# Supplementary of 'Π−nets: Deep Polynomial Neural Networks'

Grigorios G. Chrysos[1], Stylianos Moschoglou[1,2], Giorgos Bouritsas[1], Yannis Panagakis[3], Jiankang Deng[1,2], Stefanos Zafeiriou[1,2]
[1] Department of Computing, Imperial College London, UK
[2] Facesoft.io
[3] Department of Informatics and Telecommunications , University of Athens, GR
{[first letter].[surname]}@imperial.ac.uk

## 1. Introduction

The supplementary material of the paper 'Π−nets: Deep Polynomial Neural Networks' is organized as follows:

- The notation for the derivations is laid out in Sec. 2.
- The derivations for the third-order expansion are conducted in Sec. 3.
- The polynomial expansions from the recursive forms are derived in Sec. 3.
- In Sec. 4 additional details/experiments (on image generation with linear blocks) are conducted. In addition, a model comparison is performed in image generation.
- The training details of the linear classification are developed in Sec. 5.
- In Sec. 6, additional experiments on 3D meshes are presented.

**Details on image classification experiment**: Following standard practices, the following data augmentation techniques are performed during training: (1) normalization through mean RGB-channel subtraction, (2) random crop to $224 \times 224$, (3) scale from $5\%$ to $100\%$, (4) aspect ratio from $\frac{3}{4}$ to $\frac{4}{3}$, and (5) random horizontal flip. During inference, we use (1) normalization through mean RGB-channel subtraction, (2) scale to $256 \times 256$, and (3) single center crop to $224 \times 224$.

## 2. Notation

Tensors are symbolized by calligraphic letters, e.g., $\boldsymbol{\mathcal{X}}$, while matrices (vectors) are denoted by uppercase (lowercase) boldface letters e.g., $\boldsymbol{X}$, $(\boldsymbol{x})$. A set of $M$ real matrices (vectors) of varying dimensions is denoted by $\{\boldsymbol{X}_{[m]} \in \mathbb{R}^{I_m \times N}\}_{m=1}^{M}$ $(\{\boldsymbol{x}_{[m]} \in \mathbb{R}^{I_m}\}_{m=1}^{M})$.

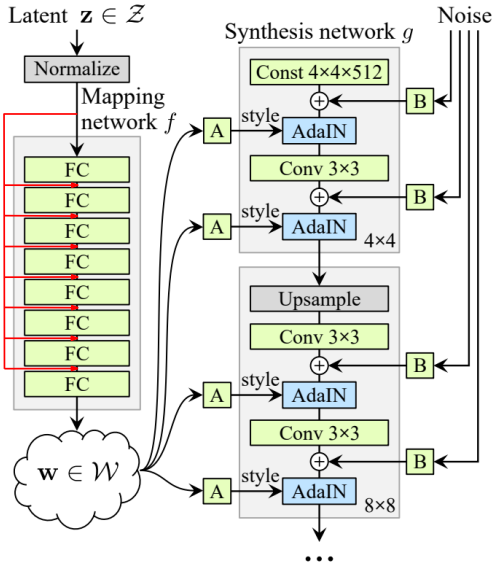



Figure 1: The modified version of StyleGAN [6], following our polynomial approach. We only slightly modify the mapping network to have the form of a polynomial. The synthesis network remains the same as in the original implementation. For a thorough description of the experiment, see Sec. 5.1 in the main paper.

**Products**: 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 *Khatri-Rao* 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 $(IJ) \times N$. For a set of matrices $\{\boldsymbol{A}_{[m]} \in \mathbb{R}^{I_m \times N}\}_{m=1}^{M}$ the Khatri-Rao product is denoted by:

$$\boldsymbol{A}_{[1]} \odot \boldsymbol{A}_{[2]} \odot \cdots \odot \boldsymbol{A}_{[M]} \doteq \bigodot_{m=1}^{M} \boldsymbol{A}_{[m]} \tag{1}$$

**Tensors**: Each element of an $M^{th}$ order tensor $\boldsymbol{\mathcal{X}}$ is addressed by $M$ indices, i.e., $(\boldsymbol{\mathcal{X}})_{i_1,i_2,\ldots,i_M} \doteq x_{i_1,i_2,\ldots,i_M}$. An

$M^{th}$-order real-valued tensor $\boldsymbol{\mathcal{X}}$ is defined over the tensor space $\mathbb{R}^{I_1 \times I_2 \times \cdots \times I_M}$, where $I_m \in \mathbb{Z}$ for $m = 1, 2, \ldots, M$. 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 $\boldsymbol{\mathcal{X}} \times_m \boldsymbol{u} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{m-1} \times I_{m+1} \times \cdots \times I_M}$, results in a tensor of order $M - 1$:

$$(\boldsymbol{\mathcal{X}} \times_m \boldsymbol{u})_{i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_M} = \sum_{i_m = 1}^{I_m} x_{i_1, i_2, \ldots, i_M} u_{i_m}. \tag{2}$$

