Learning Polynomial Activation Functions for Deep Neural Networks

Linghao Zhang [*] **Jiawang Nie** [†] **Tingting Tang** [‡]

Abstract

Activation functions are crucial for deep neural networks. This novel work frames the problem of training neural network with learnable polynomial activation functions as a polynomial optimization problem, which is solvable by the Moment-SOS hierarchy. This work represents a fundamental departure from the conventional paradigm of training deep neural networks, which relies on local optimization methods like backpropagation and gradient descent. Numerical experiments are presented to demonstrate the accuracy and robustness of optimum parameter recovery in presence of noises.

1 Introduction

One essential component for deep neural networks (DNN) is the selection of activation functions, such as sigmoid, ReLU, and their numerous variants [6, 19, 29, 31]. In recent years, there is a growing interest in designing adaptive or learnable activation functions to enhance deep neural networks [1, 2]. Among many options, polynomial activation functions offer a particularly appealing choice: they are convenient to analyze and have a variety of applications, including image generation [4, 5, 12], face detection [11], and fuzzy modeling [27, 32]. Moreover, when the activation is parameterized as a polynomial with learnable coefficients, the network's input-output map becomes a polynomial function. This allows tools from algebraic geometry to be used for a rigorous analysis of network properties, including the network's expressive power or the complexity of its training process. Such work offers a new geometric perspective for understanding deep learning [13, 15].

The key novelty of this paper arises from the following realization: with polynomial activation functions whose coefficients are to be learned, the entire training process of a deep neural network can be formulated as a nonconvex polynomial optimization problem (POP). This powerful structural insight allows us to move beyond the limitations of traditional gradient-based optimization. While classical methods like stochastic gradient descent (SGD) often guarantees convergence to stationary points, the POP formulation opens the door to global optimization techniques.

There are efficient computational methods for solving polynomial optimization. The core technique is the hierarchy of Moment-SOS relaxations. It transforms a nonconvex problem into a linear convex optimization problem with the moment cone that can be solved by a hierarchy of semidefinite programming relaxations. The dual of the moment cone is the nonnegative polynomial cone, which can be approximated by sum of squares (SOS) type representations. The sequence of these primal-dual pairs of relaxations is the called Moment-SOS hierarchy. An appealing property of this hierarchy is that almost all polynomial optimization problems can be solved exactly, for both their optimal values and optimizers, by applying Moment-SOS relaxations. We refer to [8, 16, 17, 22] for detailed

---

[*]Department of Mathematics, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, `liz010@ucsd.edu`

[†]Department of Mathematics, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, `njw@math.ucsd.edu`

[‡]Department of Mathematics and Statistics, San Diego State University Imperial Valley, 720 Heber Ave, Calexico, CA 92231, `ttang2@sdsu.edu`

introductions to moment and polynomial optimization. These methods have been successfully applied to control [14, 18, 28], data science optimization [20, 23, 25, 26, 33], tensor computation [7, 21, 24, 34], and machine learning tasks [3, 10].

In this paper, we consider feedforward neural networks in which the weight matrices are given, and all hidden layers apply polynomial activation functions whose coefficients are to be learned. We formulate the training process as a polynomial optimization problem and solve it by Moment-SOS relaxations.

For a D-hidden-layer neural network, let the input and output dimensions be m_0 and m_{D+1}. Denote by m_ℓ the number of hidden neurons in the ℓth hidden layer. For $\ell = 1, \ldots, D+1$, the weight matrices are $W_\ell \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$. For each hidden layer $\ell = 1, \ldots, D$, let $p_\ell : \mathbb{R}^1 \to \mathbb{R}^1$ be a polynomial activation function such that

$$p_\ell(t) = c_{\ell,d_\ell} t^{d_\ell} + \cdots + c_{\ell,1} t + c_{\ell,0}, \quad c_{\ell,0}, c_{\ell,1}, \ldots, c_{\ell,d_\ell} \in \mathbb{R}.$$

For uniqueness of the coefficient recovery, we usually normalize the coefficients such that $c_{1,0} = \cdots = c_{D-1,0} = 1$. The rest of the coefficients are collected in the vector

$$\mathbf{c} := (c_{1,1}, \ldots, c_{1,d_1}, \ldots, c_{D-1,1}, \ldots, c_{D-1,d_{D-1}}, c_{D,0}, c_{D,1}, \ldots, c_{D,d_D}). \tag{1.1}$$

Given an input $x \in \mathbb{R}^{m_0}$, define

$$h_0 := x, \quad v_\ell := W_\ell h_{\ell-1} \in \mathbb{R}^{m_\ell}, \quad h_\ell := p_\ell(v_\ell) \in \mathbb{R}^{m_\ell}, \quad \ell = 1, \ldots, D.$$

