POLYGAN: HIGH-ORDER POLYNOMIAL GENERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative Adversarial Networks (GANs) have become the gold standard when it comes to learning generative models for high-dimensional distributions. Since their advent, numerous variations of GANs have been introduced in the literature, primarily focusing on utilization of novel loss functions, optimization/regularization strategies and network architectures. In this paper, we turn our attention to the generator and investigate the use of high-order polynomials as an alternative class of universal function approximators. Concretely, we propose PolyGAN, where we model the data generator by means of a high-order polynomial whose unknown parameters are naturally represented by high-order tensors. We introduce two tensor decompositions that significantly reduce the number of parameters and show how they can be efficiently implemented by hierarchical neural networks that only employ linear/convolutional blocks. We exhibit for the first time that by using our approach a GAN generator can approximate the data distribution without using *any* activation functions. Thorough experimental evaluation on both synthetic and real data (images and 3D point clouds) demonstrates the merits of PolyGAN against the state of the art.

1 INTRODUCTION

Generative Adversarial Networks (GANs) are currently one of the most popular lines of research in machine learning. Research on GANs mainly revolves around: (a) how to achieve faster and/or more accurate convergence (e.g., by studying different loss functions (Nowozin et al., 2016; Arjovsky & Bottou, 2017; Mao et al., 2017) or regularization schemes (Odena et al., 2018; Miyato et al., 2018; Gulrajani et al., 2017)), and (b) how to design different hierarchical neural networks architectures composed of linear and non-linear operators that can effectively model high-dimensional distributions (e.g., by progressively training large networks (Karras et al., 2018) or by utilizing deep ResNet type of networks as generators (Brock et al., 2019)).

Even though hierarchical deep networks are efficient universal approximators for the class of continuous compositional functions (Mhaskar et al., 2016), the non-linear **activation functions** pose difficulties in their theoretical analysis, understanding, and interpretation. For instance, as illustrated in Arora et al. (2019), element-wise non-linearities pose a challenge on proving convergence, especially in an adversarial learning setting (Ji & Liang, 2018). Consequently, several methods, e.g., Saxe et al. (2014); Hardt & Ma (2017); Laurent & Brecht (2018); Lampinen & Ganguli (2019), focus only on linear models (with respect to the weights) in order to be able to rigorously analyze the neural network dynamics, the residual design principle, local extrema and generalization error, respectively. Moreover, as stated in the recent in-depth comparison of many different GAN training schemes (Lucic et al., 2018), the improvements may mainly arise from a higher computational budget and tuning and not from fundamental architectural choices.

In this paper, we depart from the choice of hierarchical neural networks that involve activation functions and investigate for the first time in the literature of GANs the use of high-order polynomials as an alternative class of universal function approximators for data generator functions. This choice is motivated by the strong evidence provided by the *Stone–Weierstrass theorem* (Stone, 1948), which states that every continuous function defined on a closed interval can be uniformly approximated as closely as desired by a polynomial function. Hence, we propose to model the vector-valued generator function $\boldsymbol{G}(\boldsymbol{z}) : \mathbb{R}^d \rightarrow \mathbb{R}^o$ by a high-order multivariate polynomial of the latent vector $\boldsymbol{z}$, whose unknown parameters are naturally represented by high-order tensors.

However, the number of parameters required to accommodate all higher-order correlations of the latent vector explodes with the desired order of the polynomial and the dimension of the latent vector. To alleviate this issue and at the same time capture interactions of parameters across different orders of approximation in a hierarchical manner, we cast polynomial parameters estimation as a coupled tensor factorization (Papalexakis et al., 2016; Sidiropoulos et al., 2017) that jointly factorizes all the polynomial parameters tensors. To this end, we introduce two specifically tailored coupled canonical polyadic (CP)-type of decompositions with shared factors. The proposed coupled decompositions of the parameters tensors result into two different hierarchical structures (i.e., architectures of neural network decoders) that do not involve *any* activation function, providing an intuitive way of generating samples with an increasing level of detail. This is pictorially shown in Figure 1. The result of the proposed PolyGAN using a fourth-order polynomial approximator is shown in Figure 1 (a), while Figure 1 (b) shows the corresponding generation when removing the fourth-order power from the generator.

Our contributions are summarized as follows:

- We model the data generator with a high-order polynomial. Core to our approach is to cast polynomial parameters estimation as a coupled tensor factorization with shared factors. To this end, we develop two coupled tensor decompositions and demonstrate how those two derivations result in different neural network architectures involving only linear (e.g., convolution) units. This approach reveals links between high-order polynomials, coupled tensor decompositions and network architectures.
- We experimentally verify that the resulting networks can learn to approximate functions with analytic expressions.
- We show how the proposed networks can be used with linear blocks, i.e., without utilizing activation functions, to synthesize high-order intricate signals, such as images.
- We demonstrate that by incorporating activation functions to the derived polynomial-based architectures, PolyGAN improves upon three different GAN architectures, namely DCGAN (Radford et al., 2015), SNGAN (Miyato et al., 2018) and SAGAN (Zhang et al., 2019).

(a) (b)

Figure 1: Generated samples by an instance of the proposed PolyGAN. (a) Generated samples using a fourth-order polynomial and (b) the corresponding generated samples when removing the terms that correspond to the fourth-order. As evidenced, by extending the polynomial terms, PolyGAN generates samples with an increasing level of detail.

2 METHOD

In this Section, we investigate the use of a polynomial expansion as a function approximator for the data generator in the context of GANs. To begin with, we introduce the notation in Section 2.1. In Section 2.2, we introduce two different polynomials models along with specifically tailored coupled tensor factorizations for the efficient estimation of their parameters.

2.1 PRELIMINARIES AND NOTATION

Matrices (vectors) are denoted by uppercase (lowercase) boldface letters e.g., $\boldsymbol{X}$, ($\boldsymbol{x}$). Tensors are denoted by calligraphic letters, e.g., $\boldsymbol{\mathcal{X}}$. The *order* of a tensor is the number of indices needed to address its elements. Consequently, each element of an Mth-order tensor $\boldsymbol{\mathcal{X}}$ is addressed by M indices, i.e., $(\boldsymbol{\mathcal{X}})_{i_1,i_2,\ldots,i_M} \doteq x_{i_1,i_2,\ldots,i_M}$.

The *mode-m unfolding* of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_M}$ maps $\boldsymbol{\mathcal{X}}$ to a matrix $\boldsymbol{X}_{(m)} \in \mathbb{R}^{I_m \times \bar{I}_m}$ with $\bar{I}_m = \prod_{\substack{k=1 \\ k \neq m}}^{M} I_k$ such that the tensor element $x_{i_1,i_2,\ldots,i_M}$ is mapped to the matrix element $x_{i_m,j}$ where $j = 1 + \sum_{\substack{k=1 \\ k \neq m}}^{M} (i_k - 1) J_k$ with $J_k = \prod_{\substack{n=1 \\ n \neq m}}^{k-1} I_n$. The *mode-$m$ vector product* of $\boldsymbol{\mathcal{X}}$ with a vector $\boldsymbol{u} \in \mathbb{R}^{I_m}$, denoted by $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{u} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$, results in a tensor of order $M - 1$:

$$(\boldsymbol{\mathcal{X}} \times_m \boldsymbol{u})_{i_1,\ldots,i_{m-1},i_{m+1},\ldots,i_M} = \sum_{i_m=1}^{I_m} x_{i_1,i_2,\ldots,i_M} u_{i_m}. \tag{1}$$

Furthermore, we denote $\boldsymbol{\mathcal{X}} \times_1 \boldsymbol{u}^{(1)} \times_2 \boldsymbol{u}^{(2)} \times_3 \cdots \times_M \boldsymbol{u}^{(M)} \doteq \boldsymbol{\mathcal{X}} \prod_{m=1}^{m} \times_m \boldsymbol{u}^{(m)}$.

The *Khatri-Rao* product (i.e., column-wise Kronecker product) of matrices $\boldsymbol{A} \in \mathbb{R}^{I \times N}$ and $\boldsymbol{B} \in \mathbb{R}^{J \times N}$ is denoted by $\boldsymbol{A} \odot \boldsymbol{B}$ and yields a matrix of dimensions $(IJ) \times N$. The Hadamard product of $\boldsymbol{A} \in \mathbb{R}^{I \times N}$ and $\boldsymbol{B} \in \mathbb{R}^{I \times N}$ is defined as $\boldsymbol{A} * \boldsymbol{B}$ and is equal to $A_{(i,j)} B_{(i,j)}$ for the (i, j) element.

The *CP decomposition* (Kolda & Bader, 2009; Sidiropoulos et al., 2017) factorizes a tensor into a sum of component rank-one tensors. An Mth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_M}$ has *rank*-1, when it is decomposed as the outer product of M vectors $\{\boldsymbol{u}^{(m)} \in \mathbb{R}^{I_m}\}_{m=1}^{M}$. That is, $\boldsymbol{\mathcal{X}} = \boldsymbol{u}^{(1)} \circ \boldsymbol{u}^{(2)} \circ \cdots \circ \boldsymbol{u}^{(M)} \doteq \bigcirc_{m=1}^{M} \boldsymbol{x}^{(m)}$, where $\circ$ denotes for the vector outer product. Consequently, the rank-R CP decomposition of an Mth-order tensor $\boldsymbol{\mathcal{X}}$ is written as:

$$\boldsymbol{\mathcal{X}} \doteq [\![\boldsymbol{U}_{[1]}, \boldsymbol{U}_{[2]}, \ldots, \boldsymbol{U}_{[M]}]\!] = \sum_{r=1}^{R} \boldsymbol{u}_r^{(1)} \circ \boldsymbol{u}_r^{(2)} \circ \cdots \circ \boldsymbol{u}_r^{(M)}, \tag{2}$$

where the factor matrices $\left\{\boldsymbol{U}_{[m]} = [\boldsymbol{u}_1^{(m)}, \boldsymbol{u}_2^{(m)}, \cdots, \boldsymbol{u}_R^{(m)}] \in \mathbb{R}^{I_m \times R}\right\}_{m=1}^{M}$ collect the vectors from the rank-one components. By considering the mode-1 unfolding of $\boldsymbol{\mathcal{X}}$, the CP decomposition can be written in matrix form as (Kolda & Bader, 2009):

$$\boldsymbol{\mathcal{X}}_{(1)} = \boldsymbol{U}_{[1]} \left(\boldsymbol{U}_{[M]} \odot \boldsymbol{U}_{[M-1]} \odot \cdots \odot \boldsymbol{U}_{[2]} \right)^T \doteq \boldsymbol{U}_{[1]} \left(\bigodot_{m=M}^{2} \boldsymbol{U}_{[m]} \right)^T \tag{3}$$

More details on tensors and multilinear operators can be found in Kolda & Bader (2009); Sidiropoulos et al. (2017).

2.2 High-order polynomial generators

GANs typically consist of two deep networks, namely a generator G and a discriminator D. G is a decoder (i.e., a function approximator of the sampler of the target distribution) which receives as input a random noise vector $\boldsymbol{z} \in \mathbb{R}^d$ and outputs a sample $\boldsymbol{x} = G(\boldsymbol{z}) \in \mathbb{R}^o$. D receives as input both $G(\boldsymbol{z})$ and real samples and tries to differentiate the fake and the real samples. During training, both G and D compete against each other till they reach an "equilibrium" (Goodfellow et al., 2014). In practice, both the generator and the discriminator are modeled as deep neural networks, involving composition of linear and non-linear operators (Radford et al., 2015).

Table 1: Nomenclature

Symbol	Dimension(s)	Definition
n, N	$\mathbb{N}$	Polynomial term order, total approximation order.
k	$\mathbb{N}$	Rank of the decompositions.
$\boldsymbol{z}$	$\mathbb{R}^d$	Input to the polynomial approximator, i.e., generator.
$\boldsymbol{C}, \boldsymbol{\beta}$	$\mathbb{R}^{o \times k}, \mathbb{R}^o$	Parameters in both decompositions.
$\boldsymbol{A}_{[n]}, \boldsymbol{S}_{[n]}, \boldsymbol{B}_{[n]}$	$\mathbb{R}^{d \times k}, \mathbb{R}^{k \times k}, \mathbb{R}^{\omega \times k}$	Matrix parameters in the hierarchical decomposition.
$\odot, *$	-	Khatri-Rao product, Hadamard product.

