Sylvain Combettes

Master of Science candidate
Ecole des Mines de Nancy
Department of Applied Mathematics

sylvain.combettes [a t] mines-nancy.org
https://www.linkedin.com/in/sylvain-combettes
https://github.com/sylvaincom

Internship report | Generative Adversarial Networks (GANs)

June 24th – September 13th, 2019

Topic:	Generating fictitious realistic patient data using GANs (generative adversarial networks)
Internship supervisors:	Fabrice COUVELARD Romain GUILLIER
Company:	Servier 50 Rue Carnot, 92284 Suresnes
Department:	*Pôle d'Expertise Méthodologie et Valorisation des Données (PEX MVD)*

Abstract

In the first chapter, we do a general presentation on GANs, in particular how they work. GANs are a revolutionary generative model invented by Ian GOODFELLOW in 2014. The key idea behind GANs is to have two neural networks competing against each other: the generator and the discriminator. GANs can synthesize samples that are impressively realistic.

In the second chapter, we apply GANs to patient data. The method is called `medGAN` (for medical GAN) and was developed by Edward CHOI in 2018. `medGAN` can only synthesize binary or count values. There are two main applications of `medGAN`: privacy and dataset augmentation. We only focus on dataset augmentation from a real-life dataset: we generate fictitious yet realistic samples that can then be concatenated with the real-life dataset into an augmented dataset (that has more samples). Training a predictive model on the augmented dataset rather than the real-life dataset can boost the prediction score (if the generated data is realistic enough). All the programs can be found on my GitHub.

Contents

Acknowledgments

I would like to thank Fabrice COUVELARD and Romain GUILLIER, my internship supervisors, for this great experience. I was able to learn a lot from them. I thank them for letting me conduct research in a autonomous way, for their advice and for creating a pleasant work atmosphere.

I would also like to thank the rest of the data science team: Antoine HAMON, Florent LEFORT, Sylvain PUYRAVAUD, Freddy LORIEAU and Cheima BOUDJENIBA. I was able to ask them questions about programming or data science and they have been very helpful.

Finally, I thank everybody at Servier for their professionalism and sympathy.

Introduction

Over the past decade, the explosion of the amount of data available – Big Data – the optimization of algorithms and the constant evolution of computing power have enabled artificial intelligence (AI) to perform more and more human tasks. In 2018, Google's CEO Sundar PICHAI predicted that AI will have a deeper impact than electricity or fire. In 2018, the McKinsey Global Institute predicted that AI will create an economic value of $2.7 trillion over the next 20 years.

AI aims at simulating human intelligence. Nowadays, AI is able to compute faster than humans and beat them at the game of Go. In his TED Talk [22], Martin FORD explains that the fact that Google DeepMind's artificial intelligence program AlphaGo was able to beat the best player at the game of Go in May 2017 is a breakthrough. Indeed, the game of Go has an infinite number of possibilities so no computer can analyze each possibility, but also because even the best players say that the game of Go is very intuitive.

How can we define artificial intelligence? In 1950, Alan TURING proposed an intelligence test for machines. According to [13], the game of imitation consists in developing a machine that is indistinguishable from a human being. Specifically, TURING suggested that a judge J exchange typed messages with a human being H on the one hand and a machine M on the other hand. These messages could cover all kinds of topics. The judge J does not know which of his two interlocutors (whom he knows as A and B) is the machine M and which is the human H. After a series of exchanges, the judge J must guess who is the machine and who is the human being. TURING thought that if one day we succeed in developing machines that make it impossible to identify correctly (i.e. lead the judge J to a 50% misidentification rate identical to what a random answer would give), then we can claim to have designed an intelligent machine or – what we will consider equivalent – a machine that thinks.

The introduction of the Deep Learning book [3] states the following:

> « In the early days of artificial intelligence, the field rapidly tackled and solved problems that are intellectually difficult for human beings but relatively straightforward for computers – problems that can be described by a list of formal, mathematical rules. The true challenge to artificial intelligence proved to be solving the tasks that are easy for people to perform but hard for people to describe formally – problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images. »

If we claim that the purpose of AI is to simulate human intelligence, the main difficulty is creativity. In the field of AI, we talk about generative models and one of the most popular model nowadays is GANs (for "generative adversarial networks").

Before discussing the technical part of the topic (i.e. scientific papers), it is very informative to read popular science articles, for example from *Sciences and Future* [17], *Les Echos* [18] [19] [20] or *MIT Technology Review* [14] [15]. They allow us to have a global and synthetic vision of GANs in order to better understand the technical details. According to [14], « Yann LeCun, Facebook's chief AI scientist, has called GANs "the coolest idea in deep learning in the last 20 years." Another AI luminary, Andrew Ng, the former chief scientist of China's Baidu, says GANs represent "a significant and fundamental advance" that's inspired a growing global community of researchers. »

In 2014, Ian GOODFELLOW invented GANs [1] and was then nicknamed "the GANfather". In his first paper that introduced GANs [1], GOODFELLOW included the bar where he first got the idea of GANs in the acknowledgments part: « Finally, we would like to thank Les Trois Brasseurs for stimulating our creativity. ». In 2019, aged 34, he was nominated in Fortune's 40 under 40 – an annual selection of the most influential young people – with the following description:

> « As one of the youngest and most respected A.I. researchers in the world, Ian Goodfellow has kept busy pushing the frontiers of deep learning. Having studied under some of the leading deep-learning practitioners like Yoshua Bengio and Andrew Ng, Goodfellow's expertise involves neural networks, the A.I. software responsible for breakthroughs in computers learning how to recognize objects in photos and understanding language. Goodfellow's creation of so-called generative adversarial networks (GANs) has enabled researchers to create realistic-looking but entirely computer-generated photos of people's faces. Although his techniques have allowed the creation of controversial "deepfake" videos [a], they've also paved the way for advanced A.I. that can create more realistic sounding audio voices, among other tasks. Apple recently took notice of Goodfellow's work and hired him to be the iPhone maker's director of machine learning in the company's special projects group. He was previously a star senior staff research scientist at Google and researcher at the high-profile nonprofit OpenAI. »
>
> Source: `https://fortune.com/40-under-40/2019/ian-goodfellow`
>
> [a] According to Wikipedia, « Deepfake (a portmanteau of "deep learning" and "fake") is a technique for human image synthesis based on artificial intelligence. It is used to combine and superimpose existing images and videos onto source images or videos using a machine learning technique known as generative adversarial network. The phrase "deepfake" was coined in 2017. Because of these capabilities, deepfakes have been used to create fake celebrity pornographic videos or revenge porn. Deepfakes can also be used to create fake news and malicious hoaxes. » One main example of deepfake is `https://www.youtube.com/watch?v=cQ54GDm1eL0`: a fake video of Obama warns us against fake news [21].

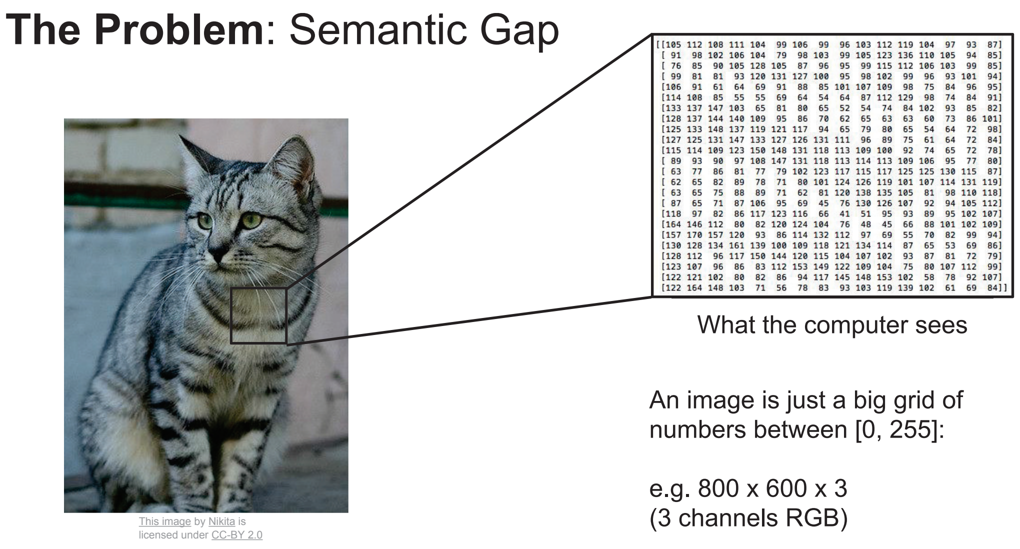
According to [14], « The magic of GANs lies in the rivalry between the two neural nets. It mimics the back-and-forth between a picture forger and an art detective who repeatedly try to outwit one another. Both networks are trained on the same data set. The first one, known as the generator, is charged with producing artificial outputs, such as photos or handwriting, that are as realistic as possible. The second, known as the discriminator, compares these with genuine images from the original data set and tries to determine which are real and which are fake. On the basis of those results, the generator adjusts its parameters for creating new images. And so it goes, until the discriminator can no longer tell what's genuine and what's bogus. » GANs can be used to imitate any data distribution (image, text, sound, etc.).

An artwork created thanks to them, based on the analysis of 15,000 examples, was recently put up for auction for a total amount of \$432,500. This artwork is displayed on figure 1. It is signed at the bottom right with $\min_{\theta_g} \max_{\theta_d} \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$, which is the fundamental equation I.3 of subsection I.2.2 on which the GAN algorithm 1 is based. Printed on canvas, the work belongs to a series of generative images called « La Famille de Belamy », where the name « Belamy » is the French translation of « Goodfellow ».

The main application of GANs (and deep learning in general) concerns computer vision as well as

Figure 1: Portrait generated in 2018 by Paris-based arts-collective Obvious with GANs sold with an auction price of 432 000$
Source: `http://obvious-art.com/edmond-de-belamy.html`

natural language processing (NLP). Note that in computer vision, images are represented as a 2-D grid of pixels, as shown in figure 2. When a computer looks at this image, it does not get the concept of cat as a human would. Instead, the computer is representing the image as a (gigantic) grid of numbers. If the size of the image is 800×600 and each pixel is represented by 3 numbers between $[0, 255]$ (giving the red, green, and blue value for that pixel), then we have an array of $800 \times 600 \times 3$ numbers. This array of $800 \times 600 \times 3$ numbers can be stretched out into a vector of dimension $1\,440\,000 = 800 \times 600 \times 3$. In short, an image is seen as a high-dimensional vector through its pixels.


The Problem: Semantic Gap
What the computer sees
An image is just a big grid of numbers between [0, 255]:
e.g. 800 x 600 x 3
(3 channels RGB)
This image by Nikita is licensed under CC-BY 2.0

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

Figure 2: The semantic gap between the concept of cat (that a human sees) and the pixel values (that the computer sees)
Source: Standord CS231n [2] lecture 2

In this report, we try to extend the range of GANs to electronic health records (EHR), more precisely to patient data.

Chapter I

General presentation on generative adversarial networks (GANs)

In AI, **generative adversarial networks** (GANs) are a generative model. GANs are an unsupervised learning method using two neural networks.

Just for information, here are some interesting websites that include references in order to understand GANs from A to Z:

- Jason BROWNLEE. Best Resources for Getting Started With Generative Adversarial Networks (GANs). 2019. `https://machinelearningmastery.com/resources-for-getting-started-with-generative-adversarial-networks/`
- Ajay Uppili ARASANIPALAI. Generative Adversarial Networks - The Story So Far. 2019. `https://blog.floydhub.com/gans-story-so-far/`
- A Beginner's Guide to Generative Adversarial Networks (GANs). `https://skymind.ai/wiki/generative-adversarial-network-gan`

The resources I highly recommend are Ian GOODFELLOW's tutorial [4] and Stanford CS231n lecture 13 "Generative models" [2]. GOODFELLOW's article [4] comes with the video on the conference he gave and is available at `https://www.youtube.com/watch?v=HGYYEUSm-0Q`. Stanford's lecture [2] was recorded and is available at `https://www.youtube.com/watch?v=5WoItGTWV54&list=PLC1qU-LWwrF64f4QKQT-Vg5Wr4qEE1Zxk&index=14&t=0s`. For any concepts of Deep Learning, GOODFELLOW's book [3] is a reference and is available at `https://www.deeplearningbook.org/`.

For information, the Deep Learning textbook [3] (by GOODFELLOW et al) dedicates 4 pages (690-693) on GANs (subsection 20.10.4).

This chapter is a general presentation on GANs. In this chapter, we do not try to apply GANs to the pharmaceutical world. We will explain some preliminary notions, go into the details of how GANs work (up to its fundamental equation (I.3)) and explain why we chose GANs over other existing methods. Finally, as GANs' main application is computer vision, we will apply GANs to medical imaging and take part in one of Servier Data Science team's project about IRM (nuclear magnetic resonance) of knees.

We will apply GANs to electronic health records (EHR) only in chapter II.

Contents

I.1 Some preliminary notions

I.1.1 Supervised vs. unsupervised learning

This subsection is taken from Stanford's lecture [2].

In **supervised learning**, we have labeled training data: we have some data x and some labels y. The goal is to learn a function that is mapping from the data x to the labels y. These labels can take different types of forms, for example categories of animals or captions of images. Regression is an example of supervised learning.

In the **unsupervised learning** setting, we have unlabeled training data: we only have some data x with no labels y. The goal is to learn some underlying hidden structure of the data x (for example grouping, axes of variation, underlying density estimation...). Clustering is an example of unsupervised learning.

Compared to supervised learning, unsupervised learning is less expensive because we do not need to train on labels. By comparison to supervised learning, unsupervised learning is still an unsolved research area. On the other hand, the potential of unsupervised learning is more interesting because it detects itself inherent structures in the data x. According to [14],

> « Today, AI programmers often need to tell a machine exactly what's in the training data it's being fed – which of a million pictures contain a pedestrian crossing a road, and which don't. This is not only costly and labor-intensive; it limits how well the system deals with even slight departures from what it was trained on. In the future, computers will get much better at feasting on raw data and working out what they need to learn from it without being told.
>
> That will mark a big leap forward in what's known in AI as "unsupervised learning." A self-driving car could teach itself about many different road conditions without leaving the garage. A robot could anticipate the obstacles it might encounter in a busy warehouse without needing to be taken around it. »

I.1.2 What is a generative model?

This subsection is taken from Stanford's lecture [2] and GOODFELLOW's tutorial [4].

Generative models are a type of unsupervised learning. The goal of a **generative model** is, given training data, to generate new samples from the same distribution (as the training data). We want to learn a model $p_{\text{model}}(x)$ which is similar to $p_{\text{data}}(x)$, with $p_{\text{data}}(x)$ being unknown. Generative models address density estimations, a core problem in unsupervised learning.

There are two types of generative models:

- **explicit density estimation**: explicitly define and solve for $p_{\text{model}}(x)$,
- **implicit density estimation**: learn a model that can sample from $p_{\text{model}}(x)$ without explicitly defining $p_{\text{model}}(x)$.

