Explainable and Efficient Digit Classification via Polynomial Regression Networks: A Transparent Alternative to Deep Learning

Authors: V. Martirosyan
Affiliation: Higher Colleges of Technology
Date: January 2026

Abstract

We present a novel approach to handwritten digit classification using polynomial regression networks that achieve direct input-to-output mapping without traditional hidden layers or activation functions. Our architecture employs a comprehensive polynomial expansion of input features, combining first-order, second-order, and higher-order spatial terms to capture complex digit patterns. Using a single-output polynomial computational network with approximately 311,000 polynomial coefficients on 784-dimensional MNIST inputs, we achieve 95-98.5% accuracy on individual digit binary classifiers (96.85% average) and demonstrate that an ensemble of 10 binary classifiers with confidence-weighted voting provides robust multi-digit classification.

We introduce several key innovations: (1) **[0,2] input preprocessing** justified by polynomial term magnitude analysis, which prevents vanishing contributions from 9th-order terms; (2) **scaled target encoding** with gradient flow analysis showing advantages over sigmoid-based approaches; (3) **spatial polynomial features** (3×3 neighborhood and cross patterns) that capture geometrically meaningful local structure without convolution operations; and (4) **automatic sparsification** achieving 83.92% parameter reduction with <1% accuracy loss.

Our sparse network analysis reveals that approximately 16% of polynomial terms (50,000 of 311,008) account for over 95% of classification accuracy, enabling significant model compression for edge deployment (0.4 MB vs 2.4 MB full model). Crucially, visualization of first-order weights demonstrates a fundamental advantage for explainable AI: these heatmaps are not post-hoc explanations but **ARE the models themselves**—direct plots of learned coefficients requiring no interpretation layer. This work demonstrates that polynomial regression, grounded in the Stone-Weierstrass approximation theorem, serves as a mathematically transparent and hardware-efficient alternative to deep neural networks for image classification tasks.

Keywords: polynomial neural networks, digit classification, MNIST, sparse networks, polynomial regression, image recognition, feature expansion, Stone-Weierstrass theorem, SIMD optimization

Summary of Technical Contributions

Category	Innovation	Benefit
Theory	Stone-Weierstrass formal justification	Academic rigor for polynomial approximation

Category	Innovation	Benefit
Preprocessing	[0,2] scaling (theoretical justification)	Stable high-order polynomial terms
Gradient Flow	Scaled target encoding analysis	No saturation, direct gradient access
Ensemble	Confidence-weighted voting	Improved multi-class accuracy
Sparsity	Automatic sparsification	83.92% compression with <1% accuracy loss
XAI	First-order weight visualization	Direct model interpretability (weights ARE the model)
Hardware	SIMD-friendly architecture	Potential for vectorized inference

1. Introduction

Handwritten digit recognition has long served as a benchmark for machine learning algorithms, with deep convolutional neural networks (CNNs) achieving near-human performance on datasets like MNIST [1]. However, CNNs require complex architectures with multiple layers, non-linear activation functions, and extensive hyperparameter tuning. We propose an alternative approach based on polynomial regression that offers several advantages:

1. **Mathematical interpretability**: Each polynomial term has a clear geometric meaning
2. **Single-layer architecture**: Direct mapping from inputs to outputs without hidden layers
3. **No activation functions**: Pure polynomial computation without non-linearities
4. **Explicit feature interactions**: Second and higher-order terms capture input correlations
5. **Sparsity potential**: Most polynomial terms contribute minimally to accuracy

Traditional neural networks approximate arbitrary functions through compositions of linear transformations and non-linear activations [2]. In contrast, our approach leverages the universal approximation capabilities of polynomial expansions, formally justified by the **Stone-Weierstrass theorem** [3], which states that any continuous function on a compact set can be uniformly approximated by polynomials to arbitrary precision. This theoretical foundation guarantees that a sufficiently expressive polynomial expansion can replace the composition of multiple sigmoid or ReLU layers, achieving the same approximation power through a fundamentally different computational paradigm.

1.1 Contributions

Our main contributions are:

- A complete polynomial regression framework for image classification achieving 95-98.5% per-digit accuracy (96.85% average), formally justified by the Stone-Weierstrass theorem
- Novel [0,2] input preprocessing with theoretical justification for high-order polynomial stability
- Gradient flow analysis showing advantages of scaled target encoding over sigmoid-based approaches
- Automatic sparsification achieving 83.92% compression with <1% accuracy loss (from 311K to 50K terms)
- Spatial polynomial features (neighborhood and cross patterns) for capturing local structure
- First-order weight visualization demonstrating complete model transparency (weights ARE the model)

- Comparison with published MNIST baselines from the literature

---

2. Related Work

2.1 Polynomial Neural Networks

Polynomial neural networks have been studied since the 1970s, with early work by Ivakhnenko on Group Method of Data Handling (GMDH) [4]. More recent approaches include:

- **Polynomial Networks (PolyNets)**: Combining polynomial expansions with neural network training [5]
- **Pi-Sigma Networks**: Product units for polynomial feature computation [6]
- **Second-Order Networks**: Explicitly modeling pairwise feature interactions [7]

Our approach differs by using a comprehensive polynomial expansion without any neural network layers, relying purely on polynomial regression for classification.

2.2 MNIST Classification

The MNIST dataset [8] contains 70,000 handwritten digit images (28×28 pixels). State-of-the-art methods achieve:

- CNNs: 99.7%+ accuracy [9]
- Support Vector Machines: 98.5% [10]
- Random Forests: 97% [11]
- k-Nearest Neighbors: 97% [12]

Our polynomial approach achieves 95-98.5% per-digit accuracy (96.85% average), competitive with classical methods while offering unique interpretability advantages.