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 *CP decomposition* [7, 10] factorizes a tensor into a sum of component rank-one tensors. The rank-$R$ CP decomposition of an $M^{th}$-order tensor $\boldsymbol{\mathcal{X}}$ is written as:

$$\boldsymbol{\mathcal{X}} \doteq [\![ \boldsymbol{U}_{[1]}, \boldsymbol{U}_{[2]}, \ldots, \boldsymbol{U}_{[M]} ]\!] = \sum_{r=1}^{R} \boldsymbol{u}_r^{(1)} \circ \boldsymbol{u}_r^{(2)} \circ \cdots \circ \boldsymbol{u}_r^{(M)}, \tag{3}$$

where $\circ$ denotes the vector outer product. The factor matrices $\{\boldsymbol{U}_{[m]} = [\boldsymbol{u}_1^{(m)}, \boldsymbol{u}_2^{(m)}, \cdots, \boldsymbol{u}_R^{(m)}] \in \mathbb{R}^{I_m \times R}\}_{m=1}^{M}$ 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 [7]:

$$\boldsymbol{X}_{(1)} \doteq \boldsymbol{U}_{[1]} \Big( \bigodot_{m=M}^{2} \boldsymbol{U}_{[m]} \Big)^T \tag{4}$$

More details on tensors and multilinear operators can be found in [7, 10].

The following lemma is useful in our method:

**Lemma 1.** *for a set of $N$ matrices $\{\boldsymbol{A}_{[\nu]} \mathbb{R}^{I_\nu \times K}\}_{\nu=1}^{N}$ and $\{\boldsymbol{B}_{[\nu]} \mathbb{R}^{I_\nu \times L}\}_{\nu=1}^{N}$, the following equality holds:*

$$(\bigodot_{\nu=1}^{N} \boldsymbol{A}_{[\nu]})^T \cdot (\bigodot_{\nu=1}^{N} \boldsymbol{B}_{[\nu]}) = (\boldsymbol{A}_{[1]}^T \cdot \boldsymbol{B}_{[1]}) * \ldots * (\boldsymbol{A}_{[N]}^T \cdot \boldsymbol{B}_{[N]}) \tag{5}$$

An indicative proof can be found in the Appendix of [3].

## 3. Model derivations

Below, we demonstrate how the recursive relationships written in the main paper can be derived from the polynomial. In all three cases, we assume a third-order approximation, i.e. $N = 3$, however this generalizes to higher order expansions. As a remainder, in the main paper, the models CCP, NCP, NCP-Skip are defined (Section 3) through their recursive relationships.

### 3.1. Third-order derivations for CCP

**Claim 1.** *The equation (3) (main paper) is a special format of a polynomial.*

*Proof.* To showcase the derivation and demonstrate the assumptions of this model, we perform a third-order expansion ($N = 3$) on the general equation:

$$\boldsymbol{x} = G(\boldsymbol{z}) = \sum_{n=1}^{N} \Big( \boldsymbol{\mathcal{W}}^{[n]} \prod_{j=2}^{n+1} \times_j \boldsymbol{z} \Big) + \boldsymbol{\beta} \tag{6}$$

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 $\boldsymbol{\mathcal{W}}^{[2]}$ 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: $\boldsymbol{W}_{(1)}^{[2]} = \boldsymbol{C}(\boldsymbol{U}_{[3]} \odot \boldsymbol{U}_{[1]})^T + \boldsymbol{C}(\boldsymbol{U}_{[2]} \odot \boldsymbol{U}_{[1]})^T$.
- Similarly, we enforce the third-order parameters tensor to admit the following CP decomposition (in matrix form) $\boldsymbol{W}_{(1)}^{[3]} = \boldsymbol{C}(\boldsymbol{U}_{[3]} \odot \boldsymbol{U}_{[2]} \odot \boldsymbol{U}_{[1]})^T$. 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:

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

Applying Lemma 1 on the last equation, we obtain:

$$\begin{aligned} G(\boldsymbol{z}) = \boldsymbol{\beta} + \boldsymbol{C}\Big\{ \Big(\boldsymbol{U}_{[3]}^T \boldsymbol{z}\Big) * \Big[ \Big(\boldsymbol{U}_{[2]}^T \boldsymbol{z}\Big) * \Big(\boldsymbol{U}_{[1]}^T \boldsymbol{z}\Big) + \\ \boldsymbol{U}_{[1]}^T \boldsymbol{z} \Big] + \Big(\boldsymbol{U}_{[2]}^T \boldsymbol{z}\Big) * \Big(\boldsymbol{U}_{[1]}^T \boldsymbol{z}\Big) + \boldsymbol{U}_{[1]}^T \boldsymbol{z} \Big\} \end{aligned} \tag{8}$$