An example of explicit density estimation is given figure I.1: Gaussian model for a one-dimensional learning database. A generative model with explicit density estimation is based on a set of examples drawn from an unknown data-generating distribution $p_{\text{data}}(x)$ and outputs an estimation $p_{\text{model}}(x)$

of this distribution. The estimated model $p_{\text{model}}(x)$ can be evaluated for a particular value x_0 in order to obtain an estimate $p_{\text{model}}(x_0)$ of the true density $p_{\text{data}}(x_0)$. We can sample new data from $p_{\text{model}}(x)$.

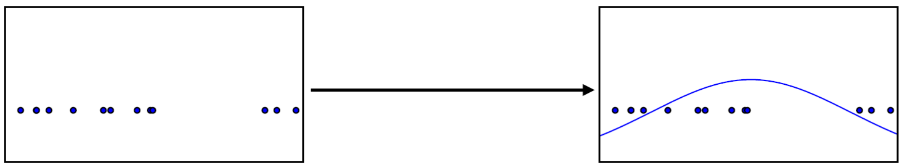


Figure I.1: An example of explicit density estimation
Source: GOODFELLOW's tutorial [4]

An example of implicit density estimation is given figure I.2. Of course, for the training data, we have a lot more images than the three that are shown. Once we have learned $p_{\text{model}}(x)$, we can generate as many samples as we want.

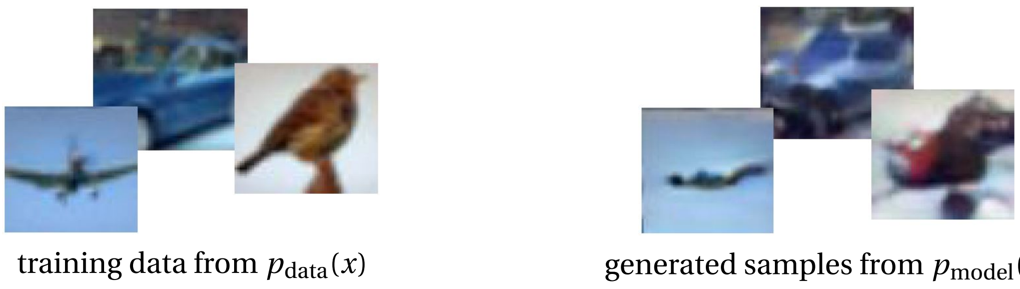

training data from $p_{\text{data}}(x)$
generated samples from $p_{\text{model}}(x)$

Figure I.2: An example of implicit density estimation
Source: Stanford CS231n [2]

GANs are an implicit density estimation problem. I will provide more details in section I.2.

A taxonomy of generative models is shown in figure I.3. The most popular are PixelRNN/CNN, Variational Autoencoder (VAE) and GANs. According to Wikipedia, in the computational complexity theory, « **tractable** » means « a problem that can be handled ». In particular, we can not optimize directly an intractable density function. For intractable problems, we can only have the approximate density (and not the exact one as with tractable problems).

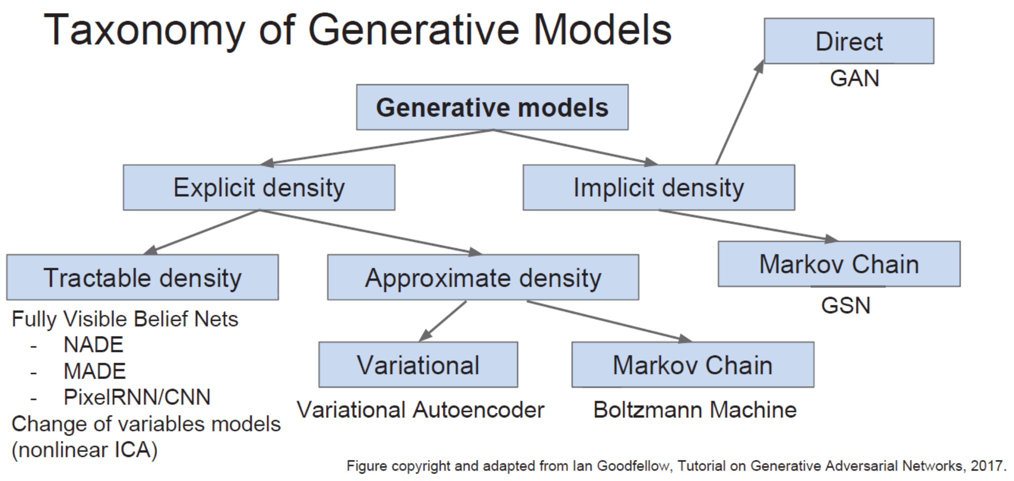


Figure I.3: A taxonomy of generative models
Source: Stanford CS231n [2]

I.1.3 Why are generative models interesting?

Generative models have several very useful applications: colorization, super-resolution, generation of artworks, etc. In general, the advantage of using a simulated model over the real model is that the computation can be faster.

Many interesting examples are given in GOODFELLOW's tutorial [4] and Stanford's lecture [2]. In particular, examples given by GOODFELLOW in the conference « Generative Adversarial Networks (NIPS 2016 tutorial) », from 4:15 to 12:33, are impressive and I highly recommend watching them. The link to this video, on which the paper [4] is based, is the following: `https://www.youtube.com/watch?v=HGYYEUSm-0Q`.

We are now going to give three examples where the generative aspect is very useful. These examples are taken from GOODFELLOW's tutorial [4].

A first example of GANs' results is given figure I.4. The generation of these fictional celebrity portraits, from the database of real portraits CELEBA-HQ composed of 30 000 images, took 19 days. The generated images have a size of 1024 × 1024 but have been compressed in this report. These portraits are very realistic.

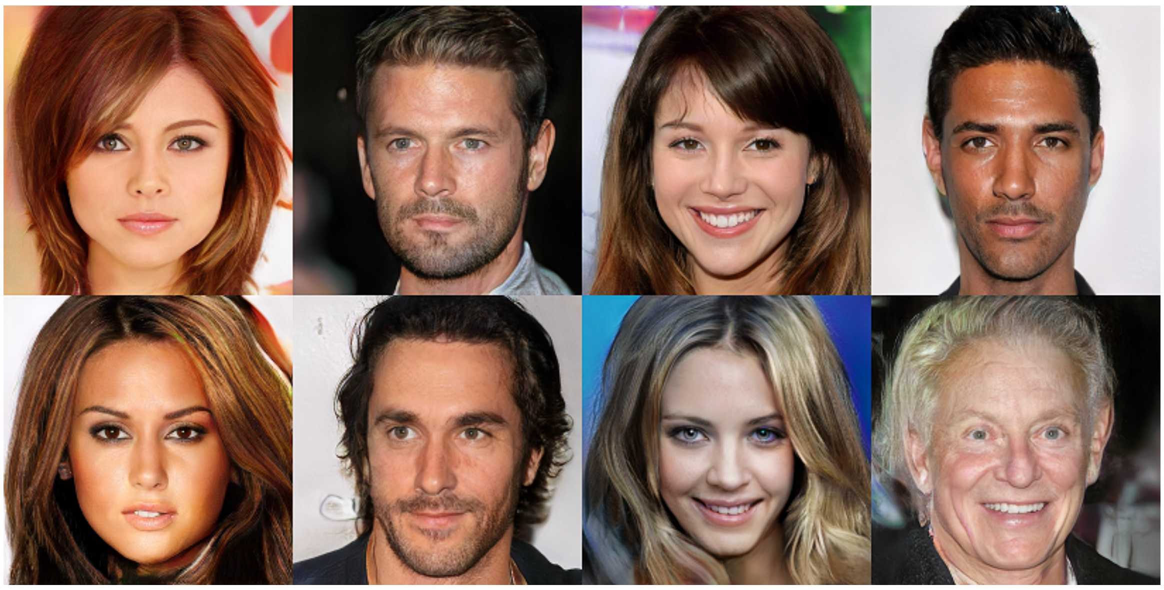


Figure I.4: Realistic fictional portraits of celebrities generated from originals using GANs
Source: Nvidia [10]

An article from the *MIT Technology Review* [14] states about figure I.4:

> « In one widely publicized example last year, researchers at Nvidia, a chip company heavily invested in AI, trained a GAN to generate pictures of imaginary celebrities by studying real ones. Not all the fake stars it produced were perfect, but some were impressively realistic. »

A second example is given figure I.5. These real images are transposed into realistic fictional images - or vice versa - with the CycleGan developed by researchers at the University of Berkeley. The concept, called **image-to-image translation**, is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training

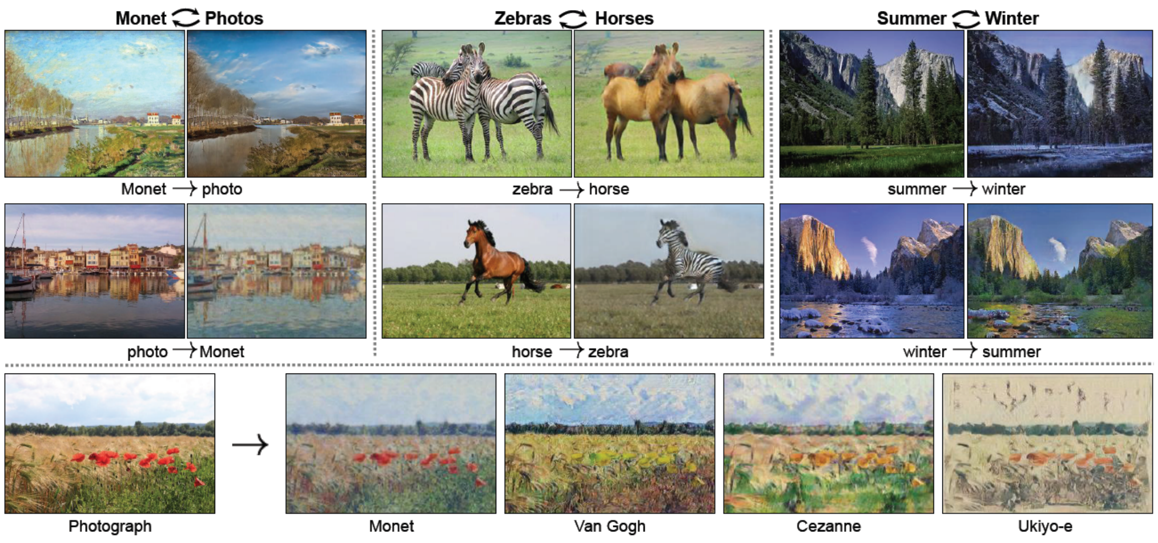


Figure I.5: CycleGAN: real images transposed into realistic fictional images using GANs
Source: Berkeley AI Research (BAIR) laboratory [11]

set of aligned image pairs. There exists a tutorial about CycleGAN on Tensorflow: `https://www.tensorflow.org/beta/tutorials/generative/cyclegan`.

The third and last example is shown in figure I.6. For example, the aerial to map feature can be very useful to Google Maps or similar applications.

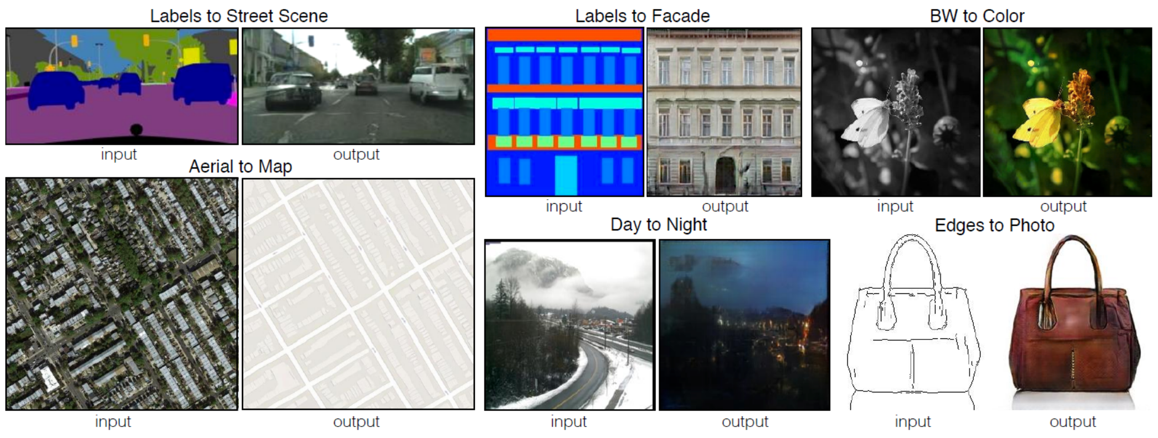


Figure I.6: Several types of image transformations using GANs
Source: Berkeley AI Research (BAIR) Laboratory [12]

We can note that most GANs' applications are computer vision (or at least image retlated). We will see in chapter II that it is possible, to a certain extent, to apply GANs to electronic health records (EHR).

I.1.4 A few important concepts and facts of machine learning

I.1.4.a About the training dataset

- Let us not be confused by the term "generative model": AI is not creative, it can only learn from the training database. Thus, in addition to code quality, a quality training database is required to obtain consistent output results: unbiased, with a lot of samples and good features.

- When our learning dataset is small (not enough samples), it is difficult to build a relevant model. For example, we may have the problem of overfitting.

  The Deep Learning textbook [3] pages 18-20 on the importance a large training dataset:

  « It is true that some skill is required to get good performance from a deep learning algorithm. Fortunately, the amount of skill required reduces as the amount of training data increases. The learning algorithms reaching human performance on complex tasks today are nearly identical to the learning algorithms that struggled to solve toy problems in the 1980s, though the models we train with these algorithms have undergone changes that simplify the training of very deep architectures. The most important new development is that today we can provide these algorithms with the resources they need to succeed. [...] The age of “Big Data” has made machine learning much easier because the key burden of statistical estimation – generalizing well to new data after observing only a small amount of data – has been considerably lightened. As of 2016, a rough rule of thumb is that a supervised deep learning algorithm will generally achieve acceptable performance with around 5,000 labeled examples per category and will match or exceed human performance when trained with a dataset containing at least 10 million labeled examples. »

  It is important to keep these orders of magnitude in mind. However, an article from the *MIT Technology Review* [14] explains that:

  « Unlike other machine-learning approaches that require tens of thousands of training images, GANs can become proficient with a few hundred. ».

- The Deep Learning textbook [3] page 3 on the importance of choosing good features for our training dataset:

  « The performance of these simple machine learning algorithms depends heavily on the representation of the data they are given. [...] [For example,] people can easily perform arithmetic on Arabic numerals but find arithmetic on Roman numerals much more time consuming. [...] Many artificial intelligence tasks can be solved by designing the right set of features to extract for that task, then providing these features to a simple machine learning algorithm. For example, a useful feature for speaker identification from sound is an estimate of the size of the speaker’s vocal tract. This feature gives a strong clue as to whether the speaker is a man, woman, or child. For many tasks, however, it is difficult to know what features should be extracted. »

- We do not really care about the performance of a classifier on training data: we use training data to find some classifier and then we apply this classifier on test data (thus we try to avoid overfitting).

