The 8th European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS Congress 2022
5–9 June 2022, Oslo, Norway

EVALUATING MULTIGRID-IN-TIME ALGORITHM FOR LAYER-PARALLEL TRAINING OF RESIDUAL NETWORKS

Chinmay V. Datar[1], **Harald Köstler**[1,2]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg
Cauerstr. 11, 91058 Erlangen, Germany
e-mail: chinmay.datar@fau.de

[2] Zentrum für Nationales Hochleistungsrechnen Erlangen (NHR@FAU)
Martensstraße 1, 91058 Erlangen, Germany
email: harald.koestler@fau.de

Key words: Residual networks, regression, multigrid reduction in time, layer-parallel, optimal control, noise

Abstract. Replacing the traditional forward and backward passes in a residual network with a Multigrid-Reduction-in-Time (MGRIT) algorithm paves the way for exploiting parallelism across the layer dimension. In this paper, we evaluate the layer-parallel MGRIT algorithm with respect to convergence, scalability, and performance on regression problems. Specifically, we demonstrate that a few MGRIT iterations solve the systems of equations corresponding to the forward and backward passes in ResNets up to reasonable tolerances. We also demonstrate that the MGRIT algorithm breaks the scalability barrier created by the sequential propagation of data during the forward and backward passes. Moreover, we show that ResNet training using the layer-parallel algorithm significantly reduces the training time compared to the layer-serial algorithm on two non-linear regression tasks.

We observe much more efficient training loss curves using layer-parallel ResNets as compared to the layer-serial ResNets on two regression tasks. We hypothesize that the error stemming from approximately solving the forward and backward pass systems using the MGRIT algorithm helps the optimization algorithm escape flat saddle-point-like plateaus or local minima on the optimization landscape. We validate this by illustrating that artificially injecting noise in a typical forward or backward propagation, allows the optimizer to escape a saddle-point-like plateau at network initialization.

1 INTRODUCTION

From applications in daily life such as web-searching, spam filtering, and image and speech recognition, to advancing scientific fields like physics, mathematics, chemistry, and material science, deep learning has revolutionized the world. One of the major factors contributing to the success of deep neural networks is believed to be an increase in depth, i.e., the number of layers in the network [1]. However, the benefits of increased depth in neural networks do not come without a price.

Deep neural networks suffer from the well-known problems of vanishing and exploding gradients [2]. One of the most cited neural network architectures of the 21st century - the 'Residual Network' (ResNet) [1] mitigates this problem by the introduction of short paths termed as 'skip connections' that help in carrying the gradient throughout the deep networks.

The second problem with deep neural networks is that the traditional sequential propagation of data during the forward and backward passes of a neural network creates a barrier to scalability with respect to layers. The multigrid-reduction-in-time (MGRIT) algorithm [3] has been proposed in [4] to replace the traditional forward and backward passes, specifically for the ResNet architecture. This introduces a new dimension of parallelism across layers that was previously inaccessible. The study of convergence, scalability, and utility of this layer-parallel MGRIT algorithm is one of the key topics of this paper.

The feasibility of this layer-parallel approach and substantial speed-ups over the layer-serial algorithm have already been shown on classification tasks in [4]. However, many real-world regression problems could potentially benefit from this layer-parallel approach. Since the loss functions, activation functions, and architectures used for regression tasks are different than those used for classification tasks, convergence behavior needs to be studied separately. Thus, implementing necessary modifications to incorporate training of layer-parallel ResNets for regression tasks and studying the performance of the algorithm on regression tasks forms another key part of this paper.

Networks that are far deeper than wider suffer from vanishing gradients at network initialization [5], even if popular strategies like He initialization [6] or Glorot initialization [7] is used. The role of noise in escaping saddle points and local minima in non-convex optimization problems has been empirically observed and studied in numerous works, although the theory is still largely unknown [8], [9]. An attempt to answer how the layer-parallel algorithm might resolve this problem of vanishing gradients at initialization has been made in this paper.

2 CONNECTION OF RESIDUAL NETWORKS TO MULTIGRID-IN-TIME

The forward propagation of data in an N-layered ResNet from layer 'n' to layer 'n+1' is described as follows:

$$x^0 = x_{in} \tag{1}$$

$$x^{n+1} = \underbrace{x^n + \beta F(x^n, W^n)}_{:=\Phi(x^n, W^n)} \quad \text{for} \quad \text{n = 0, 1, ... , N-1} \tag{2}$$

where, the function $F(x^n, W^n)$ is defined as follows:

$$F(x^n, W^n) = f_a(\omega^n x^n + b^n) \tag{3}$$

x_{in} is the input to the ResNet at layer 0. x^n is the state/output of the layer n. $W^n = [\omega^n, b^n]$ contains the trainable parameters including the weights matrix ω^n and the bias vector b^n. β is a scalar, generally chosen between $[0, 1]$. The operator $\Phi : (x^n, W^n) \longrightarrow (x^{n+1})$ is defined as shown in equation (2). f_a is the activation function. The ReLU (Rectified Linear Unit) activation function defined as: $f_a(x) = \max(x, 0)$ is used in this work.

