Downloaded 05/12/23 to 160.26.141.30 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Quadratic Residual Networks: A New Class of Neural Networks for Solving Forward and Inverse Problems in Physics Involving PDEs

Jie Bu* Anuj Karpatne*

Abstract

We propose *quadratic residual networks* (QRes) as a new type of parameter-efficient neural network architecture, by adding a quadratic residual term to the weighted sum of inputs before applying activation functions. With sufficiently high functional capacity (or expressive power), we show that it is especially powerful for solving forward and inverse physics problems involving partial differential equations (PDEs). Using tools from algebraic geometry, we theoretically demonstrate that, in contrast to plain neural networks, QRes shows better parameter efficiency in terms of network width and depth thanks to higher non-linearity in every neuron. Finally, we empirically show that QRes shows faster convergence speed in terms of number of training epochs especially in learning complex patterns.

1 Introduction

Deep neural networks (DNNs) have found remarkable success in a number of learning tasks, thanks to their ability to approximate arbitrarily complex functions [1, 2]. The composition of many nonlinearly activated neurons gives DNN high functional capacity despite the weighted sum being linear in every neuron. Intuitively, a DNN with higher capacity has a better ability to capture complex patterns from data in lesser number of training epochs. However, in order to learn generalizable patterns, we often need to proportionately balance the capacity of a DNN with the *amount of supervision* available in the training data. Indeed, it is common practice to use regularization tools such as dropout and early stopping to avoid *overfitting* the DNN model to complex spurious patterns in the training set, especially when training sizes are very small.

While data-driven supervision is limited in conventional learning tasks, there is a growing body of research on using physical knowledge as another form of supervision to train machine learning (ML) models, termed as the field of *physics-guided machine learning* (PGML) [3, 4]. In this work, we specifically study the class of problems in PGML where the physics of the system is available in the form of partial differential equations (PDEs). A promising line of work in this area is the framework of *physics-informed neural networks* (PINNs) [5, 6], where a neural network is used to model a target variable u (e.g., velocity field) given some inputs (e.g., location x and time t), based on the *physical constraint* that $u(x, t)$ satisfies a known PDE. In PINNs, the neural networks are trained not only using supervision from data (by minimizing prediction errors on labeled points) but also from physics (by evaluating the consistency of neural network predictions with PDE on plentiful unlabeled points).

[1]Discovery Analytics Center, Dept. of Computer Science, Virginia Tech, VA, USA

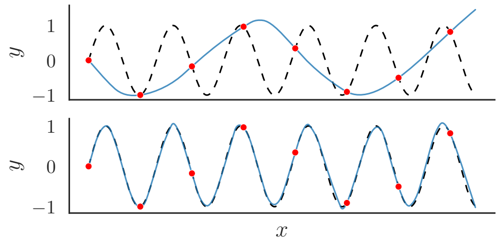
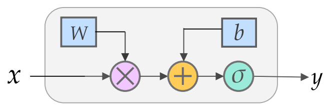


Figure 1: Toy demonstration of the importance of physics supervision in learning a target function $y = \sin(6x)$ (dashed) given limited data (8 red points). Top and bottom figures show neural network solutions (solid) trained solely using data, and both data and physics (satisfying the simple PDE constraint: $\frac{dy}{dx} = 6\cos(6x)$), respectively.

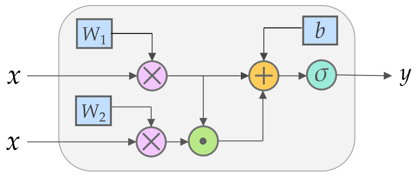
The additional supervision from physics enables PINN to support neural networks with sufficiently high capacity without running into risks of overfitting (see Figure 1 for a demonstration on a toy problem). This has sparked ample interest in the scientific community to use PINNs and their variants in a number of physical problems involving PDEs [7–9]. Despite these developments, most existing work in PINN only uses plain DNN architectures (see Figure 2a). As a result, PINN formulations typically require a vast number of network parameters and training epochs to approximate complex PDEs with acceptable accuracy [10]. This motivates us to ask the question: *Can we develop a neural network architecture with higher capacity at every layer that can approximate complex functions with less parameters than plain DNNs?*

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 05/12/23 to 160.26.141.30 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy



(a) Plain DNN Layer



(b) Quadratic Residual Network (QRes) Layer

Figure 2: Overview of our proposed Quadratic Residual Network (QRes) layer in comparison with plain DNN layer. Blue rectangular boxes represent trainable parameters and round boxes represent operations (purple "×": multiplication, orange "+": addition, green "·": Hadamard product, and cyan "σ": activation operator).

We present *Quadratic Residual networks* (QRes), a novel class of neural network architectures that impart quadratic non-linearity before applying activation functions at every layer of the network. Figure 2b shows an overview of a QRes layer where a quadratic residual term: $W_1x \circ W_2x$ is added to the weighted sum $W_1x + b$ of a plain DNN layer before passing through a non-linear activation σ. We theoretically study the expressive power of QRes to demonstrate that QRes is more parameter efficient than plain DNNs. We also conduct extensive experiments on forward and inverse problems involving PDEs by replacing DNNs with QRes in PINN frameworks and demonstrate better parameter efficiency of QRes over baselines. Finally, we empirically show that QRes converge faster than plain DNNs especially in learning higher frequency patterns.