  Our model should be "simple" to make our model work on test data: we can use regularization in the loss function to encourage our model to be "simple". The concept of "simple" depends on the task and the model. If we have many different competing hypothesis on some observations, we should prefer the simpler one because it is the one that is more likely to generalize to new observations in the future. For example, if we have to choose between several polynomial models that fit our training data, we prefer the model with the lowest degree.

I.1.4.b About deep learning

- When a distribution is complicated, it is more appropriate to try to estimate it using neural networks [2]. In particular, deep networks have the ability to learn complex patterns from data because it enables the computer to build complex concepts out of simpler concepts [3]. As we will see later, GANs use two neural networks.

- The Deep Learning textbook [3] pages 14 and 16 on the influence of neuroscience on deep learning:

  « Today, neuroscience is regarded as an important source of inspiration for deep learning researchers, but it is no longer the predominant guide for the field. The main reason for the diminished role of neuroscience in deep learning research today is that we simply do not have enough information about the brain to use it as a guide. [...] one should not view deep learning as an attempt to simulate the brain. Modern deep learning draws inspiration from many fields, especially applied math fundamentals like linear algebra, probability, information theory, and numerical optimization. »

- For computer vision, deep learning is mostly used nowadays.

- According to the Deep Learning textbook [3] page 12, « deep learning dates back to the 1940s. Deep learning only appears to be new, because it was relatively unpopular for several years preceding its current popularity, and because it has gone through many different names, only recently being called "deep learning." »

- In 2018, Yoshua BENGIO, Geoffrey HINTON and Yann LECUN won the Turing Award – the "Nobel Prize of computing" – for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing. BENGIO was one of GOODFELLOW's doctoral advisor and is the co-author of the Deep Learning textbook [3].

- The Deep Learning textbook [3] page 12 on how deep learning became so popular:

  « we identify a few key trends:

  - Deep learning has had a long and rich history, but has gone by many names, reflecting different philosophical viewpoints, and has waxed and waned in popularity.
  - Deep learning has become more useful as the amount of available training data has increased.
  - Deep learning models have grown in size over time as computer infrastructure (both hardware and software) for deep learning has improved.
  - Deep learning has solved increasingly complicated applications with increasing accuracy over time. »

- According to [3] page 14, slightly modified versions of the stochastic gradient descent (SGD) algorithm are the dominant training algorithms for deep learning models today. Let us note that the step size (also called learning rate) is a hyperparameter.

  According to Stanford CS231n lecture 3 (available at `https://www.youtube.com/watch?v=h7iBpEHGVNc&list=PLC1qU-LWwrF64f4QKQT-Vg5Wr4qEE1Zxk&index=3`), the most popular loss function for deep learning is the softmax loss (also called cross-entropy loss).

  A full lecture on loss functions and gradient descent is given by Stanford CS231n lecture 3.

I.1.4.c Maximum Likelihood Estimation

This paragraph is taken from the Deep Learning textbook page 128 [3].

The maximum likelihood estimation is a fundamental concept in machine learning. In this report, we will only make a short review.

Consider a set of m examples $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ drawn independently from the true but unknown data-generating distribution $p_{\text{data}}(\mathbf{x})$.

Let $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ be a parametric family of probability distributions over the same space indexed by $\boldsymbol{\theta}$. In other words, $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ maps any configuration $\boldsymbol{x}$ to a real number estimating the true probability $p_{\text{data}}(\boldsymbol{x})$.

The maximum likelihood estimator (MLE) for $\boldsymbol{\theta}$ is then defined as:

$$\begin{aligned}\boldsymbol{\theta}_{ML} &= \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})\end{aligned}$$

This product over many probabilities can be inconvenient for various reasons. For example, it is prone to numerical underflow. To obtain a more convenient but equivalent optimization problem, we observe that taking the logarithm of the likelihood does not change its arg max but does conveniently transform a product into a sum:

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

Because the arg max does not change when we rescale the cost function, we can divide by m to obtain a version of the criterion that is expressed as an expectation with respect to the empirical distribution $\hat{p}_{\text{data}}$ defined by the training data:

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) \tag{I.1}$$

In the rest of this report, we will use equation (I.1).

I.1.4.d Kullback-Leibler (KL) divergence

This paragraph is taken from the Deep Learning textbook pages 71 and 72 [3].

If we have two separate probability distributions $P(\mathrm{x})$ and $Q(\mathrm{x})$ over the same random variable x, we can measure how different these two distributions are using the Kullback-Leibler (KL) divergence:

$$D_{\text{KL}}(P \| Q) = \mathbb{E}_{\mathrm{x} \sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathrm{x} \sim P}\left[\log P(x) - \log Q(x)\right] \tag{I.2}$$

In the case of discrete variables, it is the extra amount of information (measured in bits if we use the base-2 logarithm, but in machine learning we usually use nats and the natural logarithm) needed to send a message containing symbols drawn from probability distribution P, when we use a code that was designed to minimize the length of messages drawn from probability distribution Q.

The KL divergence has many useful properties, most notably being non-negative. The KL divergence is 0 if and only if P and Q are the same distribution in the case of discrete variables, or equal “almost everywhere” in the case of continuous variables. Because the KL divergence is non-negative and measures the difference between two distributions, it is often conceptualized as measuring some sort of distance between these distributions. It is not a true distance measure because it is not symmetric: $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|Q)$ for some P and Q. This asymmetry means that there are important consequences to the choice of whether to use $D_{\mathrm{KL}}(P\|Q)$ or $D_{\mathrm{KL}}(Q\|P)$.

KL divergence is used in generative models to compare the real data distribution to the estimated one.

Note: We could use KL divergence to measure the accuracy of our fictitious generated dataset compared to the origiinal real-life dataset. I will not do it in this report due to the lack of time.

I.2 How do GANs work?

This section is taken from Stanford's lecture [2].

As introduced earlier, **generative adversarial networks** (GANs) are a generative model with implicit density estimation, part of unsupervised learning and are using two neural networks. We thus understand the terms "generative" and "networks" in "generative adversarial networks".

Thus, GANs do not use an explicit density function and we are only interested in the ability to generate new samples from the distribution. GANs are based on a game-theoretic approach: GANs learn to generate from a training distribution through a two-player game.

The main difficulty is that we want to sample from a complex and high-dimensional training distribution. There is no direct method. One solution (the one we will use) is to generate from a simple distribution: **random noise**. We will learn a transformation from this simple distribution to the training distribution. We use a neural network to represent this complex transformation.

As shown in figure I.7, to each sample in the training dataset, we associate a random noise z, which is a vector of chosen dimension. Note that by using random noise, we introduce a hazard and ensure that the data generated by the generator is not all identical.

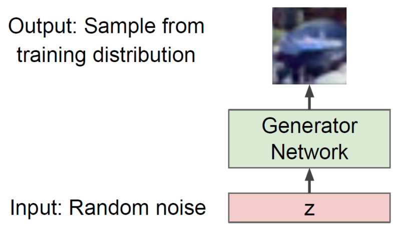


Figure I.7: Role of the random noise
Source: Stanford CS231n [2]

I.2.1 The principle: generator vs discriminator

The principle is a two-player game: a neural network called the generator and a neural network called the discriminator. The **generator** tries to fool the discriminator by generating real-looking images while the **discriminator** tries to distinguish between real images and fake images. We then understand the term "adversarial" in "generative adversarial networks". See figure I.8.

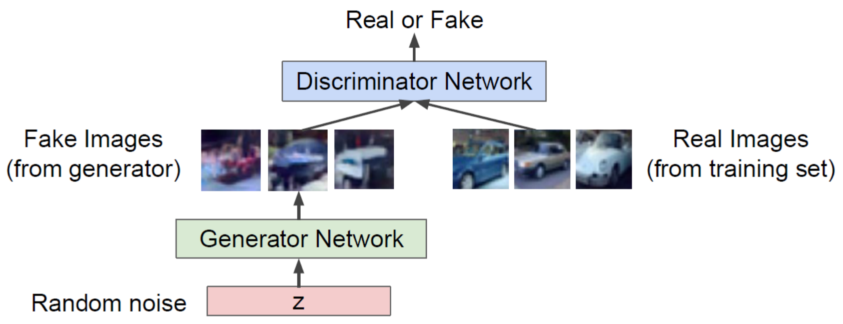


Figure I.8: Roles of the generator and the discriminator
Source: Stanford CS231n [2]

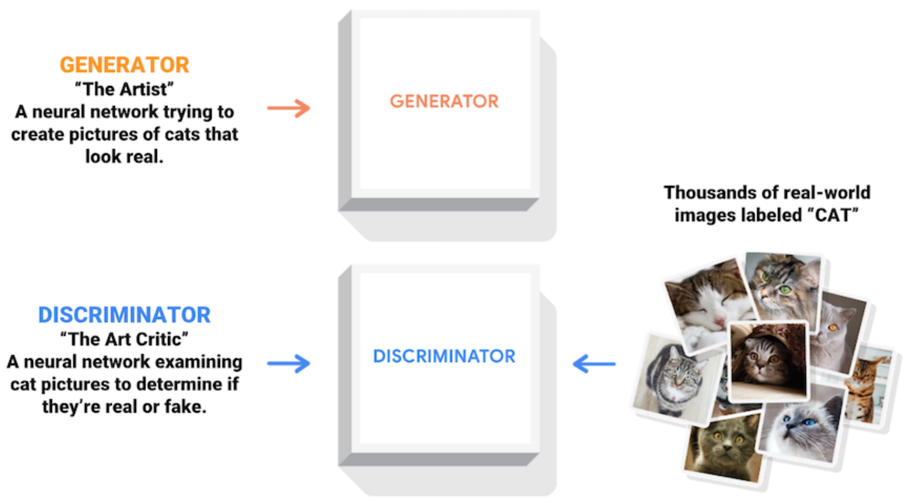

GENERATOR
"The Artist"
A neural network trying to create pictures of cats that look real.
GENERATOR
Thousands of real-world images labeled "CAT"
DISCRIMINATOR
"The Art Critic"
A neural network examining cat pictures to determine if they're real or fake.
DISCRIMINATOR

Figure I.9: Interpretation: roles of the generator and the discriminator
Source: `https://www.tensorflow.org/beta/tutorials/generative/dcgan`

The generator can be interpreted as an artist and the discriminator as an art critic. See figure I.9.

During training, the generator progressively becomes better at creating images that look real, while the discriminator becomes better at telling them apart. The process reaches equilibrium when the discriminator can no longer distinguish real images from fake. See figure I.10. Thus, if the discriminator is well trained and the generator manages to generate real-looking images that fool the discriminator, then we have a good generative model: we are generating images that look like the training set. The discriminator may then be discarded. The random noise z guarantees that the generator does not always produce the same image (which can deceive the discriminator).

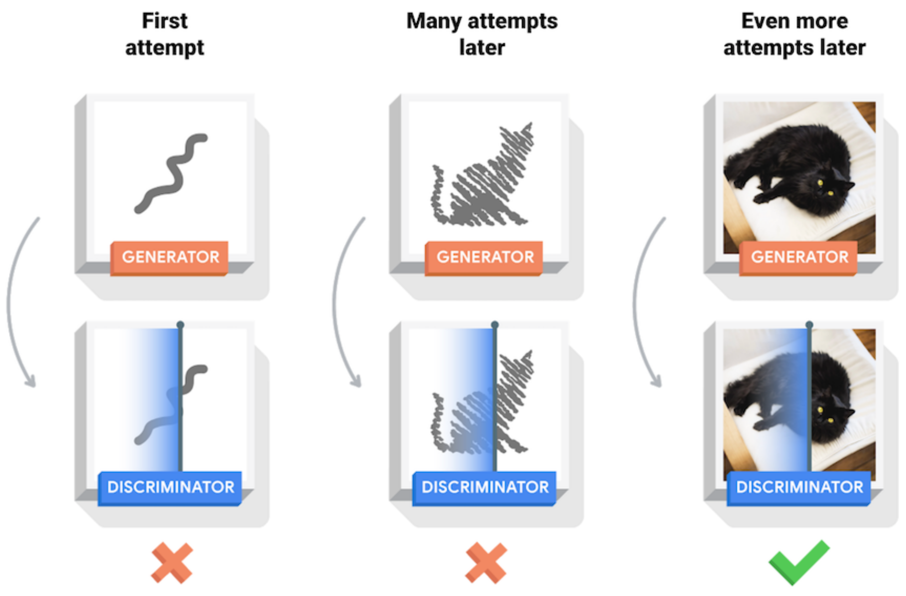


Figure I.10: Generator and discriminator training
Source: `https://www.tensorflow.org/beta/tutorials/generative/dcgan`

Note that at the beginning of the training, the generator only generates a random noise that does not resemble the training data.

I.2.2 The two-player minimax game

The generator G and the discriminator D are jointly trained in a **two-player minimax game** formulation. The minimax objective function is:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathrm{x} \sim p_{\mathrm{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{\mathrm{z} \sim p(z)} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right] \tag{I.3}$$

where θ_g is the parameters of G and θ_d is the parameters of D.

In the following, we simply refer to D_{θ_d} as D and G_{θ_g} as G.

By definition, D outputs the likelihood (see I.1.4.c) of real image in interval $[0,1]$:

- $D(x)$ equals 1 (or is close to 1) if D considers that x is a real data
- $D(x)$ equals 0 (or is close to 0) if D considers that x is a fake data (e.g. a generated data).

We can prove that, at the equilibrium, D outputs $1/2$ everywhere because D can not distinguish fake generated data from real data.

Because $\mathrm{x} \sim p_{\mathrm{data}}$, x is a real data. By definition of G, $G(z)$ is a fake generated data. x would be a real-life image of a cat and $G(z)$ would be a fake generated image of a cat. Thus, $D(x)$ is the output of the discriminator for a real input x (since $\mathrm{x} \sim p_{\mathrm{data}}$) and $D(G(z))$ is the output of the discriminator for a fake generated data $G(z)$.

GOODFELLOW wrote the two-player minimax game (I.3) such that θ_g and θ_d evolve so that the following points are true:

- The discriminator D tries to distinguish between real data x and fake data $G(z)$.

  More precisely, the discriminator D plays with θ_d (θ_g being fixed) to maximize the objective function such that $D(x)$ is close to 1 (x is real) and such that $D(G(z))$ is close to 0 (a generated data is detected as false).
- The generator G tries to fool the discriminator D into thinking that its fake generated data is real.

  More precisely, the generator G plays with θ_g (θ_d being fixed) to minimize the objective function such that $D(G(z))$ is close to 1 (a false generated data is detected as true by the discriminator).

Although we are in unsupervised learning (the data is not labeled), we choose that the data generated by G has a 0 label for false (regardless of what the discriminator returns) and the real learning data has a 1 label for true. We can thus define a loss function.

