On Expressivity and Training of Quadratic Networks

Feng-Lei Fan\textsuperscript{1}, Member, IEEE, Mengzhou Li\textsuperscript{2}, Student Member, IEEE, Fei Wang\textsuperscript{1*}, Rongjie Lai\textsuperscript{3*}, Ge Wang\textsuperscript{2*}, Fellow, IEEE,

Abstract—Inspired by the diversity of biological neurons, quadratic artificial neurons can play an important role in deep learning models. The type of quadratic neurons of our interest replaces the inner-product operation in the conventional neuron with a quadratic function. Despite promising results so far achieved by networks of quadratic neurons, there are important issues not well addressed. Theoretically, the superior expressivity of a quadratic network over either a conventional network or a conventional network via quadratic activation is not fully elucidated, which makes the use of quadratic networks not well grounded. Practically, although a quadratic network can be trained via generic backpropagation, it can be subject to a higher risk of collapse than the conventional counterpart. To address these issues, we first apply the spline theory and a measure from algebraic geometry to give two theorems that demonstrate better model expressivity of a quadratic network than the conventional counterpart with or without quadratic activation. Then, we propose an effective and efficient training strategy referred to as ReLinear to stabilize the training process of a quadratic network, thereby unleashing the full potential in its associated machine learning tasks. Comprehensive experiments on popular datasets are performed to support our findings and evaluate the performance of quadratic deep learning.

Index Terms—Neuronal diversity, quadratic neurons, quadratic networks, expressivity, training strategy

I. INTRODUCTION

In recent years, a plethora of deep artificial neural networks have been developed with impressive successes in many mission-critical tasks \cite{1, 2}. However, up to date the design of these networks focuses on architectures, such as shortcut connections \cite{3, 4}. Indeed, neural architecture search \cite{5} is to find networks of similar topological types. Almost exclusively, the mainstream network models are constructed with neurons of the same type, which is composed by two parts: inner combination and nonlinear activation (We refer to such a neuron as a conventional neuron and a network made of these neurons as a conventional network hereafter). Despite that a conventional network does simulate certain important aspects of a biological neural network such as a hierarchical representation \cite{6}, attention mechanism \cite{7}, and so on, a conventional network and a biological neural system are fundamentally different in terms of neuronal diversity and complexity. In particular, a biological neural system coordinates numerous types of neurons which contribute to all kinds of intellectual behaviors \cite{8}. Considering that an artificial network is invented to mimic the biological neural system, the essential role of neuronal diversity should be taken into account in deep learning research.

Along this direction, the so-called quadratic neurons \cite{9} were recently proposed, which replace the inner product in a conventional neuron with a quadratic operation (Hereafter, we call a neural network made of quadratic neurons as a quadratic neural network, QNN). A single quadratic neuron can implement XOR logic operation, which is not possible for an individual conventional neuron. The superior expressivity of quadratic networks over conventional networks is partially confirmed by a theorem that given the same structure there exists a class of functions that can be expressed by quadratic networks with a polynomial number of neurons, and can only be expressed by conventional networks with an exponential number of neurons \cite{10}. In addition, a quadratic autoencoder was developed for CT image denoising and produced denoising performance better than its competitors \cite{9}.

In spite of promising progress achieved by quadratic networks, there are still important theoretical and practical issues unaddressed satisfactorily. First of all, the superiority of quadratic networks in the representation power can be analyzed in a generic way instead of just showing the superiority in the case of a special class of functions. Particularly, we are interested in comparing a quadratic network with both a conventional network and a conventional network with quadratic activation \cite{11}–\cite{13}. Also, the training process of a quadratic network is subject to a higher risk of collapse than the conventional counterpart. Specifically, given a quadratic network with \( L \) layers, its output function will be a polynomial of \( 2^L \) degrees. Such a degree of freedom may lead to magnitude explosion in the training process. For example, when data are not appropriately initialized, the output of a deep quadratic network can be bizarrely huge (e.g., \( 1.1^{100} = 13,781 \)), which may destroy the training outcome. Therefore, it is essential to derive an effective and efficient strategy to facilitate the training process of a quadratic network.

To address the above issues, here we first present two theorems to reveal the superiority of a quadratic network in terms of model expressivity over either the conventional network or the conventional network with quadratic activation. The first theorem utilizes spline theory to compare the model expressivity of a quadratic network with that of a conventional network. Suppose that a ReLU activation is used, a conventional network outputs a piecewise linear function, and a quadratic network defines a piecewise polynomial function. According to spline theory, the approximation with a piecewise polynomial function is substantially more accurate than that with piecewise linear functions. Correspondingly, a quadratic...
network enjoys a better approximation accuracy. The other theorem is based on a measure in algebraic geometry to show that a quadratic network is more expressive than a conventional network with quadratic activation, which suggests that a conventional network with quadratic activation is not optimal to leverage quadratic mapping for deep learning.

\[
Y(x_1, \cdots, x_n) = a_0 + \sum_{i} a_i x_i + \sum_{i,j} a_{ij} x_i x_j + \cdots,
\]

where \( x_i \) is the \( i \)-th input variable, and \( a_i, a_{ij}, a_{ijk}, \cdots \) are coefficients. Usually, this model is terminated at the second-order terms to avoid nonlinear explosion for high-dimensional inputs. The GMDH is thought as one of the first deep learning models in the survey paper [16]. Furthermore, based on GMDH, the so-called higher-order unit was defined in [17]–[19] whose output is given by

\[
y = \sigma(Y(x_1, \cdots, x_n)),
\]

where \( \sigma(\cdot) \) is a nonlinear activation function. To maintain the power of high-order units while reducing the number of weights in high-order terms, Shin et al. reported the pi-sigma network [20], which is formulated as

\[
h_{ji} = \sum_k \omega_{kji} x_k + \theta_{ji} \quad \text{and} \quad y_i = \sigma(\prod_j h_{ji}),
\]

where \( h_{ji} \) is the output of the \( j \)-th sigma unit for the \( i \)-th output element \( y_i \), and \( \omega_{kji} \) is the weight of the \( j \)-th sigma unit associated with the input element \( x_k \). A pi-sigma network is intrinsically a shallow quadratic network. Along this direction, [21] removed all cubic and higher-order terms and proposed to use the annealing technique to find optimal quadratic terms.

Recently, higher-order units were revisited [22]–[26]. In [22], a quadratic convolutional filter of the complexity \( O(n^2) \) was proposed to replace the linear filter, while in [23], a parabolic neuron: \( \sigma\left((x^\top w^r + b^r)(x^\top w^q + b^q)\right) \) was proposed for deep learning, which is the special case of our neuron when the power term is dropped. In the work by Chrysos et al. [24], the higher-order units as described by Eq. (3) were embedded into a deep network to reduce the complexity of the individual unit via tensor decomposition and factor sharing. Such a network achieved cutting-edge performance on several tasks. Compared to [24], our group proposed a simplified quadratic neuron with \( O(3n) \) parameters and argued that more complicated neurons are not necessary based on fundamental theorem of algebra [27]. Interestingly, when only the first and second-order terms are kept, and the rank is set to two in tensor decomposition, the formulation of the polynomial neuron in [24] is \( \sigma\left((x^\top w^r + b^r)(x^\top w^q + b^q) + x^\top w^s + b^s\right) \). In this regard, the polynomial neuron in [24] is also the special case of our quadratic neuron.

On the other hand, neurons with polynomial activation [25], [26] are also relevant. However, the polynomially activated neurons are essentially different from polynomial neurons. In the former case, the networks use a polynomial activation but their neurons are still characterized by a piece-wise linear decision boundary, while in the latter case, a polynomial decision boundary is implied that can truly extract nonlinear features. Kileel et al. [28] found that a network with polynomial

II. RELATED WORK

Polynomial networks were investigated in the late 90s. The idea of polynomial networks can be traced back to the Group Method of Data Handling (GMDH [15]), which learns gradually a complicated feature extractor:

\[
\text{Fig. 1. The performance of a quadratic network trained using the proposed ReLinear method, with an observed improvement than the conventional network of the same structure. Furthermore, let each quadratic neuron evolve from a conventional neuron gradually. Moreover, regularization is imposed to control nonlinear terms of a quadratic neuron.}
\]

Main Contributions. In this paper, we present two theorems to demonstrate the superiority of quadratic networks in functional expressivity. Our results show not only that a quadratic network is powerful in the deep learning armory but also that a network with quadratic activation is sub-optimal. Of great practical importance, we propose a novel training strategy to optimize quadratic networks. Finally, we conduct comprehensive experiments to demonstrate that the quadratic network trained with the proposed training strategy can perform competitively on well-known datasets.
activation is an algebraic variety, and proposed the dimension of the algebraic variety to measure the representation power of such a network.

Development of quadratic networks. In a theoretical perspective, the representation capability of quadratic networks was partially addressed in the analyses on the role of multiplicative operations in a network [29], where the incorporation of multiplicative interactions can strictly enlarge the hypothesis space of a feedforward neural network. Fan et al. [10] showed that a quadratic network can approximate a complicated radial function with a more compact structure than a conventional model. Increasingly more results are emerging on quadratic networks applications. For example, Nguyen et al. [30] applied quadratic networks to predict the compressive strength of foamed concrete. By et al. [31] applied a quadratic network to solve forward and inverse problems in partial differential equations (PDEs).