2 Background

2.1 Physics-informed Neural Networks: There is growing body of work in the field of PINN for guiding the learning of neural networks using physics supervision available as PDEs [5–10]. A general form of a non-linear PDE can be expressed as $\mathcal{N}(u, \lambda) = 0$, where $\mathcal{N}$ is a non-linear operator involving partial derivatives of the target variable u (e.g., $u_x, u_t, u_{xx}, \ldots$), and λ represents the parameters of the PDE. There are two classes of problems in the realm of PDEs that are studied by PINNs: (a) *forward problems*, where the goal of the network is to solve for the target variable u satisfying the PDE, and (b) *inverse problems*, where the network is tasked to learn the unknown parameters of the PDE, λ, given ground-truth values of u at a collection of points.

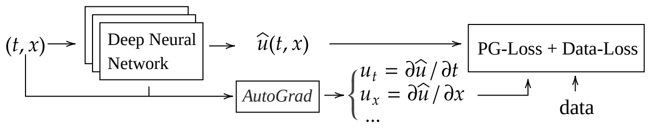


Figure 3: The general framework of PINN.

In both classes of problems, a central form of supervision used for training neural networks is the discrepancy of the network predictions w.r.t. the governing PDEs, captured as physics-guided (PG)-Loss = $\mathcal{N}(\hat{u}, \lambda)^2$. This loss is evaluated at a set of unlabeled points, where the partial derivatives involved in PG-Loss are computed analytically using automatic differentiation tools (e.g., AutoGrad). The neural networks are also supervised with data-driven loss over a set of labeled points S, Data-Loss = $\sum_{i\in S} ||\hat{u}_i - u_i||^2$, where S usually comprises of initial or boundary condition points. The combined learning objective in PINN is thus to minimize PG-Loss + Data-Loss (see Figure 3).

To accelerate training convergence of PINNs, adaptive activation functions have recently been proposed in [11], where a learnable scalar α is multiplied to the weighted sum (Wx) produced at every layer before applying activation. There has also been recent studies exposing gradient pathologies in the training of PINNs [12], where adaptive trade-off between loss terms were proposed to resolve the imbalance in loss gradients. In the same work [12], the authors further explored a modified neural network architecture for PINN problems, where, inspired by attention mechanisms, they explicitly accounted for multiplicative terms involving inputs in the network layers, similar to our work. While it empirically showed the importance of using higher-order interactions in PINN frameworks, it did not provide any theoretical justifications for its effect on the expressive power of neural networks in a principled manner as performed in our work.

2.2 Related Work on Quadratic Networks: There is a long history of research on building neural networks to capture multiplicative interactions among inputs, ranging from early works on optimal depths of plain DNNs for approximating multiplications [13] to the use of weighted products instead of summations at every unit of the network [14, 15]. In the area of graphical models, the framework of *sum-product networks* (SPNs) [16] have been developed to represent factorization operations (product nodes) in addition to mixture operations (sum nodes) for learning partition functions. Our work shares a similar motivation as SPNs to learn more expressive functions for approximating complex decision boundaries. Our work is also related to the recent framework of *neural arithmetic units*

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 05/12/23 to 160.26.141.30 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

(NAUs) [17, 18], that perform exact arithmetic operations (e.g., additions, subtractions, and multiplications) at every layer to logically extrapolate well on arithmetic tasks. Our work can be viewed as a special case of NAUs that capture quadratic residual terms, although for a different goal of expressing higher functional capacity in the process of solving non-linear PDEs.

Another line of work that bears close resemblance to our work is the quadratic deep networks (QDNs) [19], where three weight matrices are used to express quadratic products as well as linear sums at every layer before applying activations. Our work is different from QDNs on two grounds. First, we provide novel theoretical analyses of the expressive power of QRes that proves its superior parameter efficiency over plain DNNs. Second, in contrast to QDNs, we demonstrate the efficacy of using QRes in solving PINN problems, where neural networks with higher functional capacity can be better supported with the aid of physics supervision, in contrast to conventional learning tasks that only use data-driven supervision.

3 Quadratic Residual Networks

A plain DNN layer can be expressed as $y^{DNN} = \sigma(Wx + b)$, where (W, b) are the learnable parameters and σ is a non-linear activation function. Notice that $Wx + b$ is linear and it is only σ that imparts non-linearity to the outputs. As a result, we need a large number of DNN layers with reasonable widths to capture sufficient non-linearity with acceptable accuracy.

In contrast, we consider *quadratic residual terms* at every layer of our QRes network to contribute additional nonlinearity. In particular, we can express a single layer of QRes as $y^{QRes} = \sigma(W_2x \circ W_1x + W_1x + b)$, where $\circ$ denotes the Hadamard product and the term in red is the quadratic residual term (we call it "residual" as removing it simply yields a plain DNN). Hence, in problems where linear decision boundaries (activated non-linearly) are sufficient to capture the complexity of target functions, QRes can easily resort to a plain DNN by learning $W_2 = 0$. However, in problems where we need neural networks with higher functional capacity than DNNs, QRes can switch on the quadratic residual term to capture higher amounts of non-linearity using efficient network depths and widths.