GOODFELLOW's first paper on GANs [1] demonstrates that the minimax game has a global (and unique) optimum for $p_g = p_{\mathrm{data}}$ where p_g is the generative distribution and p_{data} the real data distribution.

I.2.3 Gradient descent

We consider the problem given by equation (I.3).

For the training, we will alternate between:

1. gradient ascent on the discriminator:

$$\max_{\theta_d} \left[\mathbb{E}_{x\sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right] \tag{I.4}$$

2. gradient descent on the generator:

$$\min_{\theta_g} \left[\mathbb{E}_{z\sim p(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right] \tag{I.5}$$

Let us note that for (I.5), we only keep the term on the right in (I.3) because it is the only one that is dependent on θ_g.

As can be seen in figure I.11, the landscape of the generator objective $\log(1 - D(G(z)))$ for $D(G(z))$ is not efficient. Indeed, we want to minimize this function whose slope is higher towards the right, i.e. when $D(G(z))$ is close to 1. That is tantamount to saying that when the generator succeeds in distinguishing a false data from a real one, i.e. $D(G(z))$ is close to 1, the gradient will be high. On the other hand, when the generator has not yet learned to distinguish properly between false data and actual data, i.e. $D(G(z))$ is close to 0, the gradient is almost horizontal. Thus, the gradient is high in the regions for which the generator is already well trained, whereas we want the generator to learn mainly from areas where it is bad: we want a high slope in the regions where $D(G(z))$ is close to 0.

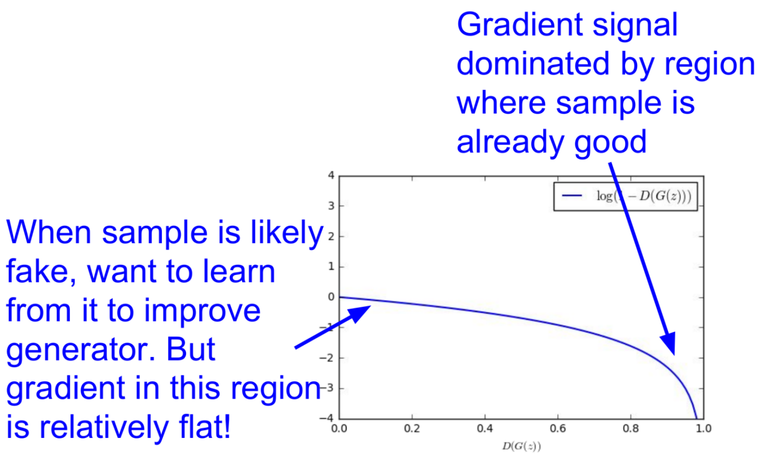


Figure I.11: Gradient descent on the generator
Source: Stanford CS231n [2]

Basically, we want the steps to be high (thus high gradient values) in the regions where we are far away from the optimum (i.e. to the left where $D(G(z))$ is close to 0) and we want the steps to be small (thus low gradient values) when we are close to the optimum (i.e. to the right where $D(G(z))$ is close to 1).

So, to improve learning, instead of equation I.5, we choose a different objective function. Equation I.5 tries to minimize the likelihood of the discriminator being right. We are now trying to maximize the likelihood of the discriminator being wrong:

$$\max_{\theta_g} \left[\mathbb{E}_{z\sim p(z)} \log\left(D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right] \tag{I.6}$$

The purpose is the same: fooling the discriminator. On the other hand, as we can see on figure I.12, the gradient is high for regions where $D(G(z))$ is close to 0: we will train the generator more in the regions where the generated samples are bad.

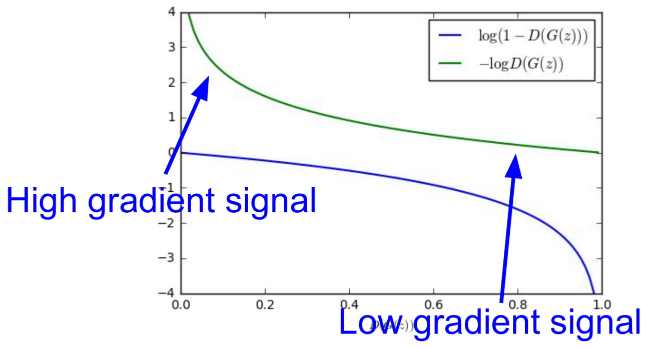


Figure I.12: Gradient ascent on the generator
Source: Stanford CS231n [2]

After taking into account the remarks we made, we obtain the algorithm 1. We successively train the discriminator first, then the generator.

Algorithm 1 GAN training

```
1: for number of training iterations do
2:     for $k$ steps do
3:         Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noize prior $p_g(z)$.    ▷ for the fake data
4:         Sample minibatch of $m$ noise samples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.    ▷ for the real data
5:         Update the discriminator by ascending its stochastic gradient:
               $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D_{\theta_d}\left(x^{(i)}\right) + \log\left(1 - D_{\theta_d}\left(G_{\theta_g}\left(z^{(i)}\right)\right)\right) \right]$
6:     end for
7:     Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noize prior $p_g(z)$.
8:     Update the generator by ascending its stochastic gradient (improved objective):
               $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log D_{\theta_d}\left(G_{\theta_g}\left(z^{(i)}\right)\right)$
9: end for    ▷ The gradient-based updates can use any standard gradient-based learning-rule.
```

Note that there are discussions on the value of k: some think that $k = 1$ gives more stability while others think that $k > 1$ is needed.

GOODFELLOW's first paper on GANs [1] demonstrates that the GAN training algorithm 1 optimizes the minimax formulation (I.3), thus obtaining the desired result.

After this training phase, we only need the generator to sample new (false) realistic data. We no longer need the discriminator.

I.2.4 Application: some simulations with TensorFlow and GAN Lab

There are some very interesting tutorials on GANs provided by TensorFlow (open source tool for machine learning developed mainly by Google, in particular available as a package on Python):

- *Generating Handwritten Digits with DCGAN* on TensorFlow 1.13: `https://GitHub.com/tensorflow/tensorflow/blob/r1.13/tensorflow/contrib/eager/python/examples/generative_examples/dcgan.ipynb`
- *Deep Convolutional Generative Adversarial Network* on TensorFlow 2.0: `https://www.tensorflow.org/beta/tutorials/generative/dcgan`
- *CycleGAN* on TensorFlow 2.0: `https://www.tensorflow.org/beta/tutorials/generative/cyclegan`

I highly recommend running at least of of the above tutorials. They provide the dataset, the code and the explanations that allow us to apply these methods, thus help understand how GANs work. These tutorials are in the `ipynb` format, they can be opened in Jupyter Notebook from Anaconda (for example). To avoid having to install TensorFlow on your computer, it is possible to execute the code directly on Google Colab's servers. Note that TensorFlow 2.0 is a beta version and that in this project we will use TensorFlow 1.2.

Another fundamental application is the GAN Lab [9], accessible at the following link: `https://poloclub.github.io/ganlab/`. The GAN Lab is a website that enables an interactive visual experimentation of the learning of the generator and the discriminator. For visualization purposes, the data are not images but two-dimensional points.

I.3 Comparison of GANs with other generative models

The goal of this section is to review the models given in the figure I.3 to justify our choice of GANs.

As this choice is based on computer vision, which is not the core of this report (electronic health records are), we are not going to detail them here. One can refer to GOODFELLOW's tutorial [4] and Stanford's lecture [2]. Stanford' lecture 13 on generative models explains only the most popular methods: PixelRNN/CNN, VAEs and GANs.

Chapter II

Application of GANs to electronic health records (EHR)

This chapter is an application of the GAN concept introduced in chapter I. As we have seen, GANs are mainly used for computer vision as well as natural language processing (NLP). In this chapter, we try to extend the range of GANs to electronic health records (EHR).

In the first section, we present the theoretical approach: `medGAN`. `medGAN` stands for "medical generative adversarial network". The scientific article on which we mostly rely is [5]:

> Edward CHOI et al. Generating Multi-label Discrete Patient Records using Generative Adversarial Networks. *arXiv:1703.06490v3*, 2018.

This article, written in particular by researchers at the Georgia Institute of Technology, introduces the concept of `medGAN` which will be presented at subsection II.1.3. `medGAN` is a combination of GANs and autoencoders in order to apply GANs to electronic health records (EHR). We present autoencoders at subsection II.1.2. In the first section, we will also see how Servier can benefit from GANs (or in this case, from `medGAN`).

I would like to thank Edward CHOI again: I was able to ask him a few questions by email and he was always very helpful. His email is `mp2893@gatech.edu`. In order to communicate with the community, we can open an issue on the GitHub repository: `https://github.com/mp2893/medgan/issues/`. Edward CHOI seems to be very responsive on his GitHub as well.

In the second section, we will implement `medGAN`, explain the code and present the results. Rather than writing everything in this report, my work on the datasets and coding can be found on my own GitHub repository about `medGAN`: `https://github.com/sylvaincom/medgan-tips`. I am glad that I was able to make a small contribution to Edward Choi's code:

- Fixing an error due to version 1.16.3 of NumPy: `https://github.com/mp2893/medgan/pull/15`
- Fixing an error when running step 2-3 with count variables: `https://github.com/mp2893/medgan/pull/17`

In this report, I add details but the core of the algorithmic implementation is on my GitHub repository.

In the third section, I present the experimental results on the MIMIC-III dataset (which is public) and also on the Datasphere data (from Servier).

Contents

II.1 Theoretical approach: `medGAN`

II.1.1 How can Servier benefit from GANs?

II.1.1.a Privacy of patients' personal data

Healthcare organizations (HCOs) have tried the de-identification method in order to protect the privacy of their patients. For de-idenfication, HCOs performed perturbation of potentialy identifiable attributes (e.g. dates of birth) via generalization, suppression or randomization. However, it is still possible to re-identify through the residual information [8].

In CHOI's paper [5], `medGAN` is mainly used for privacy purposes: instead of sending the original EHR data (that can be re-identified) to researchers, we can send them the fictitious realistic generated data.

The paper [5] concludes that « our privacy experiments indicate that `medGAN` does not simply remember the training samples and reproduce them. Rather, `medGAN` generates diverse synthetic samples that reveal little information to potential attackers unless they already possess significant amount of knowledge about the target patient. »

CHOI considers two definitions of privacy:

- **Presence disclosure** occurs when an attacker can determine that `medGAN` was trained with a dataset including the record from patient x.
- **Attribute disclosure** occurs when attackers can derive additional attributes such as diagnoses and medications about patient x based on a subset of attributes they already know about x.

II.1.1.b Dataset augmentation in order to make better predictions

Why is machine learning essential to Servier? The *PEX MVD* applies artificial intelligence (more precisely machine learning) to patient data in order to make predictions. For example, it can try to predict whether a drug will have a positive impact or not, taking into account some features of the patient (age, weight, height, etc.). An article from the *MIT Technology Review* explains how an AI system identified a potential new drug in just 46 days [16]. According to [16], « The system examines previous research and patents for molecules known to work against the drug target, prioritizing new structures that could be synthesized in the lab. It's similar to what a human chemist might do to seek new therapies – just much faster. [...] Getting a new drug to market is hugely costly and time consuming: it can take 10 years and cost as much as $2.6 billion, with the vast majority of candidates failing at the testing stage ».

In order to train machine learning algorithms, a large amount of training data is required. Computational health is on the rise. In France, the Health Data Hub, a platform for the exploitation of health data, was created in 2019 following the Villani report on AI. The Health Data Hub's goal is to increase the potential for the exploitation of health data, in particular in the areas of research, support for health personnel, health system management, monitoring and patient information. However, due to the confidentiality of patient data (among other reasons), EHR are generally limited in number: they are scarce. We are very far from the Big Data framework.

With GANs, we want to generate fictitious realistic patient data, which can then enrich the initial real-life training database. For example, my training dataset A is not large enough (let it be 500 samples with 50 features) and we want to use `medGAN` to generate a new dataset *B* of 1 000 fictitious samples (with 50 features as well). By adding *B* to *A*, we get a new training dataset *C* that has 1 500 patients. We can hope that *C* helps algorithms (any one of them) make better predictions than *A*. However, the main issue would be that we want to enrich the original training dataset *A* but the new dataset *B* is actually "based" on the original training set *A*.

I asked CHOI what he thought about using the generative model GANs for dataset augmentation. Trying to generate fictitious realistic patients with `medGAN` from a dataset of 500 samples with 250 variables seems suboptimal: there seems to be too many variables and not enough samples. There is no definite number as to how many variables we need to delete: it depends on the variance of each variable and the correlation between variables. For example, if there is a variable named "gender" and all 500 samples are from men (thus low variance), then it would be very easy for `medGAN` to replicate that variable (by putting men as gender for each generated sample).

The "Boosting Deep Learning Risk Prediction with Generative Adversarial Networks for Electronic Health Records" paper [6] aims at mimicking real patient records to augment the training dataset in order to improve the prediction performance. The introduction states the following reasons for the shortage of samples in EHR:

> « the amount of clinical data, especially with accurate labels and for rare diseases and conditions, is somewhat limited and far from most models' requirements. This comes from the following reasons: The diagnosis and patient labeling process highly relies on experienced human experts and is usually very time-consuming; Getting detailed results of lab tests and other medical features, though has become more feasible with modern facilities than ever, are still quite costly; Not to mention the privacy issues and regulations which makes it even harder to collect and keep enough medical data with desired details. These unique challenges lying in healthcare domain prevent existing deep learning models from exerting their strength with enough available and high-quality labeled data. »

Section 7.4 pages 233 and 234 of the Deep Learning textbook [3] is dedicated to dataset augmentation:

> « The best way to make a machine learning model generalize better is to train it on more data. Of course, in practice, the amount of data we have is limited. One way to get around this problem is to create fake data and add it to the training set. For some machine learning tasks, it is reasonably straightforward to create new fake data. [...] Dataset augmentation has been a particularly effective technique for a specific classification problem: object recognition. »

II.1.2 What are autoencoders?

In order to understand how `medGAN` works, we first need to understand how autoencoders work.

This subsection is taken from Stanford's lecture [2].

Autoencoders are an unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data. Autoencoders alone can not generate data.

As shown in figure II.1, we have some input data $\boldsymbol{x}$ and we want to learn some feature $\boldsymbol{z}$. The **encoder** is a function mapping from $\boldsymbol{x}$ to $\boldsymbol{z}$. The encoder can take different forms but we generally use neural networks. Originally, in the 2000s, we used linear layers of nonlinearity (sigmoid). Then, we used fully connected deeper networks. Later, we used CNNs with ReLU.