The link between ResNets and layer-parallelism comes from optimal control. Consider the following initial value problem:

$$\frac{\partial x(t)}{\partial t} = F(x^t, W^t) \quad t \in [0, T] \tag{4}$$

$$x(0) = x_{in} \tag{5}$$

where, t represents time, $x(t)$ represents the solution at time t, starting with an initial value at $t = 0$ given by x_{in}. T is the final time. The dynamical systems representation describes the evolution of the solution continuously starting from an initial value at $t = 0$. A crucial observation is that the forward propagation of the ResNet defined by equations (2), (1) could be interpreted as a forward Euler discretization in time of the initial value problem corresponding to equations (4) and (5). From the neural network perspective, x(t) represents the output or state of the neural network at layer t. Here, the network transforms the output/state $x(t)$ in discrete time-steps corresponding to successive layers.

This observation leads the ResNet training to be formulated as a discrete optimal control problem, which is then solved using the parallel-in-time algorithm [4]. Solving for the state, adjoint and design equations corresponds to executing forward pass, backward pass, and weights update steps of the ResNet training, respectively. The state and adjoint equations are then solved using the MGRIT algorithm. For instance, the state equations (forward pass) of the residual network can be formulated as a system of equations:

$$\begin{pmatrix} x^0 \\ x^1 - \Phi(x^0, W^0) \\ \vdots \\ x^N - \Phi(x^{N-1}, W^{N-1}) \end{pmatrix} = \begin{pmatrix} x_{in} \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{6}$$

where, block row k of this forward pass system represents the forward pass for layer k, starting with block row 0, which contains the input to the network. Instead of solving this state-space system sequentially for one layer at a time, it is solved using a parallel MGRIT algorithm which breaks the scalability barrier created by the sequential propagation of data across the layers. This facilitates parallelism across the layer dimension. Similarly, the MGRIT algorithm is also used to solve the adjoint system (backward pass) so that the gradients are propagated backwards in a layer-parallel manner.

3 EVALUATION OF THE MGRIT ALGORITHM

Two regression problems defined by non-linear analytical functions are used to evaluate the performance of the residual network with a layer-parallel approach. Both non-linear functions $f_1(x), f_2(x)$ are maps from $\mathbb{R}^6 \to \mathbb{R}^1$. The entries of the input vector $(x_i)_{\mathrm{i}=1,..,6}$ are sampled uniformly randomly from $[0, 4]$. The two non-linear functions $f_1(x)$ and $f_2(x)$ are defined as follows:

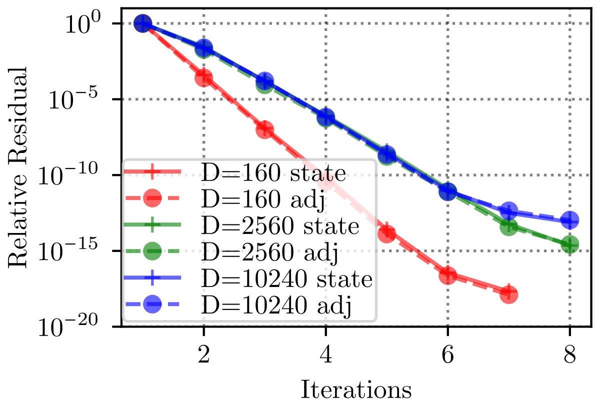
$$f_1(x) = \begin{cases} \sum_{i=1}^{6} (x_i)^i & \text{if } \sum_{i=1}^{6} (x_i)^i < 400 \\ 400 & \text{if } 400 \leq \sum_{i=1}^{6} (x_i)^i \end{cases} \tag{7}$$

$$f_2(x) = \begin{cases} \sum_{i=1}^{6}(x_i)^i & \text{if } \sum_{i=1}^{6}(x_i)^i < 400 \\ 400 & \text{if } 400 \leq \sum_{i=1}^{6}(x_i)^i < 800 \\ 800 & \text{if } 800 \leq \sum_{i=1}^{6}(x_i)^i \end{cases} \tag{8}$$

