Overview of Generative Adversarial Network

Charmettant Benoit - 960121T072 - benoitch@kth.se
Laures Benoit - 960529T153 - blaures@kth.se
Sevestre Pierre - 960420T153 - sevestre@kth.se

Abstract

Generative models are a promising field towards leveraging high dimensional and complex probability distribution that shape the world that surround us. Richard Feynmann resumed this in his quote *"What I cannot create, I cannot understand"* [1]. It enables for instance highly competitive algorithm of semi-supervised learning [2]. One of the main field concerned by these models is computer vision where generating natural real-looking images has been a fundamental problem for many years. One architecture of network was recently able to tackle this problem with outstanding results : Generative Adversarial Networks (GANs). It is this kind of network that chose to study in this project, and apply it to realistic image generation using different data sets with different complexity : MNIST [3] and CIFAR10 [4]. We implemented different versions of GANs we could find in the literature with increasing complexity (from a fully connected model with two layers to a deep convolution generative model) aiming at generating more and more realistic images. One of our goals in this study of GANs was to better understand the parameters influencing the generation quality of these networks as well as their training performance.

1 Introduction

Generative Adversarial Models were introduced in 2014 by Ian Goodfellow [5], paving the way to a whole new branch of research in generative model. This model is based on an adversarial process, between a generator G that generates images to match the original data distribution, and a discriminator D that estimates the probability of input images being generated or real. This minimax game, although asymptotically consistent, has proven to be difficult to train. New architecture were developed to tackle this issue, some of then showed great results, such as [6] and [7], who created DCGAN, that we relied on for this project. We tried to explore this field of generative networks and these different architectures with the aim of understanding them better, especially by focusing on these different problems :
- Implementing different architectures of GANs, from scratch and using the Tensorflow API, with increasing complexity, customized with ideas that proved to improve the training of such kinds of models. By improving the network step by step we better understood the influence of diverse parameters on the network's performances.
- Generating images over different datasets and trying to assess the quality of the generated images, not only visually but using a recently introduced metric, the inception score.

2 Background

As explained above, GANs appeared in Goodfellow's work in this article [5] but have been the object of a great amount of works since then. Here is a short summaries of notable improvement on the orgininal model that we might have implemented or not.

2.1 Original GANs

The general structure of GANs was introduced 4 years ago, it relies on two networks competing in a minimax game : a Generator G, generating images from noise; and a Discrimator D, classifying weather his input is from the generator or the dataset. This game is resumed in the equation below:

$$\min_G \min_D V(G,D) = E_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + E_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

Where D(**x**) represents the probability that **x** comes from the data and G(**z**) is the image generated by a noise vector **z**.

The training consists in two parallel steps: first a batch of noise is sent to the generator to create false images, these images along with real data taken from the database are sent to the discriminator so that it computes the probabilities of each image as being real data. These probabilities are then used to compute the cost functions of both parts: first, the loss of the discriminator is the usual cross-entropy loss (where label of the real data is 1 and 0 for the fake data):

$$J^{(D)} = \frac{1}{m}\sum_{i=1}^{m}[\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)})))]$$

The generator uses the output probability of the fake images it sent to the discriminator to update its parameters. Its loss is also the cross-entropy loss:

$$J^{(G)} = \frac{1}{m}\sum_{i=1}^{m}\log(1 - D(G(\mathbf{z}^{(i)})))$$

Finally, the gradients are computed using backpropagation, and both networks can be updated using the usual techniques (gradient descent...).

This game has a theoretical global optimum for $p_g = p_{data}$ where D is unable to discriminate between real and generated data (because it has been shown that D converges to

$$D^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}$$

hence D=0.5).

However this optimum situation is very difficult to achieve. In fact, this first version of the network

- with only fully connected layers - suffers important flaws such as a "collapsing" mode where the generator emits the same image-point **x** for all values of **z** (this issue is very common in the training of GANs and we faced it during the project). As it shows some promising results on simple data sets like MNIST, it performs worse on more complicated colored ones like CIFAR10. Different works have tried to improve this network and deal with its flaws, these works are presented below.