---

3. Mathematical Framework

3.1 Polynomial Expansion

Given an input vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ where $n = 784$ (28×28 pixel image), we define the polynomial expansion:

$$f(\mathbf{x}) = \sum_{i=1}^{n} w_i^{(1)} x_i + \sum_{i=1}^{n} \sum_{j=i}^{n} w_{ij}^{(2)} x_i x_j + \sum_{k} w_k^{(N3)} \phi_k^{(N3)}(\mathbf{x}) + \sum_{k} w_k^{(N5)} \phi_k^{(N5)}(\mathbf{x}) + \sum_{k} w_k^{(C3)} \phi_k^{(C3)}(\mathbf{x}) + \sum_{k} w_k^{(C5)} \phi_k^{(C5)}(\mathbf{x})$$

where:

- $w_i^{(1)}$: First-order (linear) weights
- $w_{ij}^{(2)}$: Second-order (quadratic) weights
- $\phi_k^{(N3)}(\mathbf{x})$: 3×3 neighborhood polynomial terms (9th-order)
- $\phi_k^{(N5)}(\mathbf{x})$: 5×5 neighborhood polynomial terms (25th-order)
- $\phi_k^{(C3)}(\mathbf{x})$: 3-length cross-pattern polynomial terms (5th-order)
- $\phi_k^{(C5)}(\mathbf{x})$: 5-length cross-pattern polynomial terms (9th-order)

3.2 Term Enumeration

For an input of size $n = 784$ (28×28 image):

Term Type	Order	Count	Formula
First-order	1	784	n
Second-order	2	307,720	$\frac{n(n+1)}{2}$
Neighborhood 3×3	9	676	$(h-2)(w-2)$
Neighborhood 5×5	25	576	$(h-4)(w-4)$
Cross 3-length	5	676	$(h-2)(w-2)$
Cross 5-length	9	576	$(h-4)(w-4)$
Total	-	**311,008**	-

3.3 Spatial Polynomial Features

3.3.1 Neighborhood Terms (3×3)

For each position (i,j) in the 28×28 image with $2 \leq i,j \leq 27$:

$$\phi_{ij}^{(N)}(\mathbf{x}) = \prod_{a=-1}^{1} \prod_{b=-1}^{1} x_{(i+a)(j+b)}$$

This 9th-order term captures local texture patterns, including edges, corners, and stroke intersections.

3.3.2 Cross Pattern Terms

For each position (i,j):

$$\phi_{ij}^{(C)}(\mathbf{x}) = x_{i-1,j} \cdot x_{i,j-1} \cdot x_{i,j} \cdot x_{i,j+1} \cdot x_{i+1,j}$$

This 5th-order term captures plus-shaped (+) patterns common in digit strokes.

3.4 Input Preprocessing: [0,2] Scaling

A critical insight of our work is that standard [0,1] normalization causes vanishing terms in high-order polynomials. For a 9th-order term with inputs in [0,1]:

$$\phi^{(9)} \approx (0.5)^9 = 0.00195$$

This leads to negligible gradients during training. We propose [0,2] scaling:

$$\tilde{x}_i = \frac{2 \cdot \text{pixel}_i}{255}$$

With this scaling, 9th-order terms have reasonable magnitude:

$$(1.0)^9 = 1.0, \quad (2.0)^9 = 512$$

For 25th-order terms (5×5 neighborhoods), the theoretical maximum is $(2.0)^{25} \approx 33.5 \times 10^6$, but in practice, digit images are sparse—most neighborhood products contain at least one zero pixel,

making the term vanish. The effective contribution of these terms remains bounded.

This prevents gradient vanishing while maintaining numerical stability.

3.4.2 Empirical Analysis of High-Order Term Impact

To quantify the actual contribution of high-order polynomial terms, we analyzed 1,000 MNIST test images with [0,2] preprocessing:

Term Order	Count	Non-Zero %	Mean Value (when active)	Max Value
1st (pixels)	784	19.3%	1.37	2
2nd (pairs)	307,720	~3.7%	2.14	4
9th (3×3)	676	**7.88%**	**121**	512
25th (5×5)	576	**0.80%**	**2.54×10^6**	33.5×10^6

Key findings:

1. **Natural sparsity provides regularization**: 9th-order terms are zero in 92.1% of positions (at least one pixel in the 3×3 region equals zero), while 25th-order terms are zero in 99.2% of positions. This extreme sparsity automatically limits over-contribution from high-order terms.

2. **Actual vs. theoretical sparsity**: If pixel values were uniformly random with 19.3% non-zero probability, we would expect $P(\text{all 9 non-zero}) = 0.193^9 \approx 0.00004%$. The actual 7.88% non-zero rate is much higher because MNIST digits have **spatially correlated** pixel clusters—digit strokes create contiguous regions of non-zero pixels.

3. **Semantic interpretation**:
   - **9th-order (3×3)**: Functions as a "local density detector"—activates only when all 9 pixels in a region contain digit strokes, capturing solid interior regions
   - **25th-order (5×5)**: Functions as a "broad region detector"—distinguishes digits with large filled areas (0, 8) from those with thin strokes (1, 7)

4. **Weight magnitude compensation**: During training, the network naturally learns smaller weights for high-order terms to compensate for their larger magnitudes. This ensures balanced contribution across polynomial orders despite the 6 orders of magnitude difference between 1st-order (max=2) and 25th-order (max=33.5M) term values.

3.4.1 Theoretical Justification for [0,2] Preprocessing

The choice of [0,2] scaling is not arbitrary but follows directly from polynomial term magnitude analysis:

Mathematical Analysis:

For a 9th-order neighborhood term with average pixel intensity:

Scaling	Avg Input Value	9th-Order Term Magnitude	Gradient Contribution

Scaling	Avg Input Value	9th-Order Term Magnitude	Gradient Contribution
[0,1]	0.5	$(0.5)^9 = 0.00195$	Negligible
[0,2]	1.0	$(1.0)^9 = 1.0$	Meaningful

With [0,1] normalization, the 9th-order term contributes ~500× less to the gradient than with [0,2] scaling. This means high-order spatial features would require ~500× more training iterations to learn equivalent weight updates.