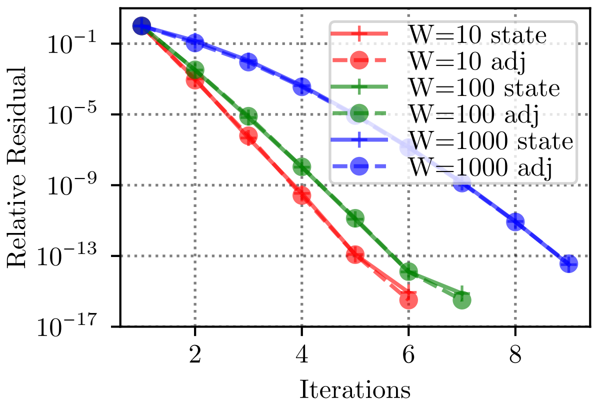
A deep residual network(ResNet) architecture with fully connected/dense layers is used in this study. The scaling factor β in equation (2) is crucial for the stability of training. The ReLU activation function has been shown to alleviate the problem of vanishing gradients [10] and is thus adopted in this study. The Xbraid package [11] has been used for the scalable parallel-in-time implementation of the multigrid-in-time algorithm. Moreover, the layer-parallel code developed in [4] has been modified and used in this work. Now we evaluate the layer-parallel MGRIT algorithm on regression tasks with respect to convergence, scalability, and accuracy.

3.1 Convergence

So first, we ask the question: Can the MGRIT algorithm solve state and adjoint equation systems up to an acceptable tolerance? To this end, the convergence history of state and adjoint equations is plotted in figure 1 for ResNets with gradually increasing depth (fixed width = 16) and for ResNets with gradually increasing width (fixed depth = 40). The relative residual for iteration k is defined as : $||r_k||/||r_1||$, where $||.||$ represents the l^2-norm and $||r_k||$, $||r_1||$ are the absolute residuals at iteration k and 1 respectively. In all the cases, we observed that the first iteration of the MGRIT algorithm yields $||r_1|| < 1$. F-cycle MGRIT with 1 relaxation sweep on each time-grid level was used in this study. We represent the number of layers (depth) by D and the number of neurons per layer (width) of the network by W.



(a) Varying depth D



(b) Varying width W

Figure 1: Convergence history of state equations (forward pass) and adjoint equations (backward pass) using MGRIT algorithm used to achieve layer-parallelism across ResNets

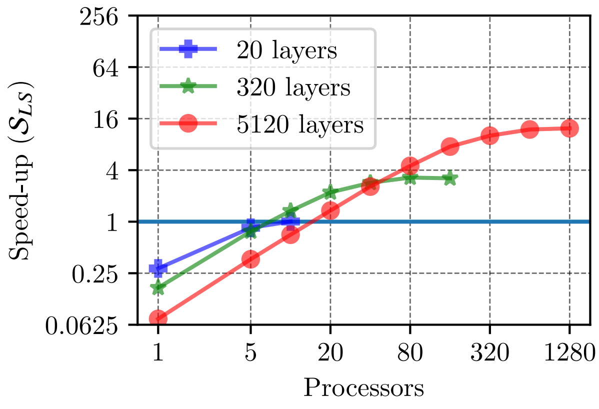
We observe that the layer-parallel MGRIT algorithm reduces the residuals of the state and adjoint equations rapidly and up to an acceptable tolerance, irrespective of the depth and the width. More details on the values of acceptable tolerances for the state and adjoint equations are discussed later. However, it is important to emphasize that higher accuracy of forward and backward passes does not necessarily translate into higher accuracy of training on a given classification or regression task. In practice, especially in deep learning, an accuracy to machine precision is often not desired.

Secondly, only 1-2 more iterations are required for the ResNet with 10240 layers to produce a relative residual of the same order of magnitude as compared with the ResNet with 160 layers, though the problem size is bigger by a factor of 64. The same holds true for increasingly wider networks. The network dimensions in consideration are much larger than the ones used in practice. Thus, the convergence results demonstrate that the MGRIT algorithm can be used in most of the practical ResNets.

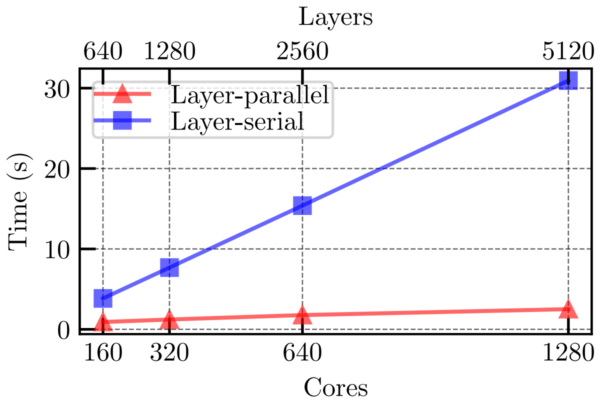
3.2 Scalability

A strong scaling study allows evaluating how speed-up over a fixed problem size scales with the number of processors. The speed-up in this context is defined as $\mathcal{S}_{LS} = \frac{T^1_{LS}}{T^{\|}_{LP}}$, where, T^1_{LS} is the time required by a single CPU using a layer-serial algorithm and $T^{\|}_{LP}$ is the time required by P processors using the layer-parallel MGRIT algorithm to execute one forward pass and one backward pass for one batch of examples.