In the above, the activation function p_ℓ is applied element-wise. The network output is

$$f(x; \mathbf{c}) := W_{D+1} h_D \in \mathbb{R}^{m_{D+1}}.$$

Equivalently, if $P_\ell : \mathbb{R}^{m_{\ell-1}} \to \mathbb{R}^{m_\ell}$ denotes the ℓth layer operator such that

$$P_\ell(u) = \big(p_\ell(w_{\ell,1}^T u), \ldots, p_\ell(w_{\ell,m_\ell}^T u) \big)$$

with $w_{\ell,j}^T$ denoting the jth row of W_ℓ, then

$$f(x; \mathbf{c}) = W_{D+1} P_D(P_{D-1}(\cdots P_1(x))). \tag{1.2}$$

Given a training dataset of input-output pairs $\{(x_i, y_i)\}_{i=1}^N$, we consider the optimization problem

$$\min_{\mathbf{c}} \quad \mathcal{L}(\mathbf{c}) := \Big\| \frac{1}{N} \sum_{i=1}^N \big(f(x_i; \mathbf{c}) - y_i \big) \Big\|_\infty, \tag{1.3}$$

where $\mathcal{L}(\mathbf{c})$ is the loss function. The epigraph reformulation of (1.3) is

$$\begin{cases} \min_{\mathbf{c}, \theta} \quad \theta \\ \text{s.t.} \quad \Big\| \frac{1}{N} \sum_{i=1}^N \big(f(x_i; \mathbf{c}) - y_i \big) \Big\|_\infty \leq \theta. \end{cases} \tag{1.4}$$

Denote by $\vartheta_{\min}$ the minimum value of (1.4). In this paper, we aim at computing the global minimum value $\vartheta_{\min}$ and finding a global minimizer $(\mathbf{c}^*, \theta^*)$.

In this work, we propose a new approach for training polynomial activation functions for deep neural networks. Our main contributions are:

- We propose a training framework for selecting polynomial activation functions whose coefficients are to be learned.

- We formulate the neural network training as a polynomial optimization problem and solve it by the Moment-SOS hierarchy.

- We provide numerical experiments that demonstrate the efficiency and accuracy of learning activation function in presence of noises. For the noiseless case, our method can recover activation functions exactly.

This paper is organized as follows. Some basics on polynomial optimization are reviewed in Section 2. In Section 3, we give the Moment-SOS hierarchy of optimization problem (1.4) and prove convergence under certain assumptions. Numerical experiments are presented in Section 4. Some conclusions are drawn in Section 5.

2 Background

Denote by $\mathbb{N}$ the set of nonnegative integers and $\mathbb{R}$ the real field. For a positive integer m, denote $[m] := \{1, \ldots, m\}$. For a polynomial p, $\deg(p)$ denotes the total degree. The ring of polynomials in $z = (z_1, \ldots, z_n)$ with real coefficients is denoted as $\mathbb{R}[z]$. For a degree d, the subset of polynomials in $\mathbb{R}[z]$ with degrees at most d is denoted as $\mathbb{R}[z]_d$. For $p \in \mathbb{R}[z]$, ∇p denotes its gradient with respect to z, and $\nabla^2 p$ denotes its Hessian with respect to z. Given a power $\alpha = (\alpha_1, \ldots, \alpha_n)$, we denote the monomial $z^\alpha := z_1^{\alpha_1} \cdots z_n^{\alpha_n}$ and $|\alpha| := \alpha_1 + \cdots + \alpha_n$. Denote by e_i the standard basis vector such that the ith entry is 1 and 0 otherwise. The infinity norm of z is $\|z\|_\infty := \max_{1 \le i \le n} |z_i|$. The Euclidean norm of z is $\|z\|_2 := (|z_1|^2 + \cdots + |z_n|^2)^{1/2}$. For a symmetric matrix X, $X \succeq 0$ means X is positive semidefinite.

A polynomial $\sigma \in \mathbb{R}[z]$ is said to be a *sum of squares* (SOS) if there are polynomials $p_1, \ldots, p_s \in \mathbb{R}[z]$ such that $\sigma = p_1^2 + \cdots + p_s^2$. For example, the polynomial

$$z_1^4 - z_1^2(1 + 2z_1 z_2) + z_1^2 z_2^2 + 2z_1 z_2 + 1 = z_1^2 + (1 - z_1^2 + z_1 z_2)^2$$

is SOS. The cone of all SOS polynomials in $\mathbb{R}[z]$ is denoted as $\Sigma[z]$. For a tuple $g := (g_1, \ldots, g_s)$ of polynomials in $\mathbb{R}[z]$, the degree-$2k$ truncation of its quadratic module is the set ($g_0 := 1$)

$$\mathrm{QM}[g]_{2k} := \Big\{ \sum_{j=0}^{s} \sigma_j g_j : \sigma_j \in \Sigma[z], \deg(\sigma_j g_j) \le 2k \Big\}.$$

The set $\mathrm{QM}[g]_{2k}$ is convex and can be represented by semidefinite programming. We refer to Section 2.5 of [22] for more details.

