ESAIM: PROCEEDINGS AND SURVEYS, August 2021, Vol. 71, p. 131-144
Didier Auroux, Jean-Baptiste Caillau, Régis Duvigneau, Abderrahmane Habbal, Olivier Pantz, Luc Pronzato, Ludovic Rifford

MULTILEVEL MINIMIZATION FOR DEEP RESIDUAL NETWORKS

Lisa Gaedke-Merzhäuser*[1], Alena Kopaničáková*[1] and Rolf Krause[1]

Abstract. We present a new multilevel minimization framework for the training of deep residual networks (ResNets), which has the potential to significantly reduce training time and effort. Our framework is based on the dynamical system's viewpoint, which formulates a ResNet as the discretization of an initial value problem. The training process is then formulated as a time-dependent optimal control problem, which we discretize using different time-discretization parameters, eventually generating multilevel-hierarchy of auxiliary networks with different resolutions. The training of the original ResNet is then enhanced by training the auxiliary networks with reduced resolutions. By design, our framework is conveniently independent of the choice of the training strategy chosen on each level of the multilevel hierarchy. By means of numerical examples, we analyze the convergence behavior of the proposed method and demonstrate its robustness. For our examples we employ a multilevel gradient-based methods. Comparisons with standard single level methods show a speedup of more than factor three while achieving the same validation accuracy.

1. Introduction

Deep residual networks or ResNets are widely used architectures that demonstrate state-of-the-art performance in complex statistical learning tasks with applications in various fields, such as computer vision [8, 26], or speech recognition [41, 42]. The popularity of ResNets originates from their remarkable performance in the ImageNet [37] and the MS COCO [33] image recognition competitions.

A major drawback of very deep ResNets is their long training time. To mitigate this issue, different strategies have been proposed, for example networks with stochastic depth [25], mollifying networks [16], spatially adaptive architectures [12], or multilevel parameter initialization strategies [7, 19].

In this work, we propose to accelerate the training of ResNets using multilevel minimization. Our work is motivated by the fact that the network depth is of paramount importance for achieving the necessary approximation properties [21, 38]. However, very deep networks are computationally expensive to train, as the cost of forward-backward propagation scales linearly with respect to the number of parameters. In contrast, shallower networks might not show the necessary approximation properties, but their training cost is relatively low. Our multilevel framework exploits a multilevel hierarchy of auxiliary networks with different depths. The training of the deepest network is then accelerated by internally training the shallower networks.

The proposed multilevel framework is inspired by multigrid methods [4, 20], which have originally been developed for the solution