To this end, the figure 2a demonstrates how speed-up scales with the processor count for ResNets with a depth of 20, 320, and 5120, respectively. The width is fixed at 16. The horizontal solid blue line indicates the threshold beyond which the layer-parallel algorithm is faster.



(a) Strong scaling



(b) Weak scaling

Figure 2: Evaluation of the MGRIT algorithm with respect to parallel scalability

We observe that the deeper the network, the higher the potential for parallelism across the layers and the greater the speed-ups. Using fewer layers restricts parallelism by restricting the maximum number of processors. Moreover, parallel efficiency is much higher for deeper networks as more processors can be exploited efficiently.

Secondly, we see that increasing the number of processors beyond a particular point does not increase the speed-up further owing to the barrier imposed by the serial fraction of the code.

The computational costs of the layer-parallel MGRIT algorithm and the layer-serial algorithm scale linearly with respect to the number of layers. On the one hand, the MGRIT algorithm requires more work (floating-point operations) per iteration as compared to the layer-serial algorithm. On the other hand, the MGRIT algorithm has a much higher parallel workload as compared to the layer-serial algorithm. Thus, the MGRIT algorithm trades parallelism for additional computational work. This explains why the MGRIT algorithm achieves speed-ups over the layer-serial algorithm by exploiting at least around 5-20 processors depending on the problem size.

In weak scaling, the workload increases with the processor count, unlike in strong scaling. Figure 2b shows the weak scaling behavior of the multigrid-in-time algorithm. The number of layers distributed per processor has been fixed to 4. The number of compute cores and the number of layers are doubled progressively from 80 to 1280 (bottom x-axis) and from 320 to 5120 (top x-axis) respectively. The blue curve indicates that the runtime for the layer-serial algorithm scales linearly with respect to the number of layers using a single CPU (the bottom x-axis is irrelevant for the blue curve). Whereas, a multigrid-in-time algorithm facilitates exploiting parallelism across the layers, thereby, breaking the scalability barrier with respect to the number of layers. The slight increase in run times for the parallel algorithm with an increasing number of resources is mainly due to the communication overhead between the processors.

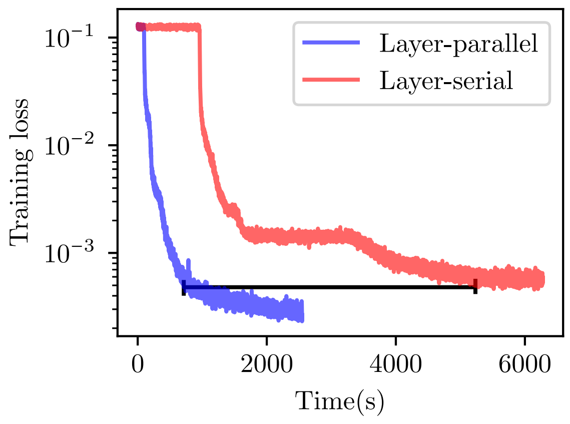
3.3 Performance

The goal of this section is to try to answer the following questions: How do the training losses obtained using layer-serial and layer-parallel algorithms compare? Does the layer-parallel algorithm outperform the layer-serial algorithm in terms of total training time, and if yes, what are the speed-up factors? Can one circumvent the problem of vanishing gradients at initialization using a layer-parallel algorithm?