For a power $\alpha = (\alpha_1, \ldots, \alpha_n)$, let $\mathbb{N}_d^n := \{\alpha \in \mathbb{N}^n : |\alpha| \le d\}$ be the set of monomial powers with degrees at most d. The symbol $\mathbb{R}^{\mathbb{N}_d^n}$ denotes the space of all real vectors labeled by $\alpha \in \mathbb{N}_d^n$. A vector w in $\mathbb{R}^{\mathbb{N}_d^n}$ is said to be a truncated multi-sequence (tms) of degree d. For a given $w \in \mathbb{R}^{\mathbb{N}_d^n}$, the *Riesz functional* generated by w is the linear functional $\mathscr{R}_w$ acting on $\mathbb{R}[z]_d$ such that

$$\mathscr{R}_w \Big(\sum_{\alpha \in \mathbb{N}_d^n} p_\alpha z^\alpha \Big) := \sum_{\alpha \in \mathbb{N}_d^n} p_\alpha w_\alpha.$$

For convenience, we denote $\langle p, w \rangle := \mathscr{R}_w(p)$ for $p \in \mathbb{R}[z]_d$. The *localizing matrix* of $p \in \mathbb{R}[z]$ generated by w is

$$L_p^{(k)}[w] := \mathscr{R}_w \big(p(z) \cdot [z]_{k_1} [z]_{k_1}^T \big), \quad k_1 := \lfloor k - \deg(p)/2 \rfloor.$$

For instance, when $n = 2$, $k = 2$, and $p = 1 - z_1^2 - z_2^2$,

$$L_p^{(2)}[w] = \begin{bmatrix} w_{00} - w_{20} - w_{02} & w_{10} - w_{30} - w_{12} & w_{01} - w_{21} - w_{03} \\ w_{10} - w_{30} - w_{12} & w_{20} - w_{40} - w_{22} & w_{11} - w_{31} - w_{13} \\ w_{01} - w_{21} - w_{03} & w_{11} - w_{31} - w_{13} & w_{02} - w_{22} - w_{04} \end{bmatrix}.$$

In particular, if $p = 1$ is the constant polynomial, we get the *moment matrix*

$$M_k[w] := L_1^{(k)}[w].$$

We refer to [16, 17, 22] for more detailed introductions to polynomial optimization.

3 The Moment-SOS Hierarchy

In this section, we give a hierarchy of Moment-SOS relaxations to solve (1.4), and discuss how to extract minimizers from the moment relaxations. For convenience of notation, we denote (**c** is defined in (1.1))

$$z := (\mathbf{c}, \theta) = (z_1, \ldots, z_n), \quad n = d_1 + \cdots + d_D + 2.$$

We rewrite the optimization problem (1.4) as follows:

$$\begin{cases} \min_{z \in \mathbb{R}^n} & z_n \\ \text{s.t.} & \|\hat{f}(z) - \hat{y}\|_\infty \le z_n. \end{cases} \tag{3.1}$$

In the above, $\hat{f}(z) := \frac{1}{N} \sum_{i=1}^{N} f(x_i; \mathbf{c})$ and $\hat{y} := \frac{1}{N} \sum_{i=1}^{N} y_i$. Observe that

$$\|\hat{f}(z) - \hat{y}\|_\infty \leq z_n \iff -z_n \leq \hat{f}_j(z) - \hat{y}_j \leq z_n, \quad j = 1, \ldots, m_{D+1}.$$

Define the polynomial tuple $g(z) := \big(g_1(z), , \ldots, g_{\hat{m}}(z)\big)$ such that

$$g(z) = \Big(z_n + (\hat{f}_1(z) - \hat{y}_1), z_n - (\hat{f}_1(z) - \hat{y}_1), \ldots,$$

$$z_n + (\hat{f}_{m_{D+1}}(z) - \hat{y}_{m_{D+1}}), z_n - (\hat{f}_{m_{D+1}}(z) - \hat{y}_{m_{D+1}})\Big),$$

where $\hat{m} = 2m_{D+1}$. Then, (3.1) is equivalent to

$$\begin{cases} \min\limits_{z \in \mathbb{R}^n} & z_n \\ \text{s.t.} & g_1(z) \geq 0, \ldots, g_{\hat{m}}(z) \geq 0. \end{cases} \tag{3.2}$$

The feasible set of (3.2) is

$$K = \{z \in \mathbb{R}^n : g_1(z) \geq 0, \ldots, g_{\hat{m}}(z) \geq 0\}.$$

Let $k_0 := \lceil \deg(g)/2 \rceil$. For a degree $k \geq k_0$, the kth order SOS-relaxation for (3.2) is

$$\begin{cases} \vartheta_{sos,k} := & \max \quad \gamma \\ & \text{s.t.} \quad z_n - \gamma \in \mathrm{QM}[g]_{2k}. \end{cases} \tag{3.3}$$

Its dual optimization problem is the kth order moment relaxation

$$\begin{cases} \vartheta_{mom,k} := & \min\limits_{w} \quad \langle z_n, w \rangle \\ & \text{s.t.} \quad L_{g_j}^{(k)}[w] \succeq 0, \ j = 1, \ldots, \hat{m}, \\ & \qquad M_k[w] \succeq 0, \ w_0 = 1, \ w \in \mathbb{R}^{\mathbb{N}_{2k}^n}. \end{cases} \tag{3.4}$$

Note that $\langle z_n, w \rangle = \mathscr{R}_w(z_n) = w_{e_n}$, the entry of w indexed by e_n. We refer to Section 2 for the above notation.