III. EXPRESSIONNESS OF QUADRATIC NETWORKS

Given an input \( x \in \mathbb{R}^n \), a quadratic neuron of our interest is characterized as

\[
\sigma(q(x)) = \sigma\left( \sum_{i=1}^{n} w_i^r x_i + b^r \right) \left( \sum_{i=1}^{n} w_i^q x_i + b^q \right) \left( x \odot x \right)^T \left( w^b + c \right),
\]

where \( \sigma(\cdot) \) is a nonlinear activation function (hereafter, we use \( \sigma(\cdot) \) to denote ReLU), \( \odot \) denotes the Hadamard product, \( w^r, w^q, w^b \in \mathbb{R}^n \), and \( b^r, b^q, c \in \mathbb{R} \) are biases. We use the superscription \( r, q, b \) for better distinguishment. For a univariate input, we have

\[
q(x) = (w^r x + b^r)(w^q x + b^q) + w^b x^2 + c.
\]

A network with higher expressiveness means that this network can either express more functions or express the same function more accurately. In this section, we show the enhanced expressiveness of our quadratic network relative to either a conventional network or a conventional network with quadratic activation. For comparison with a conventional network, we note that in spline theory, a polynomial spline has a significantly more accurate approximation power than the linear spline. Since a quadratic network can express a polynomial spline and a conventional network corresponds to a linear spline, a quadratic network is naturally more powerful than a conventional network. As far as conventional networks with quadratic activation are concerned, we leverage the dimension of algebraic variety as the model expressiveness measure defined in [28] to demonstrate that our quadratic network has a higher dimension of algebraic variety, which suggests that a quadratic network is more expressive than a conventional network with quadratic activation.

A. Spline Theory

Let \( f \) be a function in \( C^{n+1}[a, b] \) and \( p \) be a polynomial to interpolate the function \( f \) according to \( n + 1 \) distinct points \( x_0, x_1, \ldots, x_n \in [a, b] \). Then, for any \( x \in [a, b] \), there exists a point \( \xi_x \in [a, b] \) such that

\[
f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi_x) \prod_{i=0}^{n} (x - x_i).
\]

For some function such as the Runge function \( R(x) = 1/(1 + 16x^2) \), as the degree of the polynomial \( p(x) \) increases, the interpolation error goes to infinity, i.e.,

\[
\lim_{n \to \infty} \left( \max_{-1 \leq x \leq 1} |f(x) - p_n(x)| \right) = \infty.
\]

This bad behavior is referred to as the Runge phenomenon [32], which is explained by two reasons: 1) As \( n \) grows, the magnitude of the \( n \)-th derivative increases; and 2) large distances between the points makes \( \prod_{i=0}^{n} (x - x_i) \) huge.

To overcome the Runge phenomenon when a high-order polynomial is involved for interpolation, the polynomial spline [33] is invented to partition a function into pieces and fit a low-order polynomial to a small subset of points in each piece. Given a set of instances \( \{(x_i, f(x_i))\}_{i=0}^{n} \) from the function \( f(x) \), a polynomial spline is generically formulated as follows:

\[
S(x) = \begin{cases} 
  s_0(x), & x_0 \leq x < x_1 \\
  s_1(x), & x_1 \leq x < x_2 \\
  \vdots & \\
  s_{n-1}(x), & x_{n-1} \leq x \leq x_n
\end{cases},
\]

where \( S(x) \) is a polynomial spline of the order \( 2m - 1 \) (without loss of generality, we consider odd-degree polynomial splines), satisfying that (1) \( s_i(x_{i+1}) = s_{i+1}(x_{i+1}) = f(x_{i+1}) \) for \( i = 0, 1, \ldots, n \); (2) \( s_i^{(2k)}(x_{i+1}) = s_{i+1}^{(2k)}(x_{i+1}) = f^{(2k)}(x_{i+1}) \), \( i = 0, 1, \ldots, n-2, k = 1, 2, \ldots, m-1 \). The simplest piecewise polynomial is a piecewise linear function. However, a piecewise linear interpolation is generically inferior to a piecewise polynomial interpolation in terms of accuracy. To illustrate this rigorously, we have the following lemma:

Lemma 1 ([34]). Let \( S(x) \in C^{2m-2} \) be the \( (2m-1) \)-th degree spline of \( f \), as described by Eq. (8), if \( x_0, x_1, \ldots, x_n \) is a uniform partition with \( x_{i+1} - x_i = h \), then

\[
\|f - S\|_\infty \leq \frac{E_{2m}}{2^{2m}(2m)!} \|f^{(2m)}\|_\infty h^{2m},
\]

where \( \| \cdot \|_\infty \) is the \( l_\infty \) norm and \( E_{2m} \sim (-1)^m \sqrt{\pi \frac{4m}{\pi e}} )^{2m} \) denotes the \( 2m \)-th Euler number. The approximation error is bounded by \( O(h^{2m}) \). For example,

\[
\|f - S\|_\infty \approx 0.25 \cdot h^2, \quad m = 1
\]

\[
\|f - S\|_\infty \approx 0.013 \cdot h^4, \quad m = 2
\]

\[
\|f - S\|_\infty \approx 0.00000137 \cdot h^6, \quad m = 6
\]

This theorem also suggests that, to achieve the same accuracy, high degree splines require less amount of data than linear splines do. To reveal the expressivity of quadratic networks, we have the following proposition to show that a quadratic network can accurately express any univariate polynomial spline but a conventional network cannot. The method
used for the proof is to re-express $S(x)$ into a summation of several continuous functions and use quadratic network modules to express these functions one by one. Finally, we aggregate these modules together into a network, as shown by Figure 2.

![Diagram](image)

**Fig. 2. Illustration of our constructive spline approximation by a quadratic network.**

**Proposition 1** (Universal Spline Approximation). Suppose that high-order terms of the polynomial spline do not degenerate, given a univariate polynomial spline $S(x)$ expressed in Eq. (8), there exists a quadratic network with ReLU activation $Q(x)$ satisfying $Q(x) = S(x)$, while there exists no conventional networks with ReLU activation $L(x)$ such that $L(x) = S(x)$.

**Proof.** Because a function defined by a quadratic network is a continuous function, we need to re-express $S(x)$ into a formulation on continuous functions to allow the construction. Mathematically, we re-write $S(x)$ given in Eq. (8) as

$$S(x) = \sum_{i=0}^{n-1} R_i(x), \quad x \in [x_0, x_n] \quad (11)$$

where for any $i = 0, \cdots, n-1$,

$$R_i(x) = \begin{cases} 0, & x < x_i \\ s_i(x) - s_{i-1}(x), & x \geq x_i \end{cases} \quad (12)$$

For notation consistency, $s_{-1}(x) = 0$. It is straightforward to verify that Eq. (11) is equivalent to Eq. (8). For any $x \in [x_k, x_{k+1}]$,

$$\sum_{i=0}^{n-1} R_i(x) = \sum_{i=0}^{k} R_i(x) = \sum_{i=0}^{k} (s_i(x) - s_{i-1}(x)) = \sum_{i=0}^{k} s_i(x) - s_{i-1}(x) = s_k(x) = S(x). \quad (13)$$

$R_i(x)$ has the following favorable characteristics: 1) It is a truncated function which has zero function value over the domain of $x < x_i$; 2) and due to $s_i(x) - s_{i-1}(x) = 0$, $R_i(x)$ is also a continuous function. Thus, $R_i(x)$ can be succinctly expressed as

$$R_i(x) = s_i(\sigma(x - x_i) + x_i) - s_{i-1}(\sigma(x - x_i) + x_i), \quad (14)$$

where $\sigma(x - x_i) + x_i$ maps $x \in \{x|x \leq x_i\}$ into $x_i$ that has the function value of zero.

Because a quadratic network can represent any univariate polynomial \[10\), we let $Q_i(x) = s_i(x) - s_{i-1}(x)$; then,

$$R_i(x) = Q_i(\sigma(x - x_i) + x_i). \quad (15)$$

Substituting Eq. (15) into Eq. (11), we derive our construction:

$$Q(x) = s_0(x_0) + \sum_{i=0}^{n-1} Q_i(\sigma(x - x_i) + x_i) = S(x), \quad (16)$$

which concludes the proof of the first part of this proposition. For the second part, because a conventional network with ReLU activation is a piecewise linear function, $L(x)$ cannot perfectly represent $S(x)$ as long as high-order terms are non-zero.

**Remark 1.** Although the proof of Proposition 1 is constructive and demands no complicated techniques, Proposition 1 informs us an important message: A polynomial spline is a solution in the hypothesis space of quadratic networks, yet it will not appear in the hypothesis space of conventional networks. This implies that the expressivity of quadratic networks is superior to that of conventional networks, since a piecewise polynomial spline is certainly a better fitting tool than a piecewise linear spline.