It is easy to show that a linearly activated QRes (using linear activations) with depth d can learn polynomials of degree 2^{d-1}, since every layer of QRes would double the non-linearity by considering products of outputs from previous layer. As a result, even a linearly activated QRes, in theory, can approximate arbitrarily complex polynomial boundaries with sufficient network widths and depths. However, in practice, it is desirable to use non-linearly activated QRes for two reasons. First, non-linearly activated QRes can approximate polynomial decision boundaries using smaller network depths than a linearly activated QRes, thus resulting in parameter efficiency. Second, a linearly activated QRes with a large number of layers can produce unbounded activation outputs at every layer, which, if not properly scaled, can lead to unstability in training. Hence, we use non-linear activations with bounded output spaces (*tanh*) in all our implementations of QRes.

4 Theoretical Analyses of QRes

To analyze the expressive power of QRes on regression tasks, we draw inspiration from theoretical analyses of the expressive power of *deep polynomial networks* (networks with polynomial activation functions) presented in [20], using concepts from *algebraic geometry*.

4.1 Definitions and Notations:

Let us represent a network architecture[1] as a vector, $\boldsymbol{d} = \{d_0, \ldots, d_h\}$, where d_i is the width of layer i and h is the network depth. To understand the space of functions expressed by such a network architecture, let us consider the functional mapping from the space of network parameters to the space of output functions. The image (set of outputs) of this functional mapping is referred to as the *functional space* associated with the network. The dimension of the functional space gives us a measure of the expressive power of a network.

In this work, we characterize the functional space of a network using a basis set of polynomials (obtained, for example, using Taylor approximations). We also consider a special type of neural networks with polynomial activation functions of fixed degree r, which raises the input to their r-th power, namely *polynomial networks*. As shown in [20], the functional space of a polynomial network comprises of *homogeneous polynomials*, i.e., polynomials where every term (monomial) is of the same degree. We denote the space of all homogeneous polynomials of degree d in n variables with coefficients in $\mathbb{R}$ as $\mathrm{Sym}_d(\mathbb{R}^n)$.

Under these settings, the family of all networks with the same architecture $\boldsymbol{d}$ can be identified with its *functional variety*, which is the *Zariski closure*[2] of its functional space. An advantage of analyzing functional variety with polynomial equations is that it requires less strict assumptions compared to the functional space [20] and has the same dimension as the

[1]Note that the *architecture* $\boldsymbol{d}$ in this section refers to a sequence of layer widths .

[2]The Zariski closure of a set $\mathcal{X}$ is the smallest set that contains $\mathcal{X}$ and can be described by polynomial equations [20].

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 05/12/23 to 160.26.141.30 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

functional space, making the dimension of the functional variety of a network a precise measure of the networks' expressiveness.

Note that even with polynomial activation functions, the functional space of a QRes network is not a space of homogeneous polynomials. For example, for a single layer QRes network with linear activations, its functional space contains spaces of homogeneous polynomials of degrees both 1 and 2. Furthermore, neural networks with widely-used activation functions [21] also result in non-homogeneous functional spaces because popular non-linear activation functions (e.g., *tanh*) are generally not homogeneous polynomial mappings. However, every space of non-homogeneous polynomials can be viewed as comprising of subspaces of homogeneous polynomials of varying degrees. Thus, to generalize the analysis of functional spaces beyond polynomial networks, we introduce the definition of the *leading functional space* of a network as the subspace of its functional space comprising of homogeneous polynomials of highest degree. In general, for any activation function, we can always decompose it to a set of polynomial functions using Taylor approximations. The highest degree of polynomial, r, in such a decomposition can then be referred to as the *leading degree* of the activation function.

Formally, for a network architecture $\boldsymbol{d}$ with an activation function of leading degree r, we denote the *leading functional space* of a neural network as $\mathcal{F}_{\boldsymbol{d},r}$ and a QRes network as $\mathcal{F}^2_{\boldsymbol{d},r}$. The leading functional variety of a neural network and a QRes network can then be defined as the Zariski closure of its leading functional space, i.e., $\mathcal{V}_{\boldsymbol{d},r} = \overline{\mathcal{F}^2_{\boldsymbol{d},r}}$, and $\mathcal{V}_{\boldsymbol{d},r} = \overline{\mathcal{F}^2_{\boldsymbol{d},r}}$. Using these definitions, we introduce the revised concepts of *filling* functional space and variety (similar to the ones presented in [20]) as follows:

Definition 4.1. *A neural network architecture* $\boldsymbol{d} = (d_0, ..., d_h)$ *has a filling functional space for the activation degree r if its leading functional space satisfies* $F_{\boldsymbol{d},r} = \mathrm{Sym}_{r^{h-1}}(\mathbb{R}^{d_0})^{d_h}$. *For a filling functional variety, its leading functional variety satisfies* $V_{\boldsymbol{d},r} = \mathrm{Sym}_{r^{h-1}}(\mathbb{R}^{d_0})^{d_h}$.

Hence, rather than requiring the functional space or variety of a network to fill the ambient space of homogeneous polynomials, we only require it to *contain* the space of homogeneous polynomials of leading degree for it to be considered as filling.

4.2 Theoretical Analyses:

Proofs of all theoretical analyses introduced in this work are available in the full-length version of this paper [22].

Proposition 4.1. *A single-layer linearly activated* ($r = 1$) *quadratic residual networks of architecture* $\boldsymbol{d} = (d_0, d_1)$ *has a filling functional space of degree 2, i.e., its leading functional space* $\mathcal{F}^2_{\boldsymbol{d},1} = \mathrm{Sym}_2(\mathbb{R}^{d_0})^{d_1}$.