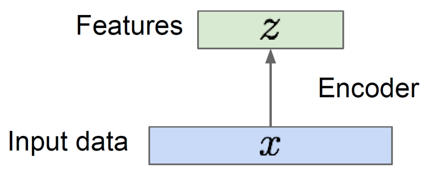


Figure II.1: Representation of the encoder
Source: Stanford CS231n [2]

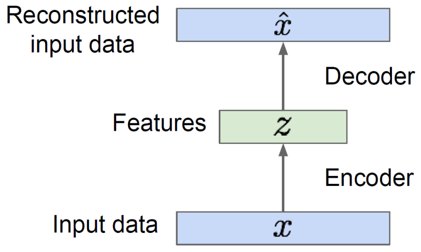


Figure II.2: Representation of the encoder and the decoder
Source: Stanford CS231n [2]

We usually specify $\boldsymbol{z}$ to be smaller than $\boldsymbol{x}$ by performing dimensionality reduction. We want to use dimensionality reduction because we want $\boldsymbol{z}$ to be able to learn features that can capture meaningful factors of variation in the data, which make $\boldsymbol{z}$ a good feature.

In order to learn this feature representation, we train the model such that the features can be used to reconstruct the original data, as shown in figure II.2. The term « autoencoding » means « encoding itself ». We use the encoder to map the input data $\boldsymbol{x}$ to some lower-dimensional features $\boldsymbol{z}$. We use the **decoder**, another neural network, to map these features $\boldsymbol{z}$ to $\hat{\boldsymbol{x}}$ which is an estimation of the original input data $\boldsymbol{x}$. In other words, $\hat{\boldsymbol{x}}$ is a reconstruction of $\boldsymbol{x}$. In particular, $\hat{\boldsymbol{x}}$ has the same dimensionality as $\boldsymbol{x}$. For the decoder, we use the same type of networks as the encoder but it is symmetric.

An example of reconstruction is given figure II.3. In this example, we are using a convolutional network for the encoder and an upconvolutional network for the decoder because $\boldsymbol{z}$ has a lower dimension than $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$. We can see that the reconstructed images are not as sharp as the original ones but the reconstructed images keep the most important characteristics of the original ones.

We try to make the pixels in the reconstructed data $\hat{\boldsymbol{x}}$ to be the same as the pixels in the input data $\boldsymbol{x}$. In order to reconstruct the input data $\boldsymbol{x}$, we use a loss function, for example $\mathcal{L}^2$:

$$\|\boldsymbol{x}-\hat{\boldsymbol{x}}\|^2 \tag{II.1}$$

It is important to note that even though we compute a loss function, there are no external labels being used in the training. We only need the unlabeled training data to compute the loss function.

Once the training is complete, we no longer need the decoder, which was only useful to compute the loss function. We can use the encoder to initialize a supervised model with better features $\boldsymbol{z}$, as explained in figure II.4.

As a conclusion, we can use a lot of unlabeled training data to try and learn good general feature representations. In cases where we do not have enough input data x, we can use the encoder to initialize a supervised learning problem with better features z. Indeed, when we do not have enough input data, it is hard to learn a good model, for example we have overfitting.

According to the Deep Learning textbook [3] page 4, the autoencoder is the quintessential example of representation learning algorithms.

The features $\boldsymbol{z}$ capture factors of variation in the training data $\boldsymbol{x}$. $\boldsymbol{z}$ is called a latent variable. According to Wikipedia, **latent variables** are variables that are not directly observed but are rather

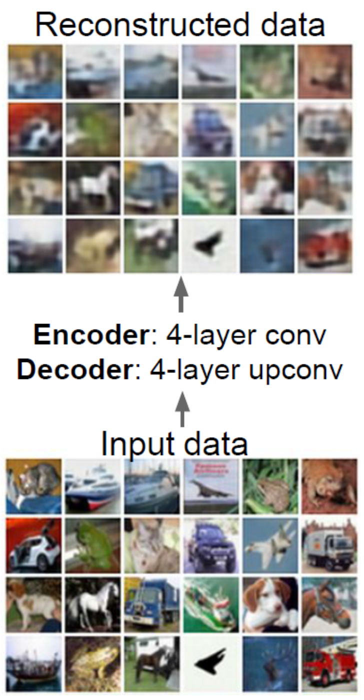


Figure II.3: An example of reconstruction
Source: Stanford CS231n [2]

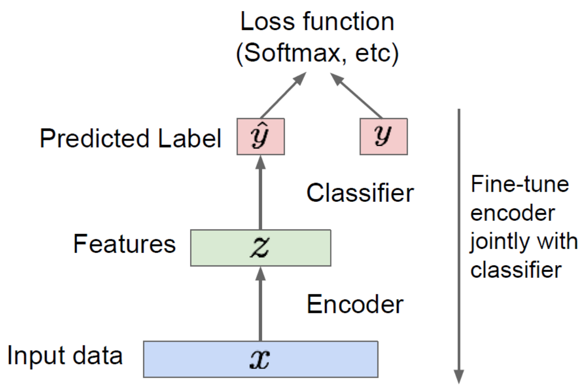


Figure II.4: The encoder can be used to initialize a supervised model.
Source: Stanford CS231n [2]

inferred (through a mathematical model) from other variables that are observed (directly measured). This latent variable z is used in **variational autoencoders** (**VAEs**), which is another generative model that we will not explain in this report.

We can find some very interesting tutorials on implementing VAEs with TensorFlow:

- *Convolutional VAE: An example with tf.keras and eager* on Tensorflow 1.13 : `https://GitHub.com/tensorflow/tensorflow/blob/r1.13/tensorflow/contrib/eager/python/examples/generative_examples/cvae.ipynb`
- *Convolutional Variational Autoencoder* on TensorFlow 2.0 : `https://www.tensorflow.org/beta/tutorials/generative/cvae?.`

II.1.3 How does `medGAN` work?

This subsection is taken from Edward CHOI's paper [5].

In this subsection, we will only address one type of method for generating synthetic electronic health records (EHR): `medGAN`. As a recall, `medGAN` stands for "medical generative adversarial network". Some of these other methods are presented in section « 2. Related work » of CHOI's paper [5].

Here is the abstract of the paper:

> « Access to electronic health record (EHR) data has motivated computational advances in medical research. However, various concerns, particularly over privacy, can limit access to and collaborative use of EHR data. Sharing synthetic EHR data could mitigate risk.
>
> In this paper, we propose a new approach, medical Generative Adversarial Network (`medGAN`), to generate realistic synthetic patient records. Based on input real patient records, `medGAN` can generate high-dimensional discrete variables (e.g., binary and count features) via a combination of an autoencoder and generative adversarial networks. We also propose minibatch averaging to efficiently avoid mode collapse, and increase the learning efficiency with batch normalization and shortcut connections. To demonstrate feasibility, we showed that `medGAN` generates synthetic patient records that achieve comparable performance to real data on many experiments including distribution statistics, predictive modeling tasks and a medical expert review. We also empirically observe a limited privacy risk in both identity and attribute disclosure using `medGAN`. »

⚠ Let us note that the term "synthetic" samples means "fabricated" samples and not "summary" samples: the generated data has the same number of variables as the input data. A *synthetic dataset* is a repository of data that is generated programmatically.

II.1.3.a What does `medGAN` do?

The `medGAN` paper was published in 2018 while the first paper about GANs was published in 2014. When the paper [5] was published, GANs had not been used for learning the distribution of discrete variables. The paper reacts as follows:

« To address this limitation, we introduce `medGAN`, a neural network model that generates high-dimensional, multi-label discrete variables that represent the events in EHRs (e.g., diagnosis of a certain disease or treatment of a certain medication). Using EHR source data, `medGAN` is designed to learn the distribution of discrete features, such as diagnosis or medication codes via a combination of an autoencoder and the adversarial framework. In this setting, the autoencoder assists the original GAN to learn the distribution of multi-label discrete variables. The specific contributions of this work are as follows:

- We define an efficient algorithm to generate high-dimensional multi-label discrete samples by combining an autoencoder with GAN, which we call `medGAN`. This algorithm is notable in that it handles both binary and count variables.
- We propose a simple, yet effective, method called *minibatch averaging* to cope with the situation where GAN overfits to a few training samples (i.e., mode collapse), which outperforms previous methods such as *minibatch discrimination*.
- We demonstrate a close-to-real data performance of `medGAN` using real EHR datasets on a set of diverse tasks, which include reporting distribution statistics, classification performance and medical expert review.
- We empirically show that `medGAN` leads to acceptable privacy risks in both presence disclosure (i.e., discovery that a patient's record contributed to the GAN) and attribute disclosure (i.e., discovery of a patient's sensitive medical data). »

In this report, we will only detail the first point about `medGAN`: the combination of an autoencoder with GANs. Autoencoders were presented in subsection II.1.2 of this report.

II.1.3.b Description of EHR Data and Notations

We assume there are $|\mathcal{C}|$ discrete variables (e.g., diagnosis, medication or procedure codes) in the EHR data that can be expressed as a fixed-size vector $\boldsymbol{x} \in \mathbb{Z}_+^{|\mathcal{C}|}$, where the value of the i-th dimension x_i indicates the number of occurrences (i.e., counts) of the i-th variable in the patient record :

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_{|\mathcal{C}|} \end{pmatrix} \tag{II.2}$$

In addition to the count variables, a visit can also be represented as a binary vector $\boldsymbol{x} \in \{0,1\}^{|\mathcal{C}|}$, where the i-th dimension indicates the absence or occurrence of the i-th variable in the patient record.

It should be noted that we can also represent demographic information, such as age and gender, as count and binary variables, respectively.

Learning the distribution of count variables is generally more difficult than learning the distribution of binary variables.

II.1.3.c Combining GANs with an autoencoder

The architecture of `medGAN` is given in figure II.5. The discrete $\boldsymbol{x}$ comes from the source EHR data, $\boldsymbol{z}$ is the random prior for the generator G ($\boldsymbol{z}$ is continuous). G is a feedforward network with shortcut connections (right-hand side figure). An autoencoder (i.e, the encoder Enc and decoder Dec) is learned from $\boldsymbol{x}$. The same decoder Dec is used after the generator G to construct the discrete output. The discriminator D tries to differentiate real input $\boldsymbol{x}$ and discrete synthetic output $Dec(G(z))$.

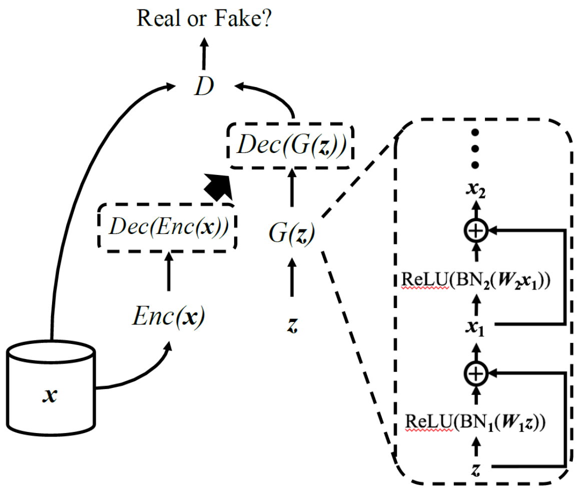


Figure II.5: Architecture of `medGAN`
Source: `medGAN` paper [5]

Since the generator G is trained by the error signal from the discriminator D via backpropagation, the original GAN can only learn to approximate discrete patient records $x \in \mathbb{Z}_+^{|\mathcal{C}|}$ with continuous values. We alleviate this limitation by leveraging the autoencoder. As seen in subsection II.1.2, autoencoders are trained to project given samples to a lower dimensional space, then project them back to the original space. Such a mechanism leads the autoencoder to learn salient features of the samples.

In this work, we apply the autoencoder to learn the salient features of discrete variables that can be applied to decode the continuous output of G. As depicted by Figure 1, an autoencoder consists of an encoder $Enc(\boldsymbol{x};\theta_{enc})$ that compresses the input $x \in \mathbb{Z}_+^{|\mathcal{C}|}$ to $Enc(\boldsymbol{x}) \in \mathbb{R}^h$, and a decoder $Dec(Enc(\boldsymbol{x});\theta_{dec})$ that decompresses $Enc(x)$ to $Dec(Enc(\boldsymbol{x}))$ as the reconstruction of the original input $\boldsymbol{x}$.

With the pre-trained autoencoder, we can allow GAN to generate distributed representation of patient records (i.e., the output of the encoder Enc), rather than generating patient records directly. Then the pretrained decoder Dec can pick up the right signals from $G(z)$ to convert it to the patient record $Dec(G(z))$. The discriminator D is trained to determine whether the given input is a synthetic sample $Dec(G(z))$ or a real sample $\boldsymbol{x}$. The architecture of the proposed model `medGAN` is depicted in Figure II.5.

It should be noted that we can round the values of $Dec(G(z))$ to their nearest integers to ensure that the discriminator D is trained on discrete values instead of continuous values. The paper [5] experimented both with and without rounding and empirically found that training D in the latter scenario led to better predictive performance. Therefore, we assume, for the remainder of this report, that D is trained without explicit rounding.

II.2 Algorithmic implementation

The code in this section is based on CHOI's paper [5] and is accessible at `https://github.com/mp2893/medgan`.

Rather than writing everything in this report, my work on the datasets and coding can be found on my own GitHub repository about `medGAN`: `https://github.com/sylvaincom/medgan-tips`. I am glad that I was able to make some contributions to Edward Choi's code:

- Fixing an error due to version 1.16.3 of NumPy: `https://github.com/mp2893/medgan/pull/15`
- Fixing an error when running step 2-3 with count variables: `https://github.com/mp2893/medgan/pull/17`

In this report, I add details but the core of the algorithmic implementation is on my GitHub repository.

II.2.1 The `medGAN` program from Edward CHOI's GitHub

II.2.1.a Edward CHOI's GitHub

We use TensorFlow 1.2. Here is the description of the code on CHOI's GitHub:

> « This code trains a generative adversarial network to generate patient records. This work currently can handle patient records that are aggregated over time, hence represented as a matrix where a row corresponds to a patient, and a column to a specific medical code (e.g. diagonsis code, medication code, or procedure code). The value of the matrix could either be binary (i.e. a specific medical code occurred in the longitudinal patient record or not) or count (i.e. how many times a specific medical code occurred in the longitudinal patient record). »

This GitHub is composed of two programs that have since been updated for Python 3:

- `process_mimic.py` (124 lines) inputs the public MIMIC-III dataset and outputs a suitable training dataset for `medGAN`,
- `medgan.py` (410 lines) inputs the output of `process_mimic.py` and outputs the generated (fake) multi-label discrete patient records.

It is important to gain access to the MIMIC-III database and then run it through `process_mimic.py` so that we can understand how we should format the input data for `medgan.py` and thus run `medgan.py` on our own data.

`medgan.py` is based on algorithm 1 from subsection I.2.3.

II.2.1.b The free and public MIMIC-III dataset

The MIMIC-III (Medical Information Mart for Intensive Care III) [24] [25] database is a free publicly available hospital database containing de-identified data from approximately 40,000 patients.

This data comes from patients who were admitted to Beth Israel Deaconess Medical Center in Boston, Massachusetts from 2001 to 2012. Upon request, this dataset can be downloaded from `https://mimic.physionet.org/gettingstarted/access/`.