**B. Dimension of Algebraic Variety**

To our best knowledge, there exist at least two ways to realize so-called polynomial networks. The first is to utilize a polynomial activation function, while the second one is to take a polynomial function for the aggregation, such as with our quadratic neurons. Despite the confusion due to the use of the same name, two realizations are dramatically different. One natural idea to check the capacity of networks is to compute the dimension of the hypothesis space provided by the network architecture. In algebraic geometry, the set of polynomials, refereed as algebraic variety, have been well studied. To put the superior expressivity of quadratic networks in perspective, we employ the *dimension of algebraic variety*, which was proposed to gauge the expressive power of polynomial networks \[28\], to compare the two realizations. We find that the dimension of algebraic variety of our quadratic network is significantly higher than that of its competitor, which suggests that our quadratic network can represent a richer class of functions than the network using quadratic activation.

**Two realizations.** Assume that a network architecture consists of $H$ layers with their widths specified by a vector $d = (d_0, d_1, d_2, \cdots, d_H)$ respectively, and $x \in \mathbb{R}^{d_0}$. A network with a quadratic activation is a function of the form

$$p_1(x) = l_H \circ \sigma_1 \circ \sigma_0 \circ l_{H-1} \circ \sigma_1 \circ l_{H-2} \cdots \circ \sigma_1 \circ l_1(x), \quad (17)$$

where $p_1(x) \in \mathbb{R}^{d_H}$, $l^h(x) = W^h x + b^h$, and $\sigma_1(z) = z^2$. In contrast, our quadratic network is of the following form:

$$p_2(x) = q_H \circ \sigma_2 \circ q_{H-1} \circ \sigma_2 \circ q_{H-2} \cdots \circ \sigma_2 \circ q_1(x), \quad (18)$$

where $p_2(x) \in \mathbb{R}^{d_H}$, $\sigma_2(z) = z^2$. To simplify our analysis, we use linear activation for our quadratic network.
as \( \sigma_2(x) = \text{ReLU}(x) - \text{ReLU}(-x) \). Given an architecture \( d \), the polynomial network with respect to the weights and biases defines a functional space, and we denote the functional spaces of two realizations as \( \mathcal{F}_{d,1} \) and \( \mathcal{F}_{d,2} \), respectively.

**Dimension of algebraic variety.** In [23], the Zariski closure \( \mathcal{V}_d = \overline{\mathcal{F}_d} \) of \( \mathcal{F}_d \) is considered, where \( \mathcal{V}_d \) is an algebraic variety, and the dimension of \( \mathcal{V}_d \) (dim \( \mathcal{V}_d \)) is a measure to the expressivity of the pertaining network. Although \( \mathcal{V}_d \) is larger than \( \mathcal{F}_d \), their dimensions agree with each other. Moreover, \( \mathcal{V}_d \) is amendable to the powerful analysis tools from algebraic geometry. In the following, based on the results in [23], we provide an estimation for the upper bound of dim \( \mathcal{V}_{d,2} \).

**Lemma 2.** Given an architecture \( d = (d_0, d_1, d_2, \ldots, d_H) \), the following holds:

\[
\text{dim} \mathcal{V}_{d,2} \leq \min \left( \sum_{h=1}^{H} (3d_{h-1}d_h + 3d_{h-1}) - \sum_{h=1}^{H-1} d_h, d_H \left( d_0 + \frac{2H^2 - 1}{2H - 1} \right) \right). 
\]  

(19)

**Proof.** For all diagonal matrices \( D_i \in \mathbb{R}^{d_i \times d_i} \) and permutation matrices \( P_i \in \mathbb{Z}^{d_i \times d_i} \), the function described in [18] returns the same output under the following replacements:

\[
\begin{align*}
\Theta_1 &\leftarrow P_1D_1\Theta_1, \\
\Theta_2 &\leftarrow P_2D_2\Theta_2D_1^{-1}P_1^T, \\
\Theta_3 &\leftarrow P_3D_3\Theta_3D_2^{-1}P_2^T, \\
& \vdots \\
\Theta_H &\leftarrow \Theta_H D_1^{-1}P_1^T,
\end{align*}
\]

(20)

where \( \Theta_h \) represents any element in \( \{W^{h,r}, W^{h,g}, b^{h,r}, b^{h,g}, c^h\} \). As a result, the dimension of a generic fiber of \( p_2(x) \) is at least \( \sum_{h=1}^{H-1} d_h \).

According to [35], the dimension of \( \mathcal{V}_{d,2} \), dim \( \mathcal{V}_{d,2} \), is equal to the dimension of the product of \( p_2(x) \) minuses the dimension of the generic fiber of \( p_2(x) \), which means

\[
\text{dim} \mathcal{V}_{d,2} \leq \sum_{h=1}^{H} (3d_{h-1}d_h + 3d_{h-1}) - \sum_{h=1}^{H-1} d_h. 
\]  

(21)

In addition, dim \( \mathcal{V}_{d,2} \) is at most the number of terms of \( p_2(x) \), which means

\[
\text{dim} \mathcal{V}_{d,2} \leq d_H \left( d_0 + \frac{2H^2 - 1}{2H - 1} \right). 
\]  

(22)

Combining the above two formulas, we conclude this proof.

For the same architecture, the upper bound provided by the network with quadratic activation [23] is

\[
\min \left( \sum_{h=1}^{H} (d_{h-1}d_h + d_{h-1}) - \sum_{h=1}^{H-1} d_h, d_H \left( d_0 + \frac{2H^2 - 1}{2H - 1} \right) \right). 
\]  

(23)

This bound is lower than what we derived in Eq. [19]. In addition to the upper bound comparison, we have the following proposition to directly compare dim \( \mathcal{V}_{d,2} \) and dim \( \mathcal{V}_{d,1} \).

**Proposition 2.** Given the same architecture \( d = (d_0, d_1, d_2, \ldots, d_H) \), we have

\[
\text{dim} \mathcal{V}_{d,2} > \text{dim} \mathcal{V}_{d,1}. 
\]  

(24)

**Proof.** It can be shown that by the following substitutions:

\[
\begin{align*}
W^{h,r} &\leftarrow W^{h,g} \quad b^{h,r} \leftarrow b^{h,g} \quad c \leftarrow 0,
\end{align*}
\]

(25)

\( p_2(x) \) turns into \( p_1(x) \), which means \( \mathcal{F}_{d,1} \subset \mathcal{F}_{d,2} \). Moreover, getting \( p_2(x) \) from \( p_1(x) \) is difficult because we need to construct interaction terms \( x_i x_j \) from \( p_1(x) \). Generally, representing a single quadratic neuron with neurons using quadratic activation requires a good number of neurons:

\[
\sum_{i=1}^{n} w_i^r x_i + \sum_{i=1}^{n} w_i^g x_i + \sum_{i=1}^{n} w_i^b x_i^2 + c
\]

\[
= \sum_{k \neq l} (w_k^r w_l^g + w_k^g w_l^r)x_k x_l + \sum_{k} (w_k^b b^r + w_k^b b^g)x_k + \sum_{k} (w_k^b b^r + w_k^b b^g)x_k^2 + c
\]

\[
= \sum_{k \neq l} (A_k x_k + B_k x_l)^2 + \sum_{k} (C_k x_k - D_k)^2 + \sum_{k} E_k x_k^2,
\]

(26)

where \( A_k, B_k, C_k, D_k, E_k \) are coefficients.

From another meaningful angle, \( p_1(x) \) can only get degree 2\( k \) polynomials, while \( p_2(x) \) can be much more flexible to get an arbitrary degree polynomial because there is a product operation in the quadratic neuron. Therefore, \( p_1(x) \) can never represent \( p_2(x) \). As a result, given the same network structure, \( \mathcal{V}_{d,1} \subset \mathcal{V}_{d,2} \), thereby we can conclude that \( \text{dim} \mathcal{V}_{d,2} > \text{dim} \mathcal{V}_{d,1} \).
Despite the superior expressivity, the training of a quadratic network may collapse, which prevents a quadratic network from achieving its full potential. When randomly initializing parameters of a quadratic network, the training is typically unstable: sometimes the model yields an exploding magnitude of the output; and in some other cases the training curve oscillates. Likely, this is because a quadratic term is nonlinear, and the composition of quadratic operations layer by layer produces a function of an exponentially high degree, causing the instability of the training process. As such, although a quadratic operation is more powerful and promises superior performance, it is necessary to balance model scalability and training stability. In this context, controlling quadratic terms is instrumental to the performance of the model, since how nonlinear a model should depend on a specific task.

A. The Training Strategy: ReLinear

To control the quadratic terms, we propose the ReLinear (referenced linear initialization), which encourages the model to learn suitable quadratic terms gradually and adaptively in reference to the corresponding linear terms. The ReLinear method has the following two steps. First, the quadratic weights in each neuron are set to $w^g = 0, b^g = 1$ and $w^b = 0, c = 0$. Such an initialization degenerates a quadratic neuron into a conventional neuron. Second, quadratic terms are regularized in the training process. Intuitively, two ways of regularization: shrinking the gradients of quadratic weights (ReLinear$^g$); and shrinking quadratic weights (ReLinear$^w$). Let $\gamma_r, \gamma_g$, and $\gamma_b$ be the learning rates for updating $w^r, b^r$, $w^g, b^g$ and $w^b, c$ respectively, $\alpha_g, \alpha_b$ be the weight factors of $w^g$ and $w_b$, and $\beta_g, \beta_b$ be the weight decay rates for $w^g$ and $w_b$ respectively. In Table I we summarize the key points of the proposed training strategy.

