ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200192

Maximal Correlation: An Alternative Criterion for Training Generative Networks

Kalliopi Basioti[1], **George V. Moustakides**[1,2] and **Emmanouil Z. Psarakis**[3]

Abstract. Generative adversarial networks (GANs) require the design of a generator and a discriminator network which is achieved by solving min-max optimization problems. Min-max/adversarial optimizations are implemented with the help of two stochastic gradient algorithms, one for each optimization problems. This data-driven approach is known to suffer from non-robustness and need for excessive computations and processing time. In this work we propose a kernel based correlation criterion which we only maximize. Under ideal conditions this non-adversarial approach is shown to achieve the same goal as the existing adversarial methods. Under a pure data-driven scenario we only need a generator network which we train with a single gradient algorithm. Since the proposed criterion is a nonlinear combination of three expectations of functions, as opposed to the standard case of a single expectation of a function, deriving a gradient algorithm that optimizes it, is not straightforward. The solution we developed for a general optimization problem involving nonlinear functions of expectations, clearly constitutes an additional interesting result.

1 BACKGROUND

Since their first appearance [2, 14], GANs have gained considerable attention and popularity, mainly due to their remarkable capability to produce, after proper training, synthetic data (usually images) that are realistically close to the data contained in their training set. The main challenge in designing GANs comes from the fact that their training algorithms require heavy computations that are primarily implementable on computationally powerful platforms. Such high computational needs arise not only because the size of the problems is usually large but also because the design of GANs requires the solution of min-max optimization problems. Stochastic gradient type algorithms employed for such cases very often exhibit non-robust behavior and slow rate of convergence, thus raising the computational needs considerably [10, 22].

In this work we focus, primarily, on the computational aspects of the training phase. Our intention is to develop a training algorithm which is simple and requires significantly less computations as compared to the current methods proposed in the literature and employed in practice. In particular in [3] it is theoretically demonstrated that certain ideas as batch processing [21] and/or gradient smoothing [19] that are used for the solution of min-max problems have, in fact, absolutely no effect in the algorithmic performance and, therefore, can be ignored. These conclusions will help us shape our algorithmic scheme and suggest a simple and efficient form.

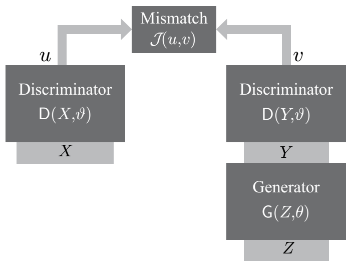

Figure 1. Representation of GAN architecture.

Figure 1 captures the architecture employed during the training phase of GANs. There is a random vector X with *unknown* probability density function (pdf) $\mathsf{f}(X)$, with X playing the role of a "prototype" random vector. The goal is to design a data-synthesis mechanism that generates realizations for the random vector X. For this goal we employ a nonlinear transformation $\mathsf{G}(Z,\theta)$, known as the *Generator*, that transforms a random vector Z of *known* pdf (e.g. Gaussian or Uniform) into a random vector Y.

We would like to design the parameters θ of the transformation so that Y is distributed according to $\mathsf{f}(Y)$. Under general assumptions such a transformation always exists [1, 8] and it can be efficiently approximated [11] by a sufficiently large neural network, with θ summarizing the network parameters.

Adversarial approaches in order to make the proper selection of θ employ a *second nonlinear transformation* $\mathsf{D}(X,\vartheta)$ that transforms X and Y into suitable *scalar* statistics $u = \mathsf{D}(X,\vartheta)$ and $v = \mathsf{D}(Y,\vartheta)$ and then compute a "mismatch" measure (not necessarily a distance) $\mathcal{J}(u,v)$ between the two random scalar quantities u, v. The second transformation $\mathsf{D}(X,\vartheta)$ is also implemented with the help of a neural network, known as the *Discriminator*. We are interested in the average mismatch between u, v namely $\mathsf{E}_{u,v}[\mathcal{J}(u,v)]$ which, after substitution, can be written as

$$J(\theta,\vartheta) = \mathsf{E}_{X,Y}\left[\mathcal{J}\big(\mathsf{D}(X,\vartheta),\mathsf{D}(Y,\vartheta)\big)\right] = \mathsf{E}_{X,Z}\left[\mathcal{J}\Big(\mathsf{D}(X,\vartheta),\mathsf{D}\big(\mathsf{G}(Z,\theta),\vartheta\big)\Big)\right]. \quad (1)$$

For every selection of the generator parameters θ we would like to select the discriminator parameters ϑ so that the average mismatch between u, v is maximized. In other words we design the discriminator to differentiate between the synthetic random vector Y and the prototype random vector X, as much as possible. This worst-case performance we then attempt to minimize by selecting properly the generator parameters θ. This leads to the following min-max opti-

[1] Computer Science, Rutgers University, New Brunswick, NJ, USA, email: kib21@scarletmail.rutgers.edu, gm463@rutgers.edu.
[2] Electrical and Computer Engineering, University of Patras, Patras, Greece, email: moustaki@upatras.gr.
[3] Computer Engineering and Informatics, University of Patras, Patras, Greece, email: psarakis@ceid.upatras.gr.

mization problem

$$\inf_{\theta}\sup_{\vartheta} J(\theta,\vartheta) = \inf_{\theta}\sup_{\vartheta} \mathsf{E}_{X,Z}\left[\mathcal{J}\Big(\mathsf{D}(X,\vartheta),\mathsf{D}\big(\mathsf{G}(Z,\theta),\vartheta\big)\Big)\right]. \quad (2)$$

Typical selections for the mismatch function $\mathcal{J}(u,v)$ are:

- $\mathcal{J}(u,v)=\log(u)+\log(1-v)$, $u,v\in[0,1]$, see [14].
- $\mathcal{J}(u,v)=\log(u)-\log(v)$, see [2].
- $\mathcal{J}(u,v)=u-v$, see [2].

It is clear that the generator generates realizations of the random vector Y by transforming the realizations of Z. But how can we be assured that these realizations have the correct pdf namely $\mathsf{f}(Y)$? To see that this is indeed the case we need to consider the generator and discriminator transformations as being general functions $\mathsf{G}(Z),\mathsf{D}(X)$ not limited to the specific classes induced by the two neural networks. This immediately implies that by properly selecting $\mathsf{G}(Z)$ we can shape the pdf $\mathsf{g}(Y)$ of Y into any pdf we desire [1]. In this idealized setup, optimizing over θ amounts to optimizing over $\mathsf{G}(Z)$ and therefore over $\mathsf{g}(Y)$ and, similarly, optimizing over ϑ amounts to optimization over $\mathsf{D}(X)$. Consequently, we can redefine the min-max optimization problem in (2) under the following idealized frame

$$\inf_{\mathsf{g}}\sup_{\mathsf{D}} J(\mathsf{g},\mathsf{D}) = \inf_{\mathsf{g}}\sup_{\mathsf{D}} \iint \mathcal{J}\big(\mathsf{D}(X),\mathsf{D}(Y)\big)\mathsf{f}(X)\mathsf{g}(Y)dXdY, \quad (3)$$