2.2 Deep convolutional GANs

One big step forward towards generating more realistic images with datasets of colored and larger images was the introduction of deep architectures with convolutional layers DCGANs by [7]; all these improvements were implemented in this project. First, they didn't use any fully connected layers on top of convolutional layers, they just use one in the generator to project the input noise to a higher dimension (Figure 1) because they found out that the network performs better and it speeds up the convergence. Second, instead of using pooling layers to increase or decrease the dimensions, they rather used strided convolutional and transpose strided convolutional layers respectively in the discriminator and the generator so that both networks learn their own representation of downsampling and upsampling. Third, they applied batch normalization (which didn't exist for the first version of the GANs in 2014) to most layers which mainly prevented the generator, among other things, from collapsing into a single output. Finally, they used ReLu activation function in the generator, except its output where tanh seems to bring better results, and LeakyReLu in the discriminator, except the last layer where sigmoid is used to ouput the probability of the image being real.

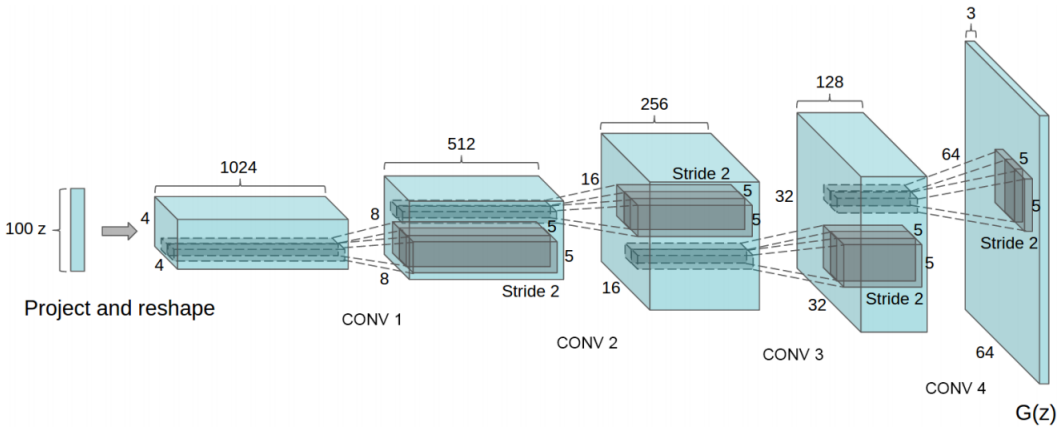

Figure 1: DCGAN architecture of the generator from [7]

2.3 Other models and improvements

We also found many tricks on the Internet that may facilitate the network convergence and its performance [8], along with some papers [9], and implemented some of them to analyse their influence. Particularly, we tried to sample the noise from a Gaussian distribution rather than a uniform one, flipped labels (a small proportion of fake images are sent to the discriminator but with the same label as real images and vice versa: this is useful to make sure the discriminator doesn't get too strong too quickly), training one network more than the other and using dropout. The results are presented later.