Specifically, for ReLinear$^g$ in Table I we set different learning rates for $w^r, b^r$ and $w^g, b^g, w^b, c$, where the learning rate for the former keeps intact as the conventional network, while the learning rate for the latter adjusts quadratic nonlinearity. Suppose that the learning rate of $w^g, b^g, w^b, c$ is zero, then quadratic terms are never updated, and a quadratic model actually turns into a linear model. As this learning rate increases, the quadratic terms become more significant. By setting the learning rate of $w^g, b^g, w^b, c$ to a smaller value, we can prevent magnitude explosion while utilizing the quadratic nonlinearity.

For ReLinear$^w$ in Table I a straightforward way is to use $l_2$ norms for $w^g, b^g, w^b, c$, and shrink the quadratic weights at each iteration, instead of cropping their gradients.

We argue that shrinking the gradients of quadratic terms (ReLinear$^g$) is better than shrinking quadratic terms (ReLinear$^w$). Shrinking quadratic terms is to adjust quadratic terms along the directions of those parameters, which may not decrease the loss due to their deviation from the directions of the gradients. In contrast, shrinking the gradients respects the directions of the gradients, thereby reducing the loss.

Furthermore, regarding the parameters $w^r, b^r$, we can use either random initialization or weight transfer that is to train a conventional network sharing the same structure of a quadratic network and then transfer the learned parameters into the quadratic network. Thus, in weight transfer, $w^r, b^r$ in each quadratic neuron are initialized by the parameters of the corresponding conventional neuron. In contrast to the random initialization, the weight transfer has an extra computational cost to train the conventional model. If the conventional model of the same structure needs to be trained, we estimate that the total cost will increase by around 20% because the number of multiplications of a conventional neuron is 20% of that of a quadratic neuron.

Remark 3. The pre-trained models are widely used in many computationally intensive or data-insufficient scenarios [37]. For example, in transfer learning, the representation and knowledge from a pre-trained model for a task can facilitate a model for another task. To train a quadratic network, we may also use a pre-trained conventional model for weight transfer. However, doing so is not an embodiment of transfer learning, as the pre-trained model is from the same task.

In our numerical experiments, consistent with Figure 1, the quadratic network trained via ReLinear always outperforms the conventional network of the same structure. Because when $\gamma_g = \gamma_b = 0$ or $\alpha_g = \alpha_b = 0$ or $\beta_g = \beta_b = 0$, the quadratic network will be a conventional network, therefore at least delivering the same performance as the conventional network. As $(\gamma_g, \gamma_b)$ or $(\alpha_g, \alpha_b)$, or $(\beta_g, \beta_b)$ gradually increases, the quadratic terms are preferably evolving to extract features and refine the workflow, making the model generally better than the corresponding conventional model. Of course, $(\gamma_g, \gamma_b)$ or $(\alpha_g, \alpha_b)$, or $(\beta_g, \beta_b)$ should not be too large or small; otherwise, the quadratic terms would be insignificant or too aggressive. Moreover, tuning these hyperparameters is simple.

<table>
<thead>
<tr>
<th>Table I</th>
<th>Proposed training strategy. ReLinear$^g$ uses shrinking gradients, while ReLinear$^w$ works with shrinking weights.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initialization $w^r, b^g, w^g, c$, Learning Rate $\gamma_r, \gamma_g, \gamma_b$, Updating Equation $w^r = w^r - \gamma_r \frac{\partial L}{\partial w^r}$, $w^g = w^g - \gamma_g \frac{\partial L}{\partial w^g}$, $w^b = w^b - \gamma_b \frac{\partial L}{\partial w^b}$</td>
</tr>
<tr>
<td>ReLinear$^g$</td>
<td>0, 1, 0, 0</td>
</tr>
<tr>
<td>ReLinear$^w$-I1</td>
<td>0, 1, 0, 0</td>
</tr>
<tr>
<td>ReLinear$^w$-I2</td>
<td>0, 1, 0, 0</td>
</tr>
<tr>
<td>Updating $b^r, b^g, c$ can be similarly done in reference to the equations for updating $w^r, w^g$ and $w^b$ respectively.</td>
<td></td>
</tr>
</tbody>
</table>
to implement and will not reduce the user friendliness.

**B. Mechanism Analysis**

In this part, we theoretically shed light on why the proposed ReLinear can avoid the magnitude explosion during the training of quadratic networks at the early stage. We also analyze the convergence behavior of the proposed ReLinear and its corresponding convergence rate.

**Stabilizing the training.** For conciseness and convenience, notations in Eqs. (17) and (18) are inherited. We formulate a fully-connected conventional network as

$$f_1(x) = l_H \circ \sigma_1 \circ l_{H-1} \circ \sigma_1 \circ l_{H-2} \cdots \circ \sigma_1 \circ l_1(x),$$

(27)

where \(l^b(x) = W^b x + b^b, W^h \in \mathbb{R}^{d_h \times d_{h-1}}, b^h \in \mathbb{R}^{d_h}, \) and \(\sigma_1(z) = \max\{z, 0\}\). A fully-connected quadratic network is formulated as

$$f_2(x) = q_H \circ \sigma_2 \circ q_{H-1} \circ \sigma_2 \circ q_{H-2} \cdots \circ \sigma_2 \circ q_1(x),$$

(28)

where \(q_h(x) = (W^{h,r} x + b^{h,r}) \odot (W^{h,g} x + b^{h,g}) + W^{h,b}(x \odot x) + c^h, W^{h,r}, W^{h,g}, W^{h,b} \in \mathbb{R}^{d_h \times d_{h-1}}, b^{h,r}, b^{h,g}, c^h \in \mathbb{R}^{d_h}, h = 1, 2, \ldots, H, d_H = 1, \) and \(\sigma_2(z) = \sigma_1(z)\).

In our experiments, we find that the training loss of a normally trained quadratic network can vary very large in the early stage because the output of the network is large. We think this is due to the nonlinearity amplification of quadratic terms to the input. The reason why the ReLinear can stabilize the training of quadratic networks is its ability to suppress the quadratic terms properly. To dissect the hidden mechanism, we first conduct Taylor expansion around 0 for conventional and quadratic networks by keeping the linear term (A ReLU network is not differentiable, but w.l.o.g., we can assume the ReLU is approximated by a smooth function):

$$f(\Delta x) \approx f(0) + \frac{\partial f}{\partial x} |_{x=0} \Delta x.$$  

(29)

Naturally, \(\| \frac{\partial f}{\partial x} \|_{x=0} \) is used to measure the amplification effect. The derivatives of \(f_1(x)\) and \(f_2(x)\) are computed as follows:

$$\| \frac{\partial f_1}{\partial x} \|_{x=0} = \| W^H \circ \sigma'((u^{H-1})^T W^{H-1} \circ \sigma'(u^{H-2})^T \cdots W^1) \| \leq \| W^H \| \| W^{H-1} \| \cdots \| W^1 \|$$

(30)

and

$$\| \frac{\partial f_2}{\partial x} \|_{x=0} = \| (W^{H,r} \circ (W^{H,g} u^{H-1} + b^{H,g}) + W^{H,g} \circ (W^{H,r} u^{H-1} + b^{H,r}) + 2 W^{H,b} \circ (W^{H,r} x + b^{H,r})) \| \leq \| (W^{H,r} \circ (W^{H,g} u^{H-1} + b^{H,g}) + W^{H,g} \circ (W^{H,r} u^{H-1} + b^{H,r}) + 2 W^{H,b} \circ (W^{H,r} x + b^{H,r})) \|,$$

(31)

where \(u^h\) is the output of the \(h\)-th layer of networks, and the Hadamard product\(^1\) between the given matrix and vector corresponds to multiply every column of the matrix by the corresponding element of the vector. Because \(x\) is around 0, we assume that \(u^h\) is infinitesimal, then Eq. (31) is reduced into

$$\| \frac{\partial f_2}{\partial x} \|_{x=0} \leq \| (W^{H,r} \circ b^{H,g} + W^{H,g} \circ b^{H,r}) \| \cdots \| (W^{1,r} \circ b^{1,g} + W^{1,g} \circ b^{1,r}) \|.$$

(32)

Per the initialization of the ReLinear, when \(W^{h,g} = 0, b^{h,r} = 1\), \(\| \frac{\partial f}{\partial x} \| = \| \frac{\partial f_2}{\partial x} \|\), which means that at the beginning, the amplification effects of conventional and quadratic networks are equal.

