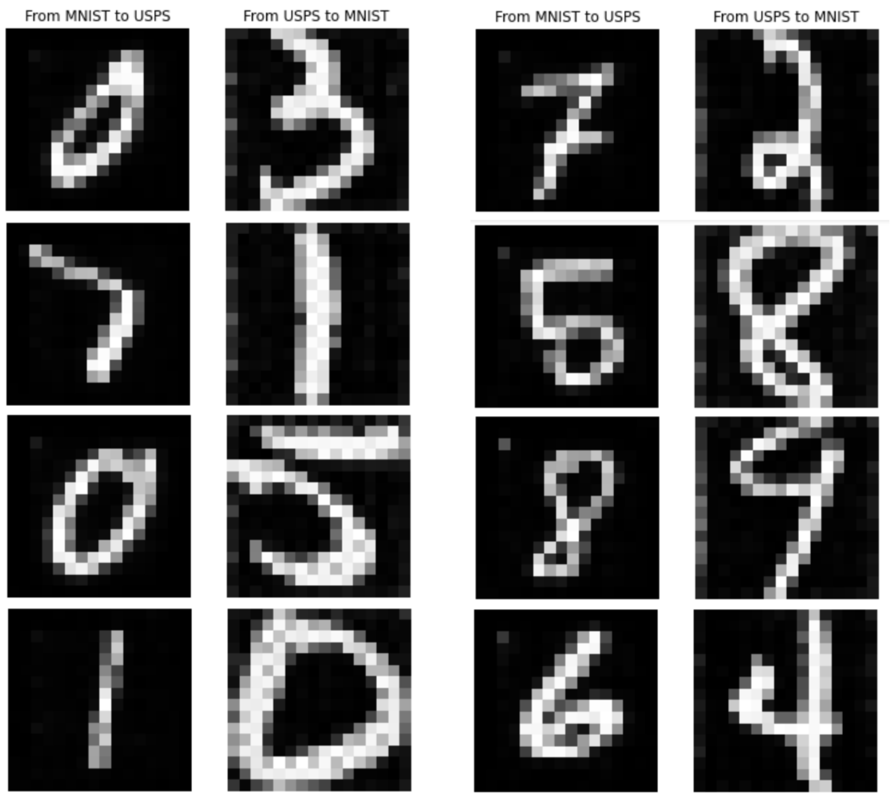

Figure 3: MNIST to USPS and USPS to MNIST

During the training phase, the generator loss decreases as we expected, this means that the generated images are much similar to true images. On the other hand, the discriminator loss does not change much. This can be explained by the considerable similarity data structure between MNIST and USPS datasets (same distribution because both of them are handwritten digits - discriminate true images from generated images which are created from similar images).

In conclusion, this project is to re-implement Cycle GAN trained on MNIST and USPS datasets. However, the result from these data sets is not much satisfied in comparing with result presented in the original paper where the authors trained on color RGB images data set such as horse2zabra, orange2apple, etc. (The future work we may do here includes to implement activity regularization loss and to try learning weights decay, however, due to limited time, I have not done this).

References

[1] A. Radford , L. Metz , S. Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* arXiv preprint arXiv:1511.06434. 2015.

[2] M. Arjovsky, S. Chintala, L. Bottou. *Wasserstein GAN.* arXiv preprint arXiv:1701.07875. 2017

[3] JY. Zhu, TS. Park, P. Isola, AA. Efros. *Unpaired image-to-image translation using cycle-consistent adversarial networks* Proceedings of the IEEE international conference on computer vision, Page 2223 - 2232. 2017

[4] IJ. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, AC. Courville, Y. Bengio. *Generative Adversarial Networks.* arXiv preprint arXiv:1406.2661. 2014.