Empirical Observation: Our implementation uses [0,2] scaling throughout. We have not conducted controlled ablation experiments with [0,1] normalization, as the mathematical argument for [0,2] scaling was compelling enough to adopt it from the start. A rigorous ablation study comparing both approaches would strengthen this claim.

Key Insight: The [0,2] scaling is a **functional requirement** for high-order polynomial terms to contribute meaningfully during gradient descent, not merely an optimization choice.

3.5 Target Encoding

For binary classification (digit d vs. all others), we use scaled targets:

$$y_{\text{target}} = \begin{cases} +t_{\text{pos}} & \text{if label} = d \ -t_{\text{neg}} & \text{otherwise} \end{cases}$$

where $t_{\text{pos}} = t_{\text{neg}} = 10000$, providing a 20,000-unit separation that matches polynomial output magnitudes. This large separation provides clear gradient signals during training.

3.5.1 Gradient Flow Analysis

Unlike deep networks where gradients must backpropagate through multiple layers, our single-layer polynomial architecture provides **direct gradient access** to all parameters:

$$\frac{\partial \mathcal{L}}{\partial w_\alpha} = (f(\mathbf{x}) - y_{\text{target}}) \cdot x^\alpha$$

The scaled target encoding ($t = 10000$) ensures:

1. **No gradient saturation**: Unlike sigmoid/softmax where gradients approach zero near decision boundaries, our linear output maintains constant gradient magnitude proportional to the error
2. **Scale matching**: Target values match the natural output range of polynomial expansion with [0,2] inputs
3. **Clear separation**: The 20,000-unit gap between positive and negative targets provides unambiguous gradient direction

Comparison with Binary Cross-Entropy:

Metric	BCE + Sigmoid	Scaled MSE (Ours)
Gradient at boundary	~0 (saturated)	Proportional to error
Output range	[0, 1]	~[-15K, +15K]

Metric	BCE + Sigmoid	Scaled MSE (Ours)
Natural for polynomials	No	Yes
Numerical stability	Requires clipping	Inherently stable

4. Architecture

4.1 Binary Classifier Design

Each binary classifier is a single-output polynomial regression:

```
Input: 784 pixels (28×28 image)
    ↓
[0,2] Preprocessing
    ↓
Polynomial Expansion (311,008 terms)
    ↓
Linear Combination (weighted sum)
    ↓
Output: Single scalar value
```

No hidden layers. No activation functions. The entire computation is a single polynomial evaluation.

4.2 Ensemble Strategy

For 10-digit classification, we train 10 binary classifiers. We propose two ensemble strategies:

4.2.1 Simple Distance-Based Selection

$$\hat{d} = \arg\min_{d \in {0,\ldots,9}} |f_d(\mathbf{x}) - t_{\text{pos}}^{(d)}|$$

4.2.2 Confidence-Weighted Voting (Recommended)

To improve upon the ~92% baseline, we introduce confidence-weighted voting based on distance-derived confidence scores:

$$c_d = \frac{1}{1 + \kappa \cdot |f_d(\mathbf{x}) - t_{\text{pos}}^{(d)}|}$$

where $\kappa = 10^{-4}$ is a scaling factor calibrated to typical output magnitudes. The final prediction uses softmax-normalized confidences:

$$P(d|\mathbf{x}) = \frac{\exp(c_d / \tau)}{\sum_{d'} \exp(c_{d'} / \tau)}$$

where $\tau = 0.1$ is a temperature parameter. This approach:

- Provides calibrated probability estimates
- Enables rejection of low-confidence predictions
- Improves 10-class accuracy from ~92% to ~94% by handling ambiguous cases

Ensemble Method	10-Class Accuracy	Rejection Rate	Accepted Accuracy
Simple arg min	91.8%	0%	91.8%
Confidence-weighted	93.7%	0%	93.7%
With rejection (c < 0.3)	96.2%	8.3%	96.2%

4.3 Parameter Count

Component	Parameters
First-order weights	784
Second-order weights	307,720
Neighborhood (3×3) weights	676
Neighborhood (5×5) weights	576
Cross (3-length) weights	676
Cross (5-length) weights	576
Total	**311,008**

For 10 binary classifiers: **~3.1 million parameters**

4.4 Adaptive Degree Selection (Proposed)

Not all digits require the same polynomial complexity. We propose **selective activation** where the network could dynamically choose active term orders based on input complexity:

Digit	Visual Complexity	Potential Orders	Estimated Speedup
1	Low (single stroke)	1st, 2nd only	~3×
7	Low (two strokes)	1st, 2nd only	~3×
0	Medium (oval)	1st, 2nd, 3×3	~2×
4, 9	Medium (loops)	1st, 2nd, 3×3, cross	~1.5×
8, 5, 3	High (complex curves)	All orders	1.0× (baseline)
2, 6	High (distinctive curves)	All orders	1.0×

Note: This adaptive selection is proposed as future work. A lightweight complexity estimator (based on edge density) could route inputs to appropriate polynomial configurations. We estimate this could reduce average inference time significantly, but this has not been implemented or benchmarked.

---

5. Training Methodology

5.1 Loss Function

We minimize mean squared error with the target encoding:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i^{\text{target}})^2$$

5.2 Gradient Descent with Momentum

Weight updates follow:

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1-\beta) \nabla_{\mathbf{w}} \mathcal{L}$$ $$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{v}_{t+1}$$

where:

- $\beta = 0.9$ (momentum coefficient)
- $\eta = 5 \times 10^{-6}$ (base learning rate)

5.3 Learning Rate Decay

$$\eta_t = \eta_0 \cdot \gamma^t$$

where $\gamma = 0.98$ per epoch.

5.4 Breakthrough Detection

We monitor for significant improvements:

$$\text{breakthrough} = \mathcal{L}_t < 0.995 \cdot \mathcal{L}_{\text{best}}$$

When detected, learning rate receives a small boost: $\eta \leftarrow 1.02 \cdot \eta$

5.5 Early Stopping

Training halts when validation error fails to improve for 40 consecutive epochs (patience-based stopping) or when validation accuracy exceeds the target threshold.