A noticeable feature of linear neural networks is that the degree of its functional variety will not grow with the network depth while the degree of the functional space of a linear QRes network can grow exponentially with network depth. The growing degree of QRes suggests it can obtain more nonlinearity from deep architectures. A QRes network with linear activation can be related to a polynomial regression, where a single layer corresponds to a quadratic regression. Proposition 4.1 can be easily generalized to QRes with deep architectures as follows.

Lemma 4.1. *For an activation function with leading degree $r \geq 1$ and network architecture* $\boldsymbol{d} = (d_0, ..., d_h)$, *the leading functional variety of a QRes network,* $\mathcal{V}^2_{\boldsymbol{d},r}$, *and a neural network,* $\mathcal{V}_{\boldsymbol{d},r}$, *satisfy* $\mathcal{V}^2_{\boldsymbol{d},r} = \mathcal{V}_{\boldsymbol{d},2r}$.

The above lemma states that, with the same network architecture, a QRes network with leading activation degree r and a neural network with leading activation degree $2r$ have functional varieties of the same degree of homogeneous polynomials. This implies that a deep QRes network can have a leading functional variety of degree $(2r)^{h-1}$ homogeneous polynomials, while a neural network of the same architecture and activation function can only reach a degree of r^{h-1}. Note that this lemma does not require the functional variety of either networks to be filling, and it holds both for linear and nonlinear activations. Using this property we can arrive at the following theorem.

Theorem 4.1. (Depth Efficiency) *For a fixed leading degree $r \geq 2$, let us assume that the functional variety of a quadratic residual network* $\boldsymbol{d_q} = (d_0, .., d_{h_q})$ *is filling,* $\mathcal{V}^2_{\boldsymbol{d_q},r}$ *is the leading functional variety for the QRes network, and* $\mathcal{V}_{\boldsymbol{d},r}$ *is the leading functional variety for a neural network* $\boldsymbol{d_n} = (d'_0, .., d'_{h_n})$, *where* $h_n, h_q > 1$ *and* $d_0 = d'_0, d_{h_q} = d'_{h_n}$. *If* $\dim \mathcal{V}_{\boldsymbol{d_n},r} \geq \dim \mathcal{V}^2_{\boldsymbol{d_q},r}$, *then*

$$h_n \geq 1 + \left(1 + \frac{\log 2}{\log r}\right)(h_q - 1) \tag{4.1}$$

The above theorem throws light on the depth efficiency of QRes as it provides a lower bound on the depth of a neural network h_n for it to have greater expressibility (i.e., larger dimension of functional variety) than a filling QRes network. Although h_n and h_q will converge for large values of r, it may need an extremely wide network to be able to have a filling functional variety of high degrees. Therefore, it is necessary to show the

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 05/12/23 to 160.26.141.30 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

efficiency of QRes in terms of network width, presented in the following.

PROPOSITION 4.2. (Minimal Filling Width) *For a neural network or a QRes network with architecture* $\boldsymbol{d} = (d_0, ..., d_h)$ *and leading activation degree* $r \geq 2$, *if*

$$d_{h-i} \geq \min \left[d_h r^{id_0}, \binom{r^{h-i} + d_0 - 1}{r^{h-i}} \right] \tag{4.2}$$

for each $i = 1, ..., h-1$, *then its functional variety is filling, and we call the lower bound of* d_{h-i} *as the minimal filling width of this layer with leading degree* r.

We refer to the architecture with minimal filling width at each intermediate layer as the *minimal filling architecture*. Using *Proposition 4.2*, we can arrive at the following theorem for width efficiency.

THEOREM 4.2. (Width Efficiency) *Suppose a neural network* $\boldsymbol{d_n} = (d_0, .., d_h)$ *is filling for leading activation degree* $r \geq 2$. *Given a quadratic residual network* $\boldsymbol{d_q} = (d'_0, .., d'_h)$ *with* $d_0 = d'_0$ *and* $d_h = d'_h$, *such that* $\dim \mathcal{V}_{\boldsymbol{d_n}, r} = \dim \mathcal{V}^2_{\boldsymbol{d_q}, r}$. *Suppose* $\boldsymbol{d_n}$ *is a minimal filling architecture, then for each* $i = 1, ..., h-1$,

$$\lim_{r \to \infty} d_{h-i} = O(2^\tau) \lim_{r \to \infty} d'_{h-i} \tag{4.3}$$

where $\tau = \min\left[id_0, (h-i)(d_0 - 1)\right]$.

The above theorem shows that a QRes network is exponentially more efficient than a neural network in terms of width to achieve the same expressive power (i.e., dimension of functional variety). Since the number of network parameters grow roughly linearly with network depth but polynomially with the network width, width efficiency is a dominating factor in the overall parameter efficiency. Further, while the above analysis was performed using polynomials as the basis set, it is easy to extend this analysis to frequencies in the spectral space (by applying Fourier decomposition). Since QRes can express higher degree of polynomials more efficiently than neural networks, QRes is also able to capture higher frequency information with comparable or even smaller number of parameters, as shown empirically in Section 5.1.

