Institut Polytechnique de Paris,
Year 2021-2022,
M2 DS

Data-based Generative Model-Project

Prénom, Nom(s): ***Benjamin COHEN, Ronny TONATO***.

The main objective of this document is to describe the methodology of the generative model G implemented for simulating realistic samples from a vector of input noise.

1 Abstract

Financial services generate large amounts of data that are extraordinarily complicated and varied. These data sets are usually kept private among several organisations for numerous reasons, including regulative needs and business desires. As a result, information sharing among completely different lines of financial businesses as outside of the organisation is still severely restricted. It's critical to analyze ways for synthesising monetary data sets that follow the same properties of the real data sets and at the same respect the necessity for privacy of parties involved. Synthetic financial helps computing to replicate shocks and to compute risk metrics. This introductory paper aims to focus on the growing want for effective artificial data generation within the monetary domain. we have a tendency to highlight 3 main areas of focus for the academic community:

- 1) Generating **realistic** artificial data sets.
- 2) **Measure the similarities** between real and generated data sets.
- 3) Guaranteeing the generative process satisfies any **privacy constraints**.

Even though those challenges are also faced in other domains, the additional regulative and privacy needs add another dimension of complexity of the problem and supply a singular opportunity to review the subject in monetary services.

2 Introduction

Log-return of stock price prediction in capital markets has been consistently researched using deep learning. Related to Time Series, recurring neural networks such as **long short-term memory** (LSTM) had been successfully tested to replicate log-return of stock price distributions. Similarly, since 2014 (see [5]), **generative adversarial networks** (GAN) have also been the subject of intense experimentation, especially in the field of computer vision.

It is an **unsupervised learning** technique with purposes to **discover the hidden patterns** in order to get a meaningful representation of the data then expend **simulating new realistic data** from the estimated density function. In this **GAN architecture** you have two **neural networks** pitted against each other, one trying to fool the other with noise, while the other trains on real data and responds with information on how to make that **noise** more realistic. We explored using a GAN with a **multi-layer perceptron** as generator and discriminator to generate the stock index. We took as an input a **noise sample Gaussian** of 1190 observations with 20 columns and the generated output is a sample of 1190 observations for each stock index.

Generative modeling involves employing a model to come up with new examples that probably come from our associate existing distribution of samples, like generating new pictures that are similar however specifically completely different from a data set of existing pictures. Here we will follow the

same principle:
Given a vector of input noise vector Z of financial data of 4 international stock indices, the goal is to make a generative model G which will simulate realistic samples.

3 Methodology

3.1 Several Methods for the same problem

The different method approaches can be broadly classified into: 1] **Applied math** and 2] **Generative Algorithms based**. Applied math techniques used to dominate this field for a huge amount for time, but recently Deep Learning has shown that the use of Generative algorithms gives us better results and is now dominating the sector of artificial knowledge generation through superior quality of information obtaining generated. The **applied math** approach solves the artificial knowledge Generation downside by **fitting a joint variable likelihood distribution** on the real data and then drawing samples from that fitted distribution leading us to define distances between distributions to evaluate how good is our distribution approximation like the **Kullback-Leibler distance**. Whereas this reasoning works for straightforward data sets that can be simply explained by likelihood distributions, most of the real data sets are very **complex** and therefore the approach is restricted by the use and need of advanced distribution.

3.2 Model GAN

GAN network consists of two models including a Discriminator and a Generator. The process of training a GAN may be decomposed in the following steps :