In computational practice, we start with relaxation order $k = k_0$. If the relaxation (3.4) is successfully solved, we extract its minimizers. Otherwise, we increase k by one and repeat this process. For $k = k_0, k_0 + 1, \ldots$, the sequence of primal-dual pairs (3.3)-(3.4) is called the Moment-SOS hierarchy. Recall that $\vartheta_{\min}$ denotes the minimum value of (1.4). Since (1.4) can be equivalently formulated as (3.1) and (3.2), $\vartheta_{\min}$ is also the minimum value for (3.1) and (3.2). For each relaxation order k, it holds that

$$\vartheta_{sos,k} \leq \vartheta_{mom,k} \leq \vartheta_{\min}. \tag{3.5}$$

Moreover, both sequences $\{\vartheta_{sos,k}\}_{k=1}^{\infty}$ and $\{\vartheta_{mom,k}\}_{k=1}^{\infty}$ are monotonically increasing. If the quadratic module $\mathrm{QM}[g]$ is archimedean [22, Section 2.6], then the Moment-SOS hierarchy of (3.3)-(3.4) has asymptotic convergence, i.e.,

$$\lim_{k \to \infty} \vartheta_{sos,k} = \lim_{k \to \infty} \vartheta_{mom,k} = \vartheta_{\min}.$$

Under some conditions, the Moment-SOS hierarchy has finite convergence, and we can extract minimizers of (3.1) from the moment relaxation (3.4). Suppose w^* is a minimizer of (3.4) for a relaxation order $k \geq k_0$. To obtain a minimizer of (3.1), we typically need the *flat truncation* condition: there exists an integer $d \in [k_0, k]$ such that

$$r := \mathrm{rank}\, M_d[w^*] = \mathrm{rank}\, M_{d-k_0}[w^*]. \tag{3.6}$$

Theorem 3.1. *[22] Suppose w^* is a minimizer of (3.4). If the flat truncation condition (3.6) holds, then the Moment-SOS relaxation is tight, i.e., $\vartheta_{\min} = \vartheta_{sos,k} = \vartheta_{mom,k}$.*

For the case $r = 1$, if w^* is a minimizer of (3.4), then $z^* = (w_{e_1}^*, \ldots, w_{e_n}^*)$ is a minimizer of (3.1). For the case $r > 1$, we refer to Section 5.3 of [22] for the extraction of minimizers.

There is a close relationship between the finite convergence theory of Moment-SOS hierarchy and local optimality conditions. Suppose u is a local minimizer of (3.2), and there exists a Lagrange multiplier $\lambda = (\lambda_1, \ldots, \lambda_{\hat{m}})$ satisfying the Karush-Kuhn-Tucker (KKT) conditions

$$\nabla z_n = e_n = \lambda_1 \nabla g_1(u) + \cdots + \lambda_{\hat{m}} \nabla g_{\hat{m}}(u), \tag{3.7}$$

$$g_j(u) \geq 0, \ \lambda_j \geq 0, \ \lambda_j g_j(u) = 0, \ j = 1, \ldots, \hat{m}. \tag{3.8}$$

For λ satisfying (3.7)-(3.8), the Lagrangian function is

$$\mathscr{L}(z) := z_n - \lambda_1 g_1(z) - \cdots - \lambda_{\hat{m}} g_{\hat{m}}(z). \tag{3.9}$$

Recall that a polynomial $\varphi(z)$ is SOS-convex if its Hessian $\nabla^2 \varphi(z) = H(z)H(z)^T$ for some matrix polynomial $H(z)$. We refer to Chapter 7 of [22] for SOS convex polynomials.

Theorem 3.2. *Let $\mathscr{L}(z)$ be the Lagrangian function for (3.2). Suppose $\mathscr{L}(z)$ is SOS-convex, then the Moment-SOS relaxation is tight for all $k \geq k_0$, i.e., $\vartheta_{sos,k} = \vartheta_{mom,k} = \vartheta_{\min}$.*

Proof. The Lagrangian function for (3.2) is (3.9) with Lagrange multipliers $\lambda_1, \ldots, \lambda_{\hat{m}} \geq 0$. Suppose z^* is the minimizer of (3.2). Then, we have $\mathscr{L}(z^*) = \vartheta_{\min}$ and $\nabla \mathscr{L}(z^*) = 0$. By the integration formula,

$$\mathscr{L}(z) = \mathscr{L}(z^*) + (z - z^*)\nabla \mathscr{L}(z^*) + \int_0^1 \int_0^t (z - z^*)^T \nabla^2 \mathscr{L}(s(z - z^*) + z^*)(z - z^*) \, \mathrm{d}s \mathrm{d}t$$

$$= \vartheta_{\min} + (z - z^*)^T \Big(\int_0^1 \int_0^t \nabla^2 \mathscr{L}(s(z - z^*) + z^*) \, \mathrm{d}s \mathrm{d}t \Big)(z - z^*).$$

Since $\mathscr{L}(z)$ is SOS-convex, the Hessian $\nabla^2 \mathscr{L}(z)$ is an SOS matrix polynomial, so $\sigma(z) := \mathscr{L}(z) - \vartheta_{\min}$ is an SOS polynomial. This implies

$$z_n - \vartheta_{\min} = \sigma(z) + \lambda_1 g_1(z) + \cdots + \lambda_{\hat{m}} g_{\hat{m}}(z) \in \mathrm{QM}[g]_{2k_0}.$$

Therefore, $\vartheta_{sos,k_0} \geq \vartheta_{\min}$. By (3.5) and the monotonicity of $\vartheta_{sos,k}$ and $\vartheta_{mom,k}$, we get the desired result. $\qquad\square$

