# LINMA2472 - Algorithms in Data Science

# Homework 1: GAN (CGAN in fact)

Bastien Massion - bastien.massion@uclouvain.be

25 September 2024 - v1

## Guidelines

Homework 1 is divided into 2 sections. The first part, given in the corresponding Notebook, introduces you to the design, the implementation in PyTorch and the training of your first Conditional GAN (CGAN). The goal is to generate handwritten digits (and letters) based on the MNIST dataset. The second part of the homework focuses on some analysis of the neural networks and on the exploration of the latent space.

You will have to make a report answering the questions and completing the tasks presented in this assignment. Make sure to answer to every question and every sub-question from this assignment (every item listed below). Note that a lot of additional indications are written in the Notebook. In your report, all answers should be concise and precise (max 10 lines). However, as several plots and example images are asked, your report could be somewhat long (up to 10 pages), but there is not too much to write. Finally, for some questions, you might be interested in looking up external resources (scientific articles, blogs, online implementations, ...). Don't forget to cite them and include a bibliography at the end of your report. Finally, using AI writing/coding assistant (such as Chat GPT) is allowed, but you absolutely have to detail where you use it (in your report and in your Notebook), which prompts were given and what answers you received.

As for the other assignments, you are making groups of 3, in the Moodle activity "Group choice for Homework 1". Register as soon as possible.

The deadline for the whole homework is: **Wednesday October 2024, 23:59**, in the "Homework 1 (GAN)" activity on Moodle.

Per group, you have to submit one zip file **group_#_Name1_Name2_Name3_HW1.zip** containing:

- Your report (in PDF format **.pdf**),
- Your Jupyter Notebook containing your codes (**.ipynb**),
- Savings of your trained models (generated with the saveModel function, see Notebook),
- A short READ_ME.txt file explaining what are the files in your zip useful for, how to run your codes and check your results easily.

Your questions or comments should be posted on the dedicated Moodle "Class forum" or kept for the Q&A session that will take place on: **Wednesday 9 October 2024, 08:30, in BARB93**. As a last resort, you can send them directly to the mail address: **bastien.massion@uclouvain.be.**.

# Introduction

Since their introduction in 2014 by Goodfellow et al. [3], Generative Adversarial Networks (GAN) have taken a predominant place in the deep learning landscape. GANs are deep neural networks composed of two parts: a generator and a discriminator. Those two parts are trained as opponents (hence "adversarial"). On one side, the generator tries, from random latent vectors, to create new fake data that can fool the discriminator into thinking they are really coming from the dataset. Meanwhile, the discriminator tries to distinguish between true and fake data.

Often, the most interesting part of this model is the generator, therefore we often refer to GANs as generative models. Together with diffusion models (such as Stable Diffusion or DALLE-3) for vision and with transformer models (such as GPT4) for language, these generative neural networks make up a large part of the AI world today. Experts think that we are currently living in the era of Generative AI. Lectures about some of those other topics are coming soon!

These generative properties are used in a lot of applications: mainly for image processing tasks (style transfer, segmentation, face generation, image inpainting, deblurring, super-resolution,...), but also for natural language processing tasks (text summarization, text generation,...), for music generation or even for medical tasks (tumor detection). Another major use of GANs is data augmentation, which can be successful in any data-driven sector for which collecting large annotated datasets is costly or risky (medicine, biology, psychology,...). [4] Of course, new applications emerge and perspectives unimaginable a few years ago could soon become reality.

Conditional GAN (CGAN), introduced in 2014 by Mirza and Osindero [6], is a variation of the classical GAN architecture that aims to give more information to the networks to improve and control the generated outputs. In particular, the generator and the discriminator both receive the label, i.e. the class, that should be generated or discriminated. Figure 1 illustrates the structural difference between GAN and CGAN. This improvement has been a huge milestone in the quest for text-to-image generation.