The last equation is precisely the one that arises from the recursive relationship from equation (3) of the main paper. □

### 3.2. Third-order derivations for NCP

Contrary to CCP, where the interactions between layers are hardcoded, we let the interactions to be learned through a joint hierarchical decomposition on the polynomial parameters. Let us first introduce learnable hyper-parameters $\{\boldsymbol{b}_{[n]} \in \mathbb{R}^{\omega}\}_{n=1}^{N}$, which act as scaling factors for each parameter tensor. Therefore, we modify (6) to:

$$G(\boldsymbol{z}) = \sum_{n=1}^{N} \left( \boldsymbol{\mathcal{W}}^{[n]} \times_2 \boldsymbol{b}_{[N+1-n]} \prod_{j=3}^{n+2} \times_j \boldsymbol{z} \right) + \boldsymbol{\beta}, \quad (9)$$

with $\{\boldsymbol{\mathcal{W}}^{[n]} \in \mathbb{R}^{o \times \omega \times \prod_{m=1}^{n} \times_m d}\}_{n=1}^{N}$.

We assume a third-order approximation ($N = 3$) to demonstrate the derivation of NCP. To estimate its parameters we jointly factorize all parameters tensors by employing nested CP decomposition 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: $\boldsymbol{W}_{(1)}^{[2]} = \boldsymbol{C}\left\{ \boldsymbol{A}_{[3]} \odot \left[ \left( \boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]} \right) \boldsymbol{S}_{[3]} \right] \right\}^T$.
- Third order parameters: $\boldsymbol{W}_{(1)}^{[3]} = \boldsymbol{C}\left\{ \boldsymbol{A}_{[3]} \odot \left[ \left( \boldsymbol{A}_{[2]} \odot \left\{ \left( \boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]} \right) \boldsymbol{S}_{[2]} \right\} \right) \boldsymbol{S}_{[3]} \right] \right\}^T$

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) for $N = 3$ is written as:

$$\begin{aligned} G(\boldsymbol{z}) = {} & \boldsymbol{\beta} + \boldsymbol{C}(\boldsymbol{A}_{[3]} \odot \boldsymbol{B}_{[3]})^T (\boldsymbol{z} \odot \boldsymbol{b}_{[3]}) + \\ & \boldsymbol{C}\left\{ \boldsymbol{A}_{[3]} \odot \left[ \left( \boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]} \right) \boldsymbol{S}_{[3]} \right] \right\}^T \left( \boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[2]} \right) + \\ & \boldsymbol{C}\left\{ \boldsymbol{A}_{[3]} \odot \left[ \left( \boldsymbol{A}_{[2]} \odot \left\{ \left( \boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]} \right) \boldsymbol{S}_{[2]} \right\} \right) \boldsymbol{S}_{[3]} \right] \right\}^T \boldsymbol{\mu} \end{aligned} \quad (10)$$

with $\boldsymbol{\mu} = \left( \boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[1]} \right)$.

In Claim 2 and Claim 3, we prove that (10) is equivalent to the recursive equation in the main paper.

**Claim 2.** *Let*

$$\boldsymbol{\omega} = \left( \boldsymbol{A}_{[2]}^T \boldsymbol{z} \right) * \left\{ \boldsymbol{B}_{[2]}^T \boldsymbol{b}_{[2]} + \boldsymbol{S}_{[2]}^T \left[ \left( \boldsymbol{A}_{[1]}^T \boldsymbol{z} \right) * \left( \boldsymbol{B}_{[1]}^T \boldsymbol{b}_{[1]} \right) \right] \right\} \quad (11)$$

*It holds that*

$$\begin{aligned} \boldsymbol{\omega} = {} & \left\{ \boldsymbol{A}_{[2]} \odot \left[ \left( \boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]} \right) \boldsymbol{S}_{[2]} \right] \right\}^T \left( \boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[1]} \right) + \\ & \left( \boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]} \right)^T \left( \boldsymbol{z} \odot \boldsymbol{b}_{[2]} \right) \end{aligned} \quad (12)$$