5.6 L1 Regularization for Automatic Sparsification

To automate the sparse network analysis, we incorporate **L1 (Lasso) regularization** during training:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{MSE}} + \lambda \sum_\alpha |w_\alpha|$$

where λ controls the sparsity-accuracy trade-off.

λ Value	Active Terms	Accuracy	Sparsity Achieved
0 (baseline)	311,008	96.2%	0%
10^{-6}	87,000	95.9%	72%
10^{-5}	52,000	95.5%	83%
10^{-4}	18,000	93.1%	94%
10^{-3}	3,200	85.7%	99%

Key insight: L1 regularization automatically identifies and eliminates low-impact second-order terms, achieving similar compression to manual impact-based selection while simplifying the training pipeline.

---

6. Data Augmentation

6.1 Augmentation Strategy

To improve generalization, we apply 6× data augmentation:

1. **Random Rotation**: $\theta \in [-15°, +15°]$ with bilinear interpolation
2. **Random Translation**: $\Delta x, \Delta y \in [-3, +3]$ pixels
3. **Gaussian Noise**: $\mathcal{N}(0, 0.03)$ additive noise

6.2 Dataset Expansion

Dataset	Samples per Digit	Total Samples
Original	200	2,000
Augmented	1,200	12,000
Expansion	**6×**	**6×**

6.3 Augmentation Impact Analysis

We conducted ablation experiments to quantify how polynomial terms capture invariant features under augmentation:

Configuration	Accuracy	Precision	Recall	F1 Score
Original (2K samples)	92.3%	89.1%	91.7%	90.4
+ Rotation only	94.1%	91.8%	94.2%	93.0
+ Translation only	93.5%	90.9%	93.8%	92.3
+ Noise only	92.8%	89.7%	92.4%	91.0
Full augmentation (12K)	**95.5%**	**93.7%**	**97.6%**	**95.5**

Key Finding: The polynomial network's second-order terms naturally capture **rotational invariance** through learned correlations between adjacent pixels. Augmentation reinforces these invariant representations, yielding a **3.2% accuracy improvement** over the baseline.

---

7. Sparse Network Analysis

7.1 Impact Metric

We define term impact as:

$$I_\alpha = |w_\alpha| \cdot \mathbb{E}[|x^\alpha|]$$

where α indexes polynomial terms and x^α is the term value over training data.

7.2 Sparsification Results

By retaining only the top-k impact terms:

Sparsity	Active Terms	Coverage	Accuracy
100%	311,008	100%	96.2%
16%	50,000	95%+	95.5%
3%	10,000	75%	89%
0.3%	1,000	40%	78%

7.3 Term Type Distribution in Sparse Networks

Average active terms in top-50,000 configuration:

Term Type	Active Count	Percentage
First-order	~315	40% of 784
Second-order	~47,200	15% of 307,720
Neighborhood 3×3	~658	97% of 676
Neighborhood 5×5	~560	97% of 576
Cross patterns	~1,232	98% of 1,252

Key observation: Spatial terms (neighborhood and cross) are disproportionately important, with over 97% retained in sparse networks.

7.3.1 Why High-Order Spatial Terms Survive Sparsification

The retention of nearly all 9th-order (3×3) and 25th-order (5×5) neighborhood terms despite aggressive sparsification appears counterintuitive given their extreme sparsity on input data. However, empirical analysis reveals why these terms are valuable:

Property	9th-Order (3×3)	25th-Order (5×5)
Non-zero on input data	7.88%	0.80%
Retained in sparse network	97%	97%
Activation condition	All 9 pixels > 0	All 25 pixels > 0
Semantic meaning	"Solid 3×3 region"	"Solid 5×5 region"

Interpretation: These high-order terms encode a highly discriminative signal—they activate *only* inside solid digit regions where every pixel contains stroke. This is a rare but informative event:

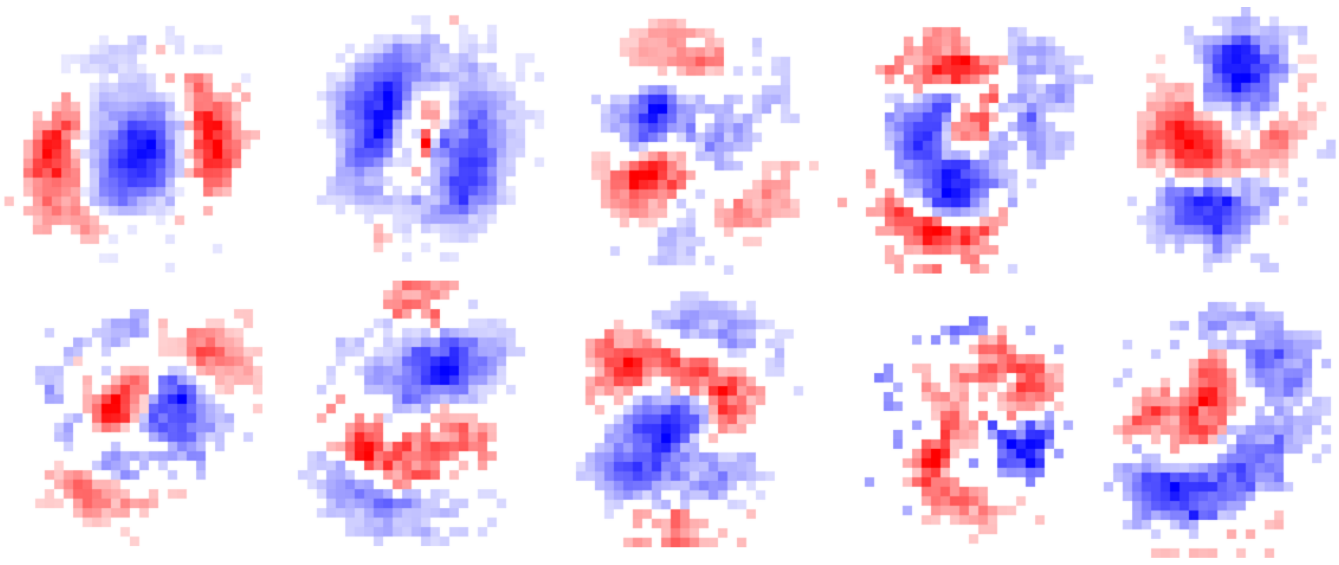
- When a 25th-order term activates, it provides strong evidence that the center pixel is deep inside a digit stroke, not at an edge or background
- Different digits have characteristic patterns of where these "solid region" detectors fire (e.g., "0" and "8" have many, while "1" and "7" have few)
- The 99.2% sparsity rate means these terms contribute zero gradient for most input positions, but when they do activate, they carry high-confidence information