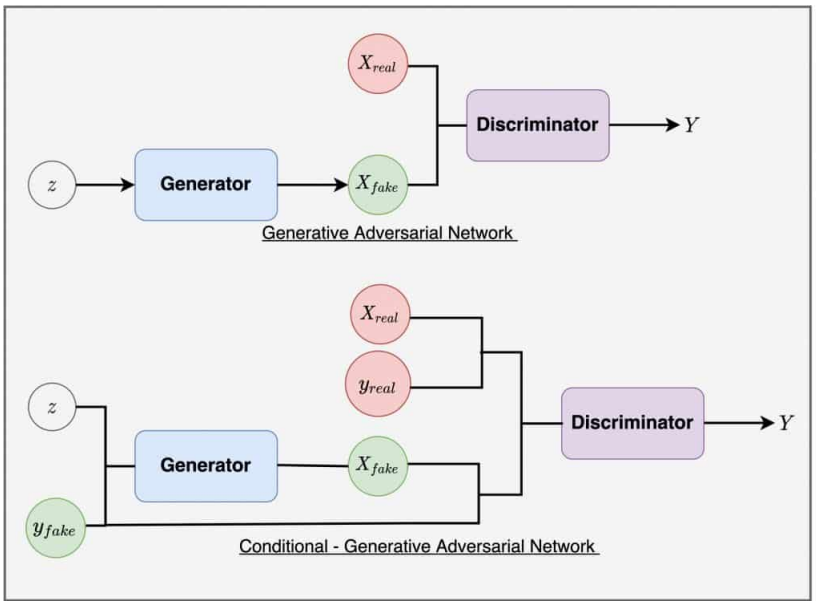


Figure 1: Architecture of classical GAN vs architecture of Conditional GAN [1]

The goal of this homework is to implement your first CGAN and explore some properties.

# Section 1. Train your CGAN on MNIST for image generation

## 1.1 Your CGAN model

- Describe your generator and discriminator architectures in great detail: number of layers, size of layers, types of layers, total number of parameters, dimension of the latent space for the generator, dimension of the label encoding, inputs and outputs. You should use the *torchinfo* package to help you. If you trained several CGANs, describe them all.

## 1.2 Training your CGAN

Train your network on the MNIST dataset (not on EMNIST Letters nor on EMNIST Balanced).

- Report the values of the optimization parameters (device used, number of epochs, batch size, algorithm/optimizer used, optimizer parameters, generator advantage discriminator advantage).
- Show new fake images generated by your generator.
- Compare them visually with real data samples.
- Show predictions made by your discriminator and comment briefly on your results.

## 1.3 Error metrics during training

- Show the evolution of the two main error metrics in function of the number of epochs: loss functions (for generator and for discriminator) and classification accuracy (for training and testing sets).
- Report the duration of the training as well.
- Discuss your results.

## 1.4 Improve your CGAN model

Suppose that you want to improve your CGAN model for digit generation according to some criterion (better outputs, faster training, more robust model or something else). As you way have encountered, proper GAN training is a very hard task. Fortunately, solutions besides simply tuning your hyperparameters have been developed.

- Propose 3 techniques that could be used and for which purpose(s) you would use them.
- Present them and cite the articles from the literature that describe the techniques.

## 1.5 BONUS: Comparison with and without improving techniques

In case you included some improving techniques in your training:

- Explain how you implemented them in practice.
- Compare the results with and without these techniques.

# Section 2. Explore the latent space

An important question about the generative part of GANs and their variations is the ability to generate what we want: the expressivity and the controllability of the generator. The expressivity relates to the ability of the model to generate any image: how close is the GAN to becoming a universal approximator? The controllability captures the ability of the GAN to generate exactly what the user wants from giving the right inputs. Those aspects are for example captured in the "GAN inversion" problem, which consists in finding the best inputs to feed our GAN with in order to produce an image as close as possible to a desired target image. This well-known problem turns out to be far from trivial. [7]

On the one hand, the diversity and the expressivity of the generation is handled by the latent space. Initially random, this feature space acquire meaningful structure during the training of the networks: this is really powerful, and somewhat magical. Thus, the latent space is a condensed space where the inherent features of images can be understood. Moreover, it gives insights into how features influence the generated image, how features are linked together, or how to search for better variations of an image in the immense space of possible images. On the other hand, CGAN makes a step towards controllable GANs, by adding an additional user-chosen input to guide the generation. Most often, the input is a label corresponding to a class, which is then represented into a more meaningful space: the label embedding space.

Therefore, understanding how those feature spaces are articulated and how to guide their structure is crucial and have lead to huge improvements in image generation. In this second part, you will explore the latent space and the label embedding to understand how your model creates a representation of the world and how it generates new images.

For your information, exploring the latent space is key to the incredible results of models such as StyleGAN2 [5]. This exploration is absolutely delightful, and I can warmly suggest you have a look at the amazing YouTube video "AI Art Evolution" by Emergent Garden [2] to be convinced. Even if the models in the video are not GANs, the ideas are similar and have been a source of inspiration for this project.

## 2.1 Diversity in each class

Generate some images of each class with your generator.

- Show some of them.
- Visually, are you satisfied with the diversity of results within each class?

Take a vector of zeros as your latent vector, i.e. the mean vector of your latent distribution (the mean of the standard normal distribution is the zero vector). Generate the corresponding images for each class.