2.4 Evaluation

Generative models suffer from the lack of metrics that can evaluate their performances. Especially for GANs, it is only possible to draw samples from p_G, and not possible to access the density $p_G(x)$ that is usually used to compute the likelihood of the dataset given this density. Salimans et al.thus introduced [9] the Inception score, using a pretrained neural network [10] to assign a class to the generated images out of thousand class available on ImageNet [11]. Having $p(y \mid x)$, the formula for the inception score is then motivated by two assumptions. First, this generated image is expected to be easily recognizable, and thus have a high posterior over a small number of labels from ImageNet, meaning the entropy of $p(y \mid x)$ should be small. Then, multiple generated images are expected to be diverse, ie should be assigned to multiple different labels, which means $\int p(y \mid \mathbf{x} = G(\mathbf{z}))\mathrm{d}z$ should have a large entropy. These assumptions incite using KL divergence, $D_{KL}(p(y \mid x)||p(y)) = \int p(y \mid x) \log(\frac{p(y|x)}{p(y)})dy$, that can be decomposed between the cross-entropy of the two densities and the entropy of the marginal : $D_{KL}(p(y \mid x)||p(y)) = H(p(y), p(y \mid x)) - H(p(y))$. Hence, based on our assumptions, a good inception score should correspond to a high value of KL divergence. [9] decided, so that it would be easier to compare, to take the exponential of the previous KL divergence, resulting in : $IS = \exp(E_x D_{KL}((p(y \mid x)||p(y))$.
The estimate we create by computing the marginal probability over a limited number of sample is slow to converge to the inception, as the original paper suggests to use 50k images. Because of computational limitation, we used a smaller amount of images, 1000, to track the inception score during the training.
Furthermore, this metric show limitation, as [12] found most model are able to reach similar inception score with an optimal number of hyperparameter. [13] gives very explicit proof of the shortcomings of inception score over simple example. One conclusion is that the Inception Network should to trained on the same dataset used to generate images. Aware of this flaws, we tried for this project to compare the inception score of our datasets with the score from the generated images. This score for the CIFAR10 dataset limited to the horse label is 3.16, with a standard deviation of 0.40.

3 Approach

Based on all of the information we could gather from these papers we tried to implement our own versions of GANs with Tensorflow. Here is what we did and how we did it. The code that join to this report is modular so it is easy to switch between the different kinds of networks we implemented (3 in total) and turn on and off different features such as distribution of the noise, learning strategies, or final activation to see how they influence results.

3.1 Models

The two first models are simpler models, made mainly to carry out preliminary tests faster, and that could be run on our own computers, yet they proved satisfactory results. We first describe their architectures below, note that "final activation" in the last layer of the generator is either sigmoid or tanh.

3.1.1 Simple model

The first version we implemented was very basic as it's a simpler version of vanilla GANs, with only 2 fully-connected layers in each network. I enabled us to rapidly have some first results especially with the MNIST dataset. Here is a description of the architecture. In the following table the final activation of the final layer (referred as final activation) can be either sigmoid or tanh. Both solvers

Table 1: Simple model

Discriminator	Generator
FC.128 - RELU	FC.128 - RELU
FC.1 - sigmoid	FC. image dimension - final activation

are Adam optimizers with default parameters (learning rate=0.001, β_1=0.9, β_2=0.999, ε=1e-08).

3.1.2 Intermediate model

The second model we implemented, still light enough so that it was possible to run it on a computer with a GPU was a simple version of DCGAN and inspired by the architectures suggested in the Infogan paper [14] where they use light DCGANs. The model is as follow: This model enabled us

Table 2: Intermediate model architecture

Discriminator	Generator
5 x 5 conv. 64 lRELU. stride 2	FC. height*width*depth*8 RELU. batchnorm
5 x 5 conv. 128 lRELU. stride 2 batchnorm	5 x 5 upconv. 64 RELU. stride 2 batchnorm
FC. 128 lRELU. batchnorm	5 x 5 upconv. depth img. final activation. stride 2
FC. 1 sigmoid	

to carry out a lot of preliminary tests so as to check which improvements were the most interesting to try on the complete model (since it's already a DCGAN), as well as debugging the code faster. Both solvers are Adam optimizers with parameters: learning rate=0.002, β_1=0.5, β_2=0.999, ε=1e-08.

3.1.3 Complete DCGAN

Below is the architecture of the complete and heavy DCGAN that can only be run on dedicated GPU (Nvidia Tesla K80 was used): As advised in the DCGAN paper, both solvers are Adam optimizers with parameters: learning rate=0.0002, β_1=0.5, β_2=0.999, ε=1e-08. From this "raw" DCGAN architecture, we then added some features and tried to understand their influence as well as evaluated the performance

[1] Depending on the shape of the data, the strides and output shape of the FC layer may change.