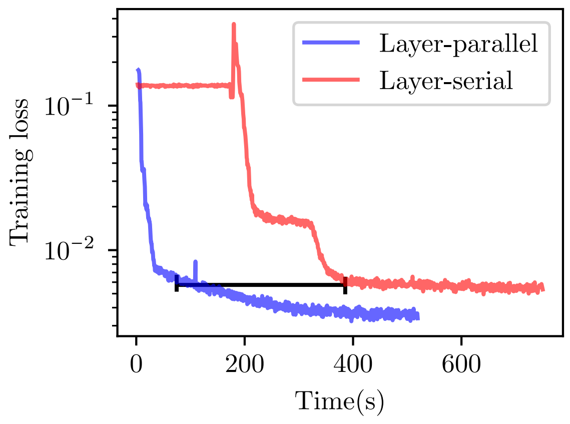
A fully connected ResNet architecture is used for the regression problems under consideration. Convolutional layers are not used to avoid enforcing local connectivity in features when none is expected. The widths of the input, output, and hidden layers are 6, 1, and 16, respectively, for both regression problems. ResNets with 322 layers and 50 layers are used for the first and the second regression tasks, respectively. ReLU activation function is used. The scaling factor $\beta = 0.1$ in equation 2 is used. The neural network is initialized with random values generated between $[0, 10^{-3}]$. For both regression tasks, each successive coarser time grid has 4 times fewer time points (layers). The MGRIT iterations are continued until l^2-norms of residuals of forward and backward pass systems are less than the user-defined tolerances for the forward and backward passes (t_f, t_b) respectively. The tolerance values in the range $[10^{-2}, 10^{-14}]$ in steps of 10^{-2} were included in the hyper-parameter study. The optimal pairs of tolerance values based on the lowest training loss achieved on the two regression tasks were $(10^{-8}, 10^{-8})$ and $(10^{-4}, 10^{-6})$ respectively. The network is trained in mini-batches using the gradient-based simultaneous optimization approach - One-shot method [12] that updates optimization parameters simultaneously while solving for time-dependent network states. Inexact gradient information is used to accelerate the training process [12], [4].

The compute nodes of the Meggie cluster of the Regional Data Center in Erlangen (RRZE) at the Friedrich-Alexander University of Erlangen-Nuremberg are used for training neural networks. The compute nodes are equipped with Intel Xeon E5-2630v4 “Broadwell” chips which have 10 cores per chip. To ensure accurate time measurements and reproducible results, the clock frequency is fixed to 2.2 GHz and the process affinity is controlled.

To gain insights into the accuracy of predictions and speed-ups based on total training time, training loss with respect to time is plotted for the layer-serial and layer-parallel algorithms for both regression tasks in figure 3.



(a) Non-linear regression task - 1 (eq (7))



(b) Non-linear regression task - 2 (eq (8))

Figure 3: Training loss with respect to time for layer-serial and layer-parallel implementation

The following table compares the lowest losses achieved on training and validation data sets using the layer-parallel and layer-serial algorithms on both regression tasks. The final losses

Table 1: Lowest losses attained on training and validation data sets

Dataset	**Training loss**		**Validation loss**	
	Layer-parallel	**Layer-serial**	**Layer-parallel**	**Layer-serial**
1	2.46×10^{-4}	4.53×10^{-4}	2.82×10^{-4}	5.52×10^{-4}
2	2.36×10^{-3}	2.84×10^{-3}	3.31×10^{-3}	3.75×10^{-3}

attained by the layer-parallel algorithm on training and validation data sets, on the first and second regression tasks are slightly lower and slightly higher, respectively, as compared to [13]. Moreover, the losses obtained on validation data sets are quite close to the ones obtained on training data sets. This confirms that the network generalizes well on unseen examples and that the network does not over-fit on examples in the training data set.

Approximately solving for forward and backward passes using the MGRIT algorithm yields slightly lower losses as compared to the exact layer-serial algorithm. Moreover, the loss using the layer-serial algorithm is observed to stagnate at initialization and midway through training in brief patches. Whereas, the layer-parallel algorithm demonstrates much more efficient loss curves - free from the stagnation of losses till the final loss is attained. Our hypothesis explaining this behavior is discussed later.

Let t_{ser} and t_{par} be the times in seconds required by layer-serial algorithm and layer-parallel algorithms, respectively, in attaining losses lower than some reference threshold - indicated by the horizontal lines in the figure 3. The speed-up based on total training times in attaining these reference losses are defined as: $\text{Speedup}^{\text{training}} = t_{\text{ser}}/t_{\text{par}}$.

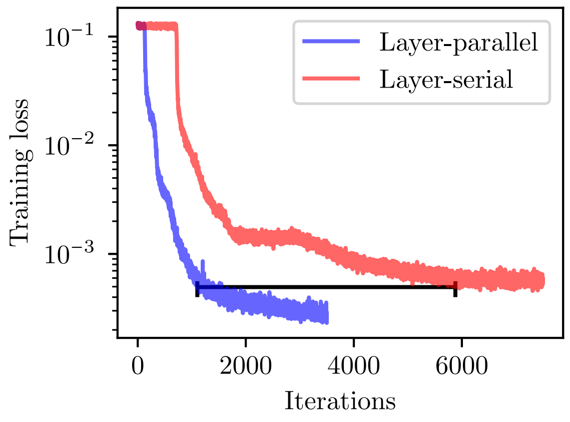
The following table summarizes the speed-ups on both regression tasks:

Table 2: Speed-up in total training time

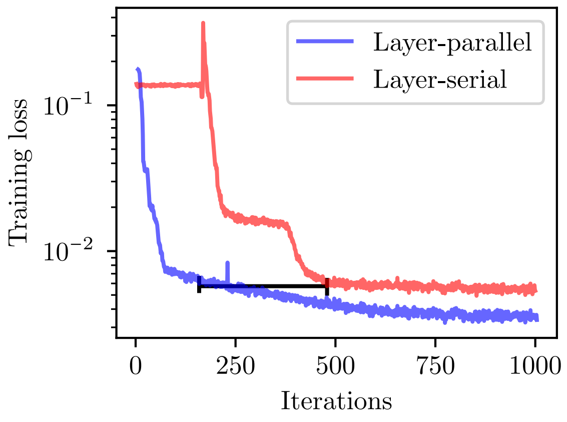
Dataset	**Reference training loss**	$\mathbf{t_{par}(s)}$	$\mathbf{t_{ser}(s)}$	$\mathbf{Speedup^{training}}$
1	5×10^{-4}	718	5233	7.2x
2	6×10^{-3}	74	385	5.2x

The substantial speed-ups successfully demonstrate the utility of the layer-parallel algorithm in reducing the total training time. However, we observe that the speed-ups stemming from the faster execution of the forward and backward passes using the layer-parallel algorithm only shrink the total training time by 42% and 70% on the first and second regression tasks respectively. So, what is responsible for the substantial speed-up factors of 7.2 and 5.2 observed before on both regression tasks, respectively?

To attempt to answer this question, training loss with respect to iterations is plotted for both regression tasks in figure 4. The reference loss is indicated by the horizontal solid black lines in figure 4.



(a) Non-linear regression task - 1 (eq (7))



(b) Non-linear regression task - 2 (eq (8))

Figure 4: Training loss using layer-serial and layer-parallel algorithms with respect to iterations

At initialization, the layer-parallel algorithm starts reducing the loss in substantially fewer iterations as compared to the layer-serial algorithm on both regression tasks. Moreover, for the same iteration number, the training loss attained by the layer-parallel algorithm is lower as compared to the layer-serial algorithm. The following table summarizes the respective number of iterations required to achieve some reference loss using layer-serial and layer-parallel algorithms, respectively.

Table 3: Training loss vs optimization iterations

Dataset	**Training loss**	$\mathbf{Iter_{parallel}}$(s)	$\mathbf{Iter_{serial}}$(s)	$\mathbf{Iter_{serial}/Iter_{parallel}}$
1	5×10^{-4}	1103	5880	5.3
2	6×10^{-3}	159	479	3.01

Factors 5.3 and 3 indicate that a large fraction of the speed-up results from efficient learning curves of the layer-parallel algorithm. However, this is not always the case. It is important to note that for different MGRIT tolerance values, the dominant fraction of the total speed-up stems from the faster execution of forward and backward passes. However, for the examples in consideration, this led to a decrease in total speed-up.

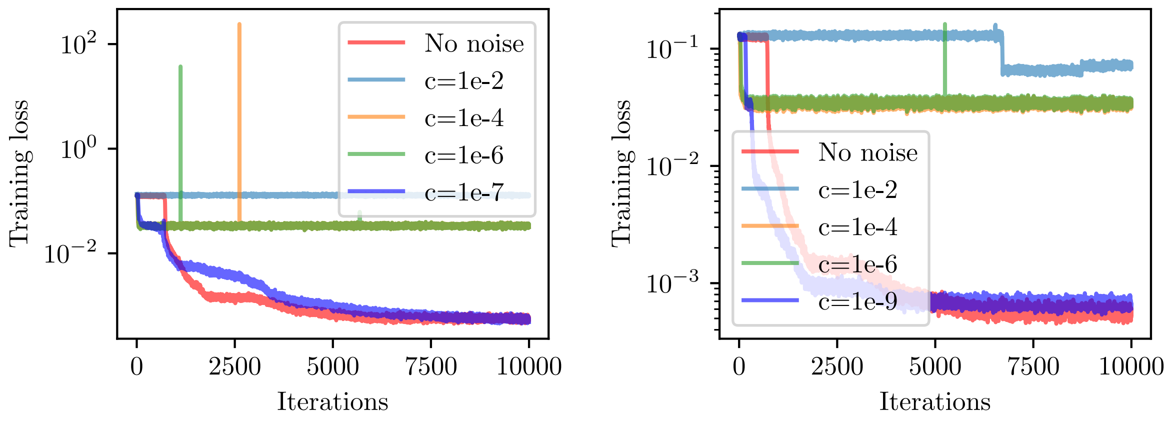
The stagnation of training loss at network initialization is an artifact of the vanishing gradient problem for deep neural networks. It is demonstrated in [5], that for networks much deeper than wider, the entries of gradient, as well as the Hessian of the objective function, get smaller. Random independent and identically distributed initialization results, it optimizers being initialized in a flat plateau, which is especially challenging for the stochastic gradient descent algorithm to escape [14]. The layer-parallel algorithm appears to be less susceptible to the optimizer getting the objective function trapped in a flat plateau at initialization as well as in the local minimum. It is important to stress that this behavior has been observed in both regression cases.