5 Empirical Results

We evaluated PINN (using DNN) and QRes on a set of forward and inverse problems (see Table 1) involving nonlinear partial differential equations, same as those used in [23]. We include both *continuous time* and *discrete time* models based on definitions from [5, 6]. Except for the forward problem on Burgers' equation,

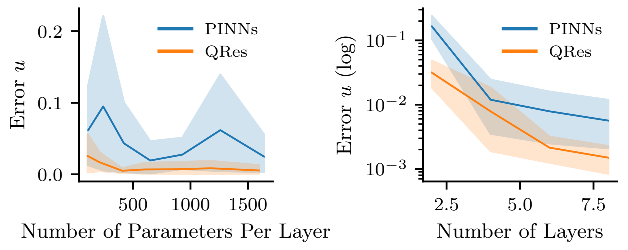


Figure 4: Parameter efficiency of PINNs and QRes networks for solving Burgers' equation. The figures show performances across 28 architectures over four values of depth (2, 4, 6, 8), and seven widths: (10, 15, 20, 15, 30, 35, 40) for PINNs and (7, 10, 14, 17, 21, 24, 28) for QRes. Lines indicate mean values and shades represent variances.

all models are trained with Adam optimizer for a fixed number of epochs and subsequently trained on L-BFGS-B for better accuracy, following the practices used in previous work on PINNs [5,6]. In addition to prediction errors, we also report the number of network parameters (including the bias term) and training epochs on Adam. We observed that the number of epochs needed for L-BFGS-B to reach termination condition is roughly the same for PINNs and QRes on all the experiments, while the convergence speed on Adam plays a dominating role for training efficiency. Additional specifications of experiments are available in the full-length version of this article [22], and all codes are on Github[3].

Better Accuracy: Tables 2 and 3 show the results of the overall evaluation of QRes and PINN on different PDEs. With the same number of parameters and epochs, QRes consistently outperforms PINN, e.g., for Navier-Stokes (N-S) in Table 2. Results on Burgers' equation in Table 2 show that even with less number of parameters, QRes still manages to have better accuracy than PINNs over most of the predictions, which is verified by results from Table 3. To push the limit of the QRes networks even further, we reduce the number of network parameters as well as training epochs for the kDV equation in Table 2. We can see that QRes maintains better accuracy over PINN with less than half of the PINN's network parameters and 1/5 training epochs.

Parameter Efficiency: To further explore the parameter efficiency of PINN and QRes, we experimented with different network widths and depths of both networks for solving Burgers' equation. Figure 4 shows how the prediction errors vary with different network widths and depths. We can see that the QRes networks outperform PINNs not only under the same settings but

[3]`https://github.com/jayroxis/qres`.

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

Table 1: Set of forward and inverse PDE problems studied in this work.

PDE	Problem Type	Model Type	Descriptions
Navier-Stokes (N-S)	Inverse	Continuous Time	$u_t + \lambda_1(uu_x + vu_y) = -p_x + \lambda_2(u_{xx} + u_{yy}),$ $v_t + \lambda_1(uv_x + vv_y) = -p_y + \lambda_2(v_{xx} + v_{yy}),$ $u_x + v_y = 0,$ Given data $u, v,$ predict λ_1, λ_2, p.
Burgers'	Inverse	Continuous Time	$u_t + \lambda_1 uu_x - \lambda_2 u_{xx} = 0,$ Given data $u,$ predict λ_1, λ_2.
Korteweg–de Vries	Inverse	Discrete Time	$u_t + \lambda_1 uu_x + \lambda_2 u_{xxx} = 0,$ Given data $u,$ predict λ_1, λ_2.
Burgers'	Forward	Continuous Time	$u_t + uu_x - (0.01/\pi)u_{xx} = 0,\ x \in [-1,1],\ t \in [0,1],$ $u(0,x) = -\sin(\pi x),$ $u(t,-1) = u(t,1) = 0.$ Predict u.
Schrödinger	Forward	Continuous Time	$ih_t + 0.5h_{xx} + \|h\|^2 h = 0,\ x \in [-5,5],\ t \in [0,\pi/2],$ $h(0,x) = 2\ \text{sech}(x),$ $h(t,-5) = h(t,5),$ $h_x(t,-5) = h_x(t,5),$ Predict h.
Allen-Cahn	Forward	Discrete Time	$u_t - 0.0001u_{xx} + 5u^3 - 5u = 0,\ x \in [-1,1],\ t \in [0,1],$ $u(0,x) = x^2\cos(\pi x),$ $u(t,-1) = u(t,1),$ $u_x(t,-1) = u_x(t,1).$ Predict u.

Table 2: Parameter identification on Navier-Stoke (N-S) equation, Burgers' equation and Korteweg–de Vries (KdV) equation. $e(\lambda)$ and $e_n(\lambda)$ represents the percentage errors of predicted parameter of the PDEs λ w.r.t. its ground truth values, while e and e_n correspond to experiments on clean data and data with 1% noise, respectively.

PDE	Model	Param.	Epochs	$e(\lambda_1)\%$	$e(\lambda_2)\%$	$e_n(\lambda_1)\%$	$e_n(\lambda_2)\%$
N-S	PINN	3.06k	200k	0.083	5.834	0.077	5.482
N-S	QRes	3.00k	200k	0.043	4.281	0.050	4.942
Burgers'	PINN	3.02k	5k	0.057	0.636	0.170	0.031
Burgers'	QRes	1.54k	5k	0.027	0.379	0.172	0.003
KdV	PINN	10.35k	50k	0.017	0.011	0.154	0.045
KdV	QRes	4.61k	10k	0.009	0.009	0.183	0.009

even with much less number of parameters. It demonstrates a huge advantage of the QRes networks in terms of parameter efficiency, which strongly supports our theoretical results.