Table 3: Complete DCGAN architecture

Discriminator	Generator
5 x 5 conv. 64 lRELU. stride 2	FC. height/8*width/8*1024 RELU.[1]
5 x 5 conv. 128 lRELU. stride 2	5 x 5 upconv. 512 RELU. stride 2[1]
5 x 5 conv. 256 lRELU. stride 2	5 x 5 upconv. 256 RELU. stride 2[1]
5 x 5 conv. 512 lRELU. stride 2	5 x 5 upconv. 128 RELU. stride 2[1]
FC. 1 sigmoid	5 x 5 upconv. image depth. final activation. stride 2[1]

3.2 Features

We implemented some of the improvements described in 2.2 and 2.3 on the complex model and evaluated their performance both visually and by computing the inception score. Note that thanks to our modular code, all these features can be easily controlled directly at the creation of the DCGAN object or when calling its `train()` method, thus without having to change manually the whole code. Below are the features we:

- Final activation function: sigmoid or tanh
- Noise type: uniform or gaussian
- Dropout
- Flipped labels: 10% of fake (real) images are fed to the discriminator as being real (respectively fake)

4 Experiment & results

Our goals for this project were first to produce some realistic images using the best GANs we could implement for different datasets (MNIST, CIFAR10-horses, a dataset of pokemon pictures); and also to compare the influence of different features we added using the inception score as an objective metric.

4.1 Generating realistic images

We display and compare here the images generated by the different networks over the MNIST and CIFAR10 (using only the picture of horses).

4.1.1 MNIST

First let's take a look at the images generated by the 3 networks (simple, intermediate and DCGAN) over the MNIST dataset. As a mean of comparison, a set of original digits is displayed in the appendix.

This dataset is rather simple to reproduce as the images are small (28x28) and in black and white (hence the depth is only 1), but also because there are a huge number of available images the networks can train on and they are rather close to each other (all digits are centered, background of the same color...). Hence, even the simplest model, which is very fast to execute, is able to reproduce fairly well the digits and some look real. However, the DCGAN's superior ability at generating real images can be appreciated here as the other two networks display more realistic digits (the noise around the digits has disappeared) and converge much quicker. Moreover, for the same number of epochs, it is obvious that the most complex architecture generates better images which can clearly be mistaken for human-made digits !