- **The Generator** uses random data to try to generate very close distribution wise generated data which has the purpose to learn the data real distribution.
- We then use this generated data randomly with some real data for **the Discriminator** to learn its parameters. The Discriminator works as a classifier and tries to understand whether the data is coming from the Generator or the real data while estimating the probabilities of the incoming sample belonging to the real data set.
- Then, we combine the losses of both the Discriminator and the Generator to propagate back through the generator. The **Generator's loss** depends on **both** the Generator and the Discriminator. We **back-propagate** through both the discriminator and generator to obtain gradients. It helps the Generator learn about the real samples distribution. If the generator doesn't make a good job generating realistic data (close in distribution) , the Discriminator's work will be very easy to distinguish generated from real data sets. Hence, the **Discriminator's loss** will be very small. Small discriminator loss will result in bigger generator loss as the equation for $\mathbf{L}(\mathbf{D}, \mathbf{G})$ shows it. This makes creating the discriminator a bit tricky, because a discriminator too strong will always result in a huge generator loss, making the generator **unable to learn**.
- We keep the process going until the Discriminator can no longer recognize which data is real and which data is generated. In the original **vanilla GAN**, we introduce a **minimax loss** derived from the cross entropy between the real and generated distributions by the **Jensen–Shannon divergence** measuring the similarity between the two probability distributions when the Discriminator is optimal. The Discriminator (D) and Generator (G) networks are trained separately, by distinctive alternating processes.
- As described above, **both models are competing** against each other in an **adversarial manner** of game theory playing a **zero-sum game**. A zero-sum game is a mathematical depiction in game and economic theory of a situation in which **an advantage that is won by one of two sides is lost by the other**. This implies that when the **Discriminator effectively**

distinguishes a test, it is **rewarded** or no upgrade is done to the model parameters, while the **Generator** is penalized with expansive changes to its model' parameters. From the other viewpoint, when **the Generator traps the Discriminator**, it gets rewarded, or no overhaul is done to its parameters, but the Discriminator is penalized, and its model parameter are modified.

In the end, the **combined loss function** we want to **minimize** looks like

$$L(D, G) = \mathbb{E}_X \left[\log D(X)\right] + \mathbb{E}_Z \left[\log(1 - D(G(Z))\right],$$

where:

- $D(X)$ is the discriminator's estimate of the probability that real data instance X is real.
- $\mathbb{E}_X[\cdot]$ is the expected value over all real data instances.
- $G(Z)$ is the generator's output when given noise Z.
- $D(G(Z))$ is the discriminator's estimate of the probability that a fake instance is real.
- $\mathbb{E}_Z[\cdot]$ is the expected value over all random inputs to the generator.

However, according to [5], we can note that the above minimax loss function can cause the GAN to get stuck in the early stages of GAN training when the discriminator's job is very easy. Therefore, [3] suggests us modifying the generator loss so that the generator tries to maximize $\log D(G(Z))$.

In our model for the Generator and Discriminator we consider a **neural network with 3 and 3 hidden layers respectively**.

3.3 Activation functions

In neural networks, it's usually common to use a **ReLU** function as activation function g in the output of each hidden layer. However, the output of a ReLU function can be fragile during training and can "die", that is, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any data point again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. To solve this, we chose to use a **Leaky ReLU function**, which is defined by:

$$\begin{aligned} g\colon \mathbb{R} &\longmapsto \mathbb{R} \\ x &\longmapsto g(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geqslant 0 \end{cases}. \end{aligned}$$

For the output of both the generator and the discriminator we chose to work with a **sigmoid** function as activation function, which is defined by:

$$\begin{aligned} g\colon \mathbb{R} &\longmapsto (0, 1) \\ x &\longmapsto g(x) = \frac{1}{1 + e^{-x}}. \end{aligned}$$

The **sigmoid activation function** allows us to introduce **non-linear** features to our network because of its non-linearity characteristic. That enables the network to **learn better representations** of non-linear relationship/data. Adding to this the fact that the sigmoid is **differentiable everywhere** is really helpful for gradient computation for example or the minimization that we will need to go through to get our network's weights through back-propagation. Since it scales down in the range of (0–1), this is why we can also use it to get probabilities.

3.4 Dropout

- An easy but useful and powerful form of **regularization** for neural networks and deep learning models is **dropout**. To prevent **over-fitting**, we built and set 3 dropout layers of ratio 0.5 for each of our Generator's hidden layers and 2 dropout layers for our Discriminator. **Dropout** is only needed during the **training** of the model and is not considered while evaluating the performance of our model.