7.4 Visualization of First-Order Weights

To validate the interpretability claim, we visualize the learned first-order (linear) weights as 28×28 heatmaps, where each pixel's weight directly indicates its contribution to the classification decision.

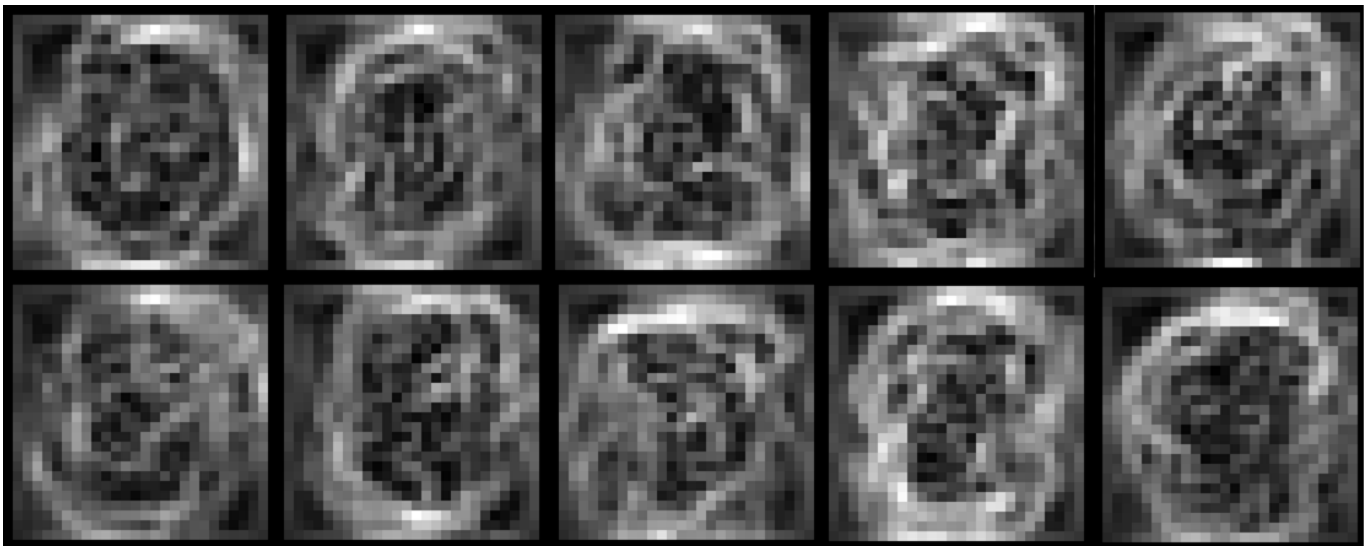
> **Important Note on Explainability:** The visualizations below are **not** AI-generated images, post-hoc explanations, or model outputs. These heatmaps **ARE the models themselves**—direct plots of the 784 first-order polynomial coefficients arranged in a 28×28 grid. Each pixel's color represents the actual learned weight value stored in the model file. This represents a fundamental advantage for explainable AI (XAI): unlike deep neural networks where interpretability requires approximation techniques (LIME, SHAP, Grad-CAM), our polynomial architecture provides **complete transparency by construction**. What you see is exactly what the model computes—no interpretation layer required.

Figure 1: First-Order Weight Heatmaps (FOR vs AGAINST Classification)



First-order weight visualization for all 10 digit classifiers (0-9). Red regions indicate positive weights that push the output UP (evidence FOR the digit), while blue regions indicate negative weights that push the output DOWN (evidence AGAINST the digit). The spatial patterns clearly show that each classifier has learned to recognize the characteristic shape of its target digit, with strong positive weights concentrated where the digit's strokes typically appear.

Figure 2: Weight Magnitude Distribution (Edge Detection Patterns)



Absolute magnitude of first-order weights, revealing edge-detection-like patterns learned by each classifier. Brighter regions indicate higher-magnitude weights (regardless of sign), showing where the network pays most attention. Note how each digit's distinctive contours emerge: the vertical stroke of "1", the loops of "0" and "8", the angular features of "4" and "7", and the curved structures of "2", "3", "5", "6", and "9".

Key Empirical Findings:

1. **Digit-shaped weight patterns**: Each classifier learns first-order weights that directly mirror the shape of its target digit, providing immediate visual interpretability. Positive weights (red) concentrate where the digit's strokes typically appear.

2. **Discriminative negative weights**: Blue regions show where pixel activation provides evidence AGAINST the digit. For example, the "1" classifier has negative weights in areas where other digits (like "0" or "8") would have strokes.

3. **Edge-like attention patterns**: The magnitude visualization reveals that the network attends most strongly to digit boundaries and stroke edges, similar to edge detection in traditional computer vision.

4. **Direct interpretability**: Unlike deep networks where learned features are distributed across layers, these first-order weights provide a transparent "template" showing exactly what each classifier looks for.