In this paper, we focus on the generator. Instead of modeling the generator as a composition of linear and non-linear functions, we assume that each generated pixel $x_i = (G(\boldsymbol{z}))_i$ may be expanded as a N^{th} order polynomial[1] in $\boldsymbol{z}$. That is,

[1] With an N^{th} order polynomial we can approximate any smooth function (Stone, 1948).

$$x_i = (G(\boldsymbol{z}))_i = \beta_i + \boldsymbol{w}_i^{[1]^T}\boldsymbol{z} + \boldsymbol{z}^T\boldsymbol{W}_i^{[2]}\boldsymbol{z} + \boldsymbol{\mathcal{W}}_i^{[3]} \times_1 \boldsymbol{z} \times_2 \boldsymbol{z} \times_3 \boldsymbol{z} + \cdots + \boldsymbol{\mathcal{W}}_i^{[N]} \prod_{n=1}^{N} \times_n \boldsymbol{z}, \quad (4)$$

where the scalar β_i, and the set of tensors $\left\{\boldsymbol{\mathcal{W}}_i^{[n]} \in \mathbb{R}^{\prod_{m=1}^{n} \times_m d}\right\}_{n=1}^{N}$ are the parameters of the polynomial expansion associated to each output of the generator, e.g., pixel. Clearly, when $n = 1$, the weights are d-dimensional vectors; when $n = 2$, the weights, i.e., $\boldsymbol{W}_i^{[2]}$, form a $d \times d$ matrix. For higher orders of approximation, i.e., when $n \geqslant 3$, the weights are n^{th} order tensors.

By stacking the parameters for all pixels, we define the parameters $\boldsymbol{\beta} \doteq [\beta_1, \beta_2, \ldots, \beta_o]^T \in \mathbb{R}^o$ and $\left\{\boldsymbol{\mathcal{W}}^{[n]} \in \mathbb{R}^{o \times \prod_{m=1}^{n} \times_m d}\right\}_{n=1}^{N}$. Consequently, the vector-valued generator function is expressed as:

$$G(\boldsymbol{z}) = \sum_{n=1}^{N} \left(\boldsymbol{\mathcal{W}}^{[n]} \prod_{j=2}^{n+1} \times_j \boldsymbol{z} \right) + \boldsymbol{\beta} \quad (5)$$

Intuitively, (5) is an expansion which allows the N^{th} order interactions between the elements of the noise latent vector $\boldsymbol{z}$. Furthermore, (5) resembles the functional form of a truncated Maclaurin expansion of vector-valued functions. In the case of a Maclaurin expansion, $\boldsymbol{\mathcal{W}}^{[n]}$ represent the n^{th} order partial derivatives of a known function. However, in our case the generator function is unknown and hence all the parameters need to be estimated from training samples.

The number of the unknown parameters in (5) is $(d^{N+1} - 1)\frac{o}{d-1}$, which grows exponentially with the order of the approximation. Consequently, the model of (5) is prone to overfitting and its training is computationally demanding.

A natural approach to reduce the number of parameters is to assume that the weights exhibit redundancy and hence the parameter tensors are of low-rank. To this end, several low-rank tensor decompositions can be employed (Kolda & Bader, 2009; Sidiropoulos et al., 2017). For instance, let the parameter tensors $\boldsymbol{\mathcal{W}}^{[n]}$ admit a CP decompostion (Kolda & Bader, 2009) of mutilinear rank-k, namely, $\{\boldsymbol{\mathcal{W}}^{[n]} = [\![\boldsymbol{U}_{[n],1}, \boldsymbol{U}_{[n],2}, \ldots, \boldsymbol{U}_{[n],(n+1)}]\!]\}_{n=1}^{N}$, with $\boldsymbol{U}_{[n],1} \in \mathbb{R}^{o \times k}$, and $\boldsymbol{U}_{[n],m} \in \mathbb{R}^{d \times k}$, for $m = 2, \ldots, n+1$. Then, (5) is expressed as

$$G(\boldsymbol{z}) = \sum_{n=1}^{N} \left([\![\boldsymbol{U}_{[n],1}, \boldsymbol{U}_{[n],2}, \ldots, \boldsymbol{U}_{[n],(n+1)}]\!] \prod_{j=2}^{n+1} \times_j \boldsymbol{z} \right) + \boldsymbol{\beta}, \quad (6)$$

which has significantly less parameters than (5), especially when $k \ll d$. However, a set of different factor matrices for each level of approximation are required in equation 6, and hence the hierarchical nature of images is not taken into account. To promote compositional structures and capture interactions among parameters in different orders of approximation we introduce next two coupled CP decompositions with shared factors.

Model 1: Coupled CP decomposition:

Instead of factorizing each parameters tensor individually we propose to jointly factorize all the parameter tensors using a coupled CP decomposition with a specific pattern of factor sharing. To illustrate the factorization, we assume a third order approximation ($N = 3$), however in the appendix a generalization to N-th order approximation is provided. Let us assume that the parameters tensors admit the following coupled CP decomposition with the factors corresponding to lower-order levels of approximation being shared across all parameters tensors. That is:

- Let $\boldsymbol{W}^{[1]} = \boldsymbol{C}\boldsymbol{U}_{[1]}^T$, be the parameters for first level of approximation.
- Let assume $\boldsymbol{\mathcal{W}}^{[2]}$ being a superposition of of two weights tensors, namely $\boldsymbol{\mathcal{W}}^{[2]} = \boldsymbol{\mathcal{W}}_{1:2}^{[2]} + \boldsymbol{\mathcal{W}}_{1:3}^{[2]}$, with $\boldsymbol{\mathcal{W}}_{i:j}^{[2]}$ denoting parameters associated with the second order interactions across the i-th and j-th order of approximation. By enforcing the CP decomposition of the above tensors to share the factor with tensors corresponding to lower-order of approximation we obtain in matrix form: $\boldsymbol{W}_{(1)}^{[2]} = \boldsymbol{C}(\boldsymbol{U}_{[3]} \odot \boldsymbol{U}_{[1]})^T + \boldsymbol{C}(\boldsymbol{U}_{[2]} \odot \boldsymbol{U}_{[1]})^T$.

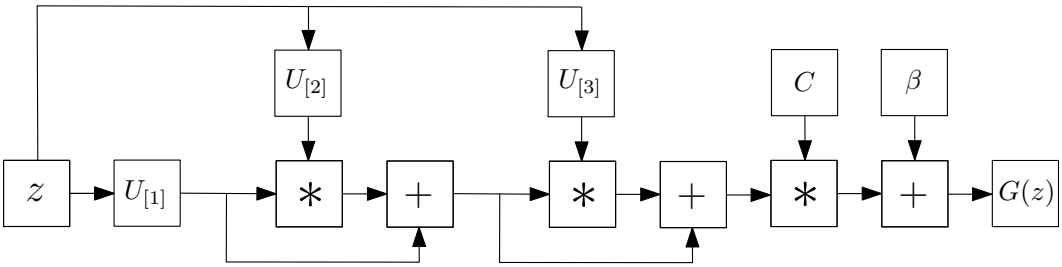


Figure 2: Schematic illustration of the Coupled CP decomposition (for third order approximation). Symbol $*$ refers to the Hadamard product.

- Similarly, we enforce the third-order parameters tensor to admit the following CP decomposition (in matrix form) $\boldsymbol{W}^{[3]}_{(1)} = \boldsymbol{C}(\boldsymbol{U}_{[3]} \odot \boldsymbol{U}_{[2]} \odot \boldsymbol{U}_{[1]})^T$. Note that all but the $\boldsymbol{U}_{[3]}$ factor matrices are shared in the factorization of tensors capturing polynomial parameters for the first and second order of approximation.

The parameters are $\boldsymbol{C} \in \mathbb{R}^{o\times k}, \boldsymbol{U}_{[m]} \in \mathbb{R}^{d\times k}$ for $m = 1, 2, 3$. Then, (6) for $N = 3$ is written as:

$$G(\boldsymbol{z}) = \boldsymbol{\beta} + \boldsymbol{C}\boldsymbol{U}^T_{[1]}\boldsymbol{z} + \boldsymbol{C}\Big(\boldsymbol{U}_{[3]} \odot \boldsymbol{U}_{[1]}\Big)^T(\boldsymbol{z} \odot \boldsymbol{z}) + \boldsymbol{C}\Big(\boldsymbol{U}_{[2]} \odot \boldsymbol{U}_{[1]}\Big)^T(\boldsymbol{z} \odot \boldsymbol{z}) + \boldsymbol{C}\Big(\boldsymbol{U}_{[3]} \odot \boldsymbol{U}_{[2]} \odot \boldsymbol{U}_{[1]}\Big)^T(\boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{z}) \tag{7}$$

The third order approximation of (7) can be implemented as a neural network with the structure of Figure 2 (proved in section B, Claim 1 of the appendix). It is worth noting that the structure of the proposed network allows for incremental network growth.

Model 2: Coupled nested CP decomposition: Instead of explicitly separating the interactions between layers, we can utilize a joint hierarchical decomposition on the polynomial parameters. Let us first introduce learnable hyper-parameters $\left\{\boldsymbol{b}_{[n]} \in \mathbb{R}^{\omega}\right\}_{n=1}^{N}$, which act as scaling factors for each parameter tensor. Therefore, we modify (5) to:

$$G(\boldsymbol{z}) = \sum_{n=1}^{N}\left(\boldsymbol{\mathcal{W}}^{[n]} \times_2 \boldsymbol{b}_{[n]} \prod_{j=3}^{n+2} \times_j \boldsymbol{z}\right) + \boldsymbol{\beta}, \tag{8}$$

with $\left\{\boldsymbol{\mathcal{W}}^{[n]} \in \mathbb{R}^{o\times\omega\times\prod_{m=1}^{n}\times_m d}\right\}_{n=1}^{N}$. For illustration purposes, we consider a third order function approximation ($N = 3$). That is,

$$G(\boldsymbol{z}) = \boldsymbol{\beta} + \boldsymbol{\mathcal{W}}^{[1]} \times_2 \boldsymbol{b}_{[1]} \times_3 \boldsymbol{z} + \boldsymbol{\mathcal{W}}^{[2]} \times_2 \boldsymbol{b}_{[2]} \times_3 \boldsymbol{z} \times_4 \boldsymbol{z} + \boldsymbol{\mathcal{W}}^{[3]} \times_2 \boldsymbol{b}_{[3]} \times_3 \boldsymbol{z} \times_4 \boldsymbol{z} \times_5 \boldsymbol{z} \tag{9}$$

To estimate its parameters we jointly factorize all parameters tensors by employing nested CP detecomposion with parameter sharing as follows (in matrix form)

- First order parameters : $\boldsymbol{W}^{[1]}_{(1)} = \boldsymbol{C}(\boldsymbol{A}_{[3]} \odot \boldsymbol{B}_{[3]})^T$.
- Second order parametes: $\boldsymbol{W}^{[2]}_{(1)} = \boldsymbol{C}\Big\{\boldsymbol{A}_{[3]} \odot \Big[\Big(\boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]}\Big)\boldsymbol{S}_{[3]}\Big]\Big\}^T$.
- Third order parameters: $\boldsymbol{W}^{[3]}_{(1)} = \boldsymbol{C}\Big\{\boldsymbol{A}_{[3]} \odot \Big[\Big(\boldsymbol{A}_{[2]} \odot \Big\{\Big(\boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]}\Big)\boldsymbol{S}_{[2]}\Big\}\Big)\boldsymbol{S}_{[3]}\Big]\Big\}^T$

with $\boldsymbol{C} \in \mathbb{R}^{o\times k}, \boldsymbol{A}_{[n]} \in \mathbb{R}^{d\times k}, \boldsymbol{S}_{[n]} \in \mathbb{R}^{k\times k}, \boldsymbol{B}_{[n]} \in \mathbb{R}^{\omega\times k}$ for $n = 1, \ldots, N$. Altogether, (9) is written as:

$$G(\boldsymbol{z}) = \boldsymbol{\beta} + \boldsymbol{C}(\boldsymbol{A}_{[3]} \odot \boldsymbol{B}_{[3]})^T(\boldsymbol{z} \odot \boldsymbol{b}_{[3]}) + \boldsymbol{C}\Big\{\boldsymbol{A}_{[3]} \odot \Big[\Big(\boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]}\Big)\boldsymbol{S}_{[3]}\Big]\Big\}^T\Big(\boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[2]}\Big) + \boldsymbol{C}\Big\{\boldsymbol{A}_{[3]} \odot \Big[\Big(\boldsymbol{A}_{[2]} \odot \Big\{\Big(\boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]}\Big)\boldsymbol{S}_{[2]}\Big\}\Big)\boldsymbol{S}_{[3]}\Big]\Big\}^T\Big(\boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[1]}\Big) \tag{10}$$

As we prove in the appendix (section B, Claim 3), (10) can be implemented in a hierarchical manner with a three-layer neural network as shown in Figure 3.