where $\mathsf{D}(X)$ is any scalar valued nonlinear transformation, and $\mathsf{g}(Y)$ is any pdf. The min-max problems corresponding to the three examples of $\mathcal{J}(u,v)$ we mentioned before accept analytic solutions. In particular in the first case, for fixed $\mathsf{f}(X),\mathsf{g}(Y)$ maximization over $\mathsf{D}(X)$ is achieved for $\mathsf{D}(X)=\frac{\mathsf{f}(X)}{\mathsf{f}(X)+\mathsf{g}(X)}$. The resulting functional is further minimized over $\mathsf{g}(Y)$ for $\mathsf{g}(Y)=\mathsf{f}(Y)$. In the second case, assuming that $|\mathsf{D}(X)|\le M$, maximization over $\mathsf{D}(X)$ is achieved when $\mathsf{D}(X)=e^{M\mathrm{sgn}(\mathsf{f}(X)-\mathsf{g}(X))}$ and minimization over $\mathsf{g}(Y)$ yields, again, $\mathsf{g}(Y)=\mathsf{f}(Y)$. Similarly for the third case, maximization over $\mathsf{D}(X)$ is achieved for $\mathsf{D}(X)=M\mathrm{sgn}(\mathsf{f}(X)-\mathsf{g}(X))$ and minimization over $\mathsf{g}(Y)$ when $\mathsf{g}(Y)=\mathsf{f}(Y)$. As we can see all three min-max problems result in different optimum discriminator functions but agree in the final solution for $\mathsf{g}(Y)$, namely Y is shaped to have the same pdf $\mathsf{f}(X)$ as the prototype random vector X.

Since in the original min-max problem (2) we limit the two transformations to be within the two classes induced by the input/output relationship of the corresponding neural network, it is clear that (2) constitutes an approximation to the ideal setup captured by (3). This implies that the output Y of the generator will follow a pdf $\mathsf{g}(Y)$ which will be an *approximation* to the desired pdf $\mathsf{f}(Y)$ of the prototype random vector X. This approximation not only depends on the richness of the transformation class induced by the generator structure but, also, on the corresponding richness of the discriminator structure. As long as one of the two structures does not approximate sufficiently close the corresponding ideal transformation (when for example the neural network does not have sufficient number of layers), *the design will fail* in the sense that the realizations of Y will not follow the desired pdf $\mathsf{f}(X)$ of the prototype X. The min-max optimization problem becomes challenging because, as we mentioned, the pdf of X is unknown and, instead, we are given a collection $\{X_1,\dots,X_N\}$ of independent realizations of X (the training set) drawn from $\mathsf{f}(X)$.

Remark 1 *Even though the goal is to design a generator network, with GANs we simultaneously require the design of an additional neural network, the discriminator. This requirement increases the number of parameters to be estimated considerably and, consequently, the computational complexity.*

Furthermore the algorithmic solution of (2) relies on alternating stochastic gradient-type algorithms and the presence of two antagonistic optimization problems translates into an increased number of updates in the implementation which are also known to be *non-robust* [4, 10, 22].

2 A NON-ADVERSARIAL APPROACH

Let us now see how we can accomplish a similar approximation for the output pdf of the generator *without the need of a discriminator*. Consider the vector space $\mathcal{V}$ of all scalar functions $\phi(U)$. We are interested in defining an *inner product* for $\mathcal{V}$. Assume that $\mathsf{K}(U,V)$ is a *positive definite kernel*, namely, a scalar function of U,V which is *symmetric* $\mathsf{K}(U,V)=\mathsf{K}(V,U)$ and satisfies

$$\iint \mathsf{K}(U,V)\phi(U)\phi(V)\,dU\,dV>0,$$

for every function $\phi(U)\in\mathcal{V}$ with $\phi(U)\neq 0$. Using a positive definite kernel, with the next lemma we present how we can define, in a straightforward way, an inner product.

Lemma 1 *For any two functions $\phi(U),\psi(U)\in\mathcal{V}$ define*

$$\prec\phi,\psi\succ=\iint \mathsf{K}(U,V)\phi(U)\psi(V)\,dU\,dV. \quad (4)$$

Then $\prec\phi,\psi\succ$ is an inner product in $\mathcal{V}$.

Proof We must show that (4) satisfies the required properties that define an inner product. Since the proof is simple we give no further details. We only mention that (4) is a direct extension of the corresponding inner product of finite dimensional vector spaces where $\phi(U)$ is replaced by a classical finite dimensional vector and $\mathsf{K}(U,V)$ by a symmetric and positive definite matrix. ■

In a vector space, equipped with an inner product (Hilbert space), we have validity of the famous *Cauchy-Schwarz inequality* [9], namely that

$$(\prec\phi,\psi\succ)^2\le\prec\phi,\phi\succ\prec\psi,\psi\succ, \quad (5)$$

furthermore, we have equality in (5), *if and only if*

$$\psi(U)=a\phi(U), \quad (6)$$

where a is a constant. Let us now define a proper optimization problem which, with the help of this well known result, will be shown to enjoy a desirable solution. As before, with $\mathsf{f}(U),\mathsf{g}(U)$ we denote probability densities. If we fix $\mathsf{f}(U)$ then we can consider the following cost as a function of $\mathsf{g}(U)$

$$\mathcal{J}(\mathsf{g})=\frac{\left(\iint \mathsf{K}(U,V)\mathsf{f}(U)\mathsf{g}(V)\,dU\,dV\right)^2}{\iint \mathsf{K}(U,V)\mathsf{g}(U)\mathsf{g}(V)\,dU\,dV}. \quad (7)$$

By recalling the definition of the inner product from (4) and using (5) we see that

$$\mathcal{J}(\mathsf{g})=\frac{(\prec\mathsf{f},\mathsf{g}\succ)^2}{\prec\mathsf{g},\mathsf{g}\succ}\le\prec\mathsf{f},\mathsf{f}\succ. \quad (8)$$

This suggests that if we *maximize* $\mathscr{J}(\mathsf{g})$ over $\mathsf{g}(U)$ then

$$\max_{\mathsf{g}} \mathscr{J}(\mathsf{g}) = \max_{\mathsf{g}} \frac{(\prec \mathsf{f}, \mathsf{g} \succ)^2}{\prec \mathsf{g}, \mathsf{g} \succ} \leq \prec \mathsf{f}, \mathsf{f} \succ = \iint \mathsf{K}(U,V)\mathsf{f}(U)\mathsf{f}(V)\, dU\, dV. \quad (9)$$

The upper bound is independent from $\mathsf{g}(U)$ consequently, if we can find a $\mathsf{g}(U)$ that attains it then this selection will necessarily be optimum. The Cauchy-Schwarz inequality is actually telling us what form this $\mathsf{g}(U)$ must have. Specifically, if we select $\mathsf{g}(U) = a\mathsf{f}(U)$ then we attain the upper bound. Furthermore, since both $\mathsf{f}(U), \mathsf{g}(U)$ are probability densities integrating to 1 this means that $a = 1$. Hence, we conclude that the optimum solution of the optimization problem in (9) has the desired form $\mathsf{g}(U) = \mathsf{f}(U)$.