Remark 3.3. *If the minimum value for (3.4) is zero, then the coefficient recovery for the activation functions is exact.*

Example 3.4. Consider a two-hidden-layer network with polynomial activation functions

$$p_1(t) = c_{1,2}t^2 + c_{1,1}t + 1, \quad p_2(t) = c_{2,1}t + c_{2,0}.$$

Let $m_0 = m_1 = m_2 = m_3 = 4$ and weight matrices

$$W_1 = \begin{bmatrix} 1 & 0 & -1 & 1 \\ 0 & 1 & 1 & 1 \\ -1 & 0 & 1 & -1 \\ -2 & 1 & -1 & 0 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 1 & -1 & 0 & 2 \\ 2 & 1 & 1 & 0 \\ 1 & 1 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \quad W_3 = \begin{bmatrix} 1 & 1 & 0 & 1 \\ -2 & -1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ -1 & 0 & -2 & 1 \end{bmatrix}.$$

Suppose the sample size $N = 2$ and let $\{(x_i, y_i)\}_{i=1}^N$ be the training data such that

$$x_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \\ -1 \end{bmatrix}, \quad y_1 = \begin{bmatrix} 66 \\ -22 \\ 106 \\ -104 \end{bmatrix}, \quad x_2 = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad y_2 = \begin{bmatrix} 38 \\ 30 \\ 46 \\ -33 \end{bmatrix}.$$

For $z := (\mathbf{c}, \theta) = (c_{1,1}, c_{1,2}, c_{2,0}, c_{2,1}, \theta)$, we have

$$\hat{f}(z) - \hat{y} = \begin{bmatrix} 3c_{2,0} + 9c_{2,1} + 29c_{1,2}c_{2,1} - 52 \\ 5c_{1,1}c_{2,1} - c_{2,0} + 5c_{1,2}c_{2,1} - 4 \\ 3c_{2,0} + 10c_{2,1} - c_{1,1}c_{2,1} + 41c_{1,2}c_{2,1} - 76 \\ \frac{5}{2}c_{1,1}c_{2,1} - 9c_{2,1} - 2c_{2,0} - \frac{73}{2}c_{1,2}c_{2,1} + \frac{137}{2} \end{bmatrix}.$$

To solve the optimization problem (3.1), we solve the moment relaxation (3.4). For relaxation order $k = 1$, the flat truncation condition (3.6) does not hold. However, (3.6) holds for $k = 2$, and we can extract the minimizer for (3.1):

$$\mathbf{c}^* = (1, -2, 1, -1), \quad \theta^* = 0.$$

4 Numerical experiments

This section provides numerical experiments for learning polynomial activation functions. All computations are implemented using MATLAB R2023b on a MacBook Pro equipped with Apple M1 Max processor and 16GB RAM. The semidefinite programs are solved by the software `GloptiPoly` [9], which calls the SDP package `SeDuMi` [30]. For neatness, only four decimal digits are displayed for computational results.

Normalization of coefficients. We remark that the normalization of coefficients is needed. To see this, consider a two-hidden-layer network with linear activation functions

$$p_1 = c_{1,1}t + c_{1,0}, \quad p_2(t) = c_{2,1}t + c_{2,0}.$$

Given any weight matrices W_1, W_2, W_3, for an input $x \in \mathbb{R}^{m_0}$, (1.2) gives the output

$$f(x; c_{1,0}, c_{1,1}, c_{2,0}, c_{2,1}) = c_{2,1}(c_{1,1}W_3W_2W_1x + c_{1,0}W_3W_2\mathbf{1}) + c_{2,0}W_3\mathbf{1},$$

where $\mathbf{1} = [1 \cdots 1]^T \in \mathbb{R}^{m_3}$. Note that

$$f(x; c_{1,0}, c_{1,1}, c_{2,0}, c_{2,1}) = f(x; \tau c_{1,0}, \tau c_{1,1}, c_{2,0}, \tau^{-1}c_{2,1})$$

for all scalar $\tau \neq 0$. So in general, we can normalize the coefficients such that $c_{1,0} = 1$. For $D > 2$ hidden layers, we can similarly normalize the coefficients such that $c_{\ell,0} = 1$ for all $\ell = 1, \ldots, D-1$.

Experiments on synthetic data. We illustrate our method on synthetic D-layer networks. For $\ell = 1, \ldots, D+1$, let $W_\ell \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$ be the randomly generated weight matrices, and let

$$p_\ell(t) = c_{\ell,d_\ell}t^{d_\ell} + \cdots + c_{\ell,1}t + c_{\ell,0}, \quad \ell = 1, \ldots, D,$$

be the polynomial activation functions. In the above, we normalize the coefficients such that $c_{1,0} = \cdots = c_{D-1,0} = 1$. We then randomly generate the rest of the coefficients $\mathbf{c}$ (defined in (1.1)), for which we denote $\mathbf{c}_{\text{true}}$. The training input-output data $\{(x_i, y_i)\}_{i=1}^N$ is generated as follows. We choose the input x_i randomly. The corresponding output y_i is obtained as (p_ℓ is applied element-wise with the coefficient vector $\mathbf{c}_{\text{true}}$)