Accessing this dataset can take a few weeks and the process is detailed in a tutorial available at `https://towardsdatascience.com/getting-access-to-mimic-iii-hospital-database-for-data-science-projects-791813feb735`. Here are the steps for getting access to the MIMIC-III dataset:

1. complete CITI "Data or Specimens Only Research" training course
2. create a PhysioNet account
3. request access to MIMIC III
4. accessing MIMIC III

In order to access the MIMIC-III dataset, we need to get a certificate – by completing a CITI course – showing that we will be respectful while using this dataset (particularly concerning privacy and consent).

II.2.2 Explanation of the code's steps

For more information, please check my GitHub repository: `https://github.com/sylvaincom/medgan-tips`.

In this subsection, I will explain the main steps of the two programs `process_mimic.py` and `medgan.py`.

II.2.2.a Explanation of the steps of `process_mimic.py`

`process_mimic.py` is a Python 3 script that processes MIMIC-III dataset and builds a binary matrix or a count matrix (depending on the input). The output matrix is a Numpy matrix of type float32 and suitable for training `medGAN`.

The file uses `if __name__ == '__main__'`. For more information, please refer to:

- the Python documentation: `https://docs.python.org/3/library/__main__.html`,
- an interesting tutorial: `https://www.afternerd.com/blog/python-__name__-__main__/`

II.2.2.b Explanation of the steps of `medgan.py`

`medgan.py` defines a Medgan class. A tutorial on classes in Python can be found at `https://docs.python.org/3/tutorial/classes.html`. Here is the first paragraph of the tutorial:

> « Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state. »

In short, by creating a class, we create a new type of objects with attributes. We can define new functions (called methods) for objects of the created class.

II.2.2.c Running the code

In order to run the code, we need to download the MIMIC-III dataset as well as `process_mimic.py` and `medgan.py`. Out of the 28 files from MIMIC-III, we must put extract `ADMISSIONS.csv` and `DIAGNOSES_ICD.csv` and put them in the same folder as the two Python codes.

II.3 Experimental results

For more information, please check my GitHub repository: `https://github.com/sylvaincom/medgan-tips`.

II.3.1 For the MIMIC-III dataset of shape (46 520, 1 071) with binary values

II.3.1.a Accuracy of the (fictitious) generated data

We wish to measure the accuracy of the (fictitious) generated dataset called `fict` considering the real-life original one called `real`. Is our (fictitious) generated dataset realistic?

Here are our parameters for `medGAN`:

dataset	number of samples	number of features
`real`	46 520	1 071
`fict`	10 000	1 071

`n_epoch`	`n_pretrain_epoch`	`batch_size`	`nSamples`
1 000	100	1 000	10 000

Table II.1: Our parameters for `medGAN`

As in Choi's paper [5], we use dimension-wise probability in figure II.6 as a measure of accuracy. Indeed, the variables are binary. The values of the x-axis and the y-axis are ordered: we successively compare the Bernoulli success probability for both datasets (`real` and `fict`) for a given feature. Given that our data is binary, for each feature (dimension), we claim that 1 corresponds to success and 0 to failure. Hence the proportion of 1s obtained is the Bernoulli success probability p. For information, we have 1 071 features thus 1 071 scatter points.

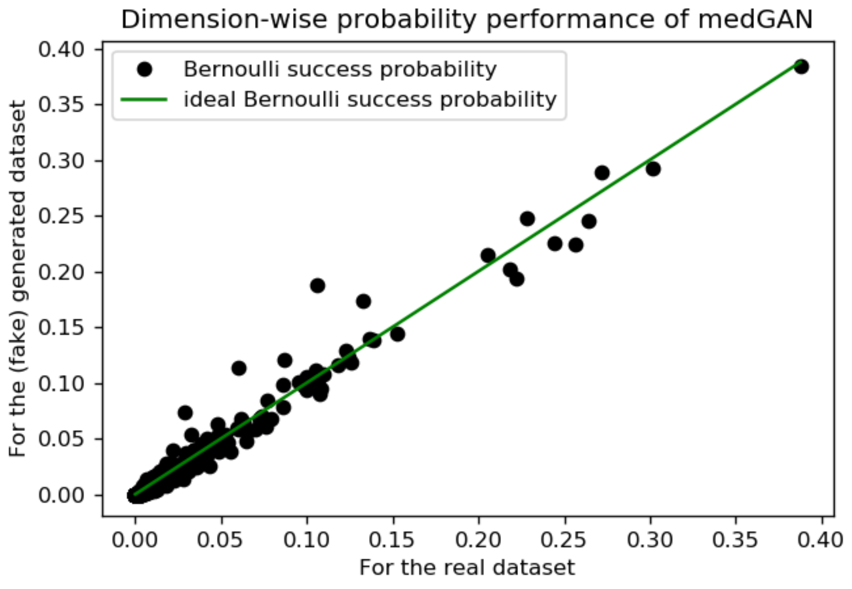


Figure II.6: Is our fictitious generated dataset realistic?

The diagonal green line indicates the ideal performance where the real and the (fake) realistic generated data show identical proportions of 1s. Based on figure II.6, as the dots are close to the diagonal green line, we can say that `medGAN` has a really good performance. I recall that we have a total of 1 071 points so it does not matter if we have a few points that are far from the diagonal green line.

> ✔ EXPERIMENTAL RESULT – The synthesis of binary values using `medGAN` works.
>
> We have observed that `medGAN` can generate fictitious samples with binary values that are realistic.

> ✍ FURTHER WORK – We could quantify the accuracy of the generated dataset.
>
> Rather than observing the accuracy of `medGAN` on a graph such as figure II.6, we could quantify the accuracy by computing the total error: the sum (on all features) of the squares of the Bernoulli success probability difference / error. Graphically, in figure II.6, an individual error (for a given feature) is the distance from the dot to the diagonal green line. We can normalize this total error by dividing it by the total number of features. This normalized squared error is actually the MSE (mean squared error). We do not compute the MSE in this report because we have nothing to compare its value to. We could also use correlations between features.

Now that we have generated fictitious samples that are realistic, we can try to perform dataset augmentation.

II.3.1.b Boosting the prediction score with dataset augmentation

One application of `medGAN` is to use the fictitious generated dataset to help enrich the original real-life dataset (for dataset augmentation) to try to boost the prediction score. Here, we act as if we were in a real-life case and all that we have at our disposal is a real-life dataset (called `real`) of shape (46 520, 1 071). We want to use `medGAN` to generate a new fictitious realistic dataset called `fict` of 10 000 fictitious realistic samples (with 1 071 features as well). By adding (meaning concatenating) `fict` to `real`, we get a new augmented dataset (called `aug`) that has 56 520 samples (patients) (and also 1 071 features). We hope that building our model on `aug` helps our prediction algorithms make better predictions than building our model on `real`. A recap is given at table II.2.

	`real` dataset	`fict` dataset	`aug` dataset
number of samples	46 520	10 000	56 520
number of features	1 071	1 071	1 071

Table II.2: Our parameters for dataset augmentation

How do we compute the prediction score of a dataset? Out of the 1 071 features of our dataset, we select one that we call `target`. We are going to try to predict the `target` feature using the remaining 1 070 features. The scores are computed with cross-validation (thus we do not divide our dataset into train / valid / test). We choose our hyper-parameters with randomized search (using a random seed for reproducibility).

How do we choose `target`? We want to predict the feature with the highest variance. Indeed, a feature with a low variance, for example, with only 1s, is very easy to predict for new unseen samples because we put 1s. Thus, we want `target` to have a proportion of 1s that is the closest to 50%.

✘ MISTAKE TO AVOID – We should not perform dataset augmentation on a `real` dataset that already has a lot of samples.

We note that the `real` dataset contains 46 520 samples. If we already have 46 520 samples (which is a lot), we probably do not need to perform dataset augmentation because we have no shortage of samples. In the next subsection II.3.2, we are going to assume that we only have 1 000 samples at our disposal. Thus, in order to keep this report concise, we are not going to present the results of dataset augmentation on the MIMIC-III dataset of shape (46 520, 1 071) with binary values. By doing so, we can also save some computing time on the prediction scores benchmark (because we have less samples).

II.3.2 For the MIMIC-III dataset of shape (1 000, 100) with binary values

In the previous subsection II.3.1, we wanted to perform dataset augmentation on the MIMIC-III dataset of shape (46 520, 1 071) with binary values. However, in practise, if we already have 46 520 samples, we probably do not need to perform dataset augmentation because we have no shortage of samples. Hence, in this subsection, we are going to randomly select 1 000 samples and randomly select 100 features from the MIMIC-III dataset. We use a random seed for reproducibility. This is our `real` dataset. Thus, we are in a situation where we have a shortage of samples so we try to perform dataset augmentation.

✘ MISTAKE TO AVOID – Do not forget to select the samples and the features of our `real` dataset randomly.

A mistake that I made was to select the first 1 000 samples and the first 100 features of the MIMIC-III dataset to obtain the `real` dataset. With this dataset, my final result was that the prediction score decreases with dataset augmentation. Actually, we should remain as general as possible by doing a random sampling for the `real` dataset.

✍ FURTHER WORK – We should work on more than one `real` dataset.

Actually, instead of doing only one prediction score benchmark of a real dataset (whose samples and features have been randomly selected), we should do several random samplings to obtain several different `real` datasets. Then, we take the mean of the prediction scores on these different `real` datasets for the benchmark.

II.3.2.a Accuracy of the (fictitious) generated data

Here are our parameters:

dataset	number of samples	number of features
`real`	1 000	100
`fict`	1 000	100

`n_epoch`	`n_pretrain_epoch`	`batch_size`	`nSamples`
1 000	100	100	1 000

Table II.3: Our parameters for `medGAN`

As in Choi's paper [5], we use dimension-wise probability in figure II.7. Indeed, the variables are binary.

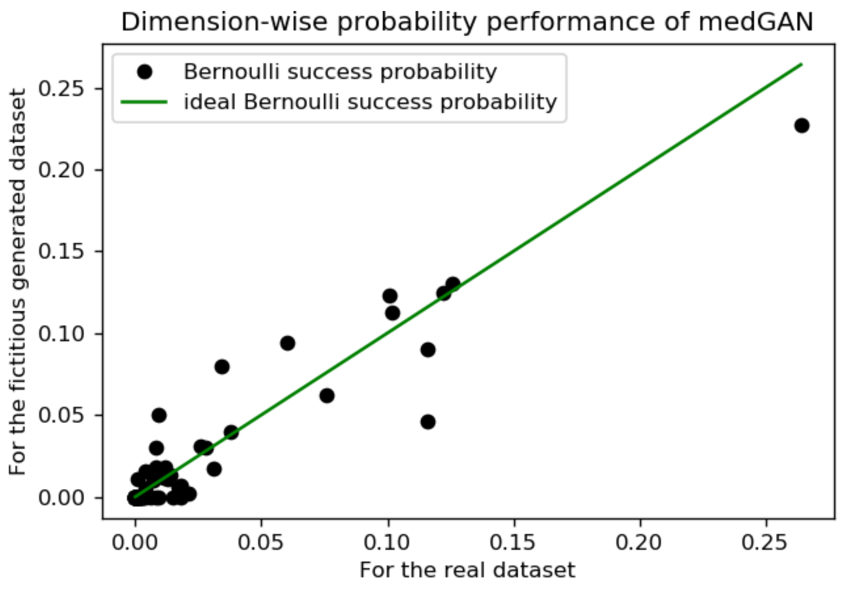


Figure II.7: Dimension-wise probability performance of `medGAN`

Based on figure II.7, as the dots are close to the diagonal green line, we can say that `medGAN` has a really good performance. I recall that that we have a total of 100 points so it does not matter if we have a few points that are far from the diagonal green line.

When comparing figure II.6 to II.7, we can observe that the generated dataset is more realistic in II.6 than II.6, thus when we have more samples, which is intuitive.

> ✍ FURTHER WORK – How to choose the parameters of `medGAN` to make our generated dataset more realistic?
>
> It could be interesting to do a benchmark of several parameters of `medGAN` (`n_epoch`, `nSamples`...) and observe how their corresponding MSE evolves. For example, we could plot the MSE against `n_epoch`. We could also explore some other parameters: number of samples, number of features, number of samples by number of features, correlation between features...

II.3.2.b Boosting the prediction score (5-fold cross-validation) with dataset augmentation

Here, the prediction scores are computed with a 5-fold cross-validation (thus we do not divide our dataset into train / valid / test). We choose our hyper-parameters with randomized search (using a random seed for reproducibility).

The index of the `target` feature is 5. The approximate proportion of 1s of `target` is 0.264 and it is the highest among all features. We could have chosen `target` as the feature the has the highest standard deviation.

> ✍ FURTHER WORK – We should try several values of `target`.
>
> The goal is to remain as general as possible.

First, we do a benchmark of the prediction scores of several machine learning (ML) models with a 5-fold cross-validation on the original real-life dataset (that we called `real`). The results are given table II.4.

ML model	Approx. mean of scores	Approx. variance of scores	Processing time
Logistic Regression	0.738	-	0:00:00.250000
Nearest Neighbors	0.749	-	0:00:00.046875
Naive Bayes	0.406	0.018	0:00:00.015625
Perceptron	0.711	0.051	0:00:00.062500
SVM	0.699	-	0:00:00.375000
Random Forest	0.753	-	0:00:01.250000
Multi-Layer Perceptron	0.742	-	0:00:00.796875

Table II.4: Benchmark of ML models on `real` of shape (1 000, 100)

⚠ The score for the Multi-Layer Perceptron can vary from a simulation to another due to the randomized search even though we use a random seed. Indeed, the randomness from computers is not truly random.

Second, we do a benchmark of the fictitious realistic generated dataset (that we called `fict`). The results are given table II.5.

ML model	Approx. mean of scores	Approx. variance of scores	Processing time
Logistic Regression	0.875	-	0:00:00.421875
Nearest Neighbors	0.865	-	0:00:00.078125
Naive Bayes	0.402	0.059	0:00:00
Perceptron	0.841	0.027	0:00:00.015625
SVM	0.863	-	0:00:00.531250
Random Forest	0.867	-	0:00:01.234375
Multi-Layer Perceptron	0.866	-	0:00:01.218750

Table II.5: Benchmark of ML models on `fict` of shape (1 000, 100)

Third, we do a benchmark of the prediction scores on the augmented dataset (that we called `aug`). The results are given table II.6. Note that II.6 is kind of a mean of II.4 and II.5 because `aug` is balanced between 50% of values from `real` and 50% of values from `fict`.

ML model	Approx. mean of scores	Approx. variance of scores	Processing time
Logistic Regression	0.792	-	0:00:00.625000
Nearest Neighbors	0.792	-	0:00:00.125000
Naive Bayes	0.391	0.038	0:00:00.015625
Perceptron	0.765	0.095	0:00:00.078125
SVM	0.784	-	0:00:00.890625
Random Forest	0.793	-	0:00:01.718750
Multi-Layer Perceptron	0.788	-	0:00:08.046875

Table II.6: Benchmark of ML models on `aug` of shape (2 000, 100)

Finally, we compute the increase of the prediction score from `real` to `aug`, thus due to dataset augmentation. The results are given table II.7. We increase the prediction score of almost all the models.