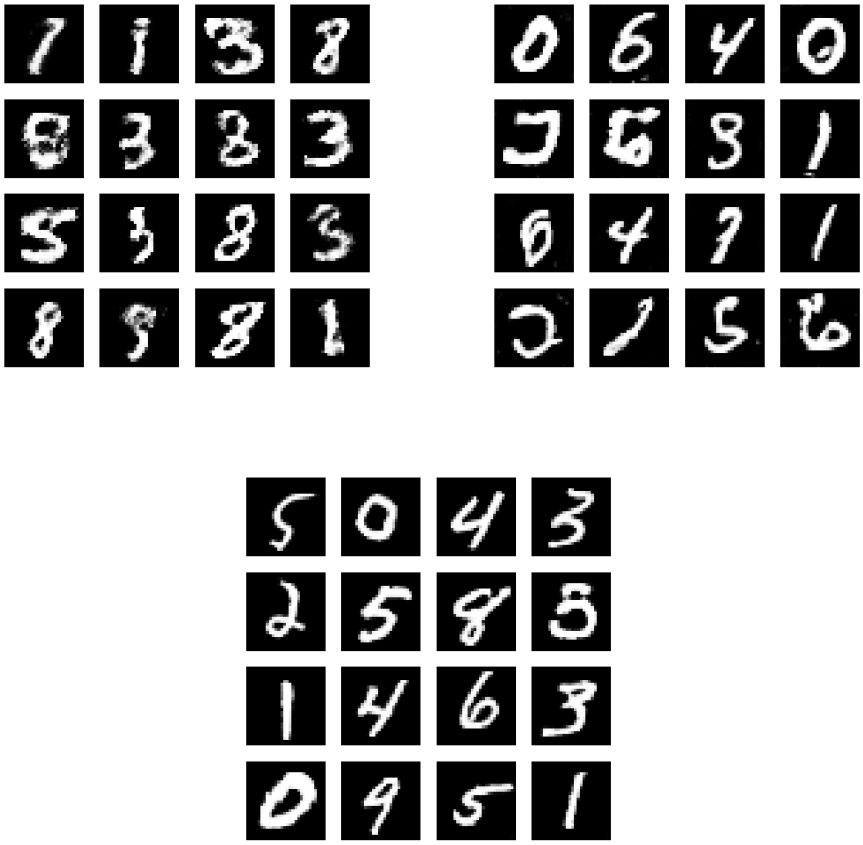

Figure 2: Generated images with simple model (up left) intermediate model (up right) and complete DCGAN architecture (middle)

4.1.2 CIFAR10 horses

Now let's have a look on the images generated from the CIFAR10 pictures labeled as horses 3. We chose not to present the images generated by the simple model over this data set because of their poor quality. Indeed, it is too simple to produce satisfying results with data of this complexity as the images are bigger than MNIST's but more especially in color and very diversed. Again, samples from the real dataset are displayed in the appendix.

We can notice on those two samples 3 that the pictures generated are not blurry, which could be expected from DCGANs. Some pictures from the samples are quite convincing and few of them are even representing the horse-rider! By comparing those two samples we were quite surprised by the quality of the images produced by the intermediate model, which takes significantly less time to train (almost three times faster); nevertheless, the most realistic images come from the complete DCGAN model whereas many images coming from the intermediate model look like paintings.

4.2 Study of the influence of the different features

In order to build an even better network, as previously explained, we implemented different improvements and features to play with and we tried to judge their influence. Besides evaluating the visual appearances of the generated images, we took into consideration the inception score computed over 100 images of horses generated by the model. We took as baseline the inception score we got from a raw DCGAN (as described in 3.1.3 trained for 100 epoch with the following features :

Figure 3: Generated images after 150 epochs with intermediate model (left) and complete DCGAN (right)

- Noise : dimension = 100, distribution = Gaussian
- Activation function at last layer of the generator : tanh
- Flip labels : disabled
- Dropout : none
- Learning strategy : D and G are updated at each batch

Then we trained different models where one of these features was changed and compared the results. We can for instance compare the pace with which the model converges and the level of inception score it can reach asymptotically. Here are the different feature we tested (the base line curve is referenced as base in legend) :

- Enabling a 10% flip label (referenced as flip)
- Dropout : 0.5 for the discriminator, 0.2 for the generator (referenced as drop)
- Noise distribution : uniform (referenced as lin)
- Final activation function of the generator : sigmoid (referenced as sig)

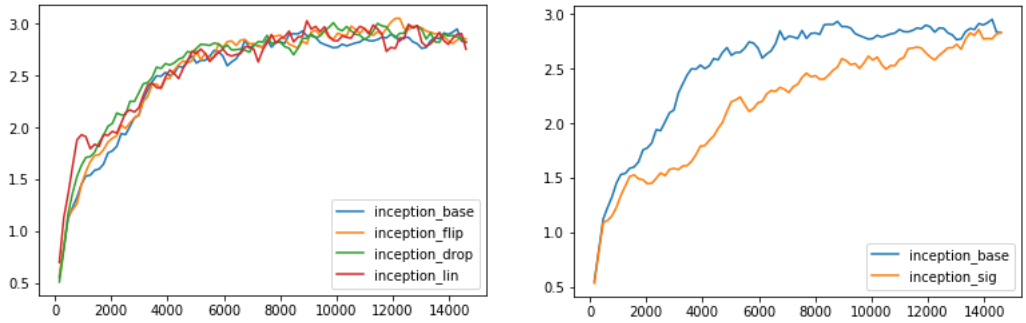

Figure 4: Evolution of the inception score as a function of the number of iterations for different models. The different curves have been smoothed to be interpreted more easily.