Remark 2 *The optimum density obtained by solving the* ***single maximization*** *problem is exactly the same as the optimum density obtained by the adversarial approaches.*

Let us now write our criterion using expectations because this will guide us in producing a stochastic gradient algorithm. If X is a random vector distributed according to $\mathsf{f}(X)$ and Y^1, Y^2 are two identically distributed but *statistically independent* random vectors following $\mathsf{g}(Y)$, then we can write for the cost function

$$\mathscr{J}(\mathsf{g}) = \frac{\mathsf{E}_{X,Y^1}[\mathsf{K}(X,Y^1)] \cdot \mathsf{E}_{X,Y^2}[\mathsf{K}(X,Y^2)]}{\mathsf{E}_{Y^1,Y^2}[\mathsf{K}(Y^1,Y^2)]}. \quad (10)$$

If we also let X^1, X^2 be two identically distributed independent random vectors following $\mathsf{f}(X)$ then, from the Cauchy-Schwarz inequality we have that the ratio

$$\mathsf{r}(\mathsf{g}) = \frac{\mathscr{J}(\mathsf{g})}{\mathsf{E}_{X^1,X^2}[\mathsf{K}(X^1,X^2)]} = \frac{(\prec \mathsf{f}, \mathsf{g} \succ)^2}{\prec \mathsf{g}, \mathsf{g} \succ \cdot \prec \mathsf{f}, \mathsf{f} \succ} \leq 1.$$

The normalized cost $\mathsf{r}(\mathsf{g})$ can play the role of a nonlinear *correlation factor* for the random vectors X, Y. This correlation, when optimized over the density $\mathsf{g}(Y)$, attains the maximal value 1. As we argued, this can happen if and only if the two random vectors X, Y enjoy the same statistical behavior, namely $\mathsf{g}(X) = \mathsf{f}(X)$. Since $\mathsf{E}_{X^1,X^2}[\mathsf{K}(X^1,X^2)]$ does not depend on $\mathsf{g}(Y)$, maximizing the correlation factor $\mathsf{r}(\mathsf{g})$ is equivalent to maximizing $\mathscr{J}(\mathsf{g})$, which is the method we propose in this work.

We must point out that kernel based, non-adversarial criteria were also adopted in the past [3, 12, 15]. However, the maximal correlation idea introduced here is presented for the first time. Given the experience of our group with these kernel approaches which is reported in [3], we would like to point out that the generator designs we obtain with the new criterion, produce higher quality synthetic images than its counterparts in [3, 12, 15]. And this is achieved with comparable computational complexity.

3 NEURAL NETWORKS AND TRAINING

The next step consists in abandoning the ideal world expressed by (10). If Y is the output $Y = \mathsf{G}(Z, \theta)$ of the generator, this suggests that the Y^1, Y^2 needed in (10) correspond to inputs Z^1, Z^2. The two random input vectors must be statistically independent in order for the same property to be inherited by the two outputs Y^1, Y^2. From (10), by substituting $Y^i = \mathsf{G}(Z^i, \theta),\ i = 1, 2$, we obtain

$$\mathscr{J}(\mathsf{g}) \approx \mathscr{J}(\theta) = \frac{\mathsf{E}_{X,Z^1}\big[\mathsf{K}\big(X, \mathsf{G}(Z^1,\theta)\big)\big] \cdot \mathsf{E}_{X,Z^2}\big[\mathsf{K}\big(X, \mathsf{G}(Z^2,\theta)\big)\big]}{\mathsf{E}_{Z^1,Z^2}\big[\mathsf{K}\big(\mathsf{G}(Z^1,\theta), \mathsf{G}(Z^2,\theta)\big)\big]}. \quad (11)$$

Furthermore, the optimization over $\mathsf{g}(Y)$ is replaced by the optimization over the network parameters θ, namely

$$\max_{\mathsf{g}} \mathscr{J}(\mathsf{g}) \approx \max_{\theta} \mathscr{J}(\theta) = \max_{\theta} \frac{\mathsf{E}_{X,Z^1}\big[\mathsf{K}\big(X, \mathsf{G}(Z^1,\theta)\big)\big] \cdot \mathsf{E}_{X,Z^2}\big[\mathsf{K}\big(X, \mathsf{G}(Z^2,\theta)\big)\big]}{\mathsf{E}_{Z^1,Z^2}\big[\mathsf{K}\big(\mathsf{G}(Z^1,\theta), \mathsf{G}(Z^2,\theta)\big)\big]}, \quad (12)$$

which constitutes our target optimization problem.

3.1 Stochastic updating algorithms for optimization and equation solving

There are two classical problems for which stochastic updating algorithms can be derived in a straightforward manner. We borrow the corresponding results from [5].

Standard optimization:
For a cost function $\mathsf{J}(\vartheta)$ which is of the form

$$\mathsf{J}(\vartheta) = \mathsf{E}_X[\mathsf{p}(X, \vartheta)], \quad (13)$$

where $\mathsf{p}(X, \vartheta)$ is a scalar function of a random vector X and a parameter vector ϑ, we are interested in the optimization problem $\max_\vartheta \mathsf{J}(\vartheta)$. If we do not know the probability density of X and instead we have a data sequence $\{X_t\}$ then we can maximize $\mathsf{J}(\vartheta)$ using the stochastic gradient algorithm which updates the parameter estimates as follows

$$\vartheta_t = \vartheta_{t-1} + \mu \nabla_\vartheta \mathsf{p}(X_t, \vartheta_{t-1}). \quad (14)$$

Scalar $\mu > 0$ is the learning rate and ∇_ϑ denotes gradient with respect to ϑ. If the data sequence is finite then we recycle the data until we reach convergence. If $\{\vartheta_t\}$ converges in the mean, then the mean limit is a (local) maximum of $\mathscr{J}(\vartheta)$.

System of nonlinear equations:
The second problem involves a *vector* function $\mathsf{J}(\vartheta)$ where the length of $\mathsf{J}(\vartheta)$ is the same as the length of ϑ (number of equations must be the same as number of unknowns). If the vector function is of the form

$$\mathsf{J}(\vartheta) = \mathsf{E}_X[\mathsf{H}(X, \vartheta)], \quad (15)$$

where $\mathsf{H}(X, \vartheta)$ is also a vector function of a random vector X and a parameter vector ϑ, we are interested in the solution of the equation $\mathsf{J}(\vartheta) = 0$. From [5] we have that, if the stochastic update algorithm (it is not necessarily a stochastic gradient)

$$\vartheta_t = \vartheta_{t-1} + \mu\, \mathsf{H}(X_t, \vartheta_{t-1}), \quad (16)$$

converges in the mean, then the corresponding mean limit is a solution of the desired system of equation.

The optimization problem we like to solve in (12), clearly, does not conform with the standard version depicted in (13) consequently, proposing a stochastic gradient algorithm is not immediate. For this reason we would like to generalize the method in (14) to cover optimization problems that have a more complicated form.

Non-standard optimization:
Let us now consider an optimization problem which refers to a cost function that is more general than the classical version in (13). We are interested in costs involving *multiple expectations* that are combined through a nonlinear function.

The following theorem introduces explicitly the problem of interest and provides all the necessary details regarding its stochastic gradient solution. Basically, we combine the existing results in order to be able to obtain the intended extension.