Furthermore, we derive the Frobenius norms of all concerning matrices and vectors in Eqs. (30) and (32). Suppose that the parameters of two networks are i.i.d. Gaussian distributed with unit variance, the typical magnitude of the Euclidean norm of \(W \in \mathbb{R}^{m \times n}\) is \(O(\sqrt{mn})\), and the typical magnitude of the Euclidean norm of \(W \circ b\), where \(W \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^{n \times 1}\) is also \(O(\sqrt{mn})\). Thus, we estimate the magnitudes of \(\| \frac{\partial f_1}{\partial x} \|_{x=0}\) and \(\| \frac{\partial f_2}{\partial x} \|_{x=0}\) as

$$\begin{cases} \| \frac{\partial f_1}{\partial x} \|_{x=0} \approx \prod_{h=1}^{H} d_H d_{h-1} \\ \| \frac{\partial f_2}{\partial x} \|_{x=0} \approx \prod_{h=1}^{H} 2 \sqrt{d_H d_{h-1}} = 2^H \prod_{h=1}^{H} \sqrt{d_H d_{h-1}}. \end{cases}$$

(33)

Based on Eq. (33), the amplification effect of a quadratic network is exponentially higher than that of a conventional network, which accounts for the magnitude explosion of a quadratic network during training. The ReLinear regularizes quadratic terms by a prescribed initialization and weight/gradient shrinkage, thereby favorably avoiding the exponential growth of the amplification effect at least at the early training stage. This is why and how the ReLinear can work.

**Convergence.** The ReLinear strategy is essentially the stochastic gradient descent (SGD) with a specific initialization and adaptive learning rates for different parameters. Since the convergence behavior of SGD has little to do with the initialization, the convergence behavior is mainly determined by adaptive learning rates. In Appendix, aided by the proofs in [38], we show under what conditions, the proposed ReLinear can converge and the associated convergence rate.

V. EXPERIMENTS

In this section, we first conduct analysis experiments (Runge function approximation and image recognition) to show the effectiveness of the ReLinear strategy in controlling quadratic terms and to analyze the performance of different schemes for the proposed ReLinear method. Then, encouraged by our theoretical analyses, we compare quadratic networks with conventional networks and the networks with quadratic activation to show that a quadratic network is a competitive model.

A. Analysis Experiments

1) Runge Function Approximation: As mentioned earlier, a polynomial spline is used to replace a complete polynomial to overcome the Runge phenomenon. Since a quadratic network with ReLU activation is a polynomial spline, we implement a fully connected quadratic network to approximate the Runge function to verify the feasibility of our proposed training
strategy in suppressing quadratic terms. This experiment is to approximate a univariate function, which enables us to conveniently compute the degree of the output function produced by a quadratic network and monitor its change.

In total, 33 points are sampled from $[-5, 5]$ with an equal distance. The width of all layers is 8. The depth is 5 such that the degree of the output function is $2^5 = 32$, meeting the minimum requirement to fit 33 instances. We compare the proposed strategy (ReLinear$^{sw}$-l1, ReLinear$^{sw}$-l2 and ReLinear$^{sg}$) with the regular training. In ReLinear$^{sw}$-l1, $\gamma_r = \gamma_g = \gamma_b = 0.0003$ and $\alpha_g = \alpha_b = 0.0001$. In ReLinear$^{sw}$-l2, $\gamma_r = \gamma_g = \gamma_b = 0.0003$ and $\beta_g = \beta_b = 0.0001$. In ReLinear$^{sg}$, the learning rates are set as $\gamma_r = 0.0003, \gamma_g = \gamma_b = 0.00015$. In contrast, we configure a learning rate of $\gamma_r = \gamma_g = \gamma_b = 0.0003$ for all parameters in regular training. The total iteration is 30000 to guarantee convergence.

![Figure 4](image)

(a) Fitting Results
(b) Polynomials from QNN
(c) Polynomials from QNN(IWL$^{sw}$-l1)
(d) Polynomials from QNN(IWL$^{sw}$-l2)
(e) Polynomials from QNN(IWL$^{sg}$)

The results are shown in Figure 4. As a spline can avoid the Runge phenomenon, regardless of how the QNN with ReLU activation is trained, it can fit the Runge function desirably without oscillations at edges, as shown in Figure 4(a). Furthermore, in Figure 4(b)-(e), we examine coefficients of randomly selected polynomials contained by functions of QNNs at different pieces. It is observed from Figure 4(b) that the polynomials associated with the regular training have unignorable high-order terms. This is counter-intuitive because a QNN partitions the interval into many pieces (24 pieces based on our computation). Since only a few samples lie in each piece, it suffices to use a low-degree polynomial in each piece. This might be due to that the space of low-degree polynomials is a measure-zero subspace in the high-degree polynomials. Thus, it is hard to obtain a low-degree polynomial fit straightforwardly. Next, we observe that coefficients of high degrees are significantly suppressed in Figure 4(c)-(e) than those in Figure 4(b). At the same time, the magnitudes of coefficients of low degrees in Figure 4(c)-(e) are put down. Such observations imply that all the proposed strategies can effectively control the quadratic terms as expected.

Next, we quantitatively compare the approximation errors of different training strategies using the rooted mean squared error (RMSE). We evenly sample 100 instances from $[-5, 5]$ as test samples, none of which appears in the training. Table II shows RMSE values of three realizations for ReLinear. It can be seen that ReLinear$^{sg}$ achieves the lowest error, suggesting that ReLinear$^{sg}$ is better at balancing between suppressing quadratic terms and maintaining approximation precision.

2) Image Classification: In the preceding part, we show that the proposed strategies can suppress the high-order terms by explicitly examining the output function of a quadratic network. Here, we focus on an image recognition task to further confirm the effectiveness of the proposed strategy. We build a quadratic ResNet (QResNet for short) by replacing conventional neurons in ResNet with quadratic neurons and keeping everything else intact. We train the QResNet on the CIFAR10 dataset. Our motivation is to gauge the characteristics and performance of the proposed different realizations of the ReLinear method through experiments.

Following configurations of [3], we use batch training with a batch of 128. The optimization is stochastic gradient descent using Adam [39]. $\gamma_r$ is set to 0.1. The total number of epochs is 200. In the 100-th and 150-th epoch, $\gamma_r, \gamma_g, \gamma_b$ decrease 1/10 at the same time. Because training curves share the same trend with the testing curves, we only show the testing curves here for conciseness.

Tuning ReLinear$^{sg}$. Here, with QResNet20, we study the impact of $\gamma_g, \gamma_b$ on the effectiveness of ReLinear$^{sg}$. Without loss of generality, we set $\gamma_g = \gamma_b$. The lower $\gamma_g$ and $\gamma_b$ are, the more the quadratic weights are constrained. We respectively set $\gamma_g, \gamma_b$ to $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ for a comprehensive analysis. The resultant accuracy curves are shown in Figure 5. It can be seen that when $\gamma_g$ is large ($\gamma_g = \gamma_b = 10^{-1}, 10^{-2}, 10^{-3}$), the training is quite unstable, the accuracy score jumps severely. However, as $\gamma_g$ and $\gamma_b$ go low, i.e., $\gamma_g = \gamma_b = 10^{-4}, 10^{-5}$, the training curves become stabilized, mirroring that the high-order terms are well controlled. The best performance (error 7.78%) is achieved when $\gamma = 10^{-4}$, consistent with the trend in Figure 1.

Tuning ReLinear$^{sw}$. Here, we investigate the impact of different parameters for ReLinear$^{sw}$-l1 and ReLinear$^{sw}$-l2, respectively. For both ReLinear$^{sw}$-l1 and ReLinear$^{sw}$-l2, $\gamma_r, \gamma_g$, and $\gamma_b$ are set to 0.1, and decay 1/10 at epochs 100 and 150 at the same time. Concerning parameters of ReLinear$^{sw}$-l1, let $\alpha_g = \alpha_b$ which are respectively set to $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ for comprehension. For ReLinear$^{sw}$-l2, we cast $\beta_g = \beta_b$ to $\{0.1, 0.5, 0.9, 0.99\}$, respectively. If the $\beta_g = \beta_b$ equal to 1, the quadratic weights...
will oscillate around 0. The accuracy curves for \text{ReLinear}\textsuperscript{\textit{w}}-\textit{l}_1 and \text{ReLinear}\textsuperscript{\textit{w}}-\textit{l}_2 are shown in Figure 6. It is observed that the \textit{l}_1-norm is not good at stabilizing the training in this task, while an appropriate \textit{l}_2 norm, i.e., \( \beta = 0.9, 0.99 \) manages to eliminate the large oscillation. The lowest error 8.21\% of \text{ReLinear}\textsuperscript{\textit{w}} is achieved by the \textit{l}_2 norm at \( \beta_2 = \beta = 0.9 \), which is worse than the lowest error 7.78\% of \text{ReLinear}\textsuperscript{\textit{g}}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig6}
\caption{Left: accuracy curves obtained from different parameters \((\gamma_g, \gamma_h)\) for \text{ReLinear}\textsuperscript{\textit{g}}. Right: accuracy curves obtained from different parameters \((\beta_g, \beta_h)\) for \text{ReLinear}\textsuperscript{\textit{w}}-\textit{l}_2.}
\end{figure}

**Weight Transfer.** As mentioned earlier, weight transfer can also be used to train a quadratic network. Because the training of a conventional ResNet has three stages (1-100 epochs, 101-150 epochs, 151-200 epochs), weight transfer also has three choices, corresponding to transferring weights from which stage. We evaluate all three choices. After transferring, the learning rate \( \gamma_r \) will inherit the learning rate of the transferred model and then decay 1/10 when the training moves to the next stage. Still, we set \( \gamma_g = \gamma_h \) to \{10\textsuperscript{-1}, 10\textsuperscript{-2}, 10\textsuperscript{-3}, 10\textsuperscript{-4}, 10\textsuperscript{-5}\} for a comprehensive analysis. Here we show the accuracy curves in Figure 7. There are two observations from Figure 7. First, transferred parameters can stabilize the training provided appropriate \((\gamma_g, \gamma_h)\). For the same \( \gamma_g \), weights transferred from the later stage make the training more robust than those transferred from the earlier stage. This is because transferred parameters from the later stages have been good for the model, there is less need to optimize the quadratic terms, thereby avoiding the risk of explosion. The second highlight is that the best performance comes from weights transferred from the first stage, which suggests that the model can be improved if quadratic terms play a significant role. The lowest errors by transferring from three stages are 7.18\%, 7.5\%, and 7.75\%.