$$h_0 := x_i, \quad v_\ell := W_\ell h_{\ell-1}, \quad h_\ell := p_\ell(v_\ell), \quad \ell = 1, \ldots, D, \quad y_i = W_{D+1}h_D + \varepsilon, \qquad (4.1)$$

where $\varepsilon = \texttt{1e-2*randn}(m_{D+1}, 1)$ is a random perturbing noise vector. We learn the activation polynomial coefficient vector $\mathbf{c}$ by solving Moment-SOS relaxations (3.3)-(3.4). We remark that for all generated instances, the learning problem (3.1) is solved successfully by the relaxation (3.4) with order $k = 3$. The learned coefficient vector $\mathbf{c}$ is denoted as $\mathbf{c}_{\text{pred}}$. The learning accuracy is assessed by the errors

$$\text{AbsErr} = \left\| \mathbf{c}_{\text{true}} - \mathbf{c}_{\text{pred}} \right\|_2, \quad \text{RelErr} = \left\| \mathbf{c}_{\text{true}} - \mathbf{c}_{\text{pred}} \right\|_2 / \left\| \varepsilon \right\|_2.$$

i) For the two-layer network with quadratic activation functions

$$p_1(t) = c_{1,2}t^2 + c_{1,1}t + 1, \quad p_2(t) = c_{2,2}t^2 + c_{2,1}t + c_{2,0},$$

the errors and consumed time for different dimensions are displayed in Table 1.

Table 1: Accuracy for 2-layer DNN with quadratic activations.

(N, m_0, m_1, m_2, m_3)	AbsErr	RelErr	Time	$\|\varepsilon\|_2$
$(50, 20, 20, 20, 20)$	0.0019	0.0452	5.3287	0.0430
$(100, 25, 25, 25, 25)$	0.0013	0.0295	5.9034	0.0485
$(150, 30, 30, 30, 30)$	0.0014	0.0265	7.0377	0.0521
$(200, 35, 35, 35, 35)$	0.0018	0.0323	6.3813	0.0557
$(250, 40, 40, 40, 40)$	0.0012	0.0201	7.9470	0.0598

ii) For the two-layer network with quadratic and cubic activation functions

$$p_1(t) = c_{1,2}t^2 + c_{1,1}t + 1, \quad p_2(t) = c_{2,3}t^3 + c_{2,2}t^2 + c_{2,1}t + c_{2,0},$$

the errors and consumed time for different dimensions are displayed in Table 2.

Table 2: Accuracy for 2-layer DNN with quadratic and cubic activations.

(N, m_0, m_1, m_2, m_3)	AbsErr	RelErr	Time	$\|\varepsilon\|_2$
$(50, 20, 20, 20, 20)$	0.0065	0.1332	17.2626	0.0480
$(100, 25, 25, 25, 25)$	0.0061	0.1203	20.7595	0.0502
$(150, 30, 30, 30, 30)$	0.0020	0.0381	22.8920	0.0513
$(200, 35, 35, 35, 35)$	0.0012	0.0197	24.5851	0.0579
$(250, 40, 40, 40, 40)$	0.0012	0.0182	26.3780	0.0609

Table 3: Accuracy for 3-layer DNN with linear activations.

$(N, m_0, m_1, m_2, m_3, m_4)$	AbsErr	RelErr	Time	$\|\varepsilon\|_2$
$(50, 20, 20, 20, 20, 20)$	0.0016	0.0368	0.9970	0.0442
$(100, 25, 25, 25, 25, 25)$	0.0012	0.0254	0.9767	0.0475
$(150, 30, 30, 30, 30, 30)$	0.0006	0.0114	1.1439	0.0549
$(200, 35, 35, 35, 35, 35)$	0.0011	0.0172	1.2455	0.0651
$(250, 40, 40, 40, 40, 40)$	0.0009	0.0151	1.3911	0.0598

iii) For the three-layer network with linear activation functions
$$p_1(t) = c_{1,1}t + 1, \quad p_2(t) = c_{2,1}t + 1, \quad p_3(t) = c_{3,1}t + c_{3,0},$$
the errors and consumed time for different dimensions are displayed in Table 3.

iv) For the three-layer network with linear and quadratic activation functions
$$p_1(t) = c_{1,2}t^2 + c_{1,1}t + 1, \quad p_2(t) = c_{2,1}t + 1, \quad p_3(t) = c_{3,1}t + c_{3,0},$$
the errors and consumed time for different dimensions are displayed in Table 4.

Table 4: Accuracy for 3-layer DNN with linear and quadratic activations.

$(N, m_0, m_1, m_2, m_3, m_4)$	AbsErr	RelErr	Time	$\|\varepsilon\|_2$
$(50, 20, 20, 20, 20, 20)$	0.0012	0.0295	2.8547	0.0422
$(100, 25, 25, 25, 25, 25)$	0.0055	0.1252	3.4459	0.0484
$(150, 30, 30, 30, 30, 30)$	0.0030	0.0540	3.8279	0.0553
$(200, 35, 35, 35, 35, 35)$	0.0009	0.0143	4.0465	0.0575
$(250, 40, 40, 40, 40, 40)$	0.0007	0.0122	4.3156	0.0628