*Proof.* We will prove the equivalence starting from (11) and transform it into (12). From (11):

$$\begin{aligned} & \boldsymbol{\omega} = \left( (\boldsymbol{A}_{[2]})^T \boldsymbol{z} \right) * \left( (\boldsymbol{B}_{[2]})^T \boldsymbol{b}_{[2]} \right) + \\ & \left( (\boldsymbol{A}_{[2]})^T \boldsymbol{z} \right) * \left\{ (\boldsymbol{S}_{[2]})^T \left[ \left( (\boldsymbol{A}_{[1]})^T \boldsymbol{z} \right) * \left( (\boldsymbol{B}_{[1]})^T \boldsymbol{b}_{[1]} \right) \right] \right\} = \\ & \left( \boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]} \right)^T \left( \boldsymbol{z} \odot \boldsymbol{b}_{[2]} \right) + \\ & \left( (\boldsymbol{A}_{[2]})^T \boldsymbol{z} \right) * \left\{ \left[ \left( \boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]} \right) \boldsymbol{S}_{[2]} \right]^T \left( \boldsymbol{z} \odot \boldsymbol{b}_{[1]} \right) \right\} \end{aligned} \quad (13)$$

where in the last equation, we have applied Lemma 1. Applying the Lemma once more in the last term of (13), we obtain (12). □

**Claim 3.** *Let*

$$\boldsymbol{\lambda} = \boldsymbol{\beta} + \boldsymbol{C}\left\{ \left( (\boldsymbol{A}_{[3]})^T \boldsymbol{z} \right) * \left[ (\boldsymbol{B}_{[3]})^T \boldsymbol{b}_{[3]} + (\boldsymbol{S}_{[3]})^T \boldsymbol{\omega} \right] \right\} \quad (14)$$

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

*Proof.* Transforming (14) into (10):

$$\begin{aligned} & \boldsymbol{\lambda} = \boldsymbol{\beta} + \boldsymbol{C}\left\{ \left( (\boldsymbol{A}_{[3]})^T \boldsymbol{z} \right) * \left( (\boldsymbol{B}_{[3]})^T \boldsymbol{b}_{[3]} \right) + \right. \\ & \left. \left( (\boldsymbol{A}_{[3]})^T \boldsymbol{z} \right) * \left( (\boldsymbol{S}_{[3]})^T \boldsymbol{\omega} \right) \right\} = \\ & \boldsymbol{\beta} + \boldsymbol{C}\left\{ \left( \boldsymbol{A}_{[3]} \odot \boldsymbol{B}_{[3]} \right)^T \left( \boldsymbol{z} \odot \boldsymbol{b}_{[3]} \right) + \right. \\ & \left. \left( (\boldsymbol{A}_{[3]})^T \boldsymbol{z} \right) * \left( (\boldsymbol{S}_{[3]})^T \boldsymbol{\omega} \right) \right\} \end{aligned} \quad (15)$$

To simplify the notation, we define $\boldsymbol{M}_1 = \boldsymbol{A}_{[2]} \odot \left[ \left( \boldsymbol{A}_{[1]} \odot \boldsymbol{B}_{[1]} \right) \boldsymbol{S}_{[2]} \right]$ and $\boldsymbol{M}_2 = \boldsymbol{A}_{[2]} \odot \boldsymbol{B}_{[2]}$. Then, $\boldsymbol{\omega} = \boldsymbol{M}_1^T \left( \boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{b}_{[1]} \right) + \boldsymbol{M}_2^T \left( \boldsymbol{z} \odot \boldsymbol{b}_{[2]} \right)$. The last term of (15) becomes:

$$\left((\boldsymbol{A}_{[3]})^T\boldsymbol{z}\right) * \left((\boldsymbol{S}_{[3]})^T\boldsymbol{\omega}\right) =$$
$$\left((\boldsymbol{A}_{[3]})^T\boldsymbol{z}\right) * \left[\left(\boldsymbol{M}_1\boldsymbol{S}_{[3]}\right)^T\left(\boldsymbol{z}\odot\boldsymbol{z}\odot\boldsymbol{b}_{[1]}\right)+\right.$$
$$\left.\left(\boldsymbol{M}_2\boldsymbol{S}_{[3]}\right)^T\left(\boldsymbol{z}\odot\boldsymbol{b}_{[2]}\right)\right] = \tag{16}$$
$$\left[\boldsymbol{A}_{[3]}\odot\left(\boldsymbol{M}_1\boldsymbol{S}_{[3]}\right)\right]^T\left(\boldsymbol{z}\odot\boldsymbol{z}\odot\boldsymbol{z}\odot\boldsymbol{b}_{[1]}\right)+$$
$$\left[\boldsymbol{A}_{[3]}\odot\left(\boldsymbol{M}_2\boldsymbol{S}_{[3]}\right)\right]^T\left(\boldsymbol{z}\odot\boldsymbol{z}\odot\boldsymbol{b}_{[2]}\right)$$

Replacing (16) into (15), we obtain (10). □

Note that the $\boldsymbol{\lambda}$ in Claim 3 (i.e. (14)) is the output of the recursive relationship in the main paper. Therefore, we have illustrated how the polynomial of (9) results in a network with the specified recursive relationship for third-order expansion.