\textbf{ReLinear+ReZero} Specially, for training a residual quadratic network, the proposed \text{ReLinear} method can be integrated with the recent proposal called \text{ReZero} (residual with zero initialization, \cite{14}), which dedicates to training a residual network by revising the residual propagation formula from \( x^{h+1} = x^h + F(x) \) to \( x^{h+1} = x^h + \zeta_h F(x) \), where \( \zeta_h \) is initialized to be zero such that the network is an identity at the beginning. \text{ReZero} can not only speed up the training but also improve the performance of the model. Here, we evaluate the feasibility of training a quadratic residual network with \text{ReLinear+ReZero}. We adopt \text{ReLinear}\textsuperscript{\textit{g}} and let \( \gamma_g = \gamma_h \). We respectively set \( \gamma_g, \gamma_h \) to \{10\textsuperscript{-1}, 10\textsuperscript{-2}, 10\textsuperscript{-3}, 10\textsuperscript{-4}\}. The results are shown in Figure 8. Comparing Figures 8 and 5 we surprisingly find that QResNet20 trained via \text{ReLinear+ReZero} is more stable. Previously, when \( \gamma_g = \gamma_h = 10^{-2}, 10^{-3} \), the accuracy curves from the only \text{ReLinear} still suffer unrest oscillations, while curves from the \text{ReLinear+ReZero} do not.

3) \textbf{Training Stability:} Here, we compare our quadratic model with a model using quadratic activation on the CIFAR100 dataset. The VGG16 \cite{40} is used as the test bed. We obtain the VGG using quadratic activation by directly revising activation function, and we prototype the Quadratic-VGG16 by replacing the conventional neurons with quadratic neurons in the VGG16. All training protocols of the quadratic VGG16 and the VGG16 using quadratic activation are the same to the authentic VGG16. We test three learning rates for the VGG16 using quadratic activation: 0.01, 0.03, 0.05. For the Quadratic-VGG16, we set the learning rate \( \gamma_r \) to 0.1 and \( \gamma_g, \gamma_h \) to 10\textsuperscript{-6}. The results are shown in Table III. We find that the VGG16 using quadratic activation does not converge regardless of different learning rates, therefore, no errors can be reported. Indeed, direct training a network using quadratic activation suffers the magnitude as well due to the exponentially high degree. Preferably, this problem is overcome in our quadratic network aided by the proposed training strategy.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Network} & \textbf{Error (%)} \\
\hline
VGG16(\text{quadratic activation}) lr=0.05 & no converge \\
VGG16(\text{quadratic activation}) lr=0.03 & no converge \\
VGG16(\text{quadratic activation}) lr=0.01 & no converge \\
\hline
Quadratic-VGG16 & 28.33 \\
\hline
\end{tabular}
\caption{\textbf{Error(\%)} of the quadratic VGG and VGG using quadratic activation on CIFAR100 validation set.}
\end{table}

\textbf{B. Comparative Study} Here, we validate the superiority of a quadratic network over a conventional network, with experiments on two classification data sets: CIFAR10 and ImageNet. The quadratic network is implemented as a drop-in replacement for the conventional network, which means that the only difference is the neuron type. Despite the straightforward replacement, aided by the proposed ReLinear, a quadratic network performs much better than its counterpart. Moreover, we implement a compact version of quadratic networks, which also demonstrates competitive performance.
The 1st Stage

The 2nd Stage

The 3rd Stage

Fig. 7. Accuracy curves from different learning rates by transferring weights from different stages.

CIFAR10. In this experiment, we systematically compare our QResNet with the ResNet. We follow the same protocol as the ResNet to train the QResNet, such as batch size, epoch number, and so on. As implied by our preceding experimental results that ReLinear$^g$ is generally better than ReLinear$^w$, we adopt ReLinear$^g$. $\gamma_g = \gamma_b = 10^{-4}, \gamma_g = \gamma_b = 10^{-5}$, $\gamma_g = \gamma_b = 10^{-5}, \gamma_g = \gamma_b = 10^{-5}$ are set for QResNet20, QResNet32, QResNet56, and QResNet110, respectively. For all quadratic models, we test the weight transfer (the first stage) and a random initialization for their linear parts. We also implement ReLinear$^g$+ReZero, where $\gamma_g = \gamma_b = 10^{-5}, \gamma_g = \gamma_b = 10^{-4}, \gamma_g = \gamma_b = 10^{-4}, \gamma_g = \gamma_b = 10^{-6}$ are set for QResNet20, QResNet32, QResNet56, and QResNet110, respectively. Table [V] summarizes the results. Regardless of ways of initialization, all quadratic models are better than their counterparts, which is consistent with our analyses that quadratic neurons can improve model expressibility. Again, the improvement by quadratic networks is warranted because the employment of the proposed strategy makes the conventional model a special case of quadratic models. At least, a quadratic model will deliver the same as the conventional model. Furthermore, combined with the weight transfer, a quadratic network trained via ReLinear$^g$ can surpass a conventional network by a larger margin, which suggests that a well-trained conventional model can effectively guide the training of a quadratic network.

In addition, we also compare QResNet with other polynomial residual networks [23], [24], which are referred to as Parabolic-ResNet and PolyResNet, respectively. Both Parabolic-ResNet and PolyResNet are trained with the ReLinear algorithm because normally-trained Parabolic-ResNet and PolyResNet do not converge. Results are summarized in Table [IV] It can be seen that as the depth increases, the performance of Parabolic-ResNet and PolyResNet does not necessarily improve. Moreover, with the same architecture, our quadratic models are consistently and substantially better than Parabolic-ResNet and PolyResNet. Since the key difference between our neuronal design and the designs in [23], [24] is the extra power term. Our results imply that the extra power term is indispensable.

Currently, an individual quadratic neuron has 3 times parameters relative to an individual convention neuron, which causes that a quadratic model is three times larger than the conventional model. To reduce the model complexity, we simplify the quadratic neuron by eradicating interaction terms, leading to a compact quadratic neuron: $q(x) = \sigma(x^\top w_r + (x \circ x)^\top w_c + c)$. The number of parameters in a compact quadratic neuron is twice that in a conventional neuron. Similarly, as a drop-in replacement, we implement the QResNet20, QResNet32, and QResNet56 with compact quadratic neurons referred to as Compact-QResNet, and compare them with conventional models. The linear parts $w_r, c$ are initialized with a conven-
TABLE IV
IMAGE CLASSIFICATION BY RESNET AND QRESNET ON CIFAR10 VALIDATION SET.

<table>
<thead>
<tr>
<th>Network</th>
<th>Params</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet20</td>
<td>0.27M</td>
<td>8.75</td>
</tr>
<tr>
<td>ResNet32</td>
<td>0.46M</td>
<td>7.51</td>
</tr>
<tr>
<td>ResNet56</td>
<td>0.86M</td>
<td>6.97</td>
</tr>
<tr>
<td>ResNet110</td>
<td>1.7M</td>
<td>6.61</td>
</tr>
<tr>
<td>ResNet1202</td>
<td>19.4M</td>
<td>7.93</td>
</tr>
<tr>
<td>Parabolic-ResNet20</td>
<td>0.54M</td>
<td>7.89</td>
</tr>
<tr>
<td>Parabolic-ResNet32</td>
<td>0.54M</td>
<td>7.25</td>
</tr>
<tr>
<td>Parabolic-ResNet56</td>
<td>0.54M</td>
<td>8.89</td>
</tr>
<tr>
<td>Poly-ResNet20</td>
<td>0.54M</td>
<td>9.80</td>
</tr>
<tr>
<td>Poly-ResNet32</td>
<td>0.54M</td>
<td>7.62</td>
</tr>
<tr>
<td>Poly-ResNet56</td>
<td>0.54M</td>
<td>8.31</td>
</tr>
<tr>
<td>QResNet20 (r. i., ReLinear)</td>
<td>0.81M</td>
<td>7.78</td>
</tr>
<tr>
<td>QResNet20 (r. i., ReLinear+ReZero)</td>
<td>0.81M</td>
<td>7.97</td>
</tr>
<tr>
<td>QResNet20 (w. t., ReLinear)</td>
<td>0.81M</td>
<td>7.17</td>
</tr>
<tr>
<td>QResNet32 (r. t., ReLinear)</td>
<td>1.39M</td>
<td>7.18</td>
</tr>
<tr>
<td>QResNet32 (r. t., ReLinear+ReZero)</td>
<td>1.39M</td>
<td>6.90</td>
</tr>
<tr>
<td>QResNet32 (w. t., ReLinear)</td>
<td>1.39M</td>
<td>6.38</td>
</tr>
<tr>
<td>QResNet56 (r. t., ReLinear)</td>
<td>2.55M</td>
<td>6.43</td>
</tr>
<tr>
<td>QResNet56 (r. t., ReLinear+ReZero)</td>
<td>2.55M</td>
<td>6.34</td>
</tr>
<tr>
<td>QResNet56 (w. t., ReLinear)</td>
<td>2.55M</td>
<td>6.22</td>
</tr>
<tr>
<td>QResNet110 (r. t., ReLinear)</td>
<td>5.1M</td>
<td>6.36</td>
</tr>
<tr>
<td>QResNet110 (r. t., ReLinear+ReZero)</td>
<td>5.1M</td>
<td>6.12</td>
</tr>
<tr>
<td>QResNet110 (w. t., ReLinear)</td>
<td>5.1M</td>
<td>5.44</td>
</tr>
</tbody>
</table>