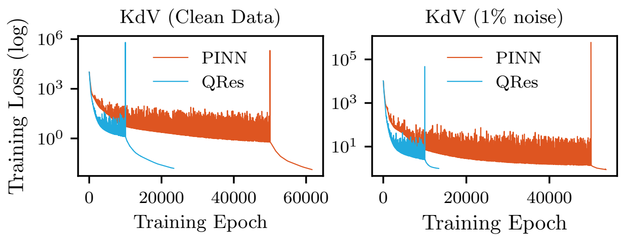


Figure 5: Training loss curves for parameter identifications on KdV equation (Table 2). The large spikes in the loss curves during training (also studied by [24]) indicate the transition from Adam to L-BFGS-B optimizers.

Faster Convergence: To analyze the convergence speeds of both networks, Figure 5 shows an example where the QRes network managed to reach comparable training loss as PINNs with roughly 1/5 of the training epochs as PINN on Adam optimizer. The QRes network reached final convergence on the L-BFGS-B optimizer with roughly 1/3 of the training epochs that PINNs used. We observed similar trends across other datasets, indicating the superior training convergence of QRes.

Comparison with Other Baselines: To demonstrate the advantages of QRes, we compared it with some other baselines for solving Burgers' equation. First, we considered APINN, which is PINN with *adaptive activations* [11] expressed as $H^{(l)} = \sigma[n\ \alpha\ (WH^{(l-1)} + b)]$, where α is the scaling parameter and n is a hyperparameter. We followed the same settings of APINN as mentioned in the original

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 05/12/23 to 160.26.141.30 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 05/12/23 to 160.26.141.30 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Table 3: Solving Schrödinger and Allen-Cahn equations. Results are reported using normalized (by ground truth values) mean square errors between predictions and ground truth values. u and v are the real and imaginary parts of the complex solution h in Schrödinger equation, and $|h| = \sqrt{u^2 + v^2}$ is the modulus.

PDE	Model	Param.	Epochs	Error u	Error v	Error $\|h\|$
Schrödinger	PINN	30.80k	50k	1.456e-03	1.878e-03	1.099e-03
Schrödinger	QRes	15.60k	50k	1.379e-03	1.751e-03	1.059e-03
Allen-Cahn	PINN	141.30k	10k	5.044e-03	N/A	N/A
Allen-Cahn	QRes	25.50k	10k	3.577e-03	N/A	N/A

Table 4: Comparing different network architectures for solving Burgers' equation.

Model	PINN	ISC	QSC	QRes-lite	APINN	QRes-full	QRes-lite
Params	3.02k	3.02k	1.54k	1.54k	3.02k	2.94k	1.54k
Optimizer	L-BFGS-B	L-BFGS-B	L-BFGS-B	L-BFGS-B	Adam	Adam	Adam
Error u	3.760e-03	1.504e-02	3.875e-03	3.386e-03	1.420e-01	8.630e-02	1.291e-01

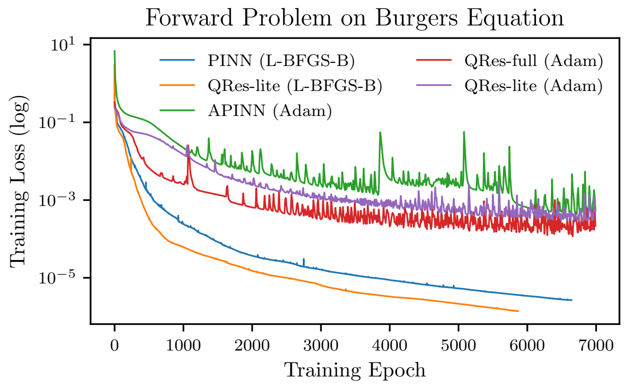


Figure 6: Training loss curves for solving Burgers' equation.

work, i.e., $n = 5$ and Adam optimizer. Next, we used two baselines architectures that share similar ideas as ResNet [25]. ISC, which is an abbreviation for *Identity ShortCut*, has the closest resemblance to ResNet, as it adds the layer input to the activation, i.e., $H^{(l)} = \sigma[WH^{(l-1)}+b]+H^{(l-1)}$. QSC denotes *Quadratic ShortCut*, which adds the quadratic residual after activation, i.e., $H^{(l)} = [W_1H^{(l-1)}] \circ [W_2H^{(l-1)}] + \sigma[W_1H^{(l-1)} + b]$.

For QRes networks, we tested two network sizes by adjusting network widths. The QRes-full has roughly the same number of parameters as the PINN and APINN, while QRes-lite has roughly half that number. Since the QSC also has two weight matrices, we set it to have the same width and depth as the QRes network (QRes-lite). While L-BFGS-B helped the models to reach higher accuracy, our experiments show that the APINN is very unstable when trained with L-BFGS-B. Therefore, we prepared QRes networks trained both with Adam and L-BFGS-B optimizers to have fair comparisons. We trained all models for 7k epochs with Adam, which is roughly the same number of epochs needed for L-BFGS-B to converge.

The results are shown in Table 4 and Figure 6. Both versions of the QRes networks outperform APINN when trained with Adam. For the group of models that were trained with L-BFGS-B optimizer, QRes-lite produces more accurate predictions than PINN and ISC, with smaller number of parameters. On the other hand, QSC performs even worse than PINN. In terms of convergence speed, Figure 6 further support that QRes networks are consistently faster than the baselines regardless of the choice of optimizers.