Theorem 1 *Let $\mathsf{a}(Z)$ be a scalar function of a vector Z and $\mathsf{P}(X,\vartheta)$ be a vector function of a random vector X and a parameter vector ϑ, that has the same size as Z. Define the cost*

$$\mathsf{J}(\vartheta) = \mathsf{a}\big(\mathsf{E}_X[\mathsf{P}(X,\vartheta)]\big), \tag{17}$$

where Z is replaced by the expectation of the vector function $\mathsf{P}(X,\vartheta)$. Then, the stochastic gradient algorithm for maximizing $\mathsf{J}(\vartheta)$ takes the following form

$$\begin{aligned} \vartheta_t &= \vartheta_{t-1} + \mu\big(\mathfrak{J}_\vartheta \mathsf{P}(X_t,\vartheta_{t-1})\big)^\intercal\big(\nabla_Z \mathsf{a}(\mathcal{Z}_{t-1})\big)\\ \mathcal{Z}_t &= \mathcal{Z}_{t-1} + \mu\{\mathsf{P}(X_t,\vartheta_{t-1}) - \mathcal{Z}_{t-1}\}, \end{aligned} \tag{18}$$

where $\mathfrak{J}_\vartheta \mathsf{P}(X,\vartheta)$ denotes the Jacobian matrix of $\mathsf{P}(X,\vartheta)$ with respect to ϑ and $\nabla_Z \mathsf{a}(Z)$ the gradient of $\mathsf{a}(Z)$ with respect to Z.

Proof The proof is interesting and surpisingly simple. It relies on the definition of the following auxiliary parameter vector

$$\mathcal{Z} = \mathsf{E}_X[\mathsf{P}(X,\vartheta)]. \tag{19}$$

We know that the value of ϑ where $\mathsf{J}(\vartheta)$ is maximized, is a root of the system of equations $\nabla_\vartheta \mathsf{J}(\vartheta) = 0$. Using the definition of the cost function in (17) and the definition of the auxiliary parameter vector in (19) this equation can be written as

$$\begin{aligned} \mathsf{E}_X\big[\big(\mathfrak{J}_\vartheta \mathsf{P}(X,\vartheta)\big)^\intercal\big(\nabla_Z \mathsf{a}(\mathcal{Z})\big)\big] &= 0\\ \mathsf{E}_X[\mathsf{P}(X,\vartheta) - \mathcal{Z}] &= 0. \end{aligned} \tag{20}$$

We note that we have enlarged the system of equations by including, in the last equation, the definition of the auxiliary parameter vector. Applying (15) with ϑ replaced by $\{\vartheta, \mathcal{Z}\}$, the extended system can be solved using the updates in (18). This completes the proof. ■

Theorem 1 is very general and can accommodate any nonlinear function $\mathsf{a}(Z)$ of any number of expectations of nonlinear functions of X and ϑ. The stochastic gradient algorithm in (18) can also be extended to include batches of size K as follows

$$\begin{aligned} \vartheta_t &= \vartheta_{t-1} + \mu\big(\textstyle\sum_{j=1}^K \mathfrak{J}_\vartheta \mathsf{P}(X_{t+j},\vartheta_{t-1})\big)^\intercal\big(\nabla_Z \mathsf{a}(\mathcal{Z}_{t-1})\big)\\ \mathcal{Z}_t &= \mathcal{Z}_{t-1} + \mu\textstyle\sum_{j=1}^K\{\mathsf{P}(X_{t+j},\vartheta_{t-1}) - \mathcal{Z}_{t-1}\}, \end{aligned} \tag{21}$$

where in each iteration t we use K data samples $\{X_{Kt+1},\ldots, X_{K(t+1)}\}$ that must be *non-overlapping* with other data batches within the same data epoch (as we recycle the data repeatedly until convergence is reached).

Remark 3 *We must emphasize that the two versions of the algorithm in* (18) *and* (21)*, contrary to the widespread belief* [12, 21]*, **have exactly the same convergence characteristics** when the batch size K is not large (micro-batch)* [3].

Even though the batch version in (21) does not exhibit any significant analytical or convergence advantage over the simple implementation in (18), there is still a very serious reason to prefer the former over the latter. If both algorithms are implemented on some parallel computing platform then, as we can see, (21) allows for the simultaneous computation of *multiple* gradients of different data points in the batch. This can result in considerable reduction of the execution time.

Unfortunately, batches cannot be selected to be overly large, something that would incur substantial execution time gains, because then we start experiencing certain negative effects in the corresponding algorithms. The most notable such problem consists in the algorithm losing its capability to escape from local extrema, a property which has been observed to be firmly present when micro-batches are adopted [18]. This means that the algorithm with large batches tends to have inferior performance compared to the case of using micro-batches.

3.2 Algorithmic implementation

We can directly apply Theorem 1 to find the solution of (12). To make the corresponding connection, we note that we need to define $Z^\intercal = [z_1, z_2, z_3]$ and $\mathsf{a}(Z) = \frac{z_1 \cdot z_2}{z_3}$. The role of the random vector X in the theorem now plays the triplet $\{X, Z^1, Z^2\}$ and ϑ must be replaced by θ. Finally, the vector function $\mathsf{P}(X,\vartheta)$ is the collection of the three scalar functions $\mathsf{K}\big(X, \mathsf{G}(Z^1,\theta)\big)$, $\mathsf{K}\big(X, \mathsf{G}(Z^2,\theta)\big)$ and $\mathsf{K}\big(\mathsf{G}(Z^1,\theta), \mathsf{G}(Z^2,\theta)\big)$.

In the algorithm that follows, at each iteration t, we use one of the available data vectors X_t and we generate two new vectors Z_t^1, Z_t^2. This, according to (18), leads to the following updating formulas

$$\begin{aligned} &Y_t^1 = \mathsf{G}(Z_t^1,\vartheta_{t-1}),\ Y_t^2 = \mathsf{G}(Z_t^2,\vartheta_{t-1})\\ &\theta_t = \theta_{t-1} +\\ &\qquad \mu\Big\{\gamma_{t-1}\beta_{t-1}\big(\mathfrak{J}_\theta \mathsf{G}(Z_t^1,\theta_{t-1})\big)^\intercal \nabla_Y \mathsf{K}(X_t, Y_t^1) +\\ &\qquad \gamma_{t-1}\alpha_{t-1}\big(\mathfrak{J}_\theta \mathsf{G}(Z_t^2,\theta_{t-1})\big)^\intercal \nabla_Y \mathsf{K}(X_t, Y_t^2) -\\ &\qquad \alpha_{t-1}\beta_{t-1}\big(\mathfrak{J}_\theta \mathsf{G}(Z_t^1,\theta_{t-1})\big)^\intercal \nabla_X \mathsf{K}(Y_t^1, Y_t^2) +\\ &\qquad \alpha_{t-1}\beta_{t-1}\big(\mathfrak{J}_\theta \mathsf{G}(Z_t^2,\theta_{t-1})\big)^\intercal \nabla_Y \mathsf{K}(Y_t^1, Y_t^2)\Big\}\\ &\alpha_t = \alpha_{t-1} + \mu\big\{\mathsf{K}(X_t, Y_t^1) - \alpha_{t-1}\big\}\\ &\beta_t = \beta_{t-1} + \mu\big\{\mathsf{K}(X_t, Y_t^2) - \beta_{t-1}\big\}\\ &\gamma_t = \gamma_{t-1} + \mu\big\{\mathsf{K}(Y_t^1, Y_t^2) - \gamma_{t-1}\big\}. \end{aligned} \tag{22}$$

The auxiliary parameter vector $\mathcal{Z}$ of Theorem 1 takes the form $\mathcal{Z}^\intercal = [\alpha, \beta, \gamma]$ where each of the three parameters captures one of the corresponding three expectations. We recall that $\mathsf{K}(X,Y)$ is the adopted kernel and $\nabla_X \mathsf{K}(X,Y)$, $\nabla_Y \mathsf{K}(X,Y)$ express its gradient with respect to X and Y respectively, while $\mathfrak{J}_\theta \mathsf{G}(Z,\theta)$ denotes the Jacobian matrix of $\mathsf{G}(Z,\theta)$ with respect to θ.