### 3.3. Third-order derivations for NCP-Skip

The derivation and assumptions of NCP-Skip resemble those of NCP. Specifically, the recursive form of NCP-Skiphas an additional 'skip connection', i.e. the term $\boldsymbol{x}_{n-1}$ in (3) of the main paper. Therefore, the derivation is quite similar to that of Sec. 3.2 and can be trivially completed.

## 4. Image generation with linear blocks

The experiments conducted with linear blocks are described in this Section. Unless mentioned otherwise, **we do not use any activation functions other than the output hyperbolic tangent ($tanh$) in this Section**. These experiments are the proof of concept for the proposed formulation, i.e. they demonstrate how even without activation functions, the performance of the generator is decent in distributions, like digit or greyscale image generation.

The generators below are trained with an adversarial loss, i.e. in a GAN setting [5], since generative models have demonstrated impressive generation performance.

**Implementation details**: The discriminator as well as the hyper-parameters, e.g. optimization options, are similar to [9]. We modify the number of generated (fake) samples in each iteration to 512, since the generator does not have activation functions; we reduce the number of discriminator steps per iteration.

### 4.1. Unsupervised digit generation

The distribution of natural images is not known analytically and learning methods typically have access to few samples (typically in the order of thousands or millions). Despite the lack of an analytic formula, images of digits can be expressed as a polynomial function of the input noise in a generator as showcased in this experiment.

The training dataset is the popular MNIST [8]. The following two baselines are considered: (a) 'Concat': we replace the Hadamard operator with concatenation (used frequently in recent methods, such as in [2]), (b) 'Orig': the Hadamard products are ditched. For our method, we implement the following two variants: (a) NCP: a single polynomial using the decomposition NCP (Sec. 3.1 in the main paper), (b) ProdPoly: a product of polynomials, where each polynomial uses NCP. NCP has the following structure: each $\boldsymbol{S}_{[n]}$ is implemented as a convolution, while we assume a $12^{th}$ order approximation. For ProdPoly, we assume the following structure of polynomials: $1^{st}$ order, $3^{rd}$ order and $12^{th}$ order (the last one is essentially the aforementioned NCP).

In Fig. 2, samples from the compared cases are visualized. We verify that the baselines do not work without the activation functions, while the two proposed models can synthesize digits. Note that the product of polynomials performs considerably better than the single polynomial; therefore, we will use the product of polynomials in the experiments below.

To assess the intraclass variation, we learn a regularized logistic regressor with HOG features on the MNIST training set. The synthesized images are visualized per class in Fig. 3. The outcomes exhibit considerable variation, e.g. in style and strokes. To our knowledge, the polynomial generation of such images has not been demonstrated previously (especially without activation functions in the generator).

### 4.2. Unsupervised generation on greyscale images

To further assess the polynomial expansion, we extend the method to synthesize greyscale images. Specifically, we use the fashion images of [11] and [4] as the training distributions. We implement a baseline, as described on Sec. 4.1 by ditching the Hadamard operators. The two variants of our method are the NCP (single polynomial) and ProdPoly (with each polynomial of type NCP). The implementations remain similar to those in Sec. 4.1.

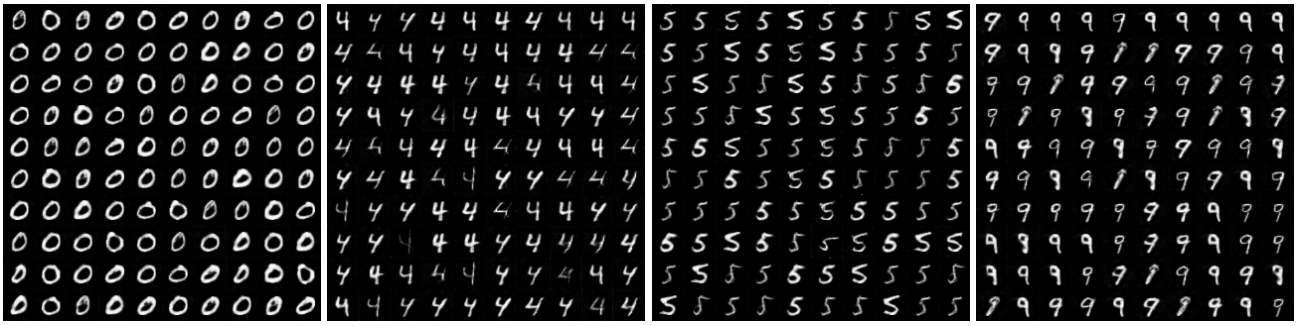
In Fig. 4 we visualize some random samples from the fashion images. As typically done in GAN, we perform a linear interpolation in the latent space of the trained model and visualize the results in Fig. 5. The images showcase how linear operations on the latent space result in plausible outcomes in the data space.