We can see on the curves that most of the implemented features do not have any significant impact on the learning performances of the model (having in mind all the inherent flaws of the inception score and the high variance of our estimate). When taking a look at the images generated by those different models it appears that they are really close which confirms our assumptions. However one difference is significant enough to be noticed. The learning pace of the model with a sigmoid activation function on the output layer of the generator is lower than with a tanh activation: hence the tanh version converges much quicker than the sigmoid version. Moreover, even though the inception scores seem to get closer around 14,000 steps, the images look very different. Just as shown below, the images generated with the sigmoid activation function are blurrier and under-saturated making them look less realistic.

Figure 5: Horses generated after 100 epochs by the DCGAN with sigmoid as the output activation function of the generator

5 Conclusion

This project gave us the opportunity to work on a very recent but also promising field of deep learning.
Although being very unstable and rather hard to train, we successfully managed to implement a working version of DCGANS with Tensorflow; as well as understanding how to evaluate them and adding some improvements that were suggested throughout the years. Although the experiments we carried out on the complete DCGAN model didn't show very different and outstanding results, they proved different points. First, it is difficult to evaluate objectively the performance of a GAN as the inception score is far from being flawless. Secondly, a lot of fine-tuning is necessary in order to improve the results and even though some features we added didn't have any positive impact, they are worth the try. Besides, we were able to prove the superiority of the tanh activation function in the output layer of the generator over the sigmoid, as advised by the DCGAN paper, hence it may be interesting to try other activation function. Finally, having built different models with an increasing complexity was an interesting idea for this model and we think it is an idea worth to pursue in other machine learning projects as it enables comparison, better understanding of the model complexity needed to solve a problem but also to carry out some tests of possible improvements faster before eventually applying them on a more complex model.
However, we would have liked to try more diverse and also more complex datasets, for instance with bigger images or with more varied images, that would have required to add some promising features on the network to make it more efficient, like the batch discrimination [9] that seems auspicious. These ideas may be a starting point for a future work.

6 Appendix

Figure 6: Sample from original CIFAR10 horses

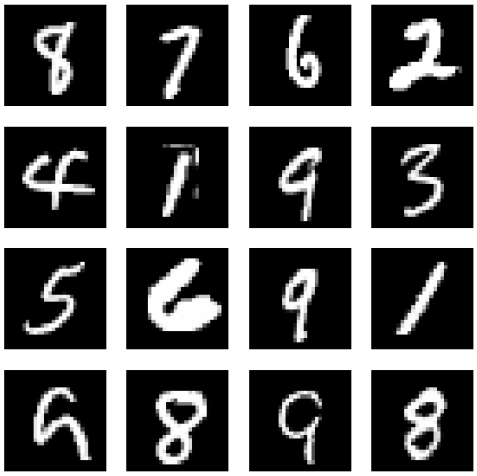

Figure 7: Samples from original MNIST dataset

References

[1] OpenAI. Generative models, Nov 2017.

[2] Diederik P. Kingma, Danilo Jimenez Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models. *CoRR*, abs/1406.5298, 2014.

[3] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[4] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[6] E. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. *ArXiv e-prints*, June 2015.

[7] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *ArXiv e-prints*, November 2015.

[8] Arjovsky M. Chintala S., Denton E. and Mathieu M. How to train a gan? tips and tricks to make gans work, Dec 2016.

[9] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[10] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *ArXiv e-prints*, February 2016.

[11] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[12] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. Are GANs Created Equal? A Large-Scale Study. *ArXiv e-prints*, November 2017.

[13] S. Barratt and R. Sharma. A Note on the Inception Score. *ArXiv e-prints*, January 2018.

[14] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016.