Regarding the updates in (22) we can have a slight simplification by observing that α_t and β_t are, on average, equal. This suggests that we could set $\beta_t = \alpha_t$ and compute α_t with the following symmetric, with respect to Y_t^1, Y_t^2, form

$$\alpha_t = \alpha_{t-1} + \mu\big\{0.5\big[\mathsf{K}(X_t, Y_t^1) + \mathsf{K}(X_t, Y_t^2)\big] - \alpha_{t-1}\big\}.$$

Finally, we must mention that an equivalent to (21) batch version is also straightforward to derive.

Interestingly, because of the Cauchy-Schwarz inequality, it is possible to *monitor whether our algorithm converges properly or not*. We note that $\frac{\alpha_t \cdot \beta_t}{\gamma_t}$ expresses the value of the corresponding cost function in (11). If the algorithm behaves correctly then this value must reach the expectation $\delta = \mathsf{E}_{X^1,X^2}[\mathsf{K}(X^1, X^2)]$. The latter can either be pre-computed from the available data vectors or estimated in parallel with the network parameters θ. For the parallel computation we observe that X_t, X_{t-1} are independent therefore they can play the role of X^1, X^2. This results in the following possible update

$$\delta_t = \delta_{t-1} + \mu\big\{\mathsf{K}(X_t, X_{t-1}) - \delta_{t-1}\big\}.$$

When we reach convergence we expect $\frac{\alpha_t \cdot \beta_t}{\gamma_t}$ to be close and randomly oscillate around δ_t. If this is not the case then this is a strong indication that the algorithm did not converge properly.

Additional simplification, giving rise to significant execution speed up, can be enjoyed if in each iteration in (22), instead of generating two random inputs Z_t^1, Z_t^2 we generate only one Z_t. For the second input vector we can simply use the one generated during the previous iteration. As it is argued in [3], imposing finite delays of computed quantities does not affect the overall performance of the training algorithm. Consequently, we can have $Z_t^1 = Z_t$ and $Z_t^2 = Z_{t-1}$. This particular selection must be accompanied by similar substitutions wherever Z_t^2 is involved. For example, for the output we compute $Y_t^1 = Y_t = \mathsf{G}(Z_t, \theta_{t-1})$ but instead of $Y_t^2 = \mathsf{G}(Z_{t-1}, \theta_{t-1})$ we can use $Y_t^2 = \mathsf{G}(Z_{t-1}, \theta_{t-2}) = Y_{t-1}$ which exists from the previous iteration. Also $\mathsf{K}(X_t, Y_t^2)$ and its gradient, can be replaced by $\mathsf{K}(X_{t-1}, Y_{t-1})$ and its corresponding gradient which are available from the previous iteration. Only $\mathsf{K}(Y_t^1, Y_t^2)$ and its gradient must be computed as $\mathsf{K}(Y_t, Y_{t-1})$ while $\mathsf{K}(Y_t^2, Y_t^1)$ and its gradient can be replaced by the already available $\mathsf{K}(Y_{t-1}, Y_{t-2})$ and its corresponding gradient.

As it is proved in [3], these approximations make sense because θ_t changes extremely slowly between iterations and because expressions as $\mathsf{K}(X_t, Y_t)$ and $\mathsf{K}(X_{t-1}, Y_{t-1})$ have, *on average*, the same value. We recall that, it is the average quantities that enter in the definition of our criterion in (11) and also, similarly to (20), in the resulting system of equations that we obtain when we equate the gradient of our criterion to 0.

A very important practical issue concerning the optimization of our criterion is clearly its unconventional form. Existing computational frameworks as TensorFlow, PyTorch, etc., which automatically derive the necessary gradient algorithm, can accommodate costs that are either purely deterministic (not involving any expectation) or under the standard form depicted in (13). Consequently, in our case they will be unable to produce the proper stochastic gradient algorithm. This means that if we like to apply the stochastic gradient algorithm in (22) we need to analytically compute the expressions for the corresponding gradients.

If, in any event, one is insisting in using the available frameworks, it is still possible by replacing expectations with averages over *large batches* namely, replace $\mathcal{J}(\theta)$ with the approximation

$$\hat{\mathcal{J}}(\theta) = \frac{\sum_{j=1}^{n} \mathsf{K}\big(X_j, \mathsf{G}(Z_j^1, \theta)\big) \cdot \sum_{j=1}^{n} \mathsf{K}\big(X_j, \mathsf{G}(Z_j^2, \theta)\big)}{n \sum_{j=1}^{n} \mathsf{K}\big(\mathsf{G}(Z_j^1, \theta), \mathsf{G}(Z_j^2, \theta)\big)}. \quad (23)$$

Cost $\hat{\mathcal{J}}(\theta)$ will be treated as deterministic by the computational frameworks thus allowing for the automatic generation of the gradients. However, the corresponding algorithm, it is possible that it may "suffer" from the negative effects we mentioned about large batches. On the other hand, it is clear that this is a reasonable starting point to test the method and the quality of the produced synthetic images, before attempting to make any analytical efforts for the gradients.

4 EXPERIMENTS

We first apply our algorithm to the MNIST database. MNIST has 70,000 training images (we combined training and testing data) of size 28×28 which are transformed into vectors of length 784. Pixels are normalized to take values in $[0, 1]$. For the generator, which is the only neural network required by our approach, we use a fully connected two-layer configuration with dimensions $10 \times 128 \times 784$ (10 inputs, 128 hidden layer size and 784 output size). In both cases the activation function for the inner layer is the ReLU. For the 10 inputs we use independent standard Gaussian random variables. For the kernel function we select the Gaussian kernel $\mathsf{K}(X, Y) = e^{-\frac{\|X-Y\|^2}{h}}$ and for kernel parameter we select $h = 20$. We also select the same configuration for the generator in the GAN implementation while for the corresponding GAN discriminator we adopt again a two-layer network of dimensions $784 \times 128 \times 1$.

The stochastic gradient algorithms are implemented for batch size equal to 600. Every time the data are exhausted we recycle them. We use a learning rate equal to $\mu = 10^{-3}$ and apply the gradient normalization scheme proposed in [24] with smoothing factor $\lambda = 0.999$, while we initialize our network parameters following [13]. Our simplified algorithm is implemented in Matlab whereas the GAN version in TensorFlow.