### 4.3. Model comparison

Three different decompositions are proposed in the main manuscript (Sec. 3.1); we conduct an evaluation of the three models in the setting of Sec. 3.2 (product of polynomials). That is, each model below is a product of polynomials; the base for each respective model is one of the three decomposi-

(a) GT (b) Orig (c) Concat (d) NCP (e) ProdPoly

Figure 2: Unsupervised digit generation samples from different methods with linear blocks.



(a) 0 (b) 4 (c) 5 (d) 9

Figure 3: Digit generation from ProdPoly. Samples from different classes are visualized to assess the intraclass variation. Even though the classes are not provided during training (unsupervised learning), the method synthesizes digits with different style and strokes.

tions, i.e. CCP, NCP, NCP-Skip. To evaluate the expressivity of each model, we train each without activation functions, i.e. similarly to Sec. 4.

The three models are originally compared in fashion image generation. The outcomes, visualized in Fig. 8, demonstrate similar generation properties.

In Fig. 9, samples of the three models (in the product of polynomials case) are synthesized; all three models can generate faces without activation functions. The three models share similar generation quality, which confirms the experiment on fashion-mnist.

## 5. Classification with linear blocks

The details of the experiment conducted in Sec. 4.2 are provided here. Specifically, in Table 1 the numerical results are provided, while the algorithm for the proposed residual block is described in Alg. 1. The training details are the following: Each method is trained for 120 epochs with batch size 128. The SGD optimizer is used with initial learning rate of 0.1. The learning rate is multiplied with a factor of 0.1 in epochs $40, 60, 80, 100$.

Table 1: Quantitative results on linear classification, i.e. the activation functions are removed from ResNet. The symbol $N$ abbreviates the order of each residual block, while 'Acc' abbreviates the accuracy.

| # blocks | $N$ | Acc CIFAR10 | Acc CIFAR100 |
|---|---|---|---|
| $[1,1,1,1]$ | 2 | $0.876 \pm 0.003$ | $0.626 \pm 0.004$ |
| | 3 | $0.870 \pm 0.003$ | $0.616 \pm 0.003$ |
| | 4 | $0.868 \pm 0.002$ | $0.609 \pm 0.002$ |
| | 5 | $0.864 \pm 0.001$ | $0.606 \pm 0.003$ |
| $[2,2,2,2]$ | 2 | $0.907 \pm 0.003$ | $0.667 \pm 0.003$ |
| | 3 | $0.891 \pm 0.001$ | $0.648 \pm 0.002$ |
| | 4 | $0.877 \pm 0.003$ | $0.626 \pm 0.004$ |
| | 5 | $0.856 \pm 0.006$ | $0.598 \pm 0.007$ |

## 6. Higher Order Polynomials for 3D Mesh Representation Learning

In this Section, the effect of higher order expressions on the task of 3D mesh representation learning is dis-

(a) GT (b) Orig (c) NCP (d) ProdPoly

Figure 4: Generation of fashion images [11] with linear blocks.