- Dropout may be described as a strategy where **haphazardly chosen neurons are overlooked during training**. They are "dropped-out" arbitrarily. This suggests that **their commitment** to the sanctioning of downward neurons is **transitory ousted** on the forward pass and any weight updates are not associated to those neurons in the backward pass (back propagation/ calculus of gradients).

- As the neural network learns, **neuron weights settle into their context** inside the network. Weights of neurons are tuned for particular highlights giving a few **specialization**. Neighboring neurons end up depending on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This dependence on setting for a neuron amid preparing is alluded to **complex co-adaptations**. We can envision that in the event that neurons are haphazardly dropped out of the network during training, then **other neurons will have to step in** and handle the representation required to form forecasts for the lost neurons.

- This method is believed to end up **learning free independent inner representations of our data** in the network. The impact is that the **organization** of the network gets to be **less delicate** to the particular weights of neurons. This helps **harden the model's robustness** and in turn comes with better **ability to generalize** because of a more robust organization and is **less likely to over-fit** the training data.

3.5 Batch normalization

Training deep neural networks with tens of hidden layers and hundred of units is **challenging**. Because all layers are changed during an update, **the update procedure is constantly chasing after an unstable target**. With backward propagation, the network's parameters are updated layer-by-layer backward from the output to the input using an **estimation of the error assuming fixed weights** in the previous layers previous of the current updated layer.
It **slows down the training** by requiring lower **learning rates** and careful **parameter initialization** to adjust the weights of the network needing a more **expansive number of epochs**. **Batch normalization** is a general technique that can be used to normalize the inputs to a layer. It can be used with several deep neural networks, such as **CNN** (Convolutional Neural Networks), **Multilayer Perceptrons** (in our case) , **RNN** (Recurrent Neural Networks).
Batch standardization acts to **normalize the mean and variance of every unit to balance out learning**, however still allowing connections among units and the nonlinear insights of a solitary unit to change. It helps **coordinating the updates** of multiple layers in the model.

We did not use it for the synthesis of our financial data because batch normalization offers some **regularization** effect, it reduces the generalization error, sometimes no longer requiring the use of dropout and we first chose to try with dropout but batch normalization is another way to avoid over-fitting and tend to a more robust model.

3.6 Optimization of the model

Optimizers upgrade the model in reaction to the output of the loss function. In quintessence, they have control over the learning process of our neural network by finding the values of parameters such

that the loss function is most reduced. The **learning rate** is a key **hyper-parameter** that scales the gradient and **sets the speed** at which the model is upgraded.

3.6.1 Stochastic Gradient Descent (SGD)

There are a few downsides of the gradient descent algorithm. Taking a look at the computation of this method we notice that for each iteration of the algorithm : for example in our case we work with 1190 data points and 4 features. We compute the **sum of squared residuals** consisting in computing as many terms as there are data points, so 1190 terms in our case. Then we compute compute the derivative of this function with respect to each of the features, so in effect we will be doing $1190 * 4 = 4760$ **computations per iteration**.

It is common to take 1000 iterations, in effect we have $4760 * 1000 = 4760000$ **computations to complete the algorithm**. That is pretty much up above the number of computations we want to allow because of the need to scale for bigger data sets and hence gradient descent is slow on huge data.

This is where **SGD** comes in touch. Instead of computing the gradient over every data points, **we randomly pick one data point** from the whole data set at each iteration to reduce the computations enormously. Understanding that we need to strike a **balance between** the **speed of SGD** and **the scale of gradient descent** using the whole data set, it is common to sample a small number of data points instead of just one point at each step and that is called **"mini-batch" gradient descent**.

3.6.2 Adam Optimizer

The **Adam optimisation rule** is an extension to stochastic gradient descent. Adam may be used rather than the classical random gradient descent procedure to update network weights repetitive based on the training data. Unlike **SGD** which keeps a single learning rate for all weight updates, Adam's learning rate change during training. Adam works by storing both the **exponentially decaying** average of past squared gradients and exponentially decaying average of past gradients. There are a number of sorts of **optimizers** being utilized in GAN literature, but **Adam** is right now among the foremost well known choices. Several properties lead us to pick Adam as optimizer to choose our learning rate :

- It is easy to implement, **computationally economical** and with **little** memory needs.
- It is **well suited** for issues that need **massive amounts of data and parameters**.
- The **hyper-parameters** have good **intuitive interpretation** and generally need very little standardisation.
- It is **adapted for non-stationary objectives**.
- It is **adapted for issues with sparse gradients or very noisy gradients**.

3.7 Evaluation score

The performance of the generative model is based on two score metrics: a marginal and a dependence error:

- **Marginals - Anderson-Darling:** We treat here the statistical problem of testing the hypothesis that n independent, identically distributed random variables have an identified continuous distribution function $F(x)$.
  According to [1] and [2], the Anderson-Darling score is defined by computing a weighted square difference between the hypothetical cumulative distribution function (c.d.f.) F from which samples have been generated, and the empirical c.d.f $\hat{F}_n$ based on$n = 1190$ observations. It is defined as:

$$W_n = \int_{\mathbb{R}} \frac{(\hat{F}_n(x) - F(x))^2}{F(x)(1-F(x))} \, \mathrm{dF}(x).$$

Let be $\tilde{u}^{\tau}_{i,n}$ the model probability of a generated variable $\tilde{X} = \mathrm{G(Z)}$ for a specific ticker, defined as:

$$\tilde{u}^{\tau}_{i,n} = \frac{1}{n+2} \left(\sum_{j=1}^{N} \mathbb{1} \left\{ X^{\tau}_j \leqslant X^{\tau}_{i,n} \right\} + 1 \right)$$

Then, we can compute the Anderson-Darling distance for each ticker τ:

$$W^{\tau}_n = -n - \frac{1}{n} \sum_{i=1}^{n} (2i-1)(\log(\tilde{u}^{\tau}_{i,n}) + \log(1 - \tilde{u}^{\tau}_{n-i+1,n})).$$

However, we will just focus on $m = n - \lceil 0.9n \rceil$ data, which represents the 10% highest values on each marginal because we want to penalize more on the extremes. So, we rewrite the last equation as:

$$W^{\tau}_m = -m - \frac{1}{m} \sum_{i=\lceil 0.9n \rceil}^{n} (2i-1)(\log(\tilde{u}^{\tau}_{i,n}) + \log(1 - \tilde{u}^{\tau}_{n-i+1,n})).$$

Then, the total metric on the marginals is just the average distance for all tickers and is computed as:

$$\mathcal{L}_M = \frac{1}{d} \sum_{\tau=1}^{d} W^{\tau}_m.$$

- **Dependence - Absolute Kendall error:** According to [4], Kendall's dependence function characterizes the dependence structure associated with a copula C and is the univariate cumulative distribution function defined by $K_C(t) = \mathbb{P}(C(U^{(1)}, \ldots, U^{(d)}) \leqslant t)$ for all $t \in [0,1]$ and $(U^{(1)}, \ldots, U^{(d)})$ a random vector with uniforms margins on $[0,1]$.

  The estimation of the Kendall's dependence function is based on the pseudo-observations

$$Z_i = \frac{1}{n-1} \sum_{j \neq i}^{n} \mathbb{1} \left\{ X^1_j < X^1_i, \ldots, X^d_j < X^d_i \right\},$$

  and we consider equivalently the ones from the generative mode

$$\tilde{Z}_i = \frac{1}{n-1} \sum_{j \neq i}^{n} \mathbb{1} \left\{ \tilde{X}^1_j < \tilde{X}^1_i, \ldots, \tilde{X}^d_j < \tilde{X}^d_i \right\},$$

  Then, the dependence metric can be computed as a L^1 norm

$$\mathcal{L}_D = \frac{1}{n} \sum_{i=1}^{n} \left| Z_{i,n} - \tilde{Z}_{i,n} \right|,$$

  where $Z_{1,n} \leqslant \cdots, \leqslant Z_{n,n}$ (resp. $\tilde{Z}_{1,n} \leqslant \cdots, \leqslant \tilde{Z}_{n,n}$) are the order statistics associated with $\{Z_1, \ldots, Z_n\}$ (resp. $\{\tilde{Z}_1, \ldots, \tilde{Z}_n\}$).

4 Challenges of Generative Modelling

- **Mode collapse :**
  The purpose of generative modelling is not only to **create realistic looking samples**, but also being able to **produce wider varieties of samples**. Because of a deficiency happening during training, **mode collapse** is a failure mode of GANs. It happens when different noises are mapped to the same output region by the Generator or when the Generator is not aware of some regions of the target data distribution. It means that **if the Generator is stuck in a local minimum it will start to generate limited samples**, the Discriminator will then learn how to differentiate the Generator's fakes and **it will end the learning algorithm and lead to monolithic sets of outputs.**

- **Lack of proper evaluation metric for GANs :**
  Another issue about **GAN development** is how to evaluate its **training accuracy**. Since GAN has initially been developed around image generation, currently the most commonly used evaluation metric is **Fréchet Inception Distance**. It is based on Fréchet distance, which is a metric that **compares statistics from two multivariate normal distributions** using their **means and covariances matrix** helping quantify the distance between distributions. It uses features extracted by a network from the imageNet dataset. Therefore this metric is limited to image generation.

- **What is being done nowadays :**
  In order to solve the main training problems, **gradient vanishing** and **mode collapse**, research has been developed into two main methods: **proposing new network architectures** and using **different loss functions**. Research has proved that the performance of GAN is related to the **network's architecture** and how we choose our **batch size**. It shows that a **well-designed architecture has a critical impact** on the output quality. The redesign of the **loss function**, including **regularization and normalization**, improves **training stability**. There is currently **no single universal solution**.

5 Data set

- As data set we will use a 4-dimensional set of vectors. This data set has been chosen as every day returns of **4 universal stock lists** (called ticker), accessible freely on normal budgetary websites. The return for each index can be negative or positive, and there's no reason that the 4 returns have the same sign at the same date. Since we are centering on extraordinary misfortunes scenarios, we have picked as it were dates for which the 4 returns are at the same time negative, it gives as it were 1190 information at the conclusion. Then, we flip their signs to bargain with positive information, for comfort.

- The key challenges across the four dimensions are **keeping the knowledge quality/complexity, protecting correlations, implementing constraints and maintaining integrity of the data set**. Any answer ought to be judged by its effectiveness in meeting these challenges.

- This means that we should be able to learn **complex distribution shapes**, maintain the **distribution statistical properties** in our data set. But also **preserve the linear and non-linear correlations** between our 4 stock indices if existing. There is also this need to understand how to **respect structural constraints** of our data if needed with prior knowledge or structural properties.

6 Results

Statistical similarities between real and generated data

	0	1	2	3
count	1190.000000	1190.000000	1190.000000	1190.000000
mean	0.081165	0.070284	0.084870	0.078455
std	0.077503	0.078015	0.085595	0.070333
min	0.000000	0.000014	0.000184	0.000026
25%	0.029669	0.019633	0.030263	0.032025
50%	0.058567	0.049106	0.062383	0.061049
75%	0.110212	0.095132	0.109364	0.102008
max	1.000000	0.903066	0.641450	0.661151

	0	1	2	3
count	1190.000000	1190.000000	1190.000000	1190.000000
mean	0.092881	0.112195	0.046708	0.092788
std	0.121575	0.135240	0.078655	0.121472
min	0.000017	0.000006	0.000000	0.000013
25%	0.014387	0.018077	0.003633	0.012729
50%	0.047881	0.060558	0.014936	0.046220
75%	0.120141	0.157318	0.052460	0.124235
max	0.963340	1.000000	0.744391	0.972304

Figure 1: Statistical description of our real data set (left) and our generated data set (right)

In **Fig. 1** we may observe that the **4 stock indices** have similar **means and standard deviations** for the two data sets. Adding to this the fact that their **quantiles** at 25%, 50%, 75% have very similar values for each stock we may understand that we get a global convergence of our model towards similar solutions of our initial data set but still unique on their own. We get an **Anderson-Darling score** around 10 and an **Absolute Kendall Error** of about 0.05 for minimum using our GAN model. Through the epochs (we chose 300 **epochs** maximal) we clearly see a **convergence toward** 0 **for our Anderson Darling score** (cf notebook) which is a sign that our Discriminator and Generator are learning and changing their weights to fit our data.

In **Fig. 2** we observe the **boxplot** of our real and generated data for each stock index. We can see there are small changes in the behavior of **outliers** between the original and generated data. However, these changes are not very significant with respect to the results expected to be obtained through the proposed methodology.

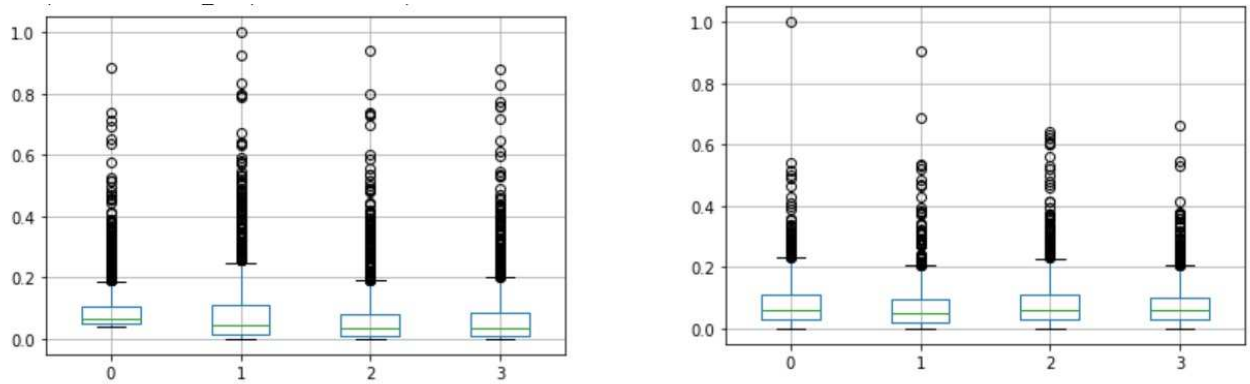

Figure 2: Boxplot description of our real data set (left) and our generated data set (right)

7 Conclusions

- As we have seen, our model GAN allows to generate realistic data which have a good fit of real data with respect to Anderson-Darling score if we considered a sufficient number of epochs. We also get a good performance of our model with the Absolute Kendall score.

However, this methodology can be a bit expensive computationally, which forces us to look for other methods to improve or/and solve this problem.

- In recent works, some alternative GAN models have been proposed in order to be used for different tasks or some real applications. For example, the models CycleGAN, DiscoGAN, StyleGAN and IsGAN had started showing promising results in generating realistic images. So, the architecture of GAN models has shown tremendous success and we will expect that there will be a lot of progress in this topic in the future.

References

[1] T.W Anderson and D.A. Darling. Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes. *The annals of mathematical statistics*, 23(2):193—-212, 1952.

[2] T.W Anderson and D.A. Darling. A test of goodness of fit. *Journal of the American Statistical Association*, 49(268):765—769, 1954.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN, 2017.

[4] C. Genest and L.-P. Rivest. Statistical inference procedures for bivariate archimedean copulas. *Journal of the American Statistical Association*, 88(423):1034—1043, 1993.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, 2014.