Figure 2 refers to the cost of our algorithm $\frac{\alpha_t \cdot \beta_t}{\gamma_t}$ and its upper bound δ_t provided by the Cauchy-Schwarz inequality. We can see how the two quantities evolve with the number of iterations. The total number of iterations applied is $N = \frac{7}{6} \times 10^6$. We observe that the cost function of the running network (blue) approximates the upper bound (red), which indicates that the algorithm behaves correctly and did not converge to an unwanted local maximum.

We have also generated a series of synthetic images correspond-

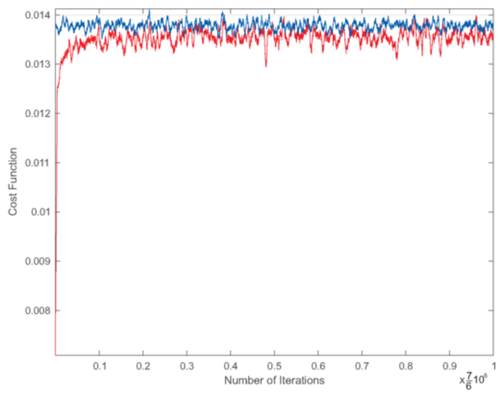

Figure 2. Evolution of proposed cost function (red) and corresponding upper bound (blue) with number of iterations for the MNIST database.

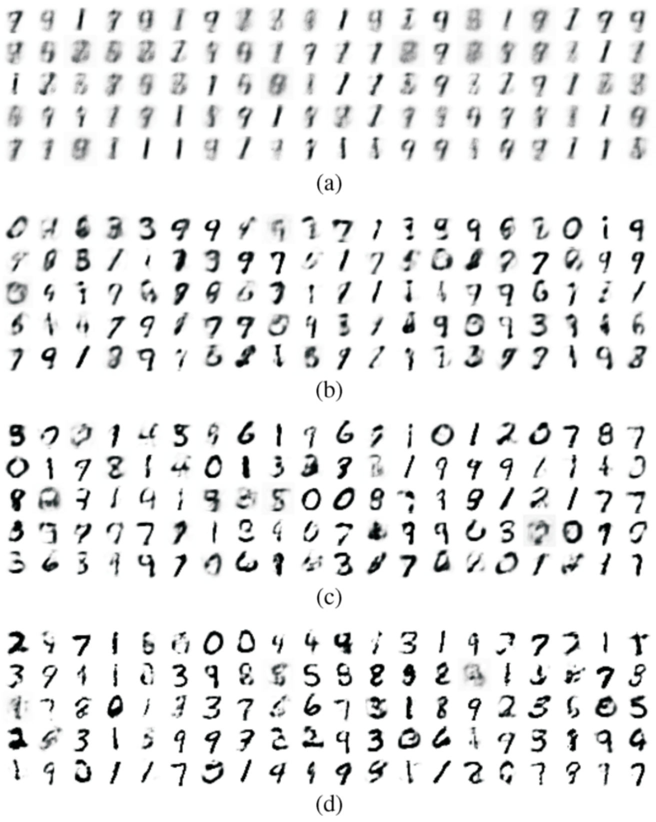

Figure 3. Synthetic numerals with proposed method after training with MNIST with (a) $\frac{7}{6}10^3$, (b) $\frac{7}{6}10^4$, (c) $\frac{7}{6}10^5$ and (d) $\frac{7}{6}10^6$ number of iterations.

ing to different points during the training with our method, in order to observe the progress of the quality of the synthetic images. In Figure 3, (a) corresponds to $\frac{7}{6}10^3$, (b) to $\frac{7}{6}10^4$, (c) to $\frac{7}{6}10^5$ and (d) to $\frac{7}{6}10^6$ iterations. It is remarkable how the network starts with a poor variety of synthetic designs and gradually, as it learns the numerals, it becomes more and more capable of producing realistically looking handwritten numbers by simply transforming, with the designed network, the randomly generated inputs.

Let us now compare our method to the GAN implementation of [2]. This comparison is, unfortunately, not very straightforward. The reason is that, in the min-max problem, with every iteration of the generator we apply five iterations for the discriminator. In each of these five iterations we need a *different batch of training data*. In other words, the GAN implementation consumes the training data *five times faster* than our approach. Despite this fact we compare the evolution of performance indexes for the two algorithms as a function of the number of iterations used in the *minimization problem* for GAN and, in our case, in the maximization problem.

A known performance index, the Inception Score [7, 16], is depicted in Figure 4 as it evolves with the number of iterations for our method (red) and GAN (blue). As we can see, GAN appears to attain better scores.

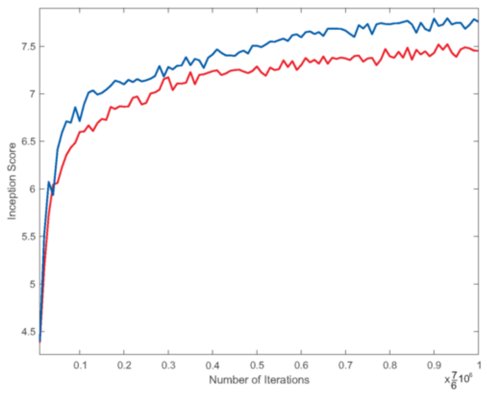

Figure 4. Evolution of inception score with number of iterations: Proposed (red), GAN (blue) for the MNIST database.

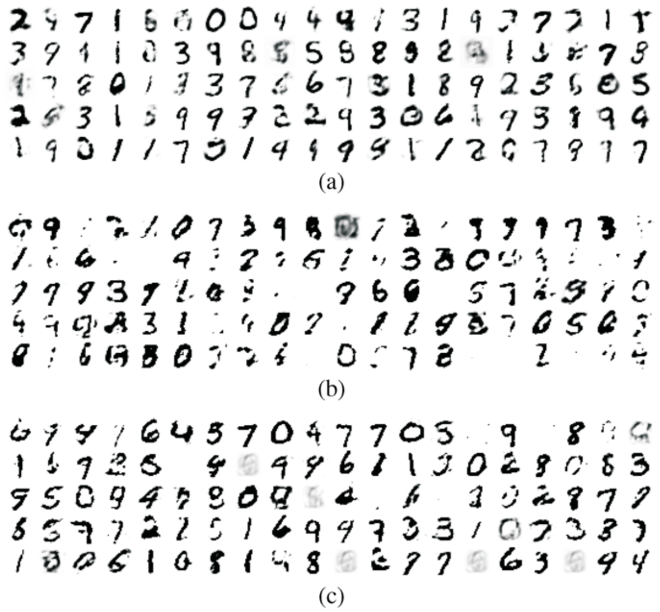

Figure 5. Results after training with the MNIST database. (a) Proposed after $\frac{7}{6}10^6$ iterations, (b) GAN after $\frac{1}{5}\frac{7}{6}10^6$ iterations and (c) GAN after $\frac{7}{6}10^6$ iterations.