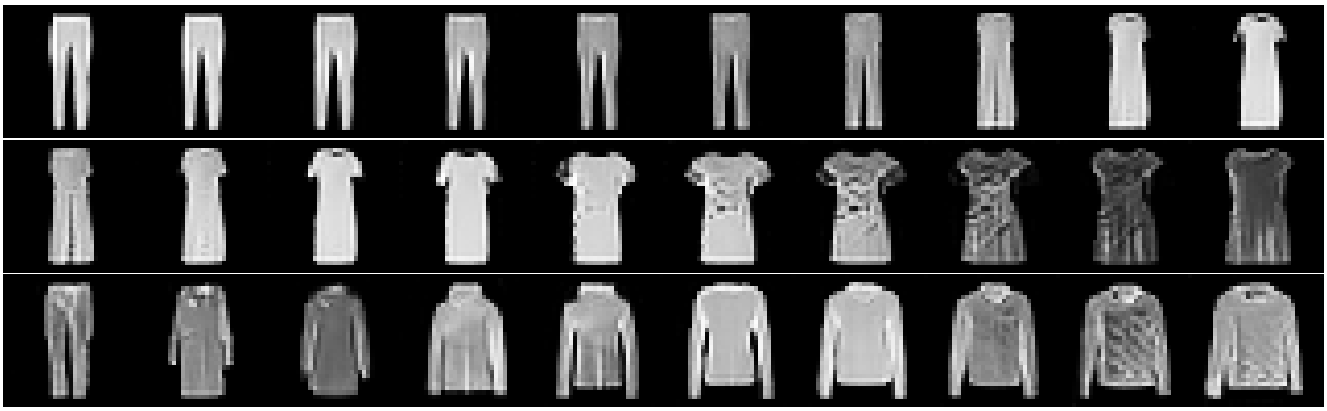


Figure 5: Linear interpolation in the latent space of ProdPoly (when trained on fashion images [11]). All the images are synthesized; the image on the leftmost column is the source, while the one in the rightmost is the target image.

cussed. The formulation relies on products of polynomials (each polynomial is based on NCP). In particular, each layer is a $N^{th}$ order polynomial given by the recursive expression $\boldsymbol{x}_n = \left(\boldsymbol{A}_{[n]}^T \boldsymbol{x}_1\right) * \left(\boldsymbol{S}_{[n]}^T \boldsymbol{x}_{n-1} + \boldsymbol{B}_{[n]}^T \boldsymbol{b}_{[n]}\right)$, $\boldsymbol{x} = \boldsymbol{C}\boldsymbol{x}_N + \boldsymbol{\beta}$. Following the convention of the main paper, we examine a special case of the above equation, where only existing building blocks of the baseline architecture are re-used. In particular:

- $2^{nd}$ order: $\boldsymbol{x}_2 = \left(\boldsymbol{S}^T \boldsymbol{x}_1\right) * \left(\boldsymbol{S}^T \boldsymbol{x}_1\right) + \boldsymbol{S}^T \boldsymbol{x}_1$, $\boldsymbol{x} = \boldsymbol{x}_2 + \boldsymbol{\beta}$, where $\boldsymbol{S}$ is the spiral convolution operator written in a matrix form. We set $\boldsymbol{A}_{[2]} = \boldsymbol{S}_{[2]} = \boldsymbol{S}$, $\boldsymbol{B}_{[2]}^T \boldsymbol{b}_{[2]} = \mathbb{1}_k$: a vector where each element is equal to 1, $\boldsymbol{C} = \boldsymbol{I}$ and $\boldsymbol{S}$ is the spiral convolution operator written in a matrix form.

- $3^{rd}$ order: $\boldsymbol{x}_3 = \left(\boldsymbol{S}^T \boldsymbol{x}_1\right) * \left(\boldsymbol{S}^T \boldsymbol{x}_1\right) * \left(\boldsymbol{S}^T \boldsymbol{x}_1\right) + \left(\boldsymbol{S}^T \boldsymbol{x}_1\right) * \left(\boldsymbol{S}^T \boldsymbol{x}_1\right) + \boldsymbol{S}^T \boldsymbol{x}_1$, $\boldsymbol{x} = \boldsymbol{x}_3 + \boldsymbol{\beta}$. We omit unfolding the recursion of the original equation for simplicity.

- $N^{th}$ order: Defined inductively, as above.

A vertex-wise instance normalization is applied on each higher-order term. In Tab. 2, the reconstruction error of the auto-encoder architecture when using up to $4^{th}$ order polynomial expansions (with or without activation functions between each layer) are summarized. In Fig. 10 we qualitatively assess the resulting reconstructions on an exemplary mesh, by plotting a color coding of the per vertex reconstruction error. The illustrations depict how higher-order information reduces the error and eliminates the areas with large reconstruction error.

## References


[1] Giorgos Bouritsas, Sergiy Bokhnyak, Stylianos Ploumpis, Michael Bronstein, and Stefanos Zafeiriou. Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation. In *IEEE Proceedings of International Conference on Computer Vision (ICCV)*, 2019. 8

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis.

(a) GT (b) Orig (c) PolyGAN-CNN (d) ProdPoly

Figure 6: Generation of facial images [4] with linear blocks.

Figure 7: Linear interpolation in the latent space of ProdPoly (when trained on facial images [4]). All the images are synthesized; the image on the leftmost column is the source, while the one in the rightmost is the target image.

| | error (mm) |
|---|---|
| ProdPoly ($2^{nd}$) | 0.530 |
| ProdPoly ($2^{nd}$ - linear) | 0.529 |
| ProdPoly ($3^{rd}$) | 0.502 |
| ProdPoly ($3^{rd}$ - linear) | 0.502 |
| ProdPoly ($4^{th}$) | **0.497** |
| ProdPoly ($4^{th}$ - linear) | 0.522 |

Table 2: ProdPoly with higher-order graph learnable operators.