Comparison between the two models: Both models are based on the polynomial expansion, however there are few differences between those. The Coupled CP decomposition has a simpler expression, however the Coupled nested CP decomposition relates to standard architectures using hierarchical composition that has recently yielded promising results in GANs (see Section 3). In the remainder of the paper, we use the Coupled nested CP decomposition by default; in Section G, we include an experimental comparison of the two models. The experimental comparison demonstrates that neither model outperforms the other in all datasets; they perform similarly.

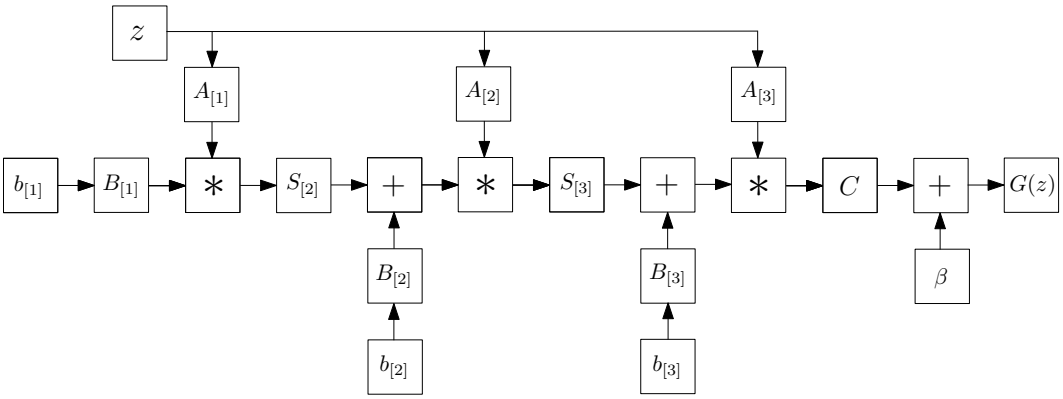


Figure 3: Schematic illustration of the Coupled nested CP decomposition (for third order approximation). Symbol $*$ refers to the Hadamard product.

3 Related work

The literature on GANs is vast; we focus only on the works most closely related to ours. The interested reader can find further information in a recent survey (Creswell et al., 2018).

Berthelot et al. (2017) use skip connections to concatenate the noise $\boldsymbol{z}$ in deeper layers in the generator. The recent BigGAN (Brock et al., 2019) performs a hierarchical composition through skip connections from the noise $\boldsymbol{z}$ to multiple resolutions of the generator. In their implementation, they split $\boldsymbol{z}$ into one chunk per resolution and concatenate each chunk (of $\boldsymbol{z}$) to the respective resolution.

Despite the propagation of the noise $\boldsymbol{z}$ to successive layers, the aforementioned works have substantial differences from ours. We introduce a well-motivated and mathematically elaborate method to achieve a more precise approximation with a polynomial expansion. In contrast to the previously mentioned works, we also do not concatenate the noise with the feature representations, but rather perform multiplication of the noise with the feature representations, which we mathematically justify.

The work that is most closely related to ours is the recently proposed StyleGAN (Karras et al., 2019), which is an improvement over the Progressive Growing of GANs (ProGAN) (Karras et al., 2018). As ProGAN, StyleGAN is a highly-engineered network that achieves compelling results on synthesized 2D images. In order to provide an explanation on the improvements of StyleGAN over ProGAN, the authors adopt arguments from the style transfer literature (Huang & Belongie, 2017). Nevertheless, the idea of style transfer proposes to use features from images for conditional image translation, which is very different to unsupervised samples (image) generation. We believe that these improvements

can be better explained under the light of our proposed polynomial function approximation. That is, as we show in Figure 1, the Hadamard products build a hierachical decomposition with increasing level of detail (rather than different styles). In addition, the improvements in StyleGAN (Karras et al., 2019) are demonstrated by using a well-tuned model. In this paper we showcase that without any complicated engineering process the polynomial generation can be applied into several architectures (or any other type of decoders) and consistently improves the performance.

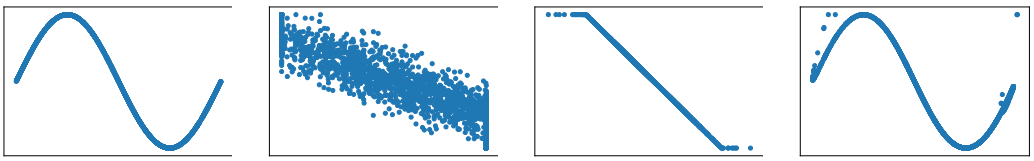
4 EXPERIMENTS

A sequence of experiments in both synthetic data ($2D$ and $3D$ data manifolds) and higher-dimensional signals are conducted to assess the empirical performance of the proposed polynomial expansion. The first experiments are conducted on a $2D$ manifolds that are analytically known (Section 4.1). Further experiments on three $3D$ manifolds are deferred to the appendix (Section D). In Section 4.2, the polynomial expansion is used for synthesizing digits. Experiments on images beyond digits are conducted in Section E; more specifically, we experiment with images of faces and natural scenes. The experiments with such images demonstrate how polynomial expansion can be used for learning highly complex distributions by using a single activation function in the generator. Lastly, we augment our polynomial-based generator with non-linearities and show that this generator is at least as powerful as contemporary architectures.

Apart from the polynomial-based generators, we implemented two variations that are considered baselines: (a) 'Concat': we replace the Hadamard operator with concatenation (used frequently in recent methods, such as in Brock et al. (2019)), (b) 'Orig': the Hadamard products are ditched, while use $\boldsymbol{b}_{[1]} \leftarrow \boldsymbol{z}$, i.e., there is a composition of linear layers that transform the noise $\boldsymbol{z}$.

4.1 SYNTHETIC EXPERIMENT ON 2D MANIFOLD

Sinusoidal: We assess the polynomial-based generator on a sinusoidal function in the bounded domain $[0, 2\pi]$. Only linear blocks, i.e., no activation functions, are used in the generator. That is, all the element-wise non-linearities (such as ReLU's, $\tanh$) are ditched. The distribution we want to match is a $\sin x$ signal. The input to the generator is $z \in \mathbb{R}$ and the output is $[x, \sin x]$ with $x \in [0, 2\pi]$. We assume a 12^{th} order approximation where each $\boldsymbol{S}_{[i]}, \boldsymbol{A}_{[i]}$ is a fully-connected layer and $\boldsymbol{B}_{[i]}$ is an identity matrix. Each fully-connected layer has width 15. In Figure 4, $2,000$ random samples are synthesized. We indeed verify that in low-dimensional distributions, such as the univariate sinusoidal, PolyGAN indeed approximates the data distribution quite accurately without using any non-linear activation functions.



(a) GT (b) Orig (c) Concat (d) PolyGAN

Figure 4: Synthesized data for learning the $[x, \sin x]$ signal. No activation functions are used in the generators. From left to right: (a) the data distribution, (b) 'Orig', (c) 'Concat', (d) PolyGAN. Notably, neither 'Orig' nor 'Concat' can learn to approximate different Taylor terms.

4.2 DIGIT GENERATION WITH LINEAR BLOCKS

The linear generator of the previous section is extended to greyscale images, in which an analytic expression of the ground-truth distribution remains elusive. To our knowledge, there has not been a generation of greyscale images based on polynomial expansion in the past.

We capitalize on the expressivity of the recent resnet-based generator (Miyato et al., 2018; Brock et al., 2019), to devise a new polynomial generator $G(\boldsymbol{z}) : \mathbb{R}^{128} \rightarrow \mathbb{R}^{32x32}$. We consider a fourth-order

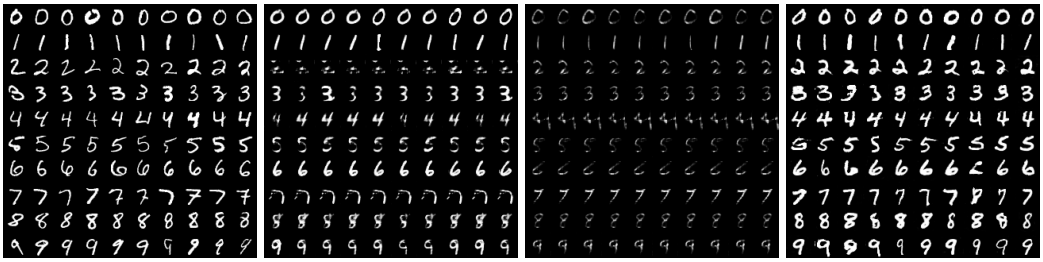
(a) GT (b) Orig (c) Concat (d) PolyGAN

Figure 5: Synthesized data for MNIST with a single activation in the generator. From left to right: (a) The ground-truth signals, (b) 'Orig', (c) 'Concat', (d) PolyGAN.

approximation (as derived in (5)) where $\boldsymbol{B}_{[i]}$ is the identity matrix, $\boldsymbol{S}_{[i]}$ is a residual block with two convolutions for $i = 1, \ldots, 4$. We emphasize that the residual block as well as all layers are *linear*, i.e., there are no activation functions. We only add a $\tanh$ in the output of the generator for normalization purposes. The discriminator and the optimization procedure are the same as in SNGAN; the only difference is that we run one discriminator step per generator step ($n_{dis} = 1$). Note that the 'Orig' resnet-based generator resembles the generator of Miyato et al. (2018) in this case.

We perform digit generation (trained on MNIST (LeCun et al., 1998)). In Figure 5, random samples are visualized for the three compared methods. Note that the two baselines have practically collapsed into a single number each, whereas PolyGAN does synthesize plausible digits.

To further assist the generation process, we utilize the labels and train a conditional GAN. That is, the class labels are used for conditional batch normalization. As illustrated in Figure 6, the samples synthesized are improved over the unsupervised setting. 'Orig' and 'Concat' still suffer from severe mode collapse, while PolyGAN synthesizes digits that have different thickness (e.g. 9), style (e.g. 2) and rotation (e.g. 1).



(a) GT (b) Orig (c) Concat (d) PolyGAN

Figure 6: Conditional digit generation. Note that both 'Orig' and 'Concat' suffer from severe mode collapse (details in section 4.2). On the contrary, PolyGAN synthesizes digits that have different thickness (e.g. 9), style (e.g. 2) and rotation (e.g. 1).

5 Conclusion

We express data generation as a polynomial expansion task. We model the high-order polynomials with tensorial factors. We introduce two tailored coupled decompositions and show how the polynomial parameters can be implemented by hierarchical neural networks, e.g. as generators in a GAN setting. We exhibit how such polynomial-based generators can be used to synthesize images by utilizing only linear blocks. In addition, we empirically demonstrate that our polynomial expansion can be used with non-linear activation functions to improve the performance of standard state-of-the-art architectures. Finally, it is worth mentioning that our approach reveals links between high-order polynomials, coupled tensor decompositions and network architectures.

REFERENCES

Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Sanjeev Arora, Nadav Cohen, Noah Golowich, and Wei Hu. A convergence analysis of gradient descent for deep linear neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

David Berthelot, Thomas Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations (ICLR)*, 2019.

Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1): 53–65, 2018.

Athinodoros S Georghiades, Peter N Belhumeur, and David J Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, (6):643–660, 2001.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems (NIPS)*, 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems (NIPS)*, pp. 5767–5777, 2017.

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems (NIPS)*, pp. 6626–6637, 2017.