TABLE V
IMAGE CLASSIFICATION ERROR(%) BY COMPACT-QRESNET ON CIFAR10 VALIDATION SET.

<table>
<thead>
<tr>
<th>Network</th>
<th>Params</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QResNet20</td>
<td>0.81M</td>
<td>7.17</td>
</tr>
<tr>
<td>Compact-QResNet20</td>
<td>0.54M</td>
<td>7.76</td>
</tr>
<tr>
<td>QResNet32</td>
<td>1.39M</td>
<td>6.38</td>
</tr>
<tr>
<td>Compact-QResNet32</td>
<td>0.92M</td>
<td>6.56</td>
</tr>
<tr>
<td>QResNet56</td>
<td>2.55M</td>
<td>6.22</td>
</tr>
<tr>
<td>Compact-QResNet56</td>
<td>1.92M</td>
<td>6.30</td>
</tr>
</tbody>
</table>

TABLE VI
IMAGE CLASSIFICATION ERROR(%) BY RESNETS WITH INCREASED CHANNEL NUMBERS ON CIFAR10 VALIDATION SET.

<table>
<thead>
<tr>
<th>Network</th>
<th>Channel Number</th>
<th>Params</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet32 (16)</td>
<td>x1</td>
<td>0.46M</td>
<td>7.51</td>
</tr>
<tr>
<td>ResNet32 (24)</td>
<td>x1.5</td>
<td>1.04M</td>
<td>6.45</td>
</tr>
<tr>
<td>ResNet32 (32)</td>
<td>x2</td>
<td>1.84M</td>
<td>6.75</td>
</tr>
<tr>
<td>ResNet32 (40)</td>
<td>x2.5</td>
<td>2.88M</td>
<td>7.44</td>
</tr>
<tr>
<td>QResNet32</td>
<td>-</td>
<td>1.39M</td>
<td>6.38</td>
</tr>
<tr>
<td>ResNet56 (16)</td>
<td>x1</td>
<td>0.86M</td>
<td>6.97</td>
</tr>
<tr>
<td>ResNet56 (24)</td>
<td>x1.5</td>
<td>1.92M</td>
<td>6.84</td>
</tr>
<tr>
<td>ResNet56 (32)</td>
<td>x2</td>
<td>3.40M</td>
<td>6.12</td>
</tr>
<tr>
<td>ResNet56 (40)</td>
<td>x2.5</td>
<td>5.31M</td>
<td>6.58</td>
</tr>
<tr>
<td>QResNet56</td>
<td>-</td>
<td>2.55M</td>
<td>6.22</td>
</tr>
</tbody>
</table>

Error (%). Here, we confirm the superior model expressivity of quadratic networks with experiments on ImageNet. The ImageNet dataset [41] is made of 1.2 million images for training and 50,000 images for validation, which are from 1,000 classes. For model configurations, we follow those in the ResNet paper [3]. We set the batch size to 256, the initial learning rate to 0.1, the weight decay to 0.0001, and the momentum to 0.9. For ReLinear⁹, we set γ₀ and γ₁ to 10⁻⁵. We adopt the standard 10-crop validation. As seen in Table VII similar to what we observed in CIFAR10 experiments, direct replacement with quadratic neurons can promote the performance, which confirms that the quadratic network is more expressive than the conventional network. For example, QResNet32 is better than ResNet32 with a considerable margin (4.1%). In addition, the performance of QResNet32 is also better than Poly-QResNet32 and Para-QResNet32, which validates that our design of quadratic networks is competitive.

VI. CONCLUSION

In this article, we have theoretically demonstrated the superior expressivity of a quadratic network over either the popular deep learning model or such a conventional model with quadratic activation. Then, we have proposed an effective and efficient strategy ReLinear for training a quadratic network, thereby improving its performance in various machine learning tasks. Finally, we have performed extensive experiments to corroborate our theoretical findings and confirm the practical gains with ReLinear. We have shared our code in [github.com/FengleiFan/ReLinear](https://github.com/FengleiFan/ReLinear) Future research includes up-scaling quadratic networks to solve more real-world problems, and characterize our quadratic approach in terms of its robustness, generalizability, and other properties.
ACKNOWLEDGEMENT

Dr. Fei Wang would like to acknowledge the support from NSF 1750326, the AWS machine learning for research award, and the Google faculty research award. Dr. Rongjie Lai’s research is supported in part by an NSF Career Award DMS–1752934 and DMS-2134168. Dr. Ge Wang would like to acknowledge the funding support from R01EB026646, R01CA233888, R01CA237267, R01HL151561, R21CA264772, and R01EB031102.

REFERENCES


APPENDIX

Here, we analyze the convergence behavior of the ReLinear. Not only is the convergence result established but also the associated convergence rate is derived. Our proofs are heavily based on [38].

Setting. The problem of interest is to optimize the following loss function:

$$\min_{x} \mathbb{E}_{x} L(\mathbb{E}, x),$$

(34)
where $\Xi$ is a collection of network parameters, and $x$ is the data. The stochastic gradient descent (SGD) with the ReLinear algorithm at the $t$-th step is as follows:

i) select a set of samples from the dataset, denoted as $X_t = \{x_m\}_{m=1}^{b_t}$ and compute the gradient accordingly

$$g_i(X_t, X_t) = \frac{1}{b_t} \sum_{i=1}^{b_t} \nabla L(\Xi_t, x^i).$$

(35)

ii) update the network parameters,

$$\Xi_{t+1} = \Xi_t - \eta_t g(X_t, X_t),$$

(36)

where $\eta_t$ is a diagonal matrix whose diagonal entries are the learning rates of different parameters. Such a setting complies with the ReLinear that assigns $(W^r, b^r)$ and $(W^g, b^g, W^b, c)$ to different learning rates. The learning rate can be either fixed or diminished. Without loss of generality, let $\eta_t$ diminish along the training step:

$$\eta_t = \frac{\beta}{\alpha + t} \text{diag}((1_K, \gamma \cdot 1_K_2)), \tag{37}$$

where $1_K$ is an all-ones vector of the length $K$, $\alpha, \beta, \gamma$ are auxiliary parameters, $K_1$ and $K_2$ represent the number of trainable parameters in $(W^r, b^r)$ and $(W^g, b^g, W^b, c)$, respectively. $\eta_t$ fulfills that $\sum_{j}^{\infty} (\eta_t)_{jj}$ diverges and $\sum_{j}^{\infty} (\eta_t)_{jj}^2$ converges.

**Assumptions.** We adopt the assumptions that are often made in proving convergence of optimization algorithms.

**H1** $L$ is $M$ -smooth, i.e., $L$ is differentiable and $\|\nabla L(\Xi_1) - \nabla L(\Xi_2)\| \leq M \|\Xi_1 - \Xi_2\|$, $\forall \Xi_1, \Xi_2$. Furthermore, $M$-smoothness implies that

$$L(\Xi_2) \leq L(\Xi_1) + \langle \nabla L(\Xi_1), \Xi_2 - \Xi_1 \rangle + \frac{M}{2} \|\Xi_2 - \Xi_1\|^2. \tag{38}$$

**H2** $L$ is $\Theta$ -Lipschitz, i.e., $|L(\Xi_1) - L(\Xi_2)| \leq \Theta \|\Xi_1 - \Xi_2\|$, $\forall \Xi_1, \Xi_2$.

**H3** The expectation of the stochastic gradient is equal to the true gradient: $E_{X}[g(\Xi, X)] = \nabla L(\Xi), \forall \Xi$.

**H4** The noise in the stochastic gradient has bounded support, i.e., $\|g(\Xi_t, X_t) - \nabla L(\Xi_t)\| \leq S, \forall \Xi_t, X_t$. Furthermore, we have

$$\|\eta_t g(\Xi_t, X_t) - \nabla L(\Xi_t)\|^2 \leq \|\eta_t g(\Xi_t, X_t)\|^2 + \|\eta_t \nabla L(\Xi_t)\|^2 - 2 \langle \eta_t g(\Xi_t, X_t), \eta_t \nabla L(\Xi_t) \rangle \leq \|\eta_t\|^2 \|g(\Xi_t, X_t) - \nabla L(\Xi_t)\|^2. \tag{39}$$

Taking the expectation of the above equation leads to

$$E[\|\eta_t g(\Xi_t, X_t)\|^2] \leq E[\|\eta_t \nabla L(\Xi_t)\|^2] + \|\eta_t\|^2 S^2. \tag{40}$$

**Lemma 3.** Let $(a_t)_{t \geq 1}$ and $(b_t)_{t \geq 1}$ be two non-negative sequences. Assume that $\sum_{t=1}^{\infty} a_t b_t$ converges and $\sum_{t=1}^{\infty} a_t$ diverges, and there exists $K \geq 0$ such that $|b_{t+1} - b_t| \leq K a_t$, then $b_t$ converges to 0.

**Proof.** The proof can be referred to Lemma 2 of [38].

**Lemma 4.** Assume (H1, H3, H4), the iterates of SGD with a learning rate diagonal matrix $\eta_t$ satisfying Eq. (37) and $0 \leq (\eta_t)_{jj} \leq \frac{1}{M}$, it holds that

$$E[\sum_{t=1}^{T} (\nabla L(\Xi_t), \eta_t \nabla L(\Xi_t))] \leq 2E[L(\Xi_1) - L(\Xi_{T+1})] + \frac{MS^2}{2} \sum_{t=1}^{T} \|\eta_t\|^2. \tag{41}$$

**Proof.** According to (H1), we have

$$L(\Xi_{t+1}) \leq L(\Xi_t) + \langle \nabla L(\Xi_t), \Xi_{t+1} - \Xi_t \rangle + \frac{M}{2} \|\Xi_{t+1} - \Xi_t\|^2$$

$$\leq L(\Xi_t) - \langle \nabla L(\Xi_t), \eta_t g(\Xi_t, X_t) \rangle + \frac{M}{2} \|\Xi_{t+1} - \Xi_t\|^2$$

$$= L(\Xi_t) + \langle \nabla L(\Xi_t), \eta_t \nabla L(\Xi_t) - g(\Xi_t, X_t) \rangle$$

$$- \langle \nabla L(\Xi_t), \eta_t \nabla L(\Xi_t) \rangle + \frac{M}{2} \|\eta_t g(\Xi_t, X_t)\|^2. \tag{42}$$

Taking the conditional expectation with respect to $X_1, X_2, \cdots, X_{t-1}$, due to (H3), we have

$$E_{X_t}[g(\Xi_t, X_t)] = \nabla L(\Xi_t). \tag{43}$$

Then, taking the expectation for Eq. (42) and applying Eq. (40), we have

$$E[\langle \nabla L(\Xi_t), \eta_t g(\Xi_t, X_t) \rangle]$$

$$\leq E[L(\Xi_t) - L(\Xi_{t+1})] + \frac{M}{2} E[\|\eta_t g(\Xi_t, X_t)\|^2]$$

$$\leq E[L(\Xi_t) - L(\Xi_{t+1})] + \frac{M}{2} E[\|\eta_t \nabla L(\Xi_t)\|^2] + \|\eta_t\|^2 S^2. \tag{44}$$

Moving $\frac{M}{2} E[\|\eta_t \nabla L(\Xi_t)\|^2]$ to the left leads to

$$E[\langle \nabla L(\Xi_t), (\eta_t - M \eta_t^2) \nabla L(\Xi_t) \rangle]$$

$$\leq E[L(\Xi_t) - L(\Xi_{t+1})] + \frac{MS^2}{2} \|\eta_t\|^2. \tag{45}$$

Because $0 \leq (\eta_t)_{jj} \leq \frac{1}{M}$, we have $\eta_t - M \eta_t^2 \geq \frac{1}{2} \eta_t$. Then,

$$E[\langle \nabla L(\Xi_t), \frac{\eta_t}{2} \nabla L(\Xi_t) \rangle]$$

$$\leq E[\langle \nabla L(\Xi_t), (\eta_t - M \eta_t^2) \nabla L(\Xi_t) \rangle]$$

$$\leq E[L(\Xi_t) - L(\Xi_{t+1})] + \frac{MS^2}{2} \|\eta_t\|^2 \tag{46}$$

Summing the above from $t = 1$ to $T$, we conclude the proof.

**Proposition 3** (Convergence Behavior). Assume (H1.2, H3, H4). Suppose that the learning rates $\eta_t$ are given by Eq. (37) and $0 \leq (\eta_t)_{jj} \leq \frac{1}{M}$, then the gradients of SGD converge to zero almost surely.
Suppose that the learning rate diagonal matrix \( \eta \) is given by Eq. (37), which naturally gives rise to

\[
\text{(Convergence Rate)}
\]

\[
\text{Proposition 4}
\]

\[
|\nabla L(\Xi_t) , \eta_t \nabla L(\Xi_t) | \leq \max_{j=1}^J \| \eta_j \| \leq \frac{2\Theta}{\gamma} (\nabla L(\Xi_t))_j^2 < \infty.
\]

Since if \( E[Y] < \infty \), with probability 1, \( Y < \infty \), we have

\[
\sum_{t=1}^{\infty} |(\nabla L(\Xi_t), \eta_t \nabla L(\Xi_t) | < \infty,
\]

which naturally gives rise to

\[
\sum_{t=1}^{\infty} (\eta_t)_{jj} |(\nabla L(\Xi_t))_j^2 < \infty.
\]

Using the fact that \( L \) is \( \Theta \)-Lipschitz and \( M \)-smooth,

\[
|(\nabla L(\Xi_{t+1}))_j - (\nabla L(\Xi_t))_j |^2
\]

\[
= |(\nabla L(\Xi_t))_j + (\nabla L(\Xi_t))_j |^2
\]

\[
\leq 2\Theta (\nabla L(\Xi_t))_j^2
\]

\[
\leq 2\Theta M \| \Xi_{t+1} - \Xi_t \|
\]

\[
= 2\Theta M \| \eta_t g(\Xi_t, X_t) \|
\]

\[
\leq 2\Theta M (\Theta + S) \frac{\sqrt{K_1 + K_2}}{\gamma} (\eta_t)_{jj},
\]

where the first inequality is due to that \( L \) is \( \Theta \)-Lipschitz, i.e.,

\[
|\nabla L(\Xi)| \leq \Theta, \text{ and the last inequality is because}
\]

\[
\| \eta_t \| \leq \sqrt{K_1 + K_2} (\eta_t)_{\max} \leq \frac{\sqrt{K_1 + K_2}}{\gamma} (\eta_t)_{jj}, \forall j.
\]

According to Lemma 3, combining Eqs. (49) and (50) as well as the fact that \( \sum_{t} (\eta_t)_{jj} \) diverges, we obtain that almost surely,

\[
\lim_{t \to \infty} (\nabla L(\Xi_t))_j^2 = 0.
\]

\[
\text{Proposition 4 (Convergence Rate). Assume (H1,H2,H3,H4).}
\]

Suppose that the learning rate diagonal matrix \( \eta_t \) is given by Eq. (37) and \( \Theta \leq (\eta_t)_{jj} \leq \frac{1}{T} \), then the iterates of SGD satisfy the following bound:

\[
\mathbb{E}[\min_{1 \leq t \leq T} \| \nabla L(\Xi_t) \|] \leq T^{-1/2} \mathcal{O}(1).
\]

\[
\text{Proof. From Proposition 3, we have}
\]

\[
\mathbb{E}[\sum_{t=1}^{T} (\nabla L(\Xi_t), \eta_t \nabla L(\Xi_t) )]
\]

\[
\leq 2(\mathbb{E}[L(\Xi_t) - L(\Xi_{t+1})] + \frac{M S^2}{2} \sum_{t=1}^{T} \| \eta_t \|^2)
\]

\[
\text{Let } \Delta = \sum_{t=1}^{T} \| \nabla L(\Xi_t) \|^2,
\]

we can bound

\[
\mathbb{E}[\sum_{t=1}^{T} (\nabla L(\Xi_t), \eta_t \nabla L(\Xi_t) )]
\]

\[
\geq \mathbb{E}[\eta_{T, \min} \Delta] \leq \eta_{T, \min} (\mathbb{E}[\Delta])^2,
\]

where the second inequality is Holder’s inequality. Let

\[
A = \frac{1}{\eta_{T, \min}} \mathbb{E}[L(\Xi_1) - L(\Xi_{T+1})],
\]

\[
B = \frac{M S^2}{2(\eta_{T, \min}) \sum_{t=1}^{T} \| \eta_t \|^2}.
\]

Then,

\[
T^{1/2} \mathbb{E}[\min_{1 \leq t \leq T} \| \nabla L(\Xi_t) \|] \leq \mathbb{E}[\Delta] \leq A + B = \mathcal{O}(1).
\]

In other words, \( \mathbb{E}[\min_{1 \leq t \leq T} \| \nabla L(\Xi_t) \|] \leq T^{-1/2} \mathcal{O}(1) \), which concludes our proof.

\[
\square
\]