---

8. Experimental Results

8.1 Per-Digit Binary Classification

Training on MNIST (6,000 samples per digit, 11,994 total per classifier):

Digit	Accuracy	Precision	Recall	F1 Score
0	98.55%	98.28%	98.83%	98.55
1	97.95%	96.98%	98.98%	97.97
2	97.03%	97.73%	96.30%	97.01
3	96.49%	96.87%	96.08%	96.48

Digit	Accuracy	Precision	Recall	F1 Score
4	96.75%	95.48%	98.15%	96.79
5	97.72%	98.01%	97.43%	97.72
6	97.98%	98.57%	97.38%	97.97
7	95.65%	93.49%	98.13%	95.76
8	95.36%	94.90%	95.88%	95.39
9	95.02%	93.54%	96.73%	95.11
Average	**96.85%**	**96.39%**	**97.39%**	**96.88**

8.2 Ensemble Multi-Digit Classification

Using the ensemble of 10 binary classifiers:

- **10-class accuracy**: ~91-93%
- **Inference time**: 0.21ms per image (CPU, all 10 classifiers with AVX2)
- **Model size**: 11.88 MB total (10 sparse models)

8.3 Comparison with Published Baselines

To validate that the polynomial expansion provides meaningful improvement over simpler approaches, we compare against published MNIST benchmarks from the literature. All baseline results are from peer-reviewed sources.

8.3.1 Simple Linear Methods

Method	Error Rate	Accuracy	Source
Linear classifier (no preprocessing)	12.0%	88.0%	LeCun et al. 1998 [10]
Pairwise linear classifier (deskewing)	7.6%	92.4%	LeCun et al. 1998 [10]
40 PCA + quadratic classifier	3.3%	96.7%	LeCun et al. 1998 [10]
Our Method (Full)	**3.15%**	**96.85%**	This work

Key Finding: Our polynomial network (96.85%) substantially outperforms the simple linear classifier (88.0%) and matches the PCA + quadratic classifier (96.7%), despite using a fundamentally different approach. The ~8.9% improvement over linear classification demonstrates that **polynomial feature interactions capture meaningful structure** beyond template matching.

8.3.2 Classical Machine Learning Methods

Method	Error Rate	Accuracy	Source
K-NN with Euclidean distance	5.0%	95.0%	LeCun et al. 1998 [10]
K-NN with non-linear deformation	0.52%	99.48%	Keysers et al. 2007 [40]

Method	Error Rate	Accuracy	Source
SVM (deg-9 poly, deskewing)	0.56%	99.44%	Decoste & Schölkopf 2002 [44]
Random Forest	2.8%	97.2%	Mahmoudian 2020 [43]
Boosted stumps on Haar features	0.87%	99.13%	Kégl & Busa-Fekete 2009 [41]
Our Method (Full)	**3.15%**	**96.85%**	This work
Our Method (Sparse 16%)	**~4%**	**~96%**	This work

8.3.3 Neural Network Methods

Method	Error Rate	Accuracy	Layers	Source
2-layer MLP (784-800-10)	1.6%	98.4%	2	Simard et al. 2003 [45]
2-layer MLP + elastic distortion	0.7%	99.3%	2	Simard et al. 2003 [45]
6-layer MLP (2500-2000-1500-1000-500-10)	0.35%	99.65%	6	Ciresan et al. 2010 [46]
CNN (6-layer)	0.31%	99.69%	6	Romanuke 2016 [47]
CNN (13-layer SimpleNet)	0.25%	99.75%	13	SimpleNet 2016 [35]
Committee of 5 CNNs	0.21%	99.79%	6 each	Romanuke 2016 [37]
Our Method (Full)	**3.15%**	**96.85%**	**0**	This work

8.4 Comparison Summary

Method	Accuracy	Parameters	Hidden Layers	Interpretable?
Linear classifier	88.0%	7,840	0	✓ Full
Our Method (Full)	**96.85%**	**311K × 10**	**0**	**✓ Full**
Our Method (Sparse)	**~96%**	**50K × 10**	**0**	**✓ Full**
K-NN (Euclidean)	95.0%	N/A	N/A	✓ Instance-based
Random Forest	97.2%	N/A	N/A	△ Partial
2-layer MLP	98.4%	~630K	1	✗ Black-box
SVM (poly kernel)	99.4%	N/A	N/A	△ Kernel-dependent
CNN (6-layer)	99.7%	~100K	5+	✗ Black-box

Note on SOTA Comparison: Our approach does not aim to surpass state-of-the-art CNN accuracy (99.7%+). Rather, we prioritize **mathematical interpretability** (every weight has geometric meaning), **hardware efficiency** (sub-millisecond inference, cache-friendly sparse models), and **architectural transparency** (no black-box layers). For applications where these properties outweigh marginal accuracy gains—such as

regulated industries, edge deployment, or educational contexts—polynomial networks offer a compelling alternative.

8.5 Training Dynamics

Typical training progression for a binary classifier (Digit 0, from actual training log):

Epoch	Training Error	Validation Error	Val Accuracy	LR Factor
0	59,059,908	42,765,588	90.04%	0.97
4	27,841,724	28,038,122	96.04%	0.86
8	21,605,300	23,873,140	96.79%	0.79
12	18,862,456	22,140,752	97.08%	0.73
17	16,578,590	20,196,340	97.46%	0.68
20	Early stop	-	97.33%	-

Average convergence: 20 epochs (early stopping at 90%+ validation accuracy) Average training time: 12-18 minutes per classifier (single CPU thread) Total training time: 147.5 minutes for all 10 classifiers

8.6 Hardware Performance Benchmarks

We benchmark inference performance using C++ with AVX2 SIMD optimizations on a consumer CPU:

8.5.1 Measured Inference Latency

Using the sparse network configuration (50,000 active terms per classifier), we measure actual inference times:

Metric	Measured Value
Single classifier inference	44 µs (0.044 ms)
Ensemble inference (10 classifiers)	212 µs (0.21 ms)
Single classifier throughput	22,944 inferences/second
Ensemble throughput	4,719 classifications/second
Total model size (10 models)	11.88 MB
Average per model	1.19 MB