Residual analysis for two-layer network. We consider a two-hidden-layer network with randomly generated weight matrices W_1, W_2, W_3. Suppose the polynomial activation functions are
$$p_1(t) = c_{1,3}t^3 + c_{1,2}t^2 + c_{1,1}t + 1, \quad p_2(t) = c_{2,2}t^2 + c_{2,1}t + c_{2,0}.$$
We randomly generate $\mathbf{c} := (c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2})$, for which we denote $\mathbf{c}_{\text{true}}$. The training input-output data $\{(x_i, y_i)\}_{i=1}^N$ is generated as follows. We choose the input x_i randomly, and the corresponding output y_i is obtained as in (4.1) with the coefficient vector $\mathbf{c}_{\text{true}}$. We learn the activation polynomial coefficient vector $\mathbf{c}$ by solving Moment-SOS relaxations (3.3)-(3.4). The learned coefficient vector $\mathbf{c}$ is denoted as $\mathbf{c}_{\text{pred}}$. To evaluate the quality of the learned $\mathbf{c}_{\text{pred}}$, we generate an independent test set $\{(\tilde{x}_i, \tilde{y}_i)\}_{i=1}^N$ in the same way as the training set. The predicted output y_i' is obtained as in (4.1) with the coefficient vector $\mathbf{c}_{\text{pred}}$ and $\varepsilon = 0$. We use two residual plots to illustrate the learning accuracy. First, for each $j = 1, \ldots, m_3$, we plot the jth entry of the residual $\varepsilon_i := y_i' - \tilde{y}_i$ against the index $i = 1, \ldots, N$ (the left plot in Figure 1). Second, we plot the norm $\|\varepsilon_i\|_2$ against the index i (the right plot in Figure 1). For neatness of the paper, we display these two residual plots only for $(N, m_0, m_1, m_2, m_3) = (50, 20, 20, 20, 20)$ in Figure 1. We can see that in both residual plots, the points are close to zeros and show no visible trends along the index i.

For different (N, m_0, m_1, m_2, m_3), we report in Table 5 the mean squared error (MSE) and root mean squared error (RMSE):
$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N \|\varepsilon_i\|_2^2, \quad \text{RMSE} = \sqrt{\text{MSE}}.$$
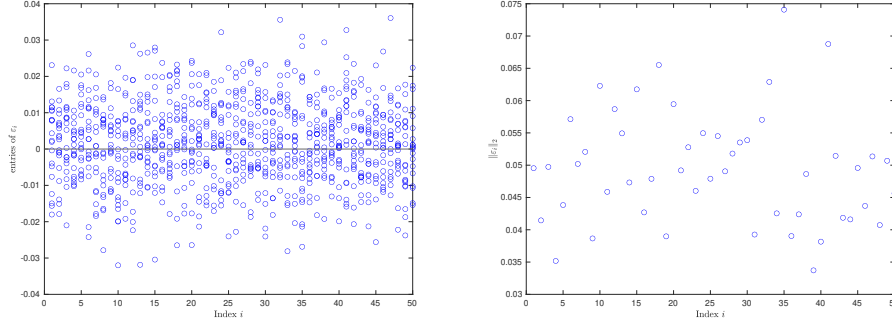
Figure 1: Plots for residual error ε_i with $i = 1, \ldots, 50$.

Table 5: Test errors for 2-layer DNN with cubic and quadratic activations.

(N, m_0, m_1, m_2, m_3)	MSE	RMSE
$(50, 20, 20, 20, 20)$	0.0090	0.0830
$(100, 25, 25, 25, 25)$	0.0026	0.0511
$(150, 30, 30, 30, 30)$	0.0046	0.0669
$(200, 35, 35, 35, 35)$	0.0078	0.0808
$(250, 40, 40, 40, 40)$	0.0133	0.1002

5 Concluding remarks

This paper proposes a new training framework for deep neural networks with learnable polynomial activation functions. This training process can be formulated as polynomial optimization, which is solvable by Moment-SOS relaxations. Numerical experiments demonstrate that our approach can learn activation polynomial coefficients exactly in absence of noises and remains robust under noises. This highlights the potential of polynomial optimization techniques in deep learning, providing new insights into the design and analysis of activation functions.

The Moment-SOS hierarchy is a powerful analytical tool for studying properties of deep neural networks. It provides globally optimal performance benchmarks and structural insights into the interplay between network weights and polynomial activation functions. The global optimality certification provided by the Moment-SOS hierarchy gives a promising direction to investigate hybrid methods, which combine gradient-based optimization methods. For example, backpropagation, with its unparalleled scalability, could be used for the bulk of the training, while the Moment-SOS hierarchy could be used to solve critical subproblems. Moreover, it could be applied to a single layer to find a globally optimal polynomial activation functions, which can be further used to initialize larger networks. It could also be used to design regularization terms that guide the gradient descent process toward a better basin of attraction.

Acknowledgments and Disclosure of Funding

References

[1] Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P. (2014). Learning activation functions to improve deep neural networks. arXiv preprint arXiv:1412.6830.

[2] Apicella, A., Donnarumma, F., Isgrò, F., Prevete, R. (2021). A survey on modern trainable activation functions. *Neural Networks*, 138, 14-32.

[3] Chen, T., Lasserre, J. B., Magron, V., Pauwels, E. (2021). Semialgebraic representation of monotone deep equilibrium models and applications to certification. *Advances in Neural Information Processing Systems*, 34, 27146-27159.