ML model	Prediction score increase (%)
Logistic Regression	7.32
Nearest Neighbors	5.74
Naive Bayes	-3.69
Perceptron	7.59
SVM	12.16
Random Forest	5.31
Multi-Layer Perceptron	6.2

Table II.7: Benchmark of scores' increase from `real` to `aug` on ML models

We can observe graphically the increase in the prediction score figure II.8. The values of the x-axis and the y-axis are ordered. The diagonal green line indicates where the real and the augmented data show identical performance for a given machine learning model. Based on figure II.8, we can say that `medGAN` can perform dataset augmentation and boost the prediction score. Indeed, the dots are mostly on top of the green line.

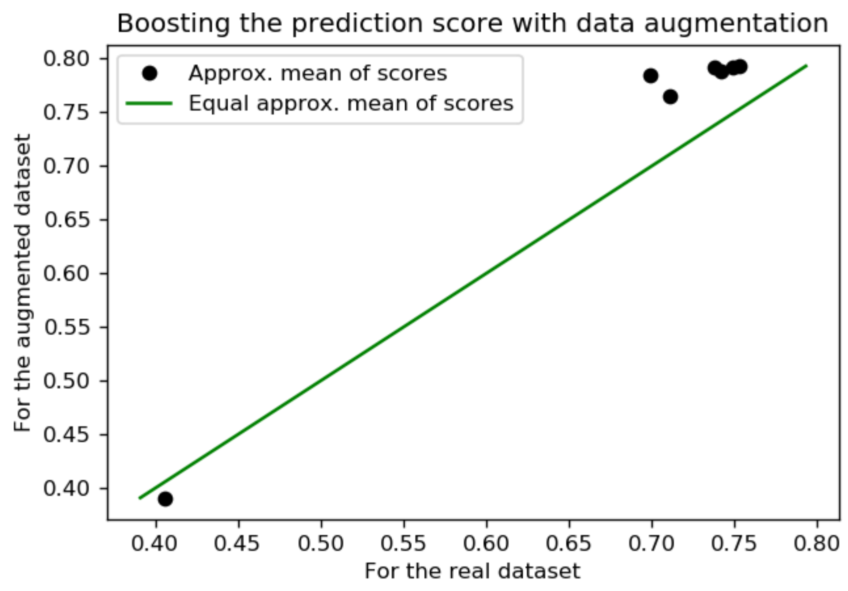


Figure II.8: Boosting the prediction score with dataset augmentation

✘ MISTAKE TO AVOID – We should not try to measure the score increase of dataset augmentation with a cross-validation because `target` would contain fictitious generated values.

In table II.7 and figure II.8, we showed that dataset augmentation can boost the prediction score. However, we kind of cheated because half of the values of `target` in `aug` are fictitious generated values from `fict`. Thus, we try to use fictitious features to predict a `target` feature that is also fictitious. Hence, the score increase is natural and not due to dataset augmentation itself.

We now try to avoid this problem by dividing our datasets into train/test and choosing our hyper-parameters with a randomzied search (using a random seed for reproducibility). We take take `real` (or `aug`) for the training set and we take only real-life values for the test set. The real-life values for the test set will come from values from the full MIMIC-III dataset. We recall that we only took 1 000 samples out of the 46 520 for `real`.

✘ MISTAKE TO AVOID – We must not replace the values of `target` in `fict` with some other ones.

By trying to avoid the previous issue, I tried to replace the `target` values of `fict` with real-life `target` values. I took real-life `target` values of samples from MIMIC-III that are not already in `real` (called `target_real`). Note that `target_real` is a vector. The result I got is that dataset augmentation decreases the score.

However, this is a mistake because in the `fict` dataset (split into `X_fict` and `y_fict`), we no longer correspond `X_fict` to `y_fict` but to `target_real`. The values in `target_real` are ordered but the values in `X_fict` are not ordered in the same way as the vector `target_real`. In other words, `y_fict` and `target_real` do not have the same order.

II.3.2.c Boosting the prediction score (on a proper test set) with dataset augmentation

Contrary to subsubsection II.3.2.b, we no longer use cross-validation to compute the prediction score of our machine learning models.

We now split our `real` dataset of shape (1 000, 100) into `X_train` and `y_train` (that is actually `target`). We try to use `X_train` and `y_train` to build a model that can predict y for an unseen X.

For the `test` set, we randomly select 250 samples (and the same 100 features as `real`) from the complete MIMIC-III dataset of shape (46 520, 1 071) that are not already samples in `real`. We split our `test` dataset of shape (250, 100) into `X_test` and `y_test` (that is actually `target`).

Let `model` be a machine learning model of our benchmark such as the perceptron. We fit the model with `model.fit(X_train, y_train)` then compute the score with `model.score(X_test, y_test)`.

✍ FURTHER WORK – For a given `real` dataset, we should take the mean of scores on several randomly chosen `test` sets in the complete MIMIC-III dataset.

The goal is to be as general as possible. Indeed, we no longer use the cross-validation function from `sklearn` so we must use different training sets to get a score that is more representative of our data.

We do a benchmark of the prediction scores of several ML models on the training set `real` and the `test` set. The results are given table II.8.

ML model	Approx. mean of scores	Processing time
Logistic Regression	0.708	0:00:00.109375
Nearest Neighbors	0.684	0:00:00.062500
Naive Bayes	0.432	0:00:00
Perceptron	0.688	0:00:00.062500
SVM	0.704	0:00:00.156250
Random Forest	0.664	0:00:01.187500
Multi-Layer Perceptron	0.684	0:00:00

Table II.8: Benchmark of ML models on the training set `real` of shape (1 000, 100)

We do a benchmark of the prediction scores of several ML models on the training set `aug` and the `test` set. The results are given table II.9.

ML model	Approx. mean of scores	Processing time
Logistic Regression	0.716	0:00:00.250000
Nearest Neighbors	0.704	0:00:00.093750
Naive Bayes	0.444	0:00:00
Perceptron	0.724	0:00:00
SVM	0.716	0:00:00.484375
Random Forest	0.68	0:00:01.656250
Multi-Layer Perceptron	0.712	0:00:13.453125

Table II.9: Benchmark of ML models on the training set `aug` of shape (2 000, 100)

We can now compute the increase of the prediction score from `real` to `aug`. The results are given table II.10. With dataset augmentation, we increased the prediction score of all the models, which is remarkable. For the perceptron, we even get an increase of more than 5%.

ML model	Prediction score increase (%)
Logistic Regression	1.13
Nearest Neighbors	2.92
Naive Bayes	2.78
Perceptron	5.23
SVM	1.70
Random Forest	2.41
Multi-Layer Perceptron	4.09

Table II.10: Benchmark of scores' increase from `real` to `aug` on ML models

We can observe graphically the increase in the prediction score figure II.9. Based on figure II.9, we can say that `medGAN` can perform dataset augmentation and boost the prediction score.

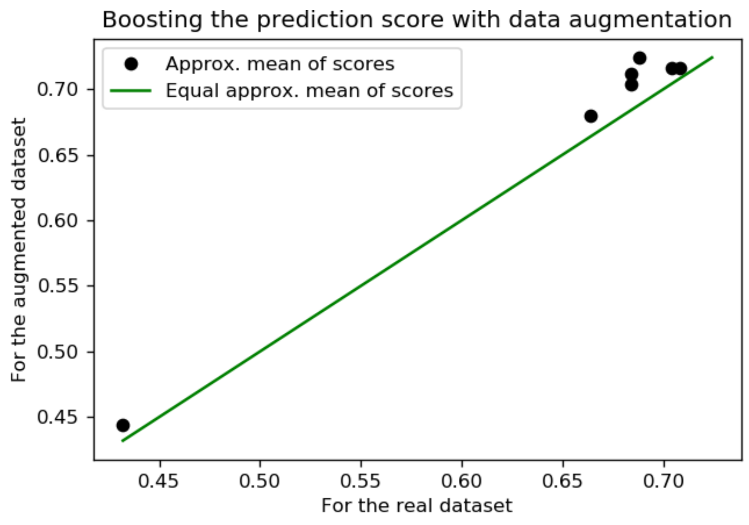


Figure II.9: Boosting the prediction score with dataset augmentation

✔ EXPERIMENTAL RESULT – Using `medGAN` to boost the prediction score works on binary values.

We have observed that `medGAN` can generate fictitious samples with binary values that are realistic. By concatenating the fictitious dataset to the real-life dataset into an augmented training dataset, we can improve the prediction score by up to 5%.

Using the exact same parameters as in table II.10, I once obtained an increase in the score of 9.82% for the Multi-Layer Perceptron (MLP). Indeed, as explained earlier, the score of the MLP can vary from a simulation due to the randomized search.

> ✍ FURTHER WORK – We should run several simulations (because of the randomized search) and take the mean of scores.
>
> The goal is to remain as general as possible. We could also use a grid search for the MLP model but the computing time would be much longer.

> ✍ FURTHER WORK – How to choose the parameters of `medGAN` to increase the prediction score?
>
> It could be interesting to do a benchmark of several parameters of the ML benchmark (the size of the test size, the accuracy of our generated dataset (MSE)...) and observe how their prediction scores evolve. For example, we could plot the prediction score increase (mean of all of the ML models) against the MSE. Instead of going through the MSE, we could also plot the score increase against the number of samples of `real` for example.

II.3.3 For the MIMIC-III dataset of shape (46 520, 1 071) with count values

Accuracy of the (fictitious) generated data

We wish to measure the accuracy of the fictitious generated dataset called `fict` considering the real-life original one called `real`. Is our fictitious generated dataset realistic?

Here are our parameters:

dataset	number of samples	number of features
`real`	46 520	1 071
`fict`	10 000	1 071

`n_epoch`	`n_pretrain_epoch`	`batch_size`	`nSamples`
1 000	100	1 000	10 000

Table II.11: Our parameters for `medGAN`

Contrary to binary values, we can no longer use dimension-wise probability as a measure of performance.

We use statistical indicators: mean, standard deviation and quantile. These indicators are very basic but they still enable us to get a quick sense that our generated data is quite close to the real-life data. In Choi's paper [5] or in the paper about dataset augmentation [6], the efficiency measure is more advanced but more complicated to implement.

We plot the mean at figure II.10. We can observe that our datasets have almost the same mean for each feature, thus that our generated data is quite accurate.

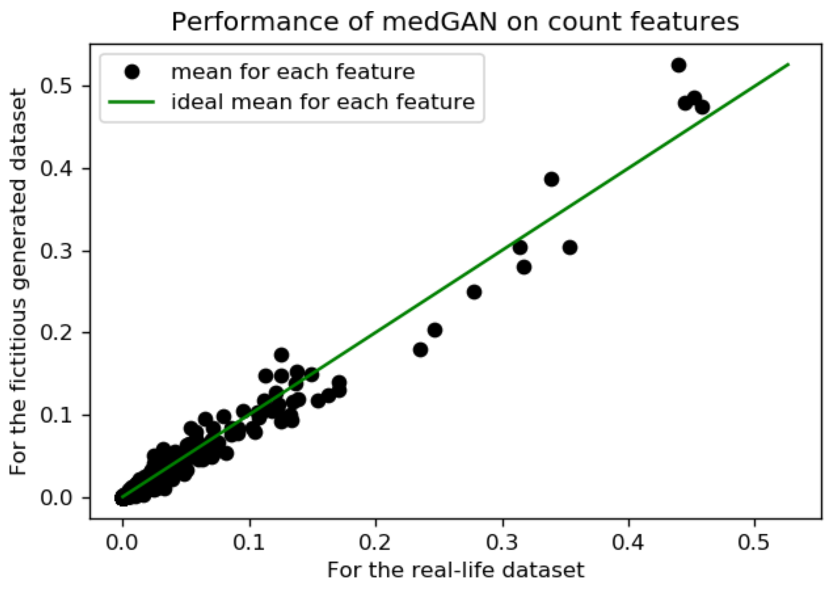


Figure II.10: Comparing the mean of the features of both datasets

We plot the standard deviation at figure II.11. We can observe that our datasets have almost the same standard deviation for each feature, thus that our generated data is quite accurate.

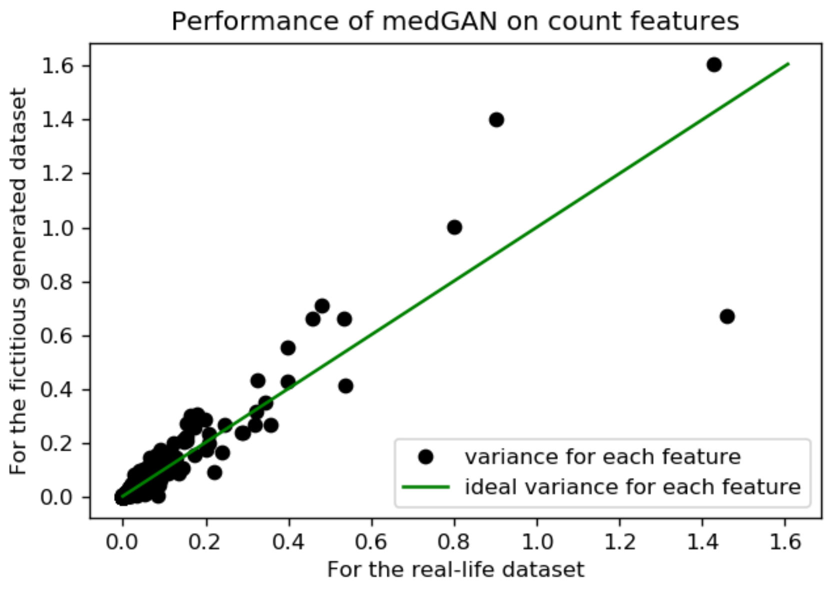


Figure II.11: Comparing the standard deviation of the features of both datasets

We plot the median (0.5 quantile) at figure II.12. We can observe that our datasets have almost the same median for each feature, thus that our generated data is quite accurate. Up to the 0.6 quantile (thus including the median), we can only observe one dot (they are superposed) at $(0,0)$ because the quantile for each column is zero for both datasets. Hence, even if we have count variables, we get a lot of zeros.

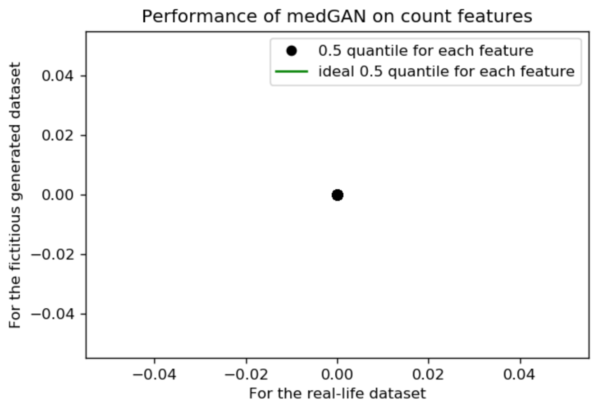


Figure II.12: Comparing the median of the features of both datasets

II.3.4 For the MIMIC-III dataset of shape (1 000, 100) with count values

II.3.4.a Accuracy of the (fictitious) generated data

Here are our parameters:

dataset	number of samples	number of features
real	1 000	100
fict	1 000	100

n_epoch	n_pretrain_epoch	batch_size	nSamples
1 000	100	1 000	1 000

Table II.12: Our parameters for `medGAN`

We plot the mean at figure II.13. We can observe that our datasets have almost the same mean for each feature, thus that our generated data is quite accurate.

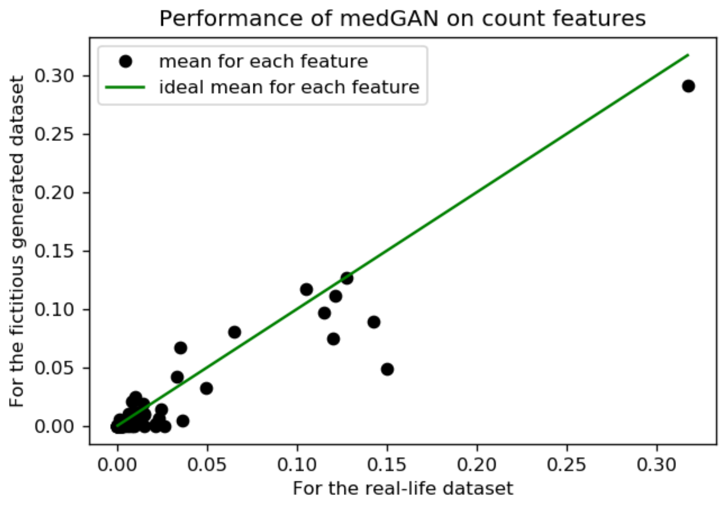


Figure II.13: Comparing the mean of the features of both datasets

We plot the standard deviation at figure II.14. We can observe that our datasets have almost the same standard deviation for each feature, thus that our generated data is not really accurate. It may be because synthesizing count values is harder than synthesizing binary ones.

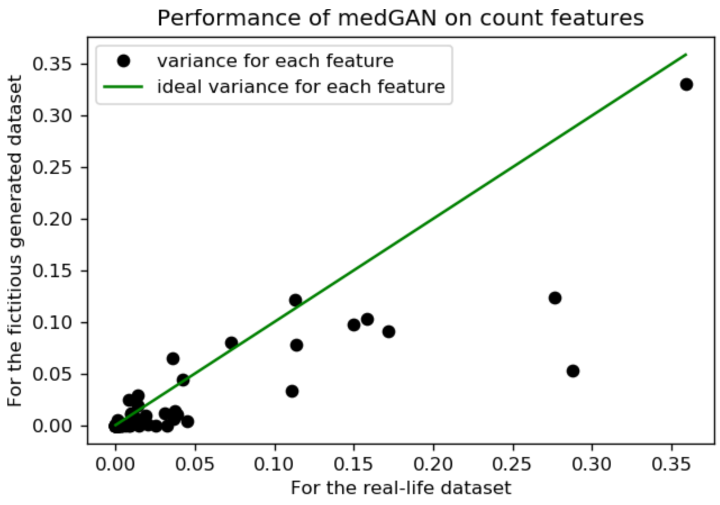


Figure II.14: Comparing the standard deviation of the features of both datasets

II.3.4.b Boosting the prediction score (on a proper test set) with dataset augmentation

We apply the same method as for subsubsection II.3.2.c.

We do a benchmark of the prediction scores of several ML models on the training set `real` and the `test` set. The results are given table II.13.

	Approx. score	Processing time
Random Forest	0.78	0:00:01.109375
Logistic Regression	0.77	0:00:00
Nearest Neighbors	0.77	0:00:00
Perceptron	0.77	0:00:00.062500
SVM	0.75	0:00:00.296875
Multi-Layer Perceptron	0.72	0:00:00
Naive Bayes	0.25	0:00:00.062500

Table II.13: Benchmark of ML models on the training set `real` of shape (1 000, 100)

We do a benchmark of the prediction scores of several ML models on the training set `aug` and the `test` set. The results are given table II.14.

	Approx. score	Processing time
Random Forest	0.79	0:00:01.125000
Logistic Regression	0.77	0:00:00.015625
Nearest Neighbors	0.77	0:00:00.046875
Multi-Layer Perceptron	0.76	0:00:05.875000
Perceptron	0.70	0:00:00.015625
SVM	0.68	0:00:01
Naive Bayes	0.25	0:00:00

Table II.14: Benchmark of ML models on the training set `aug` of shape (2 000, 100)

We can now compute the increase of the prediction score from `real` to `aug`. The results are given table II.15. With dataset augmentation, we do not increase the score, in most cases we decrease it. It seems that it is harder with count values than binary values. Actually, it would be very intuitive if the score increased with the accuracy of `fict`. Here, it is hard to generate count values that are realistic.

	Prediction score increase (%)
Logistic Regression	0.00
Nearest Neighbors	0.00
Naive Bayes	0.00
Perceptron	-9.09
SVM	-9.33
Random Forest	1.28
Multi-Layer Perceptron	5.56

Table II.15: Benchmark of scores' increase from `real` to `aug` on ML models

> ✍ FURTHER WORK – In order to make `medGAN` work better on count values, we should customize our neural networks to each dataset.
>
> For example, instead of taking CNNs, we could maybe look into other types of neural networks. The article *Automatically finding the best Neural Network for your GAN* can be interesting to read. Here is the link: `https://towardsdatascience.com/automatically-finding-the-best-neural-network-for-your-gan-c0b97a5949f2`

Conclusion

In this report, I have explained how GANs (generative adversarial networks) work. In short, GANs are a generative model with implicit density estimation, thus the term "generative". All generative models are unsupervised learning methods. The key idea behind GANs is to have two neural networks competing against each other: the generator (the artist) and the discriminator (the art critic), thus the term "adversarial". GANs use two neural networks, thus the term "networks".

GANs were discovered in 2014 by Ian GOODFELLOW [1]. Since 2014, they have mainly been applied to computer vision. Research into applying GANs to patient data is quite recent and young. An important advance occurred in 2018 with Edward CHOI's `medGAN` (for medical GAN) [5]. `medGAN` is a generative adversarial network for generating electronic health records (EHR). Basically, `medGAN` is a combination of GANs and autoencoders. The purpose of autoencoders is to make sure that our generated data is discrete (and not continuous because of the random noise).

There are two main applications of `medGAN` that can benefit Servier: privacy and dataset augmentation. We only looked into dataset augmentation. For information, some papers such as [6] have shown that GANs can boost the prediction score for EHR.

Experimentally, we have shown that `medGAN` can synthesize binary values as well as count values. It seems that it is harder to synthesize count values than binary ones. `medGAN` works badly when mixing count values with binary ones. Moreover, `medGAN` does not take into account continuous values, at least for now. However, binary values are actually very useful. Indeed, we can transform categorical features into binary ones using one-hot encoding. By doing so, we increase the number of features. Furthermore, we can transform continuous or discrete values into binary ones by using intervals. For example, if we have the age feature that is discrete (age 34, age 21...), we can transform it into several binary features: for example does the patient belong to interval of age 10-20? If yes, value 1, otherwise value 0.

Experimentally, we have also shown that `medGAN` can boost the prediction score on the MIMIC-III dataset with binary values. However, dataset augmentation does not seem to work very well on another dataset than MIMIC-III with binary values. It seems that we should customize our parameters but also our neural networks for each dataset, as `medgan.py` was customized for the MIMIC-III dataset (or so it seems).

Once again, the most updated versions of my programs can be found on my GitHub: `https://github.com/sylvaincom`.

Previously, in this report, I have made a lot of suggestions for further works, especially when presenting the experimental results at section II.3. I will not replicate them again in this conclusion.

Bibliography

[1] Ian J. GOODFELLOW, Jean POUGET-ABADIE, Mehdi MIRZA, Bing XU, David WARDE-FARLEY, Sherjil OZAIRY, Aaron COURVILLE, Yoshua BENGIO. Generative Adversarial Nets. *arXiv:1406.2661v18*, 2014.

[2] Fei-Fei LI, Justin JOHNSON, Serena YEUNG. CS231n: Convolutional Neural Networks for Visual Recognition. Lecture 13 | Generative Models. Spring 2017. `http://cs231n.stanford.edu/`

[3] Ian GOODFELLOW, Yoshua BENGIO and Aaron COURVILLE. Deep Learning. *MIT Press*, 2016. `http://www.deeplearningbook.org`

[4] Ian GOODFELLOW. NIPS 2016 Tutorial: Generative Adversarial Networks. *arXiv:1701.00160v4*, 2017.

[5] Edward CHOI, Siddharth BISWAL, Bradley MALIN, Jon DUKE, Walter F. STEWART, Jimeng SUN. Generating Multi-label Discrete Patient Records using Generative Adversarial Networks. *arXiv:1703.06490v3*, 2018.

[6] Zhengping CHE, Yu CHENG, Shuangfei ZHAI, Zhaonan SUNK, Yan LIU. Boosting Deep Learning Risk Prediction with Generative Adversarial Networks for Electronic Health Records. *arXiv:1709.01648v1*, 2017.

[7] Sanjay PURUSHOTHAM, Chuizheng MENG, Zhengping CHE, Yan LIU. Benchmarking deep learning models on large healthcare datasets. Journal of Biomedical Informatics, Volume 83, July 2018, Pages 112-134. `https://www.sciencedirect.com/science/article/pii/S1532046418300716`

[8] Khaled EL EMAM, Elizabeth JONKER, Luk ARBUCKLE, Bradley MALIN. A Systematic Review of Re-Identification Attacks on Health Data PLoS ONE, 6(12):e28071, 2011b. `https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0028071`

[9] Minsuk KAHNG, Nikhil THORAT, Duen Horng (Polo) CHAU, Fernanda B. VIÉGAS, and Martin WATTENBERG. GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation. *arXiv:1809.01587v1*, 2018.

[10] Tero KARRAS, Timo AILA, Samuli LAINE, Jaakko LEHTINEN. Progressive Growing of GANs for Improved Quality, Stability, and Variation. NVIDIA, *arXiv:1710.10196v3*, 2018.

[11] Jun-Yan ZHU, Taesung PARK, Phillip ISOLA, Alexei A. EFROS Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. Berkeley AI Research (BAIR) laboratory, UC Berkeley. *arXiv:1703.10593v6*, 2018.

[12] Phillip ISOLA, Jun-Yan ZHU, Tinghui ZHOU, Alexei A. EFROS. Image-to-Image Translation with Conditional Adversarial Networks. Berkeley AI Research (BAIR) Laboratory, UC Berkeley *arXiv:1703.10593v6*, 2018

[13] Jean-Paul DELAHAYE. L'intelligence artificielle et le test de Turing. *LNA #66*. `https://culture.univ-lille1.fr/fileadmin/lna/lna66/lna66p04.pdf`

[14] Martin GILES. The GANfather: The man who's given machines the gift of imagination. *MIT Technology Review*, 2018. `https://www.technologyreview.com/s/610253/the-ganfather-the-man-whos-given-machines-the-gift-of-imagination/`

[15] Karen HAO. Inside the world of AI that forges beautiful art and terrifying deepfakes. *MIT Technology Review*, 2018. `https://www.technologyreview.com/s/612501/inside-the-world-of-ai-that-forges-beautiful-art-and-terrifying-deepfakes/`

[16] Charlotte JEE. An AI system identified a potential new drug in just 46 days. *MIT Technology Review*, 2019. `https://www.technologyreview.com/f/614251/an-ai-system-identified-a-potential-new-drug-in-just-46-days/`

[17] Arnaud DEVILLAR. Les algorithmes collaborent pour créer de faux réalistes. *Sciences et Avenir* - Décembre 2018 - N°862.

[18] Remy DEMICHELIS. L'intelligence artificielle peut-elle être créative ? *Les Echos*, 2018. `https://www.lesechos.fr/tech-medias/intelligence-artificielle/lintelligence-artificielle-peut-elle-etre-creative-140594`

[19] Remy DEMICHELIS. Les GAN repoussent les limites de l'intelligence artificielle. *Les Echos*, 2018. `https://www.lesechos.fr/tech-medias/intelligence-artificielle/les-gan-repoussent-les-limites-de-lintelligence-artificielle-206875`

[20] Leila MARCHAND. Intelligence artificielle : un grand nom de Google rejoint d'Apple. *Les Echos*, 2019. `https://www.lesechos.fr/tech-medias/intelligence-artificielle/intelligence-artificielle-un-grand-nom-de-google-rejoint-dapple-1007080`

[21] Elisa Braun. La viralité d'une fausse vidéo d'Obama met en lumière le phénomène du «deep fake». *Le Figaro*, 2018. `http://www.lefigaro.fr/secteur/high-tech/2018/04/20/32001-20180420ARTFIG00134-la-viralite-d-une-fausse-video-d8216obama-met-en-lumiere-le-phenomene-du-deep-fake.php`

[22] Martin Ford. How we'll earn money in a future without jobs. *TED*, 2017. `https://www.youtube.com/watch?v=swB7Ivct8d8`

[23] Alexander AMINI, Ava SOLEIMANY 6.S191: Introduction to Deep Learning. MIT, 2018. `http://introtodeeplearning.com/`

[24] MIMIC-III, a freely accessible critical care database. Johnson AEW, Pollard TJ, Shen L, Lehman L, Feng M, Ghassemi M, Moody B, Szolovits P, Celi LA, and Mark RG. Scientific Data (2016). DOI: 10.1038/sdata.2016.35. Available from: `http://www.nature.com/articles/sdata201635`

[25] Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. Circulation 101(23):e215-e220 [Circulation Electronic Pages; `http://circ.ahajournals.org/content/101/23/e215.full`]; 2000 (June 13).

Notation

We provide a concise (non exhaustive) reference describing the notation used throughout this book.

a	a scalar (integer or real)
$\boldsymbol{a}$	a vector
$\boldsymbol{A}$	a matrix
a	a scalar random variable
$\mathbf{a}$	a vector-valued random variable
$\mathbf{A}$	a matrix-valued random variable
$f(\boldsymbol{x};\boldsymbol{\theta})$	function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$
$\mathbb{E}_{x\sim P}\left[f(x)\right]$	the expectation of $f(x)$ with respect to $P(x)$
p_{data}	the data generating distribution
$\widehat{p}_{\text{data}}$	the empirical distribution defined by the training set
$\mathbb{X}$	a set of training examples
$\boldsymbol{x}^{(i)}$	the i-th example (input) from a data set
$y^{(i)}$ or $\boldsymbol{y}^{(i)}$	the data set associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\boldsymbol{X}$	the $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$

List of Figures

List of Tables