Analysis of Results: Figure 7 compares visualizations of pressure field predictions of PINN and QRes for N-S equations at different epochs in training. We can see the contour lines reveal a steep drop of pressure at the left of each figure (representing a region with high frequency patterns) where PINNs struggle to learn even after 50k epochs. On the other hand, QRes manages to digest the high frequency pattern (in regions where pressure values change abruptly) much faster than PINNs within 10k epochs.

5.1 Generalizing to General ML Problems:

To analyze the ability of QRes and DNNs to learn higher frequencies in general ML problems, we performed a toy experiment to fit a composition of mono-frequency sine waves [26]. The task is to fit 1k data points on the composited curve (shown in 8a) using mean square error loss functions. All models are trained for 20k epochs with Adam optimizer. The results are shown in Figure 8, which shows two interesting characteristics of QRes. First, QRes learns higher frequencies much faster

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 05/12/23 to 160.26.141.30 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

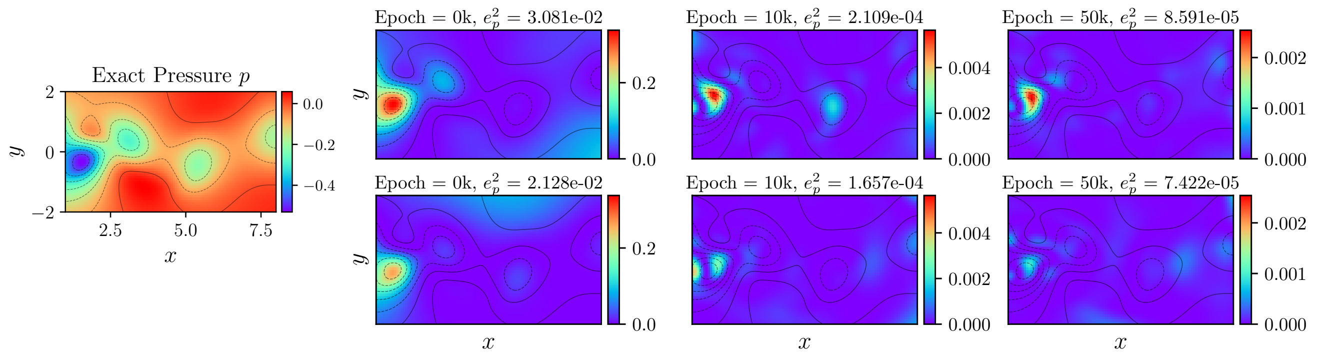
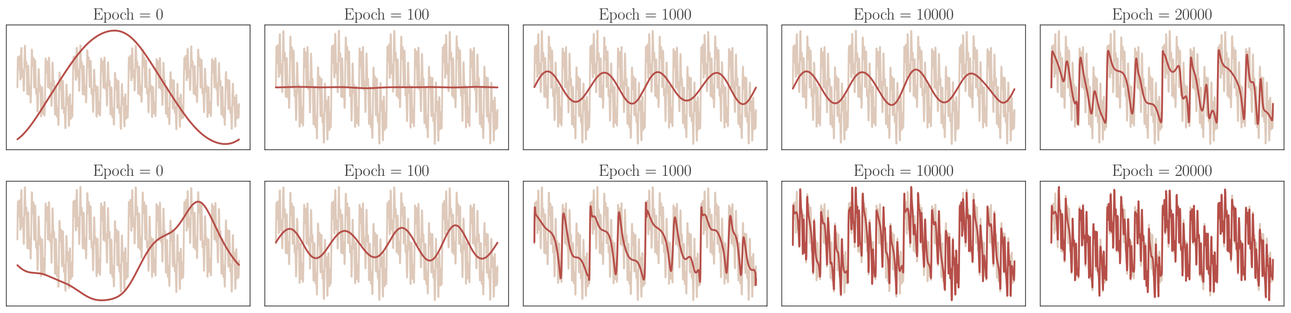
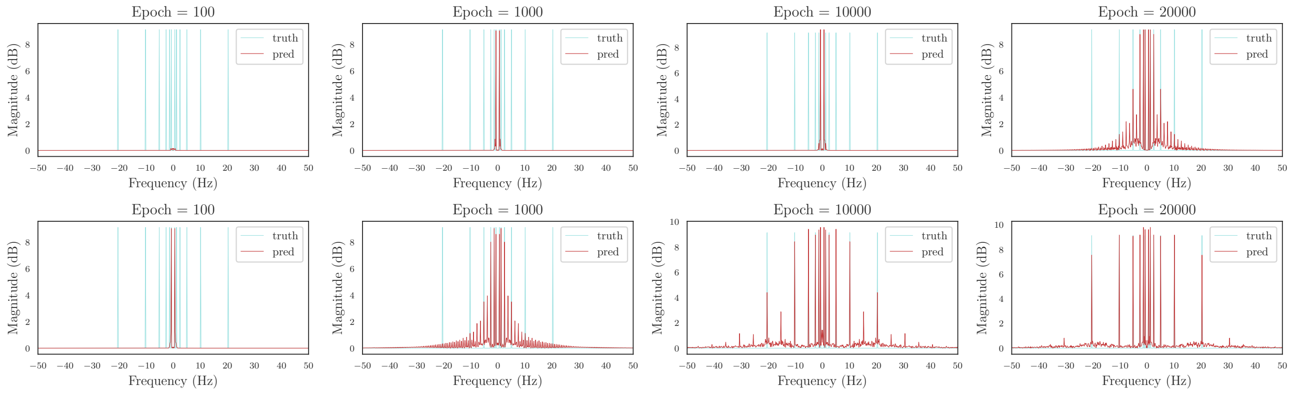


Figure 7: Visualizations of pressure field predictions for N-S equations. Left figure shows the exact pressure field while the remaining figures show error maps. The top row shows distribution of square error for PINN (1.38k parameters) and the bottom row is for the QRes networks (1.37k parameters). e_p^2 is the mean square error for the current epoch and dark contour lines represent the exact pressure field. Note that the scales of errors are different across epochs but are the same between QRes and PINN.