Emiel Hoogeboom, Rianne van den Berg, and Max Welling. Emerging convolutions for generative normalizing flows. In *International Conference on Machine Learning (ICML)*, 2019.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *IEEE Proceedings of International Conference on Computer Vision (ICCV)*, pp. 1501–1510, 2017.

Kaiyi Ji and Yingbin Liang. Minimax estimation of neural net distance. In *Advances in neural information processing systems (NIPS)*, pp. 3845–3854, 2018.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations (ICLR)*, 2018.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *IEEE Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3): 455–500, 2009.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

Andrew K Lampinen and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Thomas Laurent and James Brecht. Deep linear networks with arbitrary loss: All local minima are global. In *International Conference on Machine Learning (ICML)*, 2018.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *Advances in neural information processing systems (NIPS)*, pp. 700–709, 2018.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *IEEE Proceedings of International Conference on Computer Vision (ICCV)*, pp. 2813–2821. IEEE, 2017.

Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. Learning functions: when is deep better than shallow. *arXiv preprint arXiv:1603.00988*, 2016.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems (NIPS)*, pp. 271–279, 2016.

Augustus Odena, Jacob Buckman, Catherine Olsson, Tom B Brown, Christopher Olah, Colin Raffel, and Ian Goodfellow. Is generator conditioning causally related to gan performance? In *International Conference on Machine Learning (ICML)*, 2018.

Evangelos E Papalexakis, Tom M Mitchell, Nicholas D Sidiropoulos, Christos Faloutsos, Partha Pratim Talukdar, and Brian Murphy. Turbo-smt: Parallel coupled sparse matrix-tensor factorizations and applications. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 9(4): 269–290, 2016.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems (NIPS)*, pp. 2234–2242, 2016.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

Marshall H Stone. The generalized weierstrass approximation theorem. *Mathematics Magazine*, 21 (5):237–254, 1948.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR)*, 2016.

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *IEEE Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7794–7803, 2018.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International Conference on Machine Learning (ICML)*, 2019.

Algorithm 1: PolyGAN (model 1).

Input : Noise $\boldsymbol{z} \in \mathbb{R}^d, N \in \mathbb{N}$
Output : $\boldsymbol{x} \in \mathbb{R}^o$

% global transformation(s) of $\boldsymbol{z}$.
$\boldsymbol{v}$ = Linear($\boldsymbol{z}$)
% first layer.
$\boldsymbol{\kappa} = (\boldsymbol{U}_{[1]})^T \boldsymbol{v}$
for *n=2:N* **do**
    % Perform the Hadamard product for the n^{th} layer.
    $\boldsymbol{\kappa} = \boldsymbol{\kappa} + \left((\boldsymbol{U}_{[n]})^T \boldsymbol{v}\right) * \boldsymbol{\kappa}$
end
$\boldsymbol{x} = \boldsymbol{\beta} + \boldsymbol{C}\boldsymbol{\kappa}$.

Algorithm 2: PolyGAN (model 2).

Input : Noise $\boldsymbol{z} \in \mathbb{R}^d, N \in \mathbb{N}$
Output : $\boldsymbol{x} \in \mathbb{R}^o$

% global transformation(s) of $\boldsymbol{z}$.
$\boldsymbol{v}$ = Linear($\boldsymbol{z}$)
% first layer.
$\boldsymbol{\kappa} = \left((\boldsymbol{B}_{[1]})^T \boldsymbol{b}_{[1]}\right) * \left((\boldsymbol{A}_{[1]})^T \boldsymbol{v}\right)$
for *n=2:N* **do**
    % Multiply with the current layer weight $\boldsymbol{S}_{[n]}$ and perform the Hadamard product.
    $\boldsymbol{\kappa} = \left(\boldsymbol{S}_{[n]}\boldsymbol{\kappa} + (\boldsymbol{B}_{[n]})^T \boldsymbol{b}_{[n]}\right) * \left((\boldsymbol{A}_{[n]})^T \boldsymbol{v}\right)$
end
$\boldsymbol{x} = \boldsymbol{\beta} + \boldsymbol{C}\boldsymbol{\kappa}$.

Table 2: The pseudocode for the two models for N^{th} order polynomial approximation.

A INTRODUCTION

The appendix is organized as:

- Section B provides the Lemmas and their proofs required for our derivations.
- Section C generalizes the Coupled CP decomposition for N^{th} order expansion.
- Section D extends the experiments to 3D manifolds.
- In Section E, additional experiments on image generation with linear blocks are conducted.
- Comparisons with popular GAN architectures are conducted in Section F. Specifically, we utilize three popular generator architectures and devise their polynomial equivalent and perform comparisons on image generation. We also conduct an ablation study indicating how standard engineering techniques affect the image generation of the polynomial generator.
- In Section G, a comparison between the two proposed decompositions is conducted on data distributions from the previous Sections.

B PROOFS

For a set of matrices $\{\boldsymbol{X}_m \in \mathbb{R}^{I_m \times N}\}_{m=1}^N$ the Khatri-Rao product is denoted by:

$$\boldsymbol{X}_1 \odot \boldsymbol{X}_2 \odot \cdots \odot \boldsymbol{X}_M \doteq \bigodot_{m=1}^{M} \boldsymbol{X}_m \tag{11}$$

In this section, we prove the following identity connecting the sets of matrices $\{\boldsymbol{A}_\nu \mathbb{R}^{I_\nu \times K}\}_{\nu=1}^N$ and $\{\boldsymbol{B}_\nu \mathbb{R}^{I_\nu \times L}\}_{\nu=1}^N$:

$$(\bigodot_{\nu=1}^{N} \boldsymbol{A}_\nu)^T \cdot (\bigodot_{\nu=1}^{N} \boldsymbol{B}_\nu) = (\boldsymbol{A}_1^T \cdot \boldsymbol{B}_1) * (\boldsymbol{A}_2^T \cdot \boldsymbol{B}_2) * \ldots * (\boldsymbol{A}_N^T \cdot \boldsymbol{B}_N) \tag{12}$$

To demonstrate the simple case with two matrices, we prove first the special case with $N = 2$.

Lemma 1. *It holds that*

$$(\boldsymbol{A}_1 \odot \boldsymbol{A}_2)^T \cdot (\boldsymbol{B}_1 \odot \boldsymbol{B}_2) = (\boldsymbol{A}_1^T \cdot \boldsymbol{B}_1) * (\boldsymbol{A}_2^T \cdot \boldsymbol{B}_2) \tag{13}$$

Proof. Initially, both sides of the equation have dimensions of $K \times L$, i.e., they match. The (i, j) element of the matrix product of $(\boldsymbol{A}_1^T \cdot \boldsymbol{B}_1)$ is

$$\sum_{k_1=1}^{I_1} A_{1,(k_1,i)} B_{1,(k_1,j)} \tag{14}$$

Then the (i,j) element of the right hand side (rhs) of (13) is:

$$\begin{aligned} E_{rhs} = (\sum_{k_1=1}^{I_1} A_{1,(k_1,i)} B_{1,(k_1,j)}) \cdot (\sum_{k_2=1}^{I_2} A_{2,(k_2,i)} B_{2,(k_2,j)}) = \\ \sum_{k_1=1}^{I_1} \sum_{k_2=1}^{I_2} (A_{1,(k_1,i)} A_{2,(k_2,i)})(B_{(1,k_1,j)} B_{2,(k_2,j)}) \end{aligned} \tag{15}$$

From the definition of Khatri-Rao, it is straightforward to obtain the (ρ, i) element with $\rho = (k_1 - 1)I_2 + k_2$, (i.e. $\rho \in [1, I_1 I_2]$) of $\boldsymbol{A}_1 \odot \boldsymbol{A}_2$ as $A_{1,(k_1,i)} A_{2,(k_2,i)}$. Similarly, the (ρ, j) element of $\boldsymbol{B}_1 \odot \boldsymbol{B}_2$ is $B_{1,(k_1,j)} B_{2,(k_2,j)}$.

The respective (i,j) element of the left hand side (lhs) of (13) is:

$$\begin{aligned} E_{lhs} = \sum_{\rho=1}^{I_1 I_2} A_{1,(k_1,i)} A_{2,(k_2,i)} B_{1,(k_1,j)} B_{2,(k_2,j)} = \\ \sum_{k_1=1}^{I_1} \sum_{k_2=1}^{I_2} A_{1,(k_1,i)} A_{2,(k_2,i)} B_{1,(k_1,j)} B_{2,(k_2,j)} = E_{rhs} \end{aligned} \tag{16}$$

In the last equation, we replace the sum in ρ ($\rho \in [1, I_1 I_2]$) with the equivalent sums in k_1, k_2. □

In a similar manner, we generalize the identity to the case of $N > 2$ terms below.

Lemma 2. *It holds that*

$$(\bigodot_{\nu=1}^{N} \boldsymbol{A}_\nu)^T \cdot (\bigodot_{\nu=1}^{N} \boldsymbol{B}_\nu) = (\boldsymbol{A}_1^T \cdot \boldsymbol{B}_1) * (\boldsymbol{A}_2^T \cdot \boldsymbol{B}_2) * \ldots * (\boldsymbol{A}_N^T \cdot \boldsymbol{B}_N) \tag{17}$$

Proof. The rhs includes the Hadamard products of the matrices $\boldsymbol{A}_\nu^T \cdot \boldsymbol{B}_\nu$. Each matrix multiplication $(\boldsymbol{A}_\nu^T \cdot \boldsymbol{B}_\nu)$ results in a matrix of $K \times L$ dimensions. Thus, the rhs is a matrix of $K \times L$ dimensions.

The lhs is a matrix multiplication of two Khatri-Rao products. The first Khatri-Rao product has dimensions $K \times (\prod_\nu I_\nu)$, while the second $(\prod_\nu I_\nu) \times L$. Altogether, the lhs has $K \times L$ dimensions.

Similarly to the previous Lemma, the (i,j) element of the rhs is:

$$\begin{aligned} E_{rhs} = (\sum_{k_1=1}^{I_1} A_{1,(k_1,i)} B_{1,(k_1,j)}) \cdot (\sum_{k_2=1}^{I_2} A_{2,(k_2,i)} B_{2,(k_2,j)}) \ldots (\sum_{k_N=1}^{I_N} A_{N,(k_N,i)} B_{N,(k_N,j)}) = \\ \sum_{k_1=1}^{I_1} \sum_{k_2=1}^{I_2} \ldots \sum_{k_N=1}^{I_N} (A_{1,(k_1,i)} A_{2,(k_2,i)} \ldots A_{N,(k_N,i)})(B_{1,(k_1,j)} B_{2,(k_2,j)} \ldots B_{N,(k_N,j)}) \end{aligned} \tag{18}$$

To proceed with the lhs, it is straightforward to derive that

$$(\bigodot_{\nu=1}^{N} \boldsymbol{A}_\nu) = A_{1,(s_1,i)} A_{2,(s_2,i)} \ldots A_{N,(s_N,i)} \tag{19}$$

where $s_1 = i$ and s_ν is a recursive function of the $s_{\nu-1}$.

However, the recursive definition of s_ν is summed in the multiplication and we obtain:

$$E_{lhs} = \sum_{k_1=1}^{I_1} \sum_{k_2=1}^{I_2} \ldots \sum_{k_N=1}^{I_N} (A_{1,(k_1,i)} A_{2,(k_2,i)} \ldots A_{N,(k_N,i)})(B_{1,(k_1,j)} B_{2,(k_2,j)} \ldots B_{N,(k_N,j)}) = E_{rhs} \tag{20}$$

□

B.1 PROOFS FOR MODEL 1

Below, we prove that (7) (main paper) is equivalent to the three-layer neural network as shown in Figure 2.

Claim 1. *Let* $\boldsymbol{\omega} = \left(\boldsymbol{U}_{[2]}^T \boldsymbol{z}\right) * \left(\boldsymbol{U}_{[1]}^T \boldsymbol{z}\right) + \boldsymbol{U}_{[1]}^T \boldsymbol{z}$.