Many studies have shown that adding noise in neural network training using various approaches, allows the optimization algorithm to escape local minima and saddle-point-like plateaus of the objective function [8], [9]. To explain efficient learning curves using a layer-parallel algorithm, we hypothesize that the error stemming from the approximation of layer-serial forward and backward passes by an iterative multigrid-in-time algorithm allows the optimization algorithm to escape from this flat plateau in much fewer iterations and to avoid getting stuck into a local minimum.

To test this hypothesis, we conducted the following experiments on a traditional ResNet with layer-serial forward and backward passes. In the first experiment, random noise is artificially injected into the forward propagation step in the output of each layer. In the second experiment, random noise is injected in the backward propagation step of each layer. The magnitude of the noise is governed by the parameter c. Random numbers are sampled from a uniform distribution between [-c, c] for each neuron of each hidden layer. With all the other hyper-parameters fixed, a neural network is then trained on the first data set for different values of 'c' ranging from $[10^{-1}, 10^{-2}, ..., 10^{-10}]$.

Adding noise to the output of each layer in the forward pass results in a perturbed output of each layer, resulting in a perturbed prediction produced by the neural network. Whereas, adding noise in the backward pass, updates the weights along a direction perturbed from the steepest descent direction. The resulting training loss with respect to iterations for the first and second experiments are plotted in figure 5a and figure 5b respectively.

Loss curves in figure 5 are displayed only for a few selected values of the scaling constant c in order to avoid redundancy. The first crucial observation is that for multiple values of scaling factor c, the first rapid reduction of loss occurs in much fewer iterations on both data sets. This behavior has been observed for a wide range of values of c between $[10^{-3}, 10^{-9}]$, though not all



(a) Experiment 1: loss curve for artificially injected noise in forward pass
(b) Experiment 2: loss curve for artificially injected noise in backward pass

Figure 5: Effect of adding uniformly distributed noise in range $[-c, c]$ in the outputs of forward and backward propagation steps of each hidden layer on loss curves

loss curves corresponding to different values of c are included in figure 5.

Adding too much noise in both experiments, for instance, for $c = 10^{-2}$ renders the network incapable of extracting any patterns. For $c \in [10^{-3}, 10^{-6}]$, the optimization algorithm achieves a first rapid decline of training loss in much fewer iterations. At the beginning of the training, the added noise appears to be helping the objective function escape the flat plateau at initialization. However, once some local minimum is attained, the magnitude of noise starts dominating and the optimization algorithm fails at reducing the loss further.

These observations support our hypothesis that adding noise in training can lead the optimizer to escape the flat plateau at initialization. Much more evidence needs to be provided to support this hypothesis. However, this will be an interesting investigation for future work.

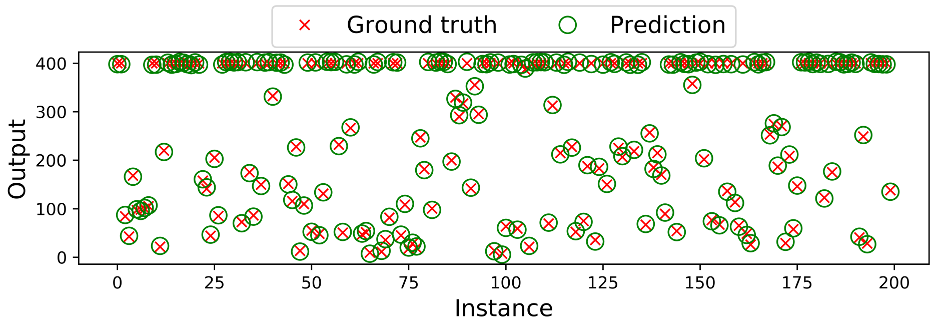
Lastly, the figure 6 demonstrates that layer-parallel ResNet predictions on both non-linear regression tasks are quite accurate.