(a) Predicted (dark) and ground truth (light) waves for DNN (top) and QRes (bottom).



(b) Spectrum of the predicted (red) and ground truth (blue) waves for DNN (top) and QRes (bottom). The spectrum is obtained using shifted double-sided fast Fourier transform (FFT) on 1000 points.

Figure 8: Training a DNN (33.54k parameters) and a QRes network (33.21k parameters) to fit a superposition of sine waves with different frequencies: $4/2\pi, 8/2\pi, 16/2\pi, 32/2\pi, 64/2\pi$ and $128/2\pi$ Hz.

than DNNs, which supports our theoretical results. Second, like DNNs, the QRes exhibits the phenomena of *spectral bias* [26], which is a well-known phenomenon that neural networks learn lower frequencies earlier in training, making it possible to apply early stopping to avoid overfitting.

6 Conclusion

In this work, we proposed *quadratic residual networks* (QRes) as a new type of neural networks with suffi-

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 05/12/23 to 160.26.141.30 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

ciently high functional capacity (or expressive power). Using tools from algebraic geometry, we developed theories that prove the efficiency of the QRes networks. Following the original PINN work, we presented empirical evidences that QRes shows consistent advantages over deep neural networks in terms of parameter efficiency, convergence speed, and accuracy. Our work suggests that physics-informed deep learning can benefit from more nonlinearity in the network (as investigated by QRes), which can be the subject of future investigations. The remarkable advantage in learning higher frequencies further suggests the promise of using QRes networks in a broader range of ML applications. A full-length version of this article is available at [22]

Acknowledgments

This work was supported by NSF grant #2026710.

References

[1] P. Kidger and T. Lyons, "Universal approximation with deep narrow networks," in *COLT*, pp. 2306–2327, 2020.

[2] E. B. Baum and D. Haussler, "What size net gives valid generalization?," in *NeurIPS* (D. S. Touretzky, ed.), pp. 81–90, Morgan-Kaufmann, 1989.

[3] A. Karpatne, G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar, "Theory-guided data science: A new paradigm for scientific discovery from data," *TKDE*, vol. 29, no. 10, pp. 2318–2331, 2017.

[4] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating physics-based modeling with machine learning: A survey," *arXiv preprint arXiv:2003.04919*, 2020.

[5] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10561*, 2017.

[6] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10566*, 2017.

[7] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, 2020.

[8] X. Jin, S. Cai, H. Li, and G. E. Karniadakis, "Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations," *arXiv preprint arXiv:2003.06496*, 2020.

[9] H. Gao, L. Sun, and J.-X. Wang, "Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parametric pdes on irregular domain," *arXiv preprint arXiv:2004.13145*, 2020.

[10] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *JCP*, vol. 378, pp. 686–707, 2019.

[11] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks," *JCP*, vol. 404, p. 109136, Mar 2020.

[12] S. Wang, X. Yu, and P. Perdikaris, "When and why pinns fail to train: A neural tangent kernel perspective," 2020.

[13] K.-Y. Siu and V. Roychowdhury, "Optimal depth neural networks for multiplication and related problems," in *NeurIPS*, pp. 59–64, 1993.

[14] R. Durbin and D. E. Rumelhart, "Product units: A computationally powerful and biologically plausible extension to backpropagation networks," *Neural computation*, vol. 1, no. 1, pp. 133–142, 1989.

[15] S. Urban and P. van der Smagt, "A neural transfer function for a smooth and differentiable transition between additive and multiplicative interactions," *arXiv preprint arXiv:1503.05724*, 2015.

[16] H. Poon and P. Domingos, "Sum-product networks: A new deep architecture," 2012.

[17] A. Trask, F. Hill, S. Reed, J. Rae, C. Dyer, and P. Blunsom, "Neural arithmetic logic units," 2018.

[18] A. Madsen and A. R. Johansen, "Neural arithmetic units," in *ICLR*, 2020.

[19] F. Fan, W. Cong, and G. Wang, "A new type of neurons for machine learning," *International journal for numerical methods in biomedical engineering*, vol. 34, no. 2, p. e2920, 2018.

[20] J. Kileel, M. Trager, and J. Bruna, "On the expressive power of deep polynomial neural networks," 2019.

[21] L. Datta, "A survey on activation functions and their relation with xavier and he normal initialization," 2020.

[22] J. Bu and A. Karpatne, "Quadratic residual networks: A new class of neural networks for solving forward and inverse problems in physics involving pdes," *arXiv preprint*, 2021.

[23] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *JCP*, vol. 378, pp. 686 – 707, 2019.

[24] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi-newton method for large-scale optimization," *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1008–1031, 2016.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, pp. 770–778, 2016.

[26] R. Basri, D. W. Jacobs, Y. Kasten, and S. Kritchman, "The convergence rate of neural networks for learned functions of different frequencies," *CoRR*, vol. abs/1906.00425, 2019.

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited