Generating fictitious realistic patient data using generative adversarial networks (GANs)

Author: Sylvain COMBETTES

Supervisors: Fabrice COUVELARD, Romain GUILLIER

September 13th, 2019

Introduction

- `https://www.youtube.com/watch?v=cQ54GDm1eL0`
  ➥ an example of deepfake (enabled by GANs)
- FaceApp
- « *[GANs are] the coolest idea in deep learning in the last 20 years* » – Yann LECUN, Facebook's chief AI scientist
- « *[GANs represent] a significant and fundamental advance* » – Andrew NG, former chief scientist of China's Baidu

Introduction

- goal of AI: simulate human intelligence → creativity
- generative models → generative adversarial networks (GANs) by Ian GOODFELLOW (« the GANfather ») in 2014
- mainly for computer vision → what about patient data?
- *Example*: portrait constructed from 15,000 examples in 2018 with GANs sold with an auction price of 432 000$

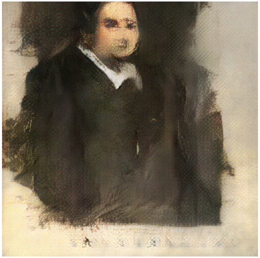


Table of contents

1. General presentation on GANs
   - Some preliminary notions
   - How do GANs work?
2. Application of GANs to patient data
   - Theoretical approach: `medGAN`
   - Algorithmic implementation
   - Experimental results

Main references for this talk

- Ian GOODFELLOW et al. Generative Adversarial Nets. 2014.
- Ian GOODFELLOW. NIPS 2016 Tutorial: Generative Adversarial Networks. 2017.
- Fei-Fei LI et al. CS231n: Convolutional Neural Networks for Visual Recognition. Lecture 13 | Generative Models. Spring 2017. `http://cs231n.stanford.edu/`
- Ian GOODFELLOW et al. Deep Learning. *MIT Press*, 2016. `http://www.deeplearningbook.org`
- Edward CHOI et al. Generating Multi-label Discrete Patient Records using Generative Adversarial Networks. 2018.

I – General presentation on GANs

I.1 Some preliminary notions

1 General presentation on GANs
- Some preliminary notions
  - Supervised vs. unsupervised learning
  - What is a generative model?
  - Why are generative models interesting?
  - A few important concepts of machine learning
- How do GANs work?

2 Application of GANs to patient data

Supervised vs. unsupervised learning

Supervised learning

- Data x, labels y
- Goal: learn a function $x \mapsto y$
- Example: linear regression

Unsupervised learning

- Data x, **no** labels y
- Goal: learn some underlying hidden structure of the data x
- Example: clustering

Advantages of unsupervised learning

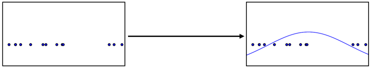
- training is cheaper
- learns some hidden structure

What is a generative model? (1/2)

☛ Given training data, generate new samples from same distribution. Learn a model $p_{\text{model}}(x)$ which is similar to $p_{\text{data}}(x)$.

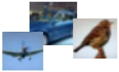
Explicit density estimation

- explicitly define and solve for $p_{\text{model}}(x)$
- Example:



Implicit density estimation

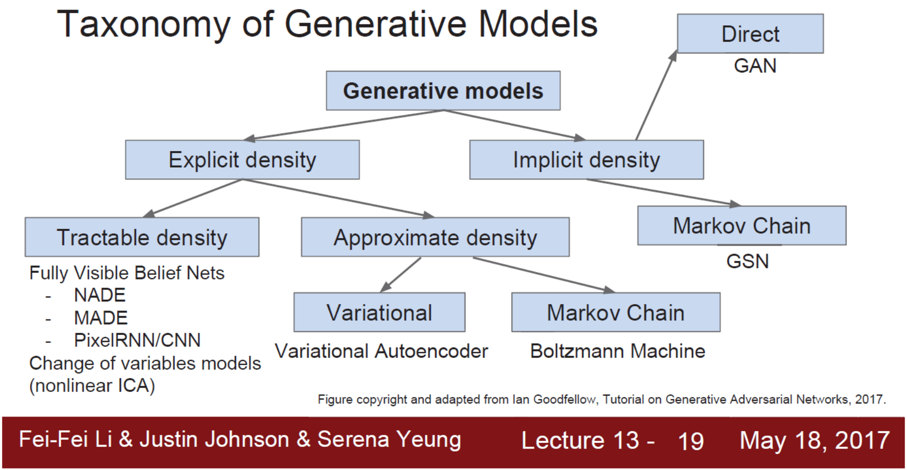
- learn a model that can sample from $p_{\text{model}}(x)$ without explicitly defining $p_{\text{model}}(x)$
- Example:



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

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

What is a generative model? (2/2)



Why are generative models interesting? (1/4)

Excerpt of GOODFELLOW's NIPS 2016 tutorial:

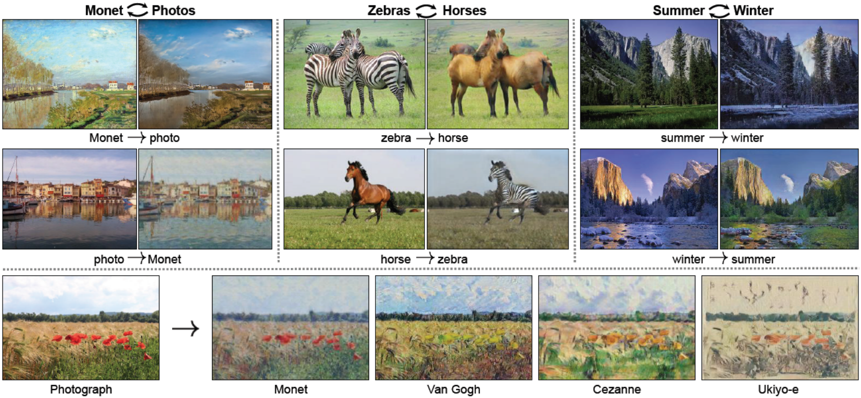
`https://youtu.be/HGYYEUSm-0Q?t=600`

Why are generative models interesting? (2/4)

Realistic fictional portraits of celebrities generated from a high-quality version of the CELEBA dataset consisting of 30 000 images using GANs:

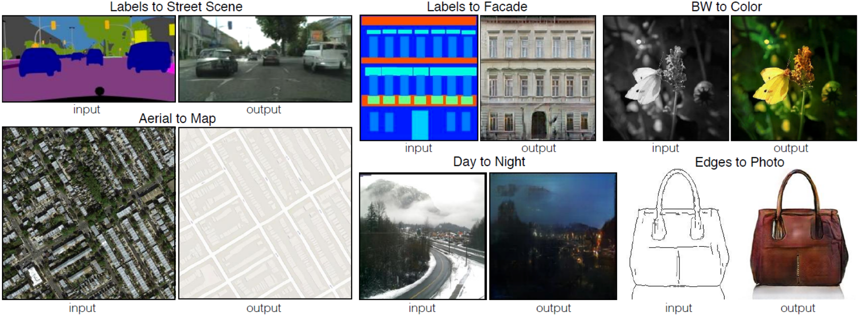
Why are generative models interesting? (3/4)

Real images transposed into realistic fictional images using GANs (image-to-image translation):



Why are generative models interesting? (4/4)

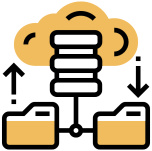
Several types of image transformations using GANs:



A few important concepts of machine learning (1/3)

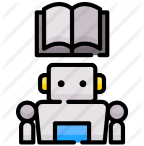
Training dataset

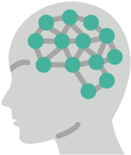
- AI is not creative: it can only learn from the training database.
- small training dataset $\rightarrow$ problems (overfitting...)
- a supervised deep learning algorithm will generally:
  - achieve acceptable performance with around 5,000 labeled examples per category
  - match or exceed human performance when trained with a dataset containing at least 10 million labeled examples
- choose good features
- performance on new unseen data (and not training data)



A few important concepts of machine learning (2/3)
Deep learning

- complicated distribution $\rightarrow$ estimate it with a neural network
- neuroscience: important source of inspiration but no longer the predominant guide
- deep learning is the most used technique in machine learning for computer vision
- deep learning dates back to the 1940s and became popular only recently:
  - amount of available training data $\nearrow$
  - computer infrastructure (both hardware and software) $\nearrow$
- dominant training algorithm: stochastic gradient descent and softmax loss function





A few important concepts of machine learning (3/3)

Maximum Likelihood Estimation (MLE)

- $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ drawn from $p_{\text{data}}(\mathbf{x})$ (unknown)
- $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$: parametric family of probability distributions over the same space indexed by $\boldsymbol{\theta}$
- MLE for $\boldsymbol{\theta}$:

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

- We prefer a sum:

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

- We divide by m:

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

I.2 How do GANs work?

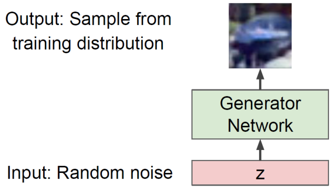
1 General presentation on GANs
- Some preliminary notions
- How do GANs work?
  - The principle: generator vs discriminator
  - The minimax game
  - Gradient descent
  - Application: TensorFlow and GAN Lab

2 Application of GANs to patient data

Introduction

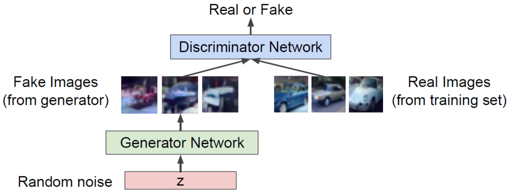
GANs:

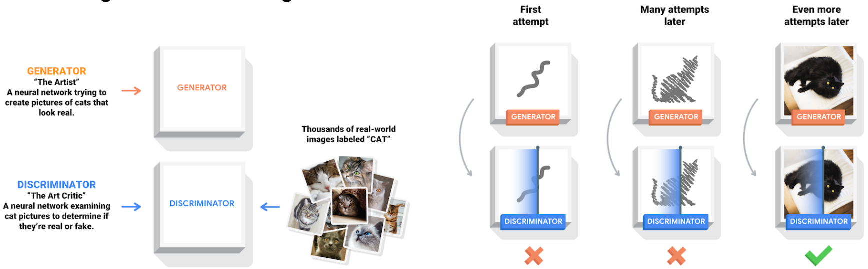
- unsupervised learning
- generative model with implicit density estimation
- using 2 neural networks
- problem: sample from a complex and high-dimensional training distribution
  - → generate from a simple distribution: random noise z
  - → the generated data is not all identical
- mainly for vision



The principle: generator vs discriminator

- generator network: try to fool the discriminator by generating real-looking images
- discriminator network: try to distinguish between real images and fake images





The minimax game (1/2)

- train jointly G and D in minimax game
- minimax objective function:

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

- D outputs the likelihood of real image in $[0, 1]$:
  - $D(x)$ equals 1 if D considers that x is a real data
  - $D(x)$ equals 0 if D considers that x is a fake data (for example a generated data).
- equilibrium when the discriminator can no longer distinguish real images from fakes $\rightarrow$ D outputs $1/2$ everywhere
- $D(x)$ is the output of the discriminator for a real input x
- $D\left(G(z)\right)$ is the output of the discriminator for a fake generated data $G(z)$

The minimax game (2/2)

- (recall) minimax objective function:

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

- Discriminator (θ_d) wants to maximize objective such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake)
- Generator (θ_g) wants to minimize objective such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real)
- unsupervised learning but:
  - the data generated by G has a 0 label for false
  - the real learning data has a 1 label for true
  - $\rightarrow$ define a loss function

Gradient descent (1/3)

(recall) minimax objective function:

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

for training, we will alternate between:

1. gradient ascent on discriminator :

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

2. gradient descent on generator :

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

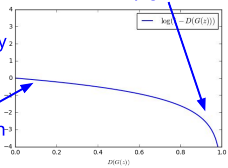
Gradient descent (2/3)

(recall) minimax objective function:

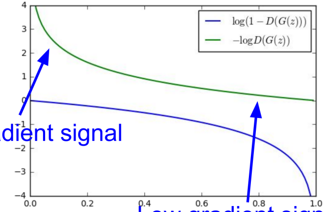
$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x\sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$$

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



minimizing likelihood of discriminator being correct



High gradient signal

Low gradient signal

maximize likelihood of discriminator being wrong

Gradient descent (3/3)

(recall) minimax objective function:

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

for training, we will alternate between:

1. gradient ascent on discriminator:

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

2. gradient ascent on generator:

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

Algorithm 1 GAN training

for number of training iterations **do**
  for k steps **do**
    Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noize prior $p_g(\boldsymbol{z})$. ▷ for the fake data
    Sample minibatch of m noise samples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$. ▷ for the real data
    Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D_{\theta_d}\left(G_{\theta_g}\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

  end for
  Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noize prior $p_g(\boldsymbol{z})$.
  Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log D_{\theta_d}\left(G_{\theta_g}\left(\boldsymbol{z}^{(i)}\right)\right)$$

Application: TensorFlow and GAN Lab

- TensorFlow's tutorials with Google Colab
  - *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`
- GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation
  - `https://poloclub.github.io/ganlab/`
  - `https://youtu.be/eTq9T_sPTYQ?t=37`

II – Application of GANs to patient data

II.1 Theoretical approach: `medGAN`

1. General presentation on GANs

2. Application of GANs to patient data
   - Theoretical approach: `medGAN`
     - How can Servier benefit from GANs?
     - What are autoencoders?
     - How does `medGAN` work?
   - Algorithmic implementation
   - Experimental results

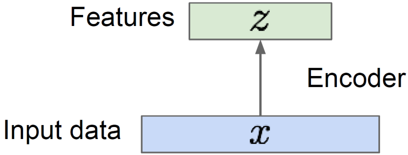
How can Servier benefit from GANs?

- privacy of patients' personal data
  - EHR data is composed of personal identifiers (dates of birth...)
  - de-identification does not work (re-identification)
  - for researchers: better access to EHR data through fake realistic generated data

- data augmentation in order to make better predictions
  - enrich the original training (small) dataset in order to make better predictions
  - very experimental
  - generating fictitious realistic patients with `medGAN` from a dataset of 500 samples with 250 variables $\rightarrow$ suboptimal

What are autoencoders? (1/3)

☛ Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

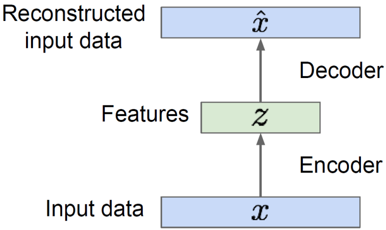


- encoder : function mapping from $\boldsymbol{x}$ to $\boldsymbol{z}$ $\rightarrow$ neural network
- $\boldsymbol{z}$ usually smaller than $\boldsymbol{x}$ (dimensionality reduction) $\rightarrow$ want features $\boldsymbol{z}$ to capture meaningful factors of variation in data $\boldsymbol{x}$

What are autoencoders? (2/3)

How to learn this feature representation?

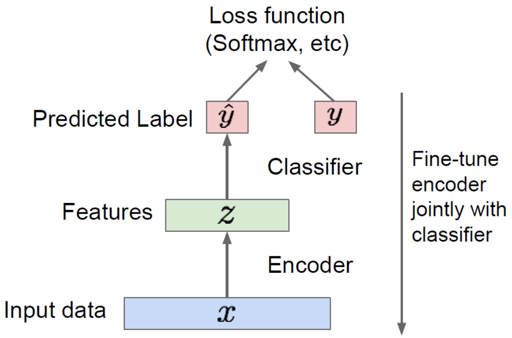
- train such that features can be used to reconstruct original data
- "autoencoding" $\leftrightarrow$ encoding itself
- role of the decoder:



- $\mathcal{L}^2$ loss function: $\|x - \widehat{x}\|^2$ (no labels!)

What are autoencoders? (3/3)

- after training: no need for the decoder
- encoder $\rightarrow$ when we do not have enough input data $\boldsymbol{x}$, we can use the encoder to initialize a supervised learning problem with better features $\boldsymbol{z}$



How does `medGAN` work? (1/2)

`medGAN`: combination of GANs and autoencoders

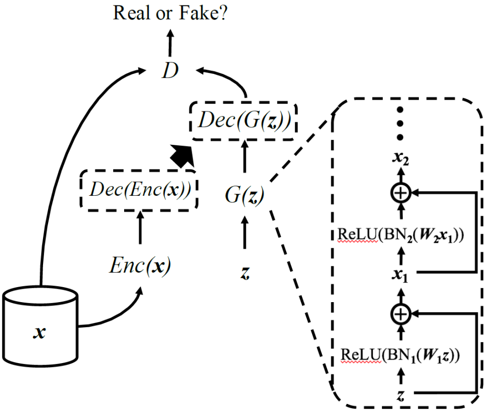
`medGAN`: neural network model that generates

- highdimensional
- multi-label
- discrete

variables that represent the events in EHRs

How does `medGAN` work? (2/2)

`medGAN`: combination of GANs and autoencoders



Why autoencoders?

- the training data $\boldsymbol{x}$ is discrete (binary or count variables)
- the generated data $G(\boldsymbol{z})$ (from the random prior $\boldsymbol{z}$) is continuous
  $\rightarrow Dec(G(\boldsymbol{z}))$ is the synthetic discrete output

II.2 Algorithmic implementation

1 General presentation on GANs

2 Application of GANs to patient data
- Theoretical approach: `medGAN`
- Algorithmic implementation
  - The `medGAN` program from CHOI's GitHub
  - Explanation of the code's steps
- Experimental results

The medGAN program from CHOI's GitHub (1/2)

CHOI's GitHub

- https://github.com/mp2893/medgan
- TensorFlow 1.2, Python 3
- 2 programs:
  - `process_mimic.py` (124 lines)
  - `medgan.py` (410 lines)
- values for `medgan.py`:
  - binary
  - or count

The medGAN program from CHOI's GitHub (2/2)

The free and public MIMIC-III dataset

- MICMIC-III dataset: free publicly available hospital database containing de-identified data from approximately 40,000 patients $\rightarrow$ access to it helps to understand how medGAN works
- features: specific medical code (ICD-9)
- a few important rules:
  - granted to someone as an individual (not for colleagues)
  - must share the code used to produce my results
  - no attempt to identify any individual

Explanation of the code's steps

https://github.com/sylvaincom/medgan-tips

II.3 Experimental results

1. General presentation on GANs
2. Application of GANs to patient data
   - Theoretical approach: `medGAN`
   - Algorithmic implementation
   - Experimental results
     - For the MIMIC-III dataset of shape (46 520, 1 071) with binary values
     - For the MIMIC-III of shape (1 000, 100) with binary values
     - For the MIMIC-III dataset of shape (46 520, 1 071) with count values
     - For the MIMIC-III dataset of shape (1 000, 100) with count values

For MIMIC-III (46 520, 1 071) with binary values (1/4)

Accuracy of the (fictitious) generated data (1/2)

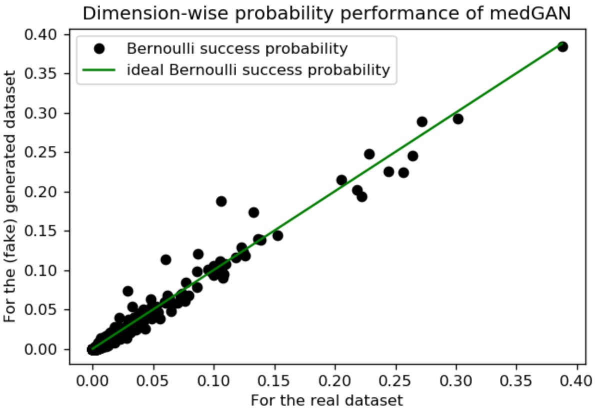
➥ Is our (fictitious) generated dataset realistic?

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

For MIMIC-III (46 520, 1 071) with binary values (2/4)

Accuracy of the (fictitious) generated data (2/2)



✔ The synthesis of binary values using `medGAN` works.

✍ We could quantify the accuracy of `fict` with MSE.

For MIMIC-III (46 520, 1 071) with binary values (3/4)

Boosting the prediction score with data augmentation (1/2)

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

How do we compute the prediction score of a dataset?

- we select one feature called `target`
- try to predict `target` using the remaining 1 070 features
- hyper-parameters $\rightarrow$ randomized search (`sklearn`)
- score $\rightarrow$ cross-validation (`sklearn`)

For MIMIC-III (46 520, 1 071) with binary values (4/4)

Boosting the prediction score with data augmentation (2/2)

How do we choose `target`?

- feature with the highest variance
- a feature with a low variance (ex. with only 1s) is very easy to predict for new unseen samples (because we put 1s)
- we want `target` to have a proportion of 1s that is the closest to 50%

✘ We should not perform data augmentation on a `real` dataset that already has a lot of samples.

For MIMIC-III (1 000, 100) with binary values (1/8)

Accuracy of the (fictitious) generated data (1/2)

Out of the (46 520, 1 071) shaped MIMIC-III dataset, we randomly select 1 000 samples and 100 features.

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

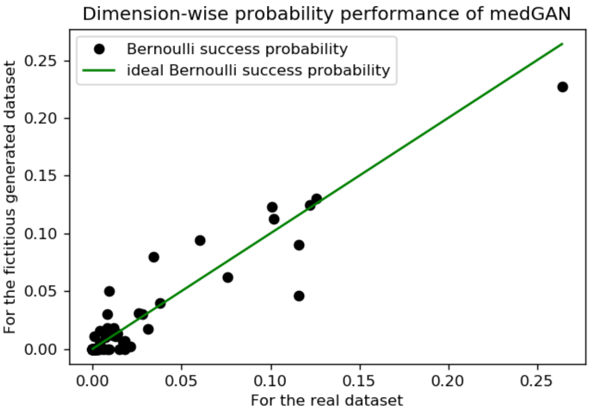
✍ We should work on more than one `real` dataset.

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

For MIMIC-III (1 000, 100) with binary values (2/8)

Accuracy of the (fictitious) generated data (2/2)

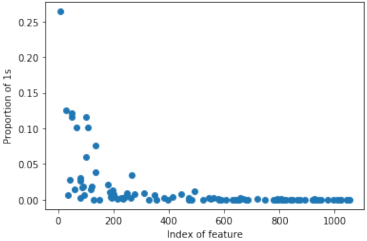


✍ How to choose the parameters of `medGAN` to make our generated dataset `fict` more realistic?

For MIMIC-III (1 000, 100) with binary values (3/8)

Boosting the prediction score (5-fold cross-validation) with data augmentation (1/3)

`target`: feature of index 5, proportion of 1s equal to 0.264



Method:

1. Benchmark of ML models on `real` of shape (1 000, 100)
2. Benchmark of ML models on `aug` of shape (2 000, 100)
3. Benchmark of scores' increase from `real` to `aug` on ML models

For MIMIC-III (1 000, 100) with binary values (4/8)

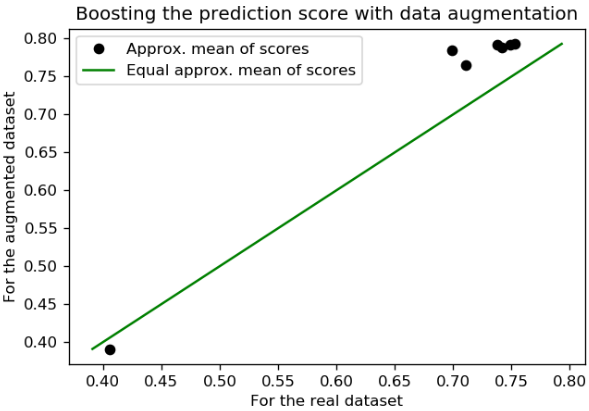
Boosting the prediction score (5-fold cross-validation) with data augmentation (2/3)

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: Benchmark of scores' increase from `real` to `aug` on ML models

For MIMIC-III (1 000, 100) with binary values (5/8)

Boosting the prediction score (5-fold cross-validation) with data augmentation (3/3)



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

For MIMIC-III (1 000, 100) with binary values (6/8)

Boosting the prediction score (on a proper test set) with data augmentation (1/3)

Method:

- Split `real` (1 000, 100) into `X_train` and `y_train` (that is actually `target`).
- Use `X_train` and `y_train` to build a model that can predict *y* for an unseen *X*.
- For `test`, we randomly select 250 samples from MIMIC-III (46 520, 1071) that are not already samples in `real`. We split `test` (250, 100) into `X_test` and `y_test` (that is actually `target`).
- We fit the model with `model.fit(X_train, y_train)` then compute the score with `model.score(X_test, y_test)`.

✍ For a given `real`, we should take the mean of scores on several randomly chosen `test`.

For MIMIC-III (1 000, 100) with binary values (7/8)

Boosting the prediction score (on a proper test set) with data augmentation (2/3)

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

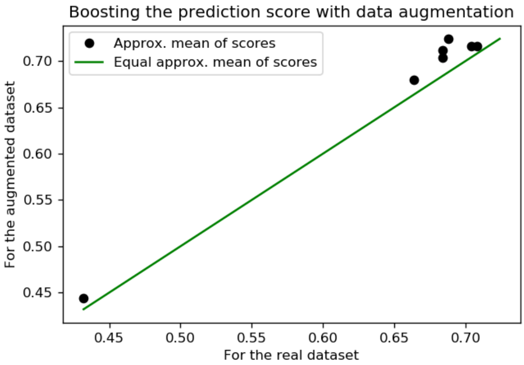
Table: Benchmark of scores' increase from `real` to `aug` on ML models

Perceptron: 0.688 $\rightarrow$ 0.724

✍ We should run several simulations (because of the randomized search) and take the mean of scores.

For MIMIC-III (1 000, 100) with binary values (8/8)

Boosting the prediction score (on a proper test set) with data augmentation (3/3)



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

✍ How to choose the parameters of `medGAN` to increase the prediction score?

For MIMIC-III (46 520, 1 071) with count values (1/2)

Accuracy of the (fictitious) generated data (1/2)

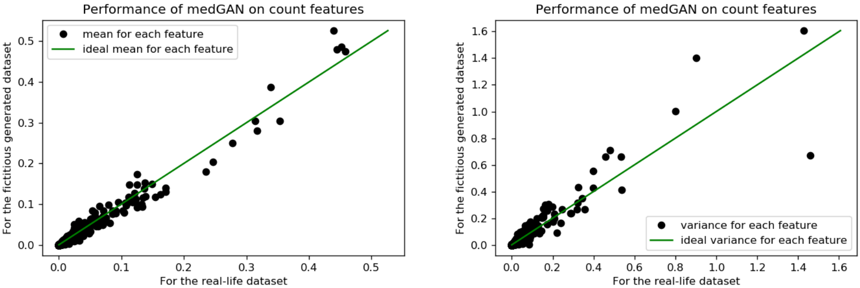
➥ Is our (fictitious) generated dataset realistic?

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

For MIMIC-III (46 520, 1 071) with count values (2/2)

Accuracy of the (fictitious) generated data (2/2)



✔ The synthesis of count values using `medGAN` works.

✍ Find a better measure of accuracy.

For MIMIC-III (1 000, 100) with count values

Boosting the prediction score (on a proper test set) with data augmentation

	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: Benchmark of scores' increase from `real` to `aug` on ML models

✍ `fict` is realistic $\nearrow \implies$ score $\nearrow$

✍ It is harder to synthesize count values than binary ones.

Conclusion

Conclusion

- using `medGAN` to synthesize:
  - ✔ binary values
  - ✔ count values
- using `medGAN` to boost the prediction score with data augmentation:
  - ✔ binary values
  - ✘ count values
- some important results:
  - `fict` is realistic $\nearrow \Longrightarrow$ score $\nearrow$
  - it is harder to synthesize count values than binary ones
  - `medGAN` does not work on continuous values
  - `medGAN` works badly when mixing count values with binary ones
  - binary values are actually very useful (categorical with one-hot encoding, intervals for continuous values...)