Then, the form of (7) is equal to:

$$G(\boldsymbol{z}) = \boldsymbol{\beta} + \boldsymbol{C}\left\{\left(\boldsymbol{U}_{[3]}^T \boldsymbol{z}\right) * \boldsymbol{\omega} + \boldsymbol{\omega}\right\} \tag{21}$$

Proof. Applying Lemma 2 on (7), we obtain:

$$\begin{aligned} G(\boldsymbol{z}) = \boldsymbol{\beta} + \boldsymbol{C}\Big\{\boldsymbol{U}_{[1]}^T \boldsymbol{z} + \left(\boldsymbol{U}_{[3]}^T \boldsymbol{z}\right) * \left(\boldsymbol{U}_{[1]}^T \boldsymbol{z}\right) + \left(\boldsymbol{U}_{[2]}^T \boldsymbol{z}\right) * \left(\boldsymbol{U}_{[1]}^T \boldsymbol{z}\right) + \\ \left(\boldsymbol{U}_{[3]}^T \boldsymbol{z}\right) * \left(\boldsymbol{U}_{[2]}^T \boldsymbol{z}\right) * \left(\boldsymbol{U}_{[1]}^T \boldsymbol{z}\right)\Big\} = \\ \boldsymbol{\beta} + \boldsymbol{C}\Big\{\left(\boldsymbol{U}_{[3]}^T \boldsymbol{z}\right) * \left[\left(\boldsymbol{U}_{[2]}^T \boldsymbol{z}\right) * \left(\boldsymbol{U}_{[1]}^T \boldsymbol{z}\right) + \boldsymbol{U}_{[1]}^T \boldsymbol{z}\right] + \left(\boldsymbol{U}_{[2]}^T \boldsymbol{z}\right) * \left(\boldsymbol{U}_{[1]}^T \boldsymbol{z}\right) + \boldsymbol{U}_{[1]}^T \boldsymbol{z}\Big\} \end{aligned} \tag{22}$$

The last equation is the same as (21). □

B.2 PROOFS FOR MODEL 2

In Claim 2 and Claim 3, we prove that (10) (main paper) is equivalent to the three-layer neural network as shown in Figure 3.

Claim 2. *Let*

$$\boldsymbol{\omega} = \left((\boldsymbol{A}_{[2]})^T \boldsymbol{z}\right) * \left\{(\boldsymbol{B}_{[2]})^T \boldsymbol{b}_{[2]} + (\boldsymbol{S}_{[2]})^T \left[\left((\boldsymbol{A}_{[1]})^T \boldsymbol{z}\right) * \left((\boldsymbol{B}_{[1]})^T \boldsymbol{b}_{[1]}\right)\right]\right\} \tag{23}$$

It holds that

$$\boldsymbol{\omega} = \left\{\boldsymbol{A}_{[2]} \odot \left[\left(\boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]}\right)\boldsymbol{S}_{[2]}\right]\right\}^T \left(\boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[1]}\right) + \left(\boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]}\right)^T \left(\boldsymbol{z} \odot \boldsymbol{b}_{[2]}\right) \tag{24}$$

Proof. We will prove the equivalence starting from (23) and transform it into (24). From (23):

$$\begin{aligned} \boldsymbol{\omega} = \left((\boldsymbol{A}_{[2]})^T \boldsymbol{z}\right) * \left((\boldsymbol{B}_{[2]})^T \boldsymbol{b}_{[2]}\right) + \left((\boldsymbol{A}_{[2]})^T \boldsymbol{z}\right) * \left\{(\boldsymbol{S}_{[2]})^T \left[\left((\boldsymbol{A}_{[1]})^T \boldsymbol{z}\right) * \left((\boldsymbol{B}_{[1]})^T \boldsymbol{b}_{[1]}\right)\right]\right\} = \\ \left(\boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]}\right)^T \left(\boldsymbol{z} \odot \boldsymbol{b}_{[2]}\right) + \left((\boldsymbol{A}_{[2]})^T \boldsymbol{z}\right) * \left\{\left[\left(\boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]}\right)\boldsymbol{S}_{[2]}\right]^T \left(\boldsymbol{z} \odot \boldsymbol{b}_{[1]}\right)\right\} \end{aligned} \tag{25}$$

where in the last equation, we have applied Lemma 1. Applying the Lemma once more in the last term of (25), we obtain (24). □

Claim 3. *Let*

$$\boldsymbol{\lambda} = \boldsymbol{\beta} + \boldsymbol{C}\left\{\left((\boldsymbol{A}_{[3]})^T \boldsymbol{z}\right) * \left[(\boldsymbol{B}_{[3]})^T \boldsymbol{b}_{[3]} + (\boldsymbol{S}_{[3]})^T \boldsymbol{\omega}\right]\right\} \tag{26}$$

with $\boldsymbol{\omega}$ *as in Claim 2. Then, it holds for* $G(\boldsymbol{z})$ *of (10) that* $G(\boldsymbol{z}) = \boldsymbol{\lambda}$.

Proof. Transforming (26) into (10):

$$\begin{aligned}\boldsymbol{\lambda} = \boldsymbol{\beta} + \boldsymbol{C}\Big\{\Big((\boldsymbol{A}_{[3]})^T\boldsymbol{z}\Big) * \Big((\boldsymbol{B}_{[3]})^T\boldsymbol{b}_{[3]}\Big) + \Big((\boldsymbol{A}_{[3]})^T\boldsymbol{z}\Big) * \Big((\boldsymbol{S}_{[3]})^T\boldsymbol{\omega}\Big)\Big\} = \\ \boldsymbol{\beta} + \boldsymbol{C}\Big\{\Big(\boldsymbol{A}_{[3]} \odot \boldsymbol{B}_{[3]}\Big)^T\Big(\boldsymbol{z} \odot \boldsymbol{b}_{[3]}\Big) + \Big((\boldsymbol{A}_{[3]})^T\boldsymbol{z}\Big) * \Big((\boldsymbol{S}_{[3]})^T\boldsymbol{\omega}\Big)\Big\}\end{aligned} \tag{27}$$

To simplify the notation, we define $\boldsymbol{M}_1 = \Big\{\boldsymbol{A}_{[2]} \odot \Big[\Big(\boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]}\Big)\boldsymbol{S}_{[2]}\Big]\Big\}$ and $\boldsymbol{M}_2 = \Big(\boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]}\Big)$. Then, $\boldsymbol{\omega} = \boldsymbol{M}_1^T\Big(\boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[1]}\Big) + \boldsymbol{M}_2^T\Big(\boldsymbol{z} \odot \boldsymbol{b}_{[2]}\Big)$. The last term of (27) becomes:

$$\begin{aligned}\Big((\boldsymbol{A}_{[3]})^T\boldsymbol{z}\Big) * \Big((\boldsymbol{S}_{[3]})^T\boldsymbol{\omega}\Big) = \Big((\boldsymbol{A}_{[3]})^T\boldsymbol{z}\Big) * \Big[\Big(\boldsymbol{M}_1\boldsymbol{S}_{[3]}\Big)^T\Big(\boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[1]}\Big) + \\ \Big(\boldsymbol{M}_2\boldsymbol{S}_{[3]}\Big)^T\Big(\boldsymbol{z} \odot \boldsymbol{b}_{[2]}\Big)\Big] = \\ \Big[\boldsymbol{A}_{[3]} \odot \Big(\boldsymbol{M}_1\boldsymbol{S}_{[3]}\Big)\Big]^T\Big(\boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[1]}\Big) + \Big[\boldsymbol{A}_{[3]} \odot \Big(\boldsymbol{M}_2\boldsymbol{S}_{[3]}\Big)\Big]^T\Big(\boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[2]}\Big)\end{aligned} \tag{28}$$

Replacing (28) into (27), we obtain (10). □

Note that the $\boldsymbol{\lambda}$ in Claim 3 is the equation behind Figure 3. By proving the claim, we have illustrated how the polynomial generator can be transformed into a network architecture for third-order approximation.

C DERIVATIONS

In this Section, we will show how the Coupled CP decomposition generalizes to the N^{th} order approximation. It suffices to find the decomposition that converts the N^{th} order polynomial into a network structure (see Alg. 1).

As done in Section 2.2, we capture the n^{th} order interactions by decomposing the parameter tensor $\boldsymbol{\mathcal{W}}^{[n]}$ (with $2 \leqslant n \leqslant N$) as:

$$\boldsymbol{\mathcal{W}}^{[n]} = \underbrace{\sum_{j_1=n}^{N}\sum_{j_2=n-1}^{j_1-1}\cdots\sum_{j_{n-1}=2}^{j_{n-2}-1}}_{(n-1)\text{sums}}\boldsymbol{\mathcal{W}}^{[n]}_{1:j_{n-1}:\ldots:j_1} \tag{29}$$

The term $\boldsymbol{\mathcal{W}}^{[n]}_{1:j_{n-1}:\ldots:j_1}$ denotes the interactions across the layers $1, j_{n-1}, \ldots, j_1$. The N^{th} order approximation becomes:

$$G(\boldsymbol{z}) = \boldsymbol{\beta} + \sum_{n=1}^{N}\Big\{\Big(\sum_{j_1=n}^{N}\sum_{j_2=n-1}^{j_1-1}\cdots\sum_{j_{n-1}=2}^{j_{n-2}-1}\Big(\boldsymbol{\mathcal{W}}^{[n]}_{1:j_{n-1}:\ldots:j_1}\Big)_{(1)}\Big)\Big(\bigodot_{m=1}^{n}\boldsymbol{z}\Big)\Big\} \tag{30}$$

By considering the mode-1 unfoding of Coupled CP decomposition (like in Section 2.2), we obtain:

$$\begin{aligned}G(\boldsymbol{z}) = \boldsymbol{\beta} + \boldsymbol{C}\sum_{n=1}^{N}\Big\{\sum_{j_1=n}^{N}\sum_{j_2=n-1}^{j_1-1}\cdots\sum_{j_{n-1}=2}^{j_{n-2}-1}\Big(\boldsymbol{U}_{[j_1]} \odot \ldots \odot \boldsymbol{U}_{[j_{n-1}]} \odot \boldsymbol{U}_{[1]}\Big)^T\Big(\bigodot_{m=1}^{n}\boldsymbol{z}\Big)\Big\} = \\ \boldsymbol{\beta} + \boldsymbol{C}\sum_{n=1}^{N}\Big\{\sum_{j_1=n}^{N}\sum_{j_2=n-1}^{j_1-1}\cdots\sum_{j_{n-1}=2}^{j_{n-2}-1}\Big(\Big(\boldsymbol{U}^T_{[j_1]}\boldsymbol{z}\Big) * \ldots * \Big(\boldsymbol{U}^T_{[j_{n-1}]}\boldsymbol{z}\Big) * \Big(\boldsymbol{U}^T_{[1]}\boldsymbol{z}\Big)\Big)\Big\} = \boldsymbol{\beta} + \boldsymbol{C}\boldsymbol{x}_N\end{aligned} \tag{31}$$

where we use $\boldsymbol{x}_N$ as an abbreviation of the sums. In the last equation, we have used Lemma 2 (Section B).

Claim 4. *The N^{th} order approximation of (30) can be implemented with a neural network as described in Alg. 1.*

Proof. We will use induction to prove the Claim. For $N = 2$, it trivially holds, while the proof for $N = 3$ is provided in Claim 1. Suppose it holds for N^{th} order approximation; we prove below that it holds for $N + 1^{th}$ order approximation.