Higher inception score must translate into better quality synthetic images. However as we can observe in Figure 5 this is clearly not the case. In Figure 5(a) we repeated for convenience Figure 3(d) produced by our method. Figure 5(b) depicts synthetic numerals produced by GAN for a number of iterations equal to $\frac{1}{5}\frac{7}{6}10^6$, namely five times less than the iterations of our method, but corresponding to the same number of consumed data batches. Finally Figure 5(c) is again synthetic numerals from GAN after $\frac{7}{6}10^6$ iterations, exactly as in our method. We observe that our neural network produces better results compared to both GAN alternatives. In particular, in GAN we observe the phenomenon of missed numerals which is not present in our method. It is, therefore, puzzling why this negative performance is not captured by the inception score and GAN appears to be superior. This is the reason why we focused on an alternative performance index which has also attracted attention recently.

The Kernel Inspection Distance (KID) [6] uses the kernel based Maximum Mean Discrepancy (MMD). If we adopt the KID to measure quality, then its evolution with the number of iterations for the

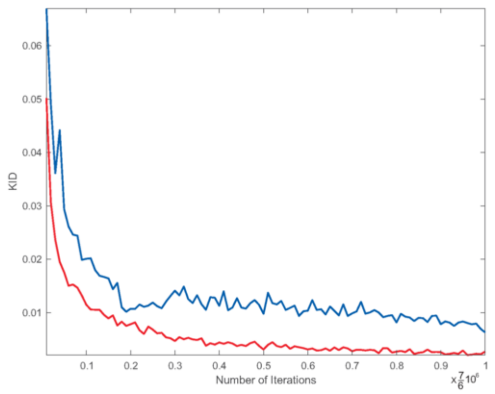

Figure 6. Evolution of the Kernel Inspection Distance (KID) with number of iterations: Proposed (red), GAN (blue) for the MNIST database.

two methods is depicted in Figure 6 where, as before, red corresponds to our method and blue to GAN. Here, lower distance values signify better results and, as we can see, KID is consistent with the synthetic images of Figure 5.

To test the approximate cost function suggested in (23) we implement it in TensorFlow. We adopt for both networks, ours and GAN,

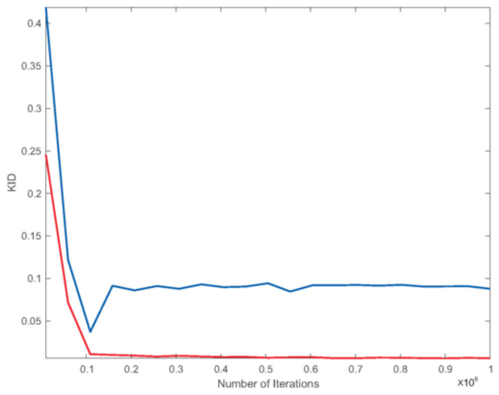

Figure 7. Evolution of the Kernel Inspection Distance (KID) with number of iterations: Proposed (red), GAN (blue) for the Fashion-MNIST database.

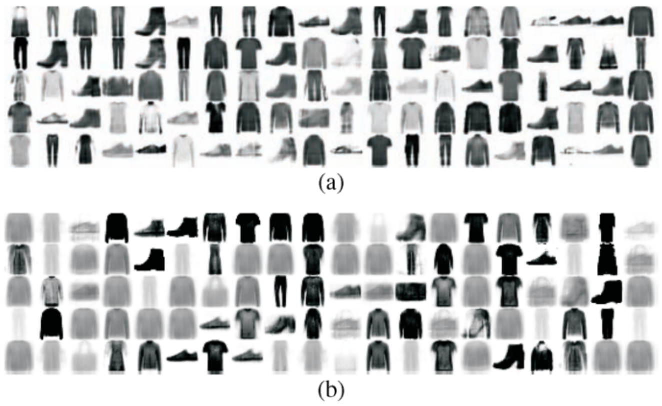

Figure 8. Results after training with Fashion-MNIST with 10^6 iterations. (a) Proposed, (b) GAN.

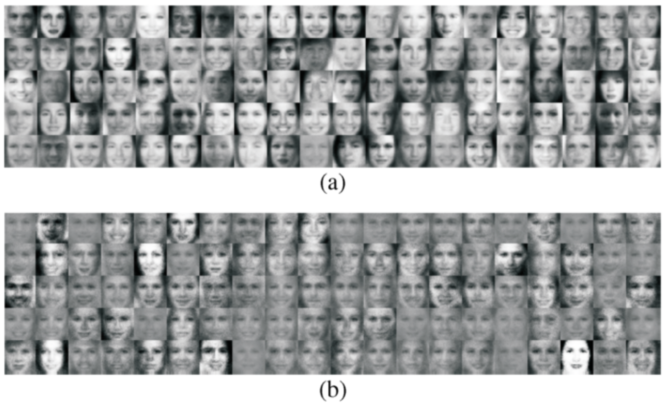

Figure 10. Results after training with CelebA with 2×10^5 iterations. (a) Proposed method, (b) GAN.

a four layer configuration of size $10 \times 20 \times 40 \times 784$. For GAN we also use a discriminator of size $784 \times 40 \times 20 \times 1$. For (23) we select $n = 2000$ and the same batch size for GAN. We train our networks with the Fashion-MNIST database [25] that contains a total of 70,000 images of 10 different types of clothes, shoes, etc. Each image is in gray scale and of dimensions 28×28. We use $\mu = 10^{-3}$, $\lambda = 0.999$ and Gaussian kernel with $h = 25$. Figure 7 depicts the KID for the two cases as a function of the training iterations. For GAN we count the iterations with respect to the minimization problem. We can see that our algorithm, even with the approximate cost, performs better than GAN. In Figure 8 we can verify this fact by observing the synthetic images generated by the two networks after they were trained for 10^6 iterations. We can clearly see that the outcomes are consistent with the KID graph.

The next set of experiments involves the CelebA database [20]. We use only 30,000 RGB images of faces that we cropped to size 32×32, transformed into gray level and reshaped into vectors of length 1024. Each pixel value is normalized to [0,1]. For the generator we use a fully connected two-layer configuration with dimensions $20 \times 300 \times 1024$. As in the previous simulations, for the 20 inputs we use independent standard Gaussian random variables. For the kernel function, again, we select the Gaussian kernel with $h = 40$. We also select the same configuration for the generator in the GAN implementation while for the corresponding GAN discriminator we adopt a two-layer network of dimensions $1024 \times 300 \times 1$. Our simplified stochastic gradient algorithm is implemented for batch size equal to 600 and we use a learning rate $\mu = 10^{-4}$. As before, we apply the gradient normalization scheme of [24] with smoothing factor $\lambda = 0.999$, while we initialize following [13]. Our simplified algorithm is implemented in Matlab. We run the algorithm for 2×10^5 iterations. In Figure 9 we can see the evolution of our cost function and the corresponding bound from the Cauchy-Schwartz. The *average* of the red curve must approach the blue for our algorithm to converge properly. Indeed this is the case with our simulation and we can verify it by smoothing out the red curve.

In Figure 10 we have synthetic faces generated by (a) the proposed method and (b) GAN, after 2×10^5 iterations. We can again observe the better convergence performance of our method as compared to GAN. We should also not forget that this specific behavior for GAN comes at a five times faster consumption of the training data and at five times higher computational complexity.

The last set of experiments is reserved for the CIFAR-10 database, which consists of 60000 RGB images of size 32x32 of 10 different categories as birds, cats, dogs, etc. We normalize each pixel color-component to [0,1] and vectorize each image to a vector of length 3072. For the generator, we adopt a convolutional neural network (CNN) with four layers. The first layer is linear followed by batch normalization [17] and a ReLU function, the next two layers are like the first one but with a deconvolutional operation instead of linear and the last layer consists of a deconvolution operation followed by a sigmoid function in order to constrain the output to the range $[0, 1]$. The generator input is an independent standard Gaussian random vector of length 128. For our kernel function, we adopt the Laplacian kernel $\mathsf{K}(X, Y) = \exp(-\frac{\|X-Y\|_1}{h})$ with $h = 200$. In the GAN

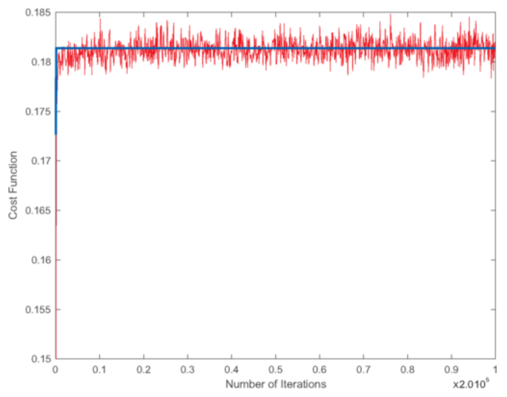

Figure 9. Evolution of proposed cost function (red) and corresponding upper bound (blue) with number of iterations for the CelebA database.

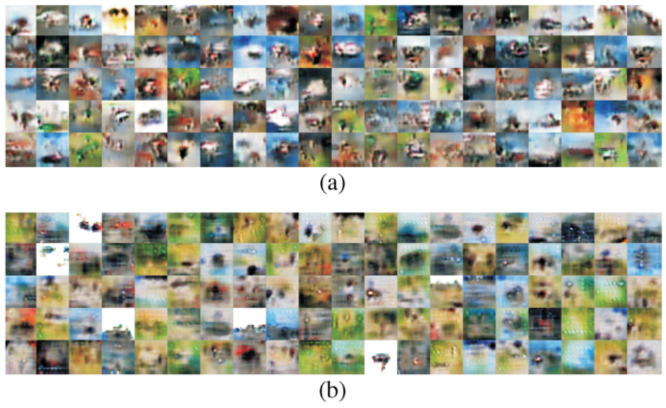

Figure 11. Results after training with CIFAR-10 with 2×10^5 iterations. (a) Proposed method, (b) GAN.

implementation the generator is the same as in our method while the discriminator has a CNN architecture with four layers as well. Its first layer is a convolutional operation with a leaky ReLU, the next two layers have the same units with the first layer but after the convolutional operation we impose batch normalization of the outputs. Finally, the last layer has only a linear operation that provides the output of the discriminator. Training is performed with a learning rate $\mu = 10^{-3}$ and batch size 2000. After training with 2×10^5 iterations our method attains a KID score of 0.0006 while the GAN 0.0016. The better score is again translated into better synthetic images as we can verify by comparing Figures 11(a) and (b).

5 CONCLUSION

We presented a non-adversarial method for designing generative networks. Introducing a kernel-based correlation criterion between the generator output and the training samples we showed that if we maximize this criterion over the neural network parameters then the resulting generator output mimics the statistical behavior of the training data. The corresponding optimization involved only a standard maximization and not an adversarial min-max problem as the known approaches in the literature.

Main challenge in the analytical part of our work constituted the cost function which was nonlinear with its corresponding optimization not being under the usual form that allows for the direct creation of a stochastic gradient algorithm. Important contribution of this work was the extension of this classical approach to provide a simple methodology for deriving stochastic gradient algorithms for non-standard optimization problems.

Finally we also discussed a simplified version of our algorithm that induces significant speed up of the execution of the stochastic gradient algorithm without sacrificing convergence performance. This is achieved by using delayed versions of quantities required by the algorithm, that were already computed during previous iterations. The resulting scheme was tested with well known databases and observed to design generators with synthesis capabilities that were comparable to GANs but requiring a significantly lower number of computations accompanied by a far more stable convergence behavior.

ACKNOWLEDGEMENT

This work was supported by the US National Science Foundation under Grant CIF 1513373, through Rutgers University.

REFERENCES

[1] D. F. Andrews, R. Gnanadesikan, and J. L. Warner. Transformations of multivariate data. *Biometrics*, 27(4):825–840, 1971.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 2017.

[3] K. Basioti, G.V. Moustakides, E.Z. Psarakis. Kernel-based training of generative networks. *arXiv*: 1811.09568, 2018.

[4] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *arXiv*: 1206.5533, 2012.

[5] A. Benveniste and M. Metivier. *Adaptive Algorithms and Stochastic Approximations*. Applications of Mathematics. Springer, 1990.

[6] M. Bińkowski, D. J. Sutherland, M. Arbel and A. Gretton. Demystifying MMD GANs. In *Proceedings of International Conference on Learning Representations*, ICLR-2018. Also *arXiv*: 1801.01401, 2018.

[7] A. Borji. Pros and cons of GAN evaluation measures. *arXiv*: 1802.03446, 2018.

[8] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B*, 26(2):211–252, 1964.

[9] A.-L. Cauchy. Sur les formules qui résultent de l'emploie du signe et sur $>$ ou $<$, et sur les moyennes entre plusieurs quantités. *Cours d'Analyse, 1er Partie: Analyse Algébrique, Œuvres Serie 2 III*, pp. 373-377, 1821.

[10] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, January 2018.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[12] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv*: 1505.03906, 2015.

[13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings International Conference on Artificial Intelligence and Statistics*, 2010.

[14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *arXiv*: 1406.2661, 2014.

[15] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *J. Mach. Learn. Res.*, 13(1):723–773, Mar. 2012.

[16] G. Huang, Y. Yuan, Q. Xu, C. Guo, Y. Sun, F. Wu, and K. Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *arXiv*: 1806.07755, 2018.

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *J. Mach. Learn. Res.*, 37: 448–456, 2015.

[18] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *Proceedings International Conference on Learning Representation*, ICLR-2017. Also *arXiv*: 1609.04836, 2016.

[19] D. P. Kingma and J. L. Ba. ADAM: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations*, ICLR-2015.

[20] Z. Liu, P. Luo,X. Wang and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision*, ICCV-2015.

[21] D. Masters and C. Luschi. Revisiting small batch training for deep neural networks. *arXiv*: 1804.07612, 2018.

[22] L. M. Mescheder, S. Nowozin, and A. Geiger. The numerics of GANs. In *Proceedings Advances Neural Information Processing Systems Conference*, 2017.

[23] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv*: 1511.06434, 2015.

[24] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop. *COURSERA: Neural Networks for Machine Learning*, 2012.

[25] H. Xiao, K. Rasul and R¿ Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv*: 1708.07747, 2017.