[4] Chrysos, G. G., Moschoglou, S., Bouritsas, G., Deng, J., Panagakis, Y., Zafeiriou, S. (2021). Deep polynomial neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 44(8), 4021-4034.

[5] Chrysos, G., Moschoglou, S., Panagakis, Y., Zafeiriou, S. (2019). Polygan: High-order polynomial generators. arXiv preprint arXiv:1908.06571.

[6] Glorot, X., Bordes, A., Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the 14th international conference on artificial intelligence and statistics* (pp. 315-323).

[7] Guo, B., Nie, J., Yang, Z. (2022). Learning diagonal Gaussian mixture models and incomplete tensor decompositions. *Vietnam Journal of Mathematics*, 50(2), 421-446.

[8] Henrion, D., Korda, M., Lasserre, J. B. (2020). *The Moment-sos Hierarchy*, World Scientific, Singapore.

[9] Henrion, D., Lasserre, J. B., Löfberg, J. (2009). GloptiPoly 3: moments, optimization and semidefinite programming. *Optimization Methods & Software*, 24(4-5), 761-779.

[10] Huang, L., Nie, J., Wang, J. (2025). Optimization over the weakly Pareto set and multi-task learning. arXiv preprint arXiv:2504.00257.

[11] Huang, L. L., Shimizu, A., Hagihara, Y., Kobatake, H. (2003). Face detection from cluttered images using a polynomial neural network. *Neurocomputing*, 51, 197-211.

[12] Karras, T., Laine, S., Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 4401-4410).

[13] Kileel, J., Trager, M., Bruna, J. (2019). On the expressive power of deep polynomial neural networks. *Advances in Neural Information Processing Systems*, 32.

[14] Korda, M., Henrion, D., Jones, C. N. (2014). Convex computation of the maximum controlled invariant set for polynomial control systems. *SIAM Journal on Control and Optimization*, 52(5), 2944-2969.

[15] Kubjas, K., Li, J., Wiesmann, M. (2024). Geometry of polynomial neural networks. *Algebraic Statistics*, 15(2), 295–328.

[16] Lasserre, J. B. (2015). An introduction to polynomial and semi-algebraic optimization (Vol. 52). *Cambridge University Press*.

[17] Laurent, M. (2009). Sums of squares, moment matrices and optimization over polynomials. In *Emerging applications of algebraic geometry* (pp. 157-270). New York, NY: Springer New York.

[18] Lebarbé, C., Flayac, E., Fournié, M., Henrion, D., Korda, M. (2024). Optimal Control of 1D Semilinear Heat Equations with Moment-SOS Relaxations. arXiv preprint arXiv:2411.11528.

[19] Nair, V., Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning* (ICML-10) (pp. 807-814).

[20] Nie, J. (2012). Sum of squares methods for minimizing polynomial forms over spheres and hypersurfaces. *Frontiers of mathematics in china*, 7(2), 321-346.

[21] Nie, J. (2017). Low rank symmetric tensor approximations. *SIAM Journal on Matrix Analysis and Applications*, 38(4), 1517-1540.

[22] Nie, J. (2023). *Moment and polynomial optimization*, SIAM.

[23] Nie, J., Wang, L., Ye, J. J. (2017). Bilevel polynomial programs and semidefinite relaxation methods. *SIAM Journal on Optimization*, 27(3), 1728-1757.

[24] Nie, J., Wang, L., Zheng, Z. (2023). Low rank tensor decompositions and approximations. *Journal of the Operations Research Society of China*, 1-27.

[25] Nie, J., Yang, L., Zhong, S., Zhou, G. (2023). Distributionally robust optimization with moment ambiguity sets. *Journal of Scientific Computing*, 94(1), 12.

[26] Nie, J., Zhong, S. (2025). Distributionally Robust Optimization with Polynomial Robust Constraints. *Journal of Global Optimization*, 1-26.

[27] Park, B. J., Pedrycz, W., Oh, S. K. (2002). Fuzzy polynomial neural networks: hybrid architectures of fuzzy modeling. *IEEE Transactions on Fuzzy Systems*, 10(5), 607-621.

[28] Pauwels, E., Henrion, D., Lasserre, J. B. (2017). Positivity certificates in optimal control. In *Geometric and Numerical Foundations of Movements* (pp. 113-131).

[29] Ramachandran, P., Zoph, B., Le, Q. V. (2017). Searching for activation functions. arXiv preprint arXiv:1710.05941.

[30] Sturm, J. F. (1999). Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4), 625-653.

[31] Xu, B., Wang, N., Chen, T., Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853.

[32] Zarandi, M. F., Türksen, I. B., Sobhani, J., Ramezanianpour, A. A. (2008). Fuzzy polynomial neural networks for approximation of the compressive strength of concrete. *Applied Soft Computing*, 8(1), 488-498.

[33] Zhong, S., Cui, Y., Nie, J. (2024). Towards global solutions for nonconvex two-stage stochastic programs: a polynomial lower approximation approach. *SIAM Journal on Optimization*, 34(4), 3477-3505.

[34] Zhou, J., Nie, J., Peng, Z., Zhou, G. (2025). The rank-1 completion problem for cubic tensors. *SIAM Journal on Matrix Analysis and Applications*, 46(1), 151-171.