Let us denote the approximation of (30) as $G_N(\boldsymbol{z})$. The $(N+1)^{th}$ order approximation from (30) is:

$$G_{N+1}(\boldsymbol{z}) = \boldsymbol{\beta} + \sum_{n=1}^{N+1} \Bigg\{ \Bigg(\sum_{j_1=n}^{N+1} \sum_{j_2=n-1}^{j_1-1} \cdots \sum_{j_{n-1}=2}^{j_{n-2}-1} \Big(\boldsymbol{\mathcal{W}}^{[n]}_{1:j_{n-1}:\ldots:j_1} \Big)_{(1)} \Bigg) \Bigg(\bigodot_{m=1}^{n} \boldsymbol{z} \Bigg) \Bigg\} =$$

$$\boldsymbol{\beta} + \sum_{n=1}^{N} \Bigg\{ \Bigg(\sum_{j_1=n}^{N} \sum_{j_2=n-1}^{j_1-1} \cdots \sum_{j_{n-1}=2}^{j_{n-2}-1} \Big(\boldsymbol{\mathcal{W}}^{[n]}_{1:j_{n-1}:\ldots:j_1} \Big)_{(1)} \Bigg) \Bigg(\bigodot_{m=1}^{n} \boldsymbol{z} \Bigg) + \tag{32}$$

$$\sum_{j_2=n-1}^{N} \cdots \sum_{j_{n-1}=2}^{j_{n-2}-1} \Big(\boldsymbol{\mathcal{W}}^{[n]}_{1:j_{n-1}:\ldots:j_2:(N+1)} \Big)_{(1)} \Big(\bigodot_{m=1}^{n} \boldsymbol{z} \Big) \Bigg\} + \Big(\boldsymbol{\mathcal{W}}^{[N+1]}_{1:2:\ldots:(N+1)} \Big)_{(1)} \Big(\bigodot_{m=1}^{N+1} \boldsymbol{z} \Big)$$

In the last equation, the first term in the sums is $\boldsymbol{x}_N$; for the rest two terms we apply Lemma 2:

$$G_{N+1}(\boldsymbol{z}) = \boldsymbol{\beta} + \boldsymbol{C}\boldsymbol{x}_N + \boldsymbol{C} \Big\{ \Big(\boldsymbol{U}^T_{[N+1]} \boldsymbol{z} \Big) * \Big(\boldsymbol{U}^T_{[N]} \boldsymbol{z} \Big) * \ldots * \Big(\boldsymbol{U}^T_{[2]} \boldsymbol{z} \Big) * \Big(\boldsymbol{U}^T_{[1]} \boldsymbol{z} \Big) \Big\} +$$

$$\boldsymbol{C} \sum_{n=1}^{N} \Bigg\{ \sum_{j_2=n-1}^{N} \cdots \sum_{j_{n-1}=2}^{j_{n-2}-1} \Bigg(\Big(\boldsymbol{U}^T_{[N+1]} \boldsymbol{z} \Big) * \Big(\boldsymbol{U}^T_{[j_2]} \boldsymbol{z} \Big) * \ldots * \Big(\boldsymbol{U}^T_{[j_{n-1}]} \boldsymbol{z} \Big) * \Big(\boldsymbol{U}^T_{[1]} \boldsymbol{z} \Big) \Bigg) \Bigg\} = \tag{33}$$

$$\boldsymbol{\beta} + \boldsymbol{C}\boldsymbol{x}_N + \boldsymbol{C} \Big\{ \Big(\boldsymbol{U}^T_{[N+1]} \boldsymbol{z} \Big) * \Bigg[\Big(\boldsymbol{U}^T_{[N]} \boldsymbol{z} \Big) * \ldots * \Big(\boldsymbol{U}^T_{[2]} \boldsymbol{z} \Big) * \Big(\boldsymbol{U}^T_{[1]} \boldsymbol{z} \Big) + \boldsymbol{\lambda} \Bigg] \Big\}$$

where

$$\boldsymbol{\lambda} = \sum_{n=1}^{N} \sum_{j_2=n-1}^{N} \cdots \sum_{j_{n-1}=2}^{j_{n-2}-1} \Bigg(\Big(\boldsymbol{U}^T_{[j_2]} \boldsymbol{z} \Big) * \ldots * \Big(\boldsymbol{U}^T_{[j_{n-1}]} \boldsymbol{z} \Big) * \Big(\boldsymbol{U}^T_{[1]} \boldsymbol{z} \Big) \Bigg) \tag{34}$$

The term $\boldsymbol{\lambda}$ is equal to the $\kappa = (n-1)^{th}$ order of (31), while there is only a single term for $n = N$. Therefore, (33) is transformed into:

$$G_{N+1}(\boldsymbol{z}) = \boldsymbol{\beta} + \boldsymbol{C}\boldsymbol{x}_N + \boldsymbol{C} \Big\{ \Big(\boldsymbol{U}^T_{[N+1]} \boldsymbol{z} \Big) * \boldsymbol{x}_N \Big\} \tag{35}$$

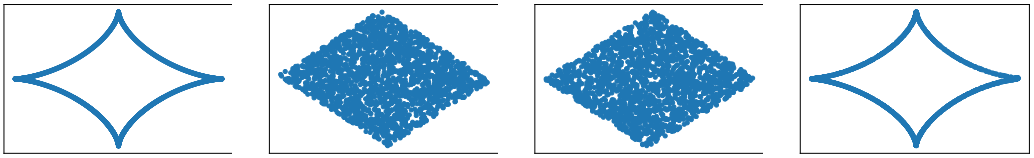
which is exactly the form described by Alg. 1. This concludes the induction proof.

□

D EXPERIMENTS ON SURFACES

Astroid: We implement a superellipse with parametric expression $[\alpha \cos^3 t, \alpha \sin^3 t]$ for $t \in [-\alpha, \alpha]$. This has a more complex distribution and four sharp edges. The random samples are visualized in Figure 7. PolyGAN models the data distribution accurately in contrast to the two baselines.

We conduct three experiments in which the data distribution is analytically derived. The experiments are:

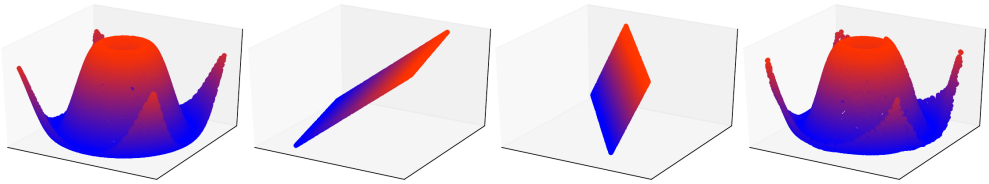


(a) GT (b) Orig (c) Concat (d) PolyGAN

Figure 7: Synthesized data for learning the 'astroid' signal. No activation functions are used in the generators.

Sin3D: The data manifold is an extension over the 2D manifold of the sinusoidal experiment (Section 4.1). The function we want to learn is $G(\boldsymbol{z}) : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ with the data manifold described by the vector $[x, y, \sin(10 * \sqrt{x^2 + y^2})]$ for $x, y \in [-0.5, 0.5]$.

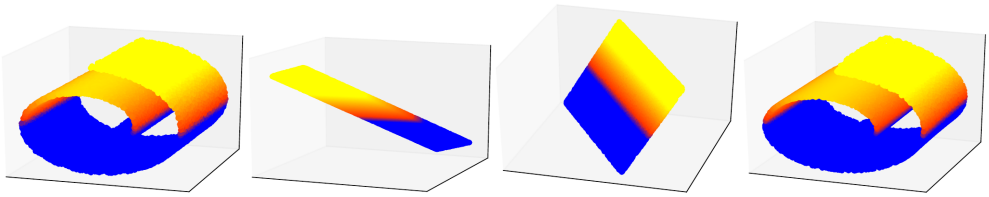
In Figure 8, $20,000$ samples are sampled from the generators and visualized. PolyGAN captures the data distribution, while 'Orig' and 'Concat' fail.



(a) GT (b) Orig (c) Concat (d) PolyGAN

Figure 8: Experiment on 3D synthetic data. From left to right: (a) the data distribution, (b) 'Orig', (c) 'Concat', (d) PolyGAN. As expected, the 'Orig' and the 'Concat' cannot capture the data distribution.

Swiss roll: The three dimensional vector $[t \cdot \sin t, y, t \cdot \cos t] + 0.05 \cdot s$ for $t, y \in [0, 1]$ and $s \sim \mathcal{N}(0, 1)$ forms the data manifold[2]. In Figure 9, $20,000$ samples are visualized.



(a) GT (b) Orig (c) Concat (d) PolyGAN

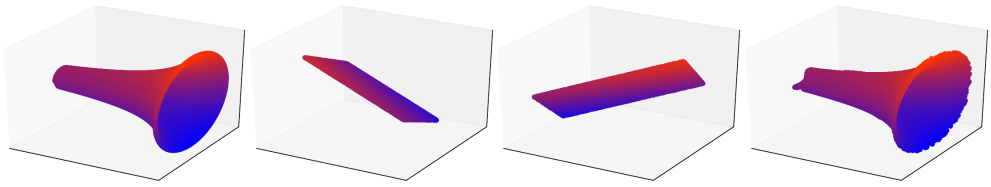
Figure 9: Experiment on 3D synthetic data ('swiss roll'). From left to right: (a) the data distribution, (b) 'Orig', (c) 'Concat', (d) PolyGAN.

Gabriel's Horn: The three dimensional vector $[x, \alpha \cdot \frac{\cos t}{x}, \alpha \cdot \frac{\sin t}{x}]$ for $t \in [0, 160\pi]$ and $x \in [1, 4]$ forms the data manifold. The dependence on both sinusoidal and the function $\frac{1}{x}$ makes this curve challenging for a polynomial expansion. In Figure 10, the synthesized samples are plotted. PolyGAN learns how to generate samples on the manifold despite the fraction in the parametric form.

E IMAGE GENERATION WITH LINEAR BLOCKS

Apart from digit generation (Section 4.2), we conduct two experiments on image generation of face and natural scenes. Since both distributions are harder than the digits one, we extend the

[2]This is a standard synthetic distribution in popular machine learning frameworks such as scikit-learn.



(a) GT (b) Orig (c) Concat (d) PolyGAN

Figure 10: Synthesized data on 'Gabriel's Horn'. From left to right: (a) the data distribution, (b) 'Orig', (c) 'Concat', (d) PolyGAN.

approximation followed on Section 4.2 by one order, i.e., we assume a fifth-order approximation. We emphasize that each block is a residual block with no activation functions.

Faces: In the experiment with faces, we utilize as the training samples the YaleB (Georghiades et al., 2001) dataset. The dataset includes greyscale images of faces under extreme illuminations. We rescale all of the images into 64×64 for our analysis.

Random samples are illustrated in Figure 11. Our method generates diverse images and captures the case of illuminating either half part of the face, while 'Orig' and 'Concat' generate images that have a dark side only on the left and right side, respectively. The difference becomes profound in the finer details of the face (please zoom in), where both baselines fail to synthesize realistic semantic parts of the face.

(a) GT (b) Orig (c) Concat (d) PolyGAN

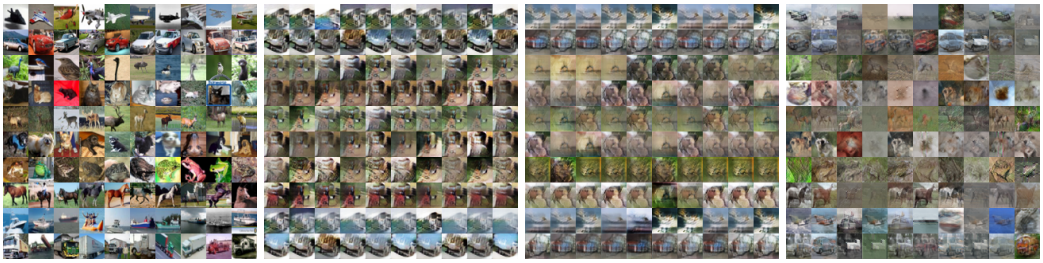
Figure 11: Image generation on faces (YaleB (Georghiades et al., 2001)) for a generator with linear blocks and a single activation function only on the output (i.e., $\tan h$). Notice that our method can illuminate either the left or right part of the face, in contrast to 'Orig' (and 'Concat') which generate images that have a dark side only on the left (respectively right) side. In addition, both 'Orig' and 'Concat' fail to capture the fine details of the facial structure (please zoom in for the details).

Natural scenes: We further evaluate the generation of natural images, specifically by training on CIFAR10 (Krizhevsky et al., 2014). CIFAR10 includes $50,000$ training images of $32 \times 32 \times 3$ resolution.

In Table 3, we evaluate the standard metrics of Inception Score (IS) and Frechet Inception Distance (FID) (see more details for the metrics in section F). Our model outperforms both 'Orig' and 'Concat' by a considerable margin. In Figure 12, some random synthesized samples are presented.

Table 3: IS/FID scores on CIFAR10 (Krizhevsky et al., 2014) with linear blocks.

conditional SNGAN with linear blocks on CIFAR10		
Model	IS (↑)	FID (↓)
Orig	4.47 ± 0.21	156.67 ± 12.29
Concat	4.23 ± 0.37	188.08 ± 17.00
PolyGAN	$\mathbf{6.43 \pm 0.11}$	$\mathbf{53.50 \pm 2.71}$



(a) GT (b) Orig (c) Concat (d) PolyGAN

Figure 12: Conditional image generation on CIFAR10 for a generator with linear blocks and a single activation function. Our approach generates more realistic samples in comparison to the compared methods, where severe mode collapse also takes place.