4 CONCLUSIONS

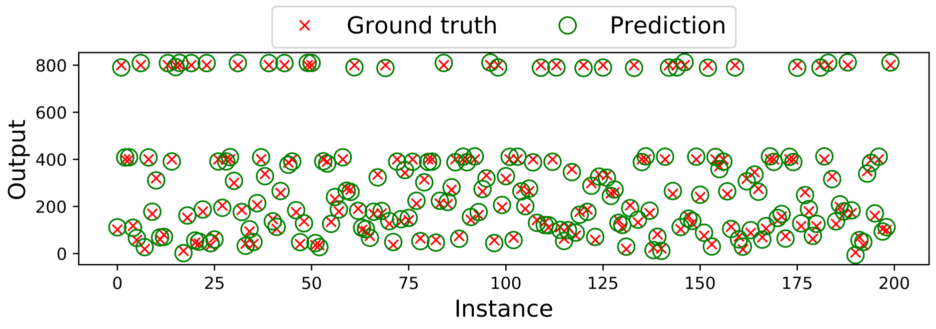
We first extend the framework [4] to facilitate training ResNets in a layer-parallel manner on regression tasks. In this paper, we evaluate the MGRIT algorithm that allows layer-parallel training of deep residual networks with respect to convergence, scalability, and performance on regression tasks.

First, we demonstrate that the layer-parallel algorithm can solve systems of equations corresponding to the forward and the backward passes respectively, up to a reasonable tolerance with a few MGRIT iterations. We demonstrate this for networks ranging from very deep (with up to 10,000 layers) to very wide (with up to 1000 neurons per layer). We recommend using 1 relaxation sweep on each time-grid level in V and F cycles, as additional relaxation sweeps do not increase the convergence rate significantly.

In our experiments, we see that the number of processors required by the layer-parallel



(a) Non-linear regression task - 1 (eq (7))



(b) Non-linear regression task - 2 (eq (8))

Figure 6: Layer-parallel ResNet prediction Vs ground truth on 200 instances from the test data for data sets 1 and 2

algorithm to outperform the layer-serial algorithm lies between 5 to 20 depending on the network depth and tolerance. We recommend performing a hyper-parameter search for finding optimal tolerances for forward and backward propagation systems, as these influence the run time of the layer-parallel algorithm. Secondly, we demonstrate that the layer-parallel algorithm breaks the scalability barrier with respect to the depth of the network on regression tasks. Moreover, deeper networks have higher parallel efficiency and exhibit higher speed-ups over the layer-serial algorithm.

In our experiments, the ResNet training with a layer-parallel algorithm was able to attain a lower loss and much faster (roughly 7.2x and 5.2x faster) as compared to the ResNet trained using a layer-serial algorithm.

Lastly, we demonstrate that adding noise to a layer-serial forward or a backward pass helps the optimizer escape the flat saddle-point-like plateau at initialization. This supports our hypothesis that the noise induced in the training by approximately solving the forward and backward pass systems helps the optimization algorithm to escape a flat plateau at initialization and local minima. However, much more evidence and theoretical studies are needed to bolster these empirical results.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[2] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," *Diploma, Technische Universität München*, vol. 91, no. 1, 1991.

[3] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder, "Parallel time integration with multigrid," *SIAM Journal on Scientific Computing*, vol. 36, no. 6, pp. C635–C661, 2014.

[4] S. Gunther, L. Ruthotto, J. B. Schroder, E. C. Cyr, and N. R. Gauger, "Layer-parallel training of deep residual neural networks," *SIAM Journal on Mathematics of Data Science*, vol. 2, no. 1, pp. 1–23, 2020.

[5] A. Orvieto, J. Kohler, D. Pavllo, T. Hofmann, and A. Lucchi, "Vanishing curvature and the power of adaptive methods in randomly initialized deep networks," *arXiv preprint arXiv:2106.03763*, 2021.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[7] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

[8] Y. Fang, Z. Yu, and F. Chen, "Noise helps optimization escape from saddle points in the synaptic plasticity," *Frontiers in neuroscience*, vol. 14, p. 343, 2020.

[9] T. Liu, Y. Li, S. Wei, E. Zhou, and T. Zhao, "Noisy gradient descent converges to flat minima for nonconvex matrix factorization," in *International Conference on Artificial Intelligence and Statistics*, pp. 1891–1899, PMLR, 2021.

[10] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011.

[11] "XBraid: Parallel multigrid in time." http://llnl.gov/casc/xbraid.

[12] T. Bosse, N. R. Gauger, A. Griewank, S. Günther, and V. Schulz, "One-shot approaches to design optimzation," *Trends in PDE Constrained Optimization*, pp. 43–66, 2014.

[13] D. Chen, F. Hu, G. Nian, and T. Yang, "Deep residual learning for nonlinear regression," *Entropy*, vol. 22, no. 2, p. 193, 2020.

[14] H. Daneshmand, J. Kohler, A. Lucchi, and T. Hofmann, "Escaping saddles with stochastic gradients," in *International Conference on Machine Learning*, pp. 1155–1164, PMLR, 2018.