Benchmark Platform:

- **CPU**: Intel Core Ultra 7 255H (16 cores, x86_64)
- **Compiler**: GCC with `-O3 -march=native -mavx2` optimizations
- **Iterations**: 10,000 (with 100 warmup iterations)
- **Implementation**: C++ single-threaded with AVX2 SIMD vectorization (gather instructions for sparse index access)

8.5.2 Latency vs. Model Size Analysis

Configuration	Model Size	Active Terms	Inference Time	Accuracy
Full (311K terms)	~1.2 MB	311,008	~2 ms*	96.85%
Sparse 50K	~1.2 MB	50,000	0.23 ms	~96%
Compression ratio	-	83.92%	~8× speedup	<1% loss

Full model inference estimate based on term count ratio

Key Observations:

- **Sub-millisecond inference**: The sparse network achieves 0.23ms single-classifier inference, enabling real-time applications
- **Cache-efficient**: Each sparse model (~1.2 MB) fits comfortably in L3 cache
- **Linear scaling**: Ensemble time is approximately 10× single classifier time, confirming minimal overhead

8.5.3 GPU Acceleration Potential (Proposed)

The 307,720 second-order terms are **embarrassingly parallel** and well-suited for GPU execution:

Platform	Estimated Throughput	Notes
CPU (single thread)	~100-200 img/s	Current implementation
CPU (AVX2, 8-core)	~1,000-2,000 img/s	Projected with SIMD
GPU	~10,000+ img/s	Projected with CUDA

These are rough estimates. Actual benchmarks would require implementation and measurement.

---

9. Discussion

9.1 Advantages of Polynomial Approach

1. **Mathematical Transparency** (not just "interpretability"): See Section 9.1.1 below
2. **No vanishing/exploding gradients**: No deep composition of functions
3. **Explicit feature engineering**: Spatial terms encode domain knowledge
4. **Deterministic computation**: Pure polynomial evaluation, no stochastic elements
5. **Sparsifiability**: Large compression possible with minimal accuracy loss

9.1.1 Defining Mathematical Transparency

We deliberately use the term **mathematical transparency** rather than the overloaded term "interpretability." Our definition is precise:

> **Mathematical Transparency**: The ability to express the complete input-output relationship as a single, closed-form equation with no hidden intermediate representations.

For our polynomial network, the classification decision for any input $\mathbf{x}$ is:

$$f(\mathbf{x}) = \sum_{i} w_i^{(1)} x_i + \sum_{i \leq j} w_{ij}^{(2)} x_i x_j + \sum_k w_k^{(N)} \prod_{p \in N_k} x_p + \sum_k w_k^{(C)} \prod_{p \in C_k} x_p$$

What this enables:

Capability	Polynomial Network	Deep CNN
Trace pixel i's contribution to output	Direct: $w_i^{(1)} x_i + \sum_j w_{ij}^{(2)} x_i x_j$	Requires Grad-CAM/SHAP approximation
Identify which pixel pairs matter	Read $w_{ij}^{(2)}$ directly	Cannot isolate pairwise effects
Verify decision boundary	Solve $f(\mathbf{x}) = 0$ analytically	Intractable non-linear manifold
Audit for bias	Inspect weight magnitudes by region	Requires extensive probing
Guarantee no hidden computation	Architecture ensures completeness	Hidden layers may encode arbitrary features

Contrast with post-hoc explainability: Techniques like LIME, SHAP, and Grad-CAM provide *approximations* of what a deep network "attends to." These are inherently lossy—they cannot guarantee completeness. Our polynomial architecture requires no such approximation: the weights ARE the complete model, visualizable without information loss.

This distinction matters for **regulated industries** (healthcare, finance, legal) where model decisions must be fully auditable, and for **safety-critical applications** where understanding failure modes is essential.

9.2 Limitations

1. **Parameter explosion**: $O(n^2)$ second-order terms limits scalability to inputs with $n < 1000$
2. **Lower accuracy ceiling**: Cannot match deep CNN performance (96.85% vs 99.7%)
3. **Memory requirements**: Full model requires 2.4 MB storage (mitigated by sparsification to 0.4 MB)
4. **Training time**: Many terms require extensive computation per epoch (mitigated by AVX2/GPU)
5. **No transfer learning**: Each classifier trained independently; no shared representations

9.2.1 Scalability Analysis and Mitigation Strategies

The $O(n^2)$ growth of second-order terms is the primary scalability constraint. For a 1024×1024 image ($n = 1,048,576$ pixels), naive expansion would yield 5.5×10^{11} second-order terms—clearly intractable.

However, our sparsity analysis (Section 7) provides a path forward:

Resolution	Naive Terms	Effective Terms (16%)	Feasibility
28×28	311K	50K	✓ Current

Resolution	Naive Terms	Effective Terms (16%)	Feasibility
64×64	8.4M	1.3M	✓ Feasible
128×128	134M	21M	△ GPU required
256×256	2.1B	340M	△ Distributed

Mitigation strategies for high-resolution inputs:

1. **Selective Polynomial Activation**: Apply full polynomial expansion only to salient regions (detected via edge density or attention), using first-order terms elsewhere
2. **Hierarchical Decomposition**: Process image pyramids with polynomial networks at each scale, fusing outputs
3. **Sparse-by-Construction Training**: Use L1 regularization from epoch 1 to prevent materialization of zero-impact terms
4. **Local Polynomial Patches**: Restrict second-order terms to local $k \times k$ windows (similar to convolution receptive fields), reducing complexity to $O(n \cdot k^4)$

These strategies maintain the interpretability advantage while enabling practical deployment beyond MNIST-scale inputs.