F IMAGE GENERATION WITH ACTIVATION FUNCTIONS

To demonstrate the flexibility of the PolyGAN, we utilize three different popular generators. The three acrhitectures chosen are DCGAN (Radford et al., 2015), SNGAN (Miyato et al., 2018), and SAGAN (Zhang et al., 2019). Each original generator is converted into a polynomial expansion, while we use the non-linearities to boost the performance of the polynomial generator. The hyper-parameters are kept the same as the corresponding baseline. Algorithms 3 and 4 succinctly present the key differences of our approach compared to the traditional one (in the case of SNGAN, similarly for other architectures).

In addition to the baseline, we implement the most closely related alternative to our framework, namely instead of using the Hadamard operator as in Figure 3, we concatenate the noise with the feature representations at that block. The latter approach is frequently used in the literature (Berthelot et al., 2017; Brock et al., 2019) (referred as "Concat" in the paper). The number of the trainable parameters of the generators are reported in Table 13. Our method has only a minimal increase of the parameters, while the concatenation increases the number of parameters substantially.

To reduce the variance often observed during GAN training (Lucic et al., 2018; Odena et al., 2018), each reported score is averaged over 10 runs utilizing different seeds. The metrics we utilize are Inception Score (IS) (Salimans et al., 2016) and Frechet Inception Distance (FID) (Heusel et al., 2017).

Below, we perform an ablation study on Section F.2, and then present the experiments on unsupervised (Section F.3) and conditional image generation (Section F.4) respectively.

Datasets: We use CIFAR10 (Krizhevsky et al., 2014) and Imagenet (Russakovsky et al., 2015) as the two most widely used baselines for GANs:

- **CIFAR10** (Krizhevsky et al., 2014) includes $60,000$ images of 32×32 resolution. We use $50,000$ images for training and the rest for testing.
- **Imagenet** (Russakovsky et al., 2015) is a large scale dataset that includes over one million training images and $50,000$ validation images. We reshape the images to 128×128 resolution.

Baseline architectures: The architectures employed are:

- DCGAN (Radford et al., 2015), as implemented in `https://github.com/pytorch/examples/tree/master/dcgan`. This is a widely used baseline.
- SNGAN (Miyato et al., 2018), as implemented in `https://github.com/pfnet-research/sngan_projection`. SNGAN is a strong performing GAN that introduced a spectral normalization in the discriminator.
- SAGAN (Zhang et al., 2019), as implemented in `https://github.com/voletiv/self-attention-GAN-pytorch`. This is a recent network architecture that utilizes

the notion of self-attention (Wang et al., 2018) in a GAN setting, achieving impressive results on Imagenet (Russakovsky et al., 2015).

The default hyper-parameters are left unchanged. The aforementioned codes are used for reporting the results of both the baseline and our method to avoid any discrepancies, e.g. different frameworks resulting in unfair comparisons. The source code will be released to enable the reproduction of our results.

Evaluation metrics: The popular Inception Score (IS) (Salimans et al., 2016) and Frechet Inception Distance (FID) (Heusel et al., 2017) are used for the quantitative evaluation. Both scores extract feature representations from a pretrained classifier (in practice the Inception network (Szegedy et al., 2015)). Despite their shortcomings, IS and FID are widely used (Lucic et al., 2018; Creswell et al., 2018), since alternative metrics fail for generative models (Theis et al., 2016).

The Inception Score is defined as

$$\exp\left(\mathbb{E}_{\boldsymbol{x}\in P_\theta}[KL(p(y|\boldsymbol{x})\|p(y)]\right) \tag{36}$$

where $\boldsymbol{x}$ is a generated sample and $p(y|\mathbf{x})$ is the conditional distribution for labels y. The distribution $p(y)$ over the labels is approximated by $\frac{1}{M}\sum_{n=1}^{M} p(y|\boldsymbol{x}_n)$ for $\boldsymbol{x}_n$ generated samples. Following the methods in the literature (Miyato et al., 2018), we compute the inception score for $M = 5,000$ generated samples per run (10 splits for each run).

The Frechet Inception Distance (FID) utilizes feature representations from a pretrained network (Szegedy et al., 2015) and assumes that the distributions of these representations are Gaussian. Denoting the representations of real images as $\mathcal{N}(\boldsymbol{\mu}_r, \boldsymbol{C}_r)$ and the generated (fake) as $\mathcal{N}(\boldsymbol{\mu}_f, \boldsymbol{C}_f)$, FID is:

$$\|\boldsymbol{\mu}_r - \boldsymbol{\mu}_f\|_2^2 + trace\big(\boldsymbol{C}_r + \boldsymbol{C}_f - 2\big(\boldsymbol{C}_r\boldsymbol{C}_f\big)^{1/2}\big) \tag{37}$$

In the experiments, we use $M = 10,000$ to compute the mean and covariance of the real images and $M = 10,000$ synthesized samples for $\boldsymbol{\mu}_f, \boldsymbol{C}_r$.

For both scores the original tensorflow inception network weights are used; the routines of tensorflow.contrib.gan.eval are called for the metric evaluation.

F.1 Implementation details

We experimentally define that a (series of) affine transformation(s) on the input noise $\boldsymbol{z}$ are beneficial before using the transformed $\boldsymbol{z}$ for the Hadamard products.[3] These affine transformations are henceforth mentioned as *global transformations* on $\boldsymbol{z}$.

The implementation details for each network are the following:

- DCGAN: We use a global transformation followed by a RELU non-linearity. WThe rest details remain the same as the baseline model.
- SNGAN: Similarly to DCGAN, we use a global transformation with a RELU non-linearity.

We consider each residual block as one order of approximation and compute the Hadamard product after each block (see algorithm 4).

F.2 Ablation study on CIFAR10 with non-linear generators

We conduct an ablation study based on SNGAN architecture (or our variant of SNGAN-poly), since most recent methods are based on similar generators Zhang et al. (2019); Brock et al. (2019). Unless explicitly mentioned otherwise, the SNGAN is trained on CIFAR10 for unsupervised image generation.

[3] A similar transformation is performed in other GAN architectures, such as in Karras et al. (2019).

Algorithm 3: Original SNGAN generator.

Input : Noise $\boldsymbol{z} \in \mathbb{R}^{128}$, $\phi = \text{RELU}$
Output : $\boldsymbol{x} \in \mathbb{R}^{32 \times 32 \times 3}$

;
;
% fully-connected layer for reshaping.
$\boldsymbol{h}$ = ϕ(Linear($\boldsymbol{z}$)) % dims out: $4 \times 4 \times 256$.;
;
;
;
for *n=1:3* **do**
    % resnet blocks.
    $\boldsymbol{h}$ = resblock($\boldsymbol{h}$)
    % dims out: $(4 \cdot 2^n) \times (4 \cdot 2^n) \times 256$.;
    ;
    ;
    ;
end
$\boldsymbol{x}$ = tanh(Conv($\boldsymbol{h}$)) % dims out: $32 \times 32 \times 3$.

Algorithm 4: Modified SNGAN-poly.

Input : Noise $\boldsymbol{z} \in \mathbb{R}^{128}$, $\phi = \text{RELU}$
Output : $\boldsymbol{x} \in \mathbb{R}^{32 \times 32 \times 3}$

% global transformation of $\boldsymbol{z}$.
$\boldsymbol{v}$ = ϕ(Linear($\boldsymbol{z}$))
% fully-connected layer for reshaping.
$\boldsymbol{h}$ = ϕ(Linear($\boldsymbol{v}$)) % dims out: $4 \times 4 \times 256$.
% perform a hadamard product here.
$\boldsymbol{v}_0 = (\boldsymbol{A}_{[0]})^T \cdot \boldsymbol{v}$
$\boldsymbol{h} = \boldsymbol{h} * \boldsymbol{v}_0$ % dims out: $4 \times 4 \times 256$.
for *n=1:3* **do**
    % resnet blocks.
    $\boldsymbol{h}$ = resblock($\boldsymbol{h}$)
    % dims out: $(4 \cdot 2^i) \times (4 \cdot 2^i) \times 256$.
    % reshape $\boldsymbol{v}$ for hadamard product.
    $\boldsymbol{v}_n = (\boldsymbol{A}_{[n]})^T \cdot \boldsymbol{v}$
    $\boldsymbol{h} = \boldsymbol{h} * \boldsymbol{v}_n$
end
$\boldsymbol{x}$ = tanh(Conv($\boldsymbol{h}$)) % dims out: $32 \times 32 \times 3$.

Table 4: The algorithm on the left describes the SNGAN generator. The algorithm on the right preserves the resnet blocks of the SNGAN generator, but converts it into a polynomial (named SNGAN-poly). The different lines are emphasized with blue color.

Global transformation validation: We add a global transformation on $\boldsymbol{z}$, i.e. a fully-connected layer and use the transformed noise as input to the generator. In the first experiment, we evaluate whether to add a non-linear activation to the global transformation. The two alternatives are: i) with linear global transformation ('Ours-linear-global'), i.e. no non-linearity, and ii) with global transformation followed by a RELU non-linearity ('Ours-RELU-global').

Table 5: Global transformation validation on SNGAN. The first two results assess the addition of a non-linear activation function after the global transformation. The last two rows compare the addition of a global transformation on the original generator.

SNGAN on CIFAR10		
Model	IS (↑)	FID (↓)
Ours-linear-global	8.23 ± 0.10	18.85 ± 0.59
Ours-RELU-global	$\mathbf{8.30 \pm 0.09}$	$\mathbf{17.65 \pm 0.76}$
Original	8.06 ± 0.10	19.06 ± 0.50
Original-RELU-global	7.98 ± 0.27	37.61 ± 7.16

The first two results in Table 5 demonstrate that both metrics marginally improve when using a non-linear activation function. We add this global transformation with RELU on the original SNGAN. The results are reported in the last two rows of Table 5 (where the original is mentioned as 'Orig', while the alternative of adding a global transformation as 'Original-RELU-global').

Split $\boldsymbol{z}$ into chunks: The recent BigGAN of (Brock et al., 2019) performs hierarchical synthesis of images by splitting the latent vector $\boldsymbol{z}$ into one chunk per resolution (block). Each chunk is then concatenated into the respective resolution.

We scrutinize this splitting against our method; we split the noise $\boldsymbol{z}$ into $(k+1)$ non-overlapping chunks of equal size for performing k injections. The injection with splitting is mentioned as 'Inject-split' below. Our splitting deteriorates the scores on the task as reported in Table 6. It is possible that more elaborate splitting techniques, such as those in Brock et al. (2019) are beneficial.

Normalization before Hadamard product: In Karras et al. (2019) they normalize the transformed noise through ADAIN, while in Karras et al. (2018) they similarly perform a feature vector normalization.

Table 6: Ablation experiment on splitting the noise $\boldsymbol{z}$ into non-overlapping chunks for the injection.

SNGAN on CIFAR10		
Model	IS (↑)	FID (↓)
Original	8.06 ± 0.10	19.06 ± 0.50
Inject-split	7.75 ± 0.12	22.08 ± 0.98
Ours-RELU-global	$\mathbf{8.30 \pm 0.09}$	$\mathbf{17.65 \pm 0.76}$

We scrutinize a feature normalization on the baseline of 'Ours-RELU-global'. For each layer i we divide the $\boldsymbol{A}_{[i]}\boldsymbol{z}$ vector with its standard deviation. The variant with global transformation followed by RELU and normalization before the Hadamard product is called 'Ours-norm'. The results in Table 7 illustrate that normalization improves the metrics.

Table 7: Ablation experiment on normalizing the $\boldsymbol{A}_{[i]}\boldsymbol{z}$ vector before the Hadamard product.