- Show for each class the mean generated images.
- Compare the quality of these "mean" images with randomly generated ones and comment briefly.

## 2.2 Discrimination on each class

For each class, generate a lot of images (for example 500) with your generator and extract the same number of true data points with the same label from your dataset.

- Report the accuracy of your discriminator for each class. Is your discriminator stronger (or weaker) for some classes than for others?

- Count the percentage of predictions Fake/Real to see if the discriminator is biased.
- Are there more false positives or more false negatives? Why could it be?

## 2.3 Outside of label distribution

Neural networks do not like to work with categorical or integer entries, they prefer to work with continuous multidimensional inputs. This work is done by the *nn.Embedding* layer which takes an integer label as input and returns a continuous vector of predetermined dimension. Each label corresponds to one particular label-embedding vector. Importantly, the Embedding module refuses to receive anything else than one known integer label, as it raises an IndexError otherwise. This is unfortunate because we would love to discover what happens if we try to generate points with labels that are not present in the training set, as it could be crucial in several applications. For example, that could allow the generation of images with features from several distinct labels, or even with brand-new features not corresponding to any known label.

The easiest method to bypass this limitation and generate out-of-distribution labels is to directly generate embedding vectors which do not correspond to any actual label and feed them to the Generator, skipping the entire Embedding module. Note that even if those random intermediate vectors do not correspond to any actual label, they can be associated with one label by building a multi-class classifier in the Embedding space. In particular, a very naive 1-NN classifier (nearest neighbour) is implemented in the code. To any random embedding vector, the classifier assigns the label of the closest label-embedding vector in the embedding space (according to the Euclidean norm), making a guess on what image class such vector should produce.

Apply the method presented above.

- Do the out-of-distribution generated images visually make sense?
- From your human perspective, do those generated images suffer from a lack of diversity?
- Do the generated images visually correspond to the naive 1-NN classifier prediction?

## 2.4 Interpolation for exploration

The goal is to examine how digits transform one into the other according to your model.

First, within the same class, choose two latent vectors that you find visually satisfying. We call those latent vectors "extreme latent vectors". Create intermediate/interpolating latent vectors by interpolating linearly between the extreme latent vectors and generate their associated images.

- Show how one extreme image evolves into the other one by linear interpolation in the latent space.
- Compare the interpolants to the extremes, for 3 distinct classes. Are they all realistic and satisfying?

Secondly, choose two satisfying images from different classes and their associated extreme latent vectors. Compute the generator's embedding vectors corresponding to the two different labels and interpolate linearly both between the extreme latent vectors and between the embedding vectors.

- Show how one image evolves into the other one by linear interpolation in the extended latent space.
- Comment on your findings.

### 2.5 Noise for exploration

Generate "noise latent vectors" with $0 < \sigma$ from a normal zero-mean distribution, i.e. sampled in $\mathcal{N}(0, \sigma I_{\text{dim\_lat}})$. Add those noise vectors to the mean latent vectors to create "variation latent vectors", with the following values of $\sigma$: try $\sigma \in \{0.1, 0.5, 1.0, 1.5, 2.0\}$. These vectors are close variations of the mean latent vectors, i.e. they are neighbours in the latent space.

- Compare your variation latent vectors with the mean latent vector, in terms of labels and in terms of visual quality.
- Analyze the impact of the value of $\sigma$ on the variation latent vectors.

Repeat the same operation, but this time, add noise on the embedding vectors and not on the latent vector. The embedding noise vectors follow a normal distribution $\mathcal{N}(0, \sigma I_{\text{dim\_lab\_enc}})$, with $\sigma \in \{0.1, 0.5, 1.0, 1.5, 2.0\}$.

- Compare your variation embedding vectors with the mean latent vector, in terms of labels and in terms of visual quality.
- Analyze the impact of the value of $\sigma$ on the variation embedding vectors.

## References


[1] Aditya Sharma. Conditional GAN (cGAN) in PyTorch and TensorFlow, July 2021.

[2] Emergent Garden. AI Art Evolution, October 2022.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[4] Jonathan Hui. GAN — Some cool applications of GAN, March 2020.

[5] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN, March 2020. arXiv:1912.04958 [cs, eess, stat].

[6] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets, November 2014. arXiv:1411.1784 [cs, stat].

[7] Weihao Xia, Yulun Zhang, Yujiu Yang, Jing-Hao Xue, Bolei Zhou, and Ming-Hsuan Yang. GAN Inversion: A Survey, March 2022. arXiv:2101.05278 [cs].