**Algorithm 1:** The $t^{th}$ residual block with order $N$. In the linear blocks, $\phi$ represents the identity function. The abbreviations 'BN' and 'IN' stand for batch and instance normalization respectively.

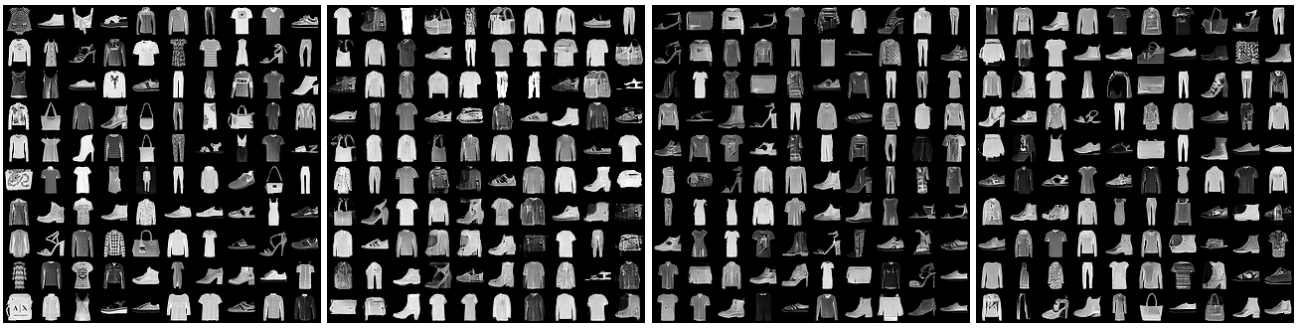
**Input** : $\boldsymbol{x}_t, N, \phi, a$
**Output** : $\boldsymbol{x}_{t+1}$

$\boldsymbol{o} \leftarrow \boldsymbol{x}_t$;
% (Original) convolutions.;
**for** *n=1:2* **do**
  $\boldsymbol{o} \leftarrow \phi(\text{BN}(\text{conv}(\boldsymbol{o})))$;
**end**
$\boldsymbol{s} \leftarrow \boldsymbol{o}$;
% Add the higher-order terms.;
**for** *n=1:N* **do**
  $\boldsymbol{\tau} \leftarrow \boldsymbol{o} * \boldsymbol{x}_t$;
  $\boldsymbol{o} \leftarrow \boldsymbol{\tau}$;
  $\boldsymbol{s} \leftarrow \boldsymbol{s} + IN(\boldsymbol{\tau})$;
**end**
$\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t + \boldsymbol{s}$


In *International Conference on Learning Representations (ICLR)*, 2019. 4

[3] Grigorios Chrysos, Stylianos Moschoglou, Yannis Panagakis, and Stefanos Zafeiriou. Polygan: High-order polynomial generators. *arXiv preprint arXiv:1908.06571*, 2019. 2

[4] 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. 4, 7, 8

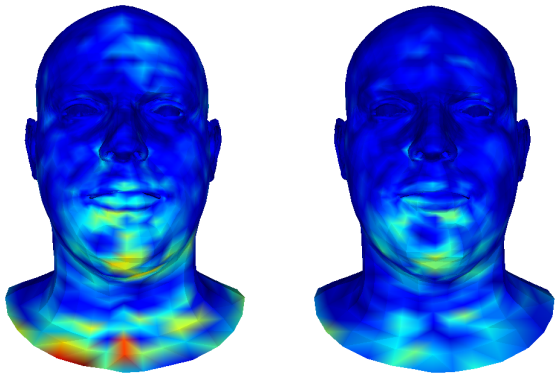


(a) GT (b) ProdPoly - CCP (c) ProdPoly - NCP (d) ProdPoly - NCP-Skip

Figure 8: Comparison of the proposed models in fashion image [11] generation with linear blocks.

(a) GT (b) ProdPoly - CCP (c) ProdPoly - NCP (d) ProdPoly - NCP-Skip

Figure 9: Comparison of the proposed models in facial image [4] generation with linear blocks.



(a) [1] (b) $2^{nd}$ order - NCP

Figure 10: A color coding of the per vertex reconstruction error on an exemplary mesh. The values closer to blue indicate a smaller reconstruction error, while areas closer to red indicate a larger error.


[5] 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. 4

[6] 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. 1

[7] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009. 2

[8] 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. 4

[9] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018. 4

[10] 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. 2

[11] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 4, 6, 8