SNGAN on CIFAR10		
Model	IS (↑)	FID (↓)
Ours-RELU-global	8.30 ± 0.09	17.65 ± 0.76
Ours-norm	$\mathbf{8.37 \pm 0.11}$	$\mathbf{17.14 \pm 0.58}$

Skip the Hadamard product: Motivated by the skip connection of our Coupled CP decomposition, we add a skip connection to each Hadamard product. For instance, we modify the term $\left((\boldsymbol{A}_{[1]})^T\boldsymbol{z}\right) * \left((\boldsymbol{B}_{[1]})^T\boldsymbol{b}_{[1]}\right)$ into $\left((\boldsymbol{A}_{[1]})^T\boldsymbol{z}\right) * \left((\boldsymbol{B}_{[1]})^T\boldsymbol{b}_{[1]}\right) + (\boldsymbol{B}_{[1]})^T\boldsymbol{b}_{[1]}$.

In Table 8, we use 'Ours-RELU-global' as baseline against the model with the skip connection ('Ours-skip').

Table 8: Ablation experiment on adding a skip connection to each Hadamard product.

SNGAN on CIFAR10		
Model	IS (↑)	FID (↓)
Ours-RELU-global	8.30 ± 0.09	$\mathbf{17.65 \pm 0.76}$
Ours-skip	$\mathbf{8.43 \pm 0.11}$	21.54 ± 1.59

Since we use SNGAN both for unsupervised/conditional image generation, we verify the aforementioned results in the *conditional setting*, i.e. when the class information is also provided to the generator and the discriminator.

Normalization before Hadamard product: Similarly to the experiment above, for each layer i we divide the $\boldsymbol{A}_{[i]}\boldsymbol{z}$ vector with its standard deviation. The quantitative results in Table 9 improve the IS score, but the FID deteriorates.

Skip the Hadamard product: Similarly to the aforementioned unsupervised case, we assess the performance if we add a skip connection in the Hadamard. In Table 10, the quantitative results comparing the baseline and the skip case are presented.

F.3 Unsupervised image generation

In this experiment, we study the image generation problem without any labels or class information for the images. The architectures of DCGAN and resnet-based SNGAN are used for image generation in CIFAR10 (Krizhevsky et al., 2014). Table 11 summarizes the results of the IS/FID scores of the compared methods. In all of the experiments, PolyGAN outperforms the compared methods.

F.4 Conditional image generation

Frequently class information is available. We can utilize the labels, e.g. use conditional batch normalization or class embeddings, to synthesize images conditioned on a class. We train two networks,

Table 9: Ablation experiment (conditional GAN setting) on normalizing the $\boldsymbol{A}_{[i]}\boldsymbol{z}$ vector before the Hadamard product.

conditional SNGAN on CIFAR10		
Model	IS (↑)	FID (↓)
Ours-RELU-global	8.66 ± 0.14	$\mathbf{13.52 \pm 0.60}$
Ours-norm	$\mathbf{8.76 \pm 0.11}$	15.40 ± 1.29

Table 10: Ablation experiment (conditional GAN setting) on adding a skip connection to each Hadamard product.

conditional SNGAN on CIFAR10		
Model	IS (↑)	FID (↓)
Ours-RELU-global	8.66 ± 0.14	$\mathbf{13.52 \pm 0.60}$
Ours-skip	$\mathbf{8.77 \pm 0.10}$	13.62 ± 0.69

i.e., SNGAN (Miyato et al., 2018) in CIFAR10 (Krizhevsky et al., 2014) and SAGAN (Zhang et al., 2019) in Imagenet (Russakovsky et al., 2015). SAGAN uses self-attention blocks (Wang et al., 2018) to improve the resnet-based generator.

Despite our best efforts to show that our method is both architecture and database agnostic, the recent methods are run for hundreds of thousands or even million iterations till "convergence". In SAGAN the authors report that for each training multiple GPUs need to be utilized for weeks to reach the final reported Inception Score. We report the metrics for networks that are run with batch size 64 (i.e., four times less than the original 256) to fit in a single 16GB NVIDIA V100 GPU. Following the current practice in ML, due to the lack of computational budget (Hoogeboom et al., 2019), we run SAGAN for $400,000$ iterations (see Figure 3 of the original paper for the IS during training)[4]. Each such experiment takes roughly 6 days to train. The FID/IS scores of our approach compared against the baseline method can be found in Table 12. In both cases, our proposed method yields a higher Inception Score and a lower FID.

G EXPERIMENTAL MODEL COMPARISON

An experimental comparison of the two models described in Section 2 is conducted below. Unless explicitly mentioned otherwise, the networks used below do not include any non-linear activation functions, they are polynomial expansions with linear blocks. We use the following four experiments:

Sinusoidal on 2D: The data distribution is described by $[x, \sin(x)]$ with $x \in [0, 2\pi]$ (see Section 4.1 for further details). We assume 8^{th} order approximation for Coupled CP decomposition and 12^{th} order for Coupled nested CP decomposition. Both have width 15 units. The comparison between the two models in Figure 13 demonstrates that they can both capture the data manifold. Impressively, the Coupled CP decomposition does not synthesize a single point that is outside of the manifold.

Astroid: The data distribution is described on Section D. The samples comparing the two models are visualized in Figure 15.

Sin3D: The data distribution is described on Section D. In Figure 15 the samples from the two models are illustrated.

Swiss roll: The data distribution is described on Section D. In Figure 16 the samples from the two models are illustrated.

Digit generation: We conduct an experiment on images to verify that both architectures can learn higher-dimensional distributions. We select the digit images as described in Section 4.2. In this case, Coupled CP decomposition is implemented as follows: each $\boldsymbol{U}_{[i]}$ is a series of linear convolutions with stride 2 for $i = 1, \ldots, 4$, while $\boldsymbol{C}$ is a linear residual block. We emphasize that in both models

[4]Given the batch size difference, our training corresponds to roughly the $100,000$ steps of the authors' reported results.

Table 11: IS/FID scores on CIFAR10 (Krizhevsky et al., 2014) utilizing DCGAN (Radford et al., 2015) and SNGAN (Miyato et al., 2018) architectures for unsupervised image generation. Each network is run for 10 times and the mean and standard deviation are reported. In both cases, inserting block-wise noise injections to the generator (i.e., converting to our proposed PolyGAN) results in an improved score. Higher IS / lower FID score indicate better performance.

DCGAN		
Model	IS (↑)	FID (↓)
Orig	6.25 ± 0.06	47.29 ± 2.06
Concat	6.03 ± 0.06	49.35 ± 2.17
PolyGAN	$\mathbf{6.61 \pm 0.05}$	$\mathbf{42.86 \pm 1.02}$

SNGAN		
Model	IS (↑)	FID (↓)
Orig	8.06 ± 0.10	19.06 ± 0.50
Concat	8.28 ± 0.16	20.77 ± 2.91
PolyGAN	$\mathbf{8.30 \pm 0.09}$	$\mathbf{17.65 \pm 0.76}$

Table 12: Quantitative results on conditional image generation. We implement both SNGAN trained on CIFAR10 and SAGAN trained on Imagenet (for $400,000$ iterations). Each network is run for 10 times and the mean and variance are reported.

SNGAN (CIFAR10)		
Model	IS (↑)	FID (↓)
Orig	8.30 ± 0.11	14.70 ± 0.97
PolyGAN	$\mathbf{8.66 \pm 0.14}$	$\mathbf{13.52 \pm 0.60}$

SAGAN (Imagenet)		
Model	IS (↑)	FID (↓)
Orig	13.81 ± 0.21	138.20 ± 8.71
PolyGAN	$\mathbf{14.60 \pm 0.15}$	$\mathbf{84.37 \pm 6.37}$

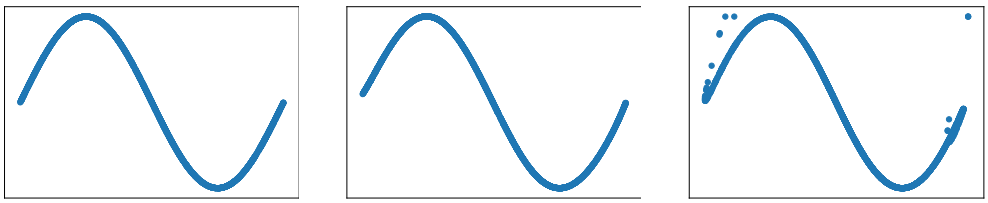
all the activation functions are removed and there is a single $tanh$ in the output of the generator for normalization purposes.

Table 13: Number of parameters for the generators of each approach and on various databases. As can be seen, our method only marginally increases the parameters while substantially improving the performance. On the other hand, "Concat" significantly increases the parameters without analogous increase in the performance.

DCGAN (CIFAR10)	
Model	Params
Orig	$3,573,440$
Concat	$6,416,448$
PolyGAN	$3,663,936$

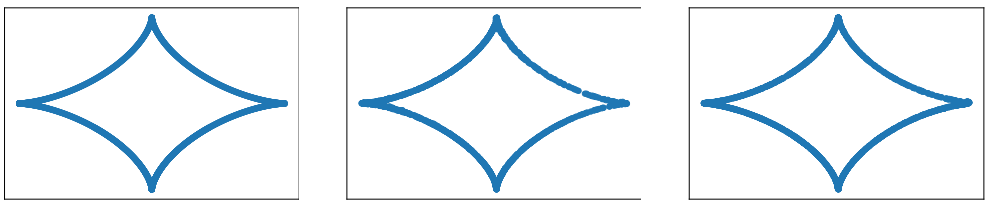
SNGAN (CIFAR10)	
Model	Params
Orig	$4,276,739$
Concat	$6,383,875$
PolyGAN	$4,408,835$

SAGAN (Imagenet)	
Model	Params
Orig	$42,079,300$
PolyGAN	$42,351,748$



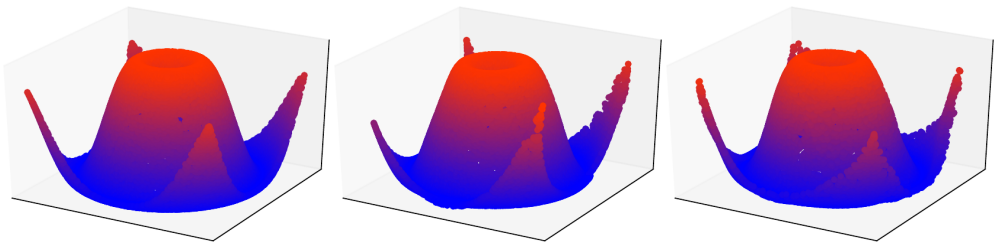
(a) GT (b) Coupled CP decomposition (c) Coupled nested CP decomposition

Figure 13: Synthesized data for learning the $[x, sin(x)]$ signal. No activation functions are used in the generators. From left to right: (a) the data distribution, (b) Coupled CP decomposition, (c) Coupled nested CP decomposition.



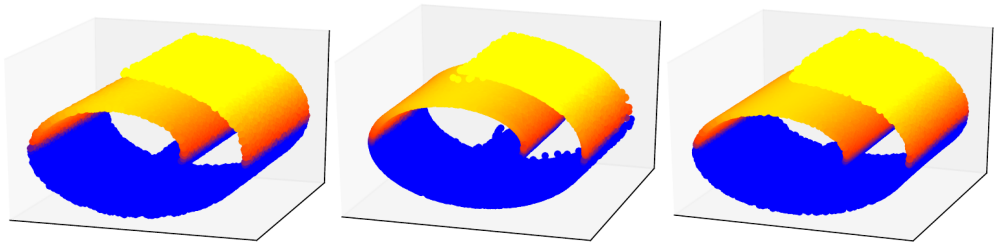
(a) GT (b) Coupled CP decomposition (c) Coupled nested CP decomposition

Figure 14: Synthesized data for learning the Astroid. Both models generate only plausible examples.



(a) GT (b) Coupled CP decomposition (c) Coupled nested CP decomposition

Figure 15: Experiment on 3D synthetic data. From left to right: (a) the data distribution, (b) Coupled CP decomposition, (c) Coupled nested CP decomposition.



(a) GT (b) Coupled CP decomposition (c) Coupled nested CP decomposition

Figure 16: Experiment on 3D synthetic data ('swiss roll'). From left to right: (a) the data distribution, (b) Coupled CP decomposition, (c) Coupled nested CP decomposition. Note that Coupled CP decompositiongenerates some noisy samples in contrast to Coupled nested CP decomposition.

(a) GT (b) Coupled CP decomposition (c) Coupled nested CP decomposition

Figure 17: Comparison of the two decompositions on digit generation.