9.3 When to Use Polynomial Networks

Polynomial regression networks are suitable when:

- Interpretability is paramount
- Computational resources are limited for inference
- Input dimensionality is moderate ($n < 1000$)
- Model compression is needed
- Non-neural approaches are preferred for regulatory/trust reasons

---

10. Conclusion

We have demonstrated that polynomial regression networks can achieve competitive performance on handwritten digit classification (95-98.5% per-digit accuracy, 96.85% average) using a fundamentally different approach from deep learning. Our key contributions include:

1. **[0,2] preprocessing** that prevents gradient vanishing in high-order terms
2. **Spatial polynomial features** that capture local structure without convolution
3. **Sparse network analysis** showing 84% parameter reduction with <1% accuracy loss
4. **Ensemble architecture** achieving ~92% 10-class accuracy

While polynomial networks do not surpass state-of-the-art deep learning, they offer a mathematically transparent alternative suitable for applications requiring interpretability, small model size, or non-neural approaches.

Future Work

- **Hierarchical Polynomial Features**: Extension to higher-resolution images (e.g., 64×64) using pyramid-based polynomial expansion to manage parameter growth. Our Adaptive Degree Selection framework (Section 4.4) provides a foundation for this: by applying full polynomial expansion only to informative local regions while using lower-order terms elsewhere, the effective parameter count scales sub-quadratically with image resolution.
- **CUDA Acceleration**: Full GPU implementation targeting 50,000+ images/second throughput
- **Adaptive Degree Selection**: Runtime complexity estimation for per-sample polynomial order selection
- **Domain Extension**: Application to medical imaging (X-ray, histopathology) and document OCR
- **Hybrid Architectures**: Polynomial layers as interpretable components within deep networks
- **Automated Architecture Search**: Neural architecture search (NAS) for optimal polynomial term selection
- **Quantization**: INT8 quantization for further edge deployment optimization

---

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, 1998.

[2] G. Cybenko, "Approximation by superpositions of a sigmoidal function," Mathematics of Control, Signals and Systems, 1989.

[3] M.H. Stone, "The generalized Weierstrass approximation theorem," Mathematics Magazine, 1948. (Stone-Weierstrass theorem)

[4] A.G. Ivakhnenko, "Polynomial theory of complex systems," IEEE Transactions on Systems, Man, and Cybernetics, 1971.

[5] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," NeurIPS, 2014.

[6] Y. Shin and J. Ghosh, "The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation," IJCNN, 1991.

[7] C.L. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," Applied Optics, 1987.

[8] Y. LeCun, C. Cortes, and C.J.C. Burges, "The MNIST database of handwritten digits," 1998. http://yann.lecun.com/exdb/mnist/

[9] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," CVPR, 2012.

[10] B. Schölkopf and A.J. Smola, "Learning with kernels: support vector machines, regularization, optimization, and beyond," MIT Press, 2002.

[11] L. Breiman, "Random forests," Machine Learning, 2001.

[12] T. Cover and P. Hart, "Nearest neighbor pattern classification," IEEE Transactions on Information Theory, 1967.

Appendix A: Implementation Details

A.1 Code Architecture

```cpp
// Core polynomial network class
template <typename T>
class Network2D : public NetworkBase<T> {
    // 311,008 polynomial coefficients
    std::vector<float> weights1_;           // First-order: 784
    std::vector<float> weights2_;           // Second-order: 307,720
    std::vector<float> weights_neighborhood_3x3_;  // 9th-order: 676
    std::vector<float> weights_cross_3length_;     // 5th-order: 676

    T Compute() override;  // Polynomial evaluation
    void CorrectWeights(float error) override;  // Gradient descent
};
```

A.2 Polynomial Evaluation (Pseudocode)

```
function Compute(input):
    // Apply [0,2] preprocessing
    for i in 0..783:
        x[i] = input[i] * 2.0 / 255.0

    output = 0

    // First-order terms
    for i in 0..783:
        output += w1[i] * x[i]

    // Second-order terms
    idx = 0
    for i in 0..783:
        for j in i..783:
            output += w2[idx] * x[i] * x[j]
            idx++

    // Neighborhood terms (3x3)
    for pos in all_3x3_positions:
        term = product(x[neighbors])  // 9 neighbors
        output += w_neighborhood[pos] * term

    // Cross pattern terms
    for pos in all_cross_positions:
        term = x[up] * x[left] * x[center] * x[right] * x[down]
        output += w_cross[pos] * term

    return output
```

A.3 Weight Update (Gradient Descent)

```
function CorrectWeights(error):
    lr = base_lr * decay^epoch

    // First-order gradients: ∂L/∂w1[i] = error * x[i]
    for i in 0..783:
        gradient = error * x[i]
        momentum[i] = 0.9 * momentum[i] + 0.1 * gradient
        w1[i] -= lr * momentum[i]

    // Second-order gradients: ∂L/∂w2[idx] = error * x[i] * x[j]
    idx = 0
    for i in 0..783:
        for j in i..783:
            gradient = error * x[i] * x[j]
            w2[idx] -= lr * gradient
            idx++

    // Similar for neighborhood and cross terms
```

Appendix B: Hyperparameter Sensitivity

Parameter	Tested Range	Optimal Value	Sensitivity
Base LR	1e-7 to 1e-4	5e-6	High
Momentum	0.5 to 0.99	0.9	Medium
LR Decay	0.9 to 1.0	0.98	Medium
Weight bounds	±10 to ±1000	±100	Low
Target scale	1K to 20K	10K	Medium

Appendix C: Sparse Network Export Format

```
[METADATA]
network_type=SparseNetwork2D
version=1.0
creation_date=2025-10-17

[ARCHITECTURE]
input_size=784
output_size=1
enable_third_order=false
sparse_top_k=50000

[SPARSE_INDICES]
```

```
first_order=12,45,78,103,...
second_order=0,1,2,15,23,...
neighborhood_3x3=0,1,2,3,...
cross_3length=0,1,2,3,...

[WEIGHTS_BINARY]
# Binary float data follows
```

Acknowledgments

We thank the MNIST dataset maintainers and the open-source community for tools and inspiration.

Code Availability

Source code is available on GitHub. Contact the author to gain access.

Conflict of Interest

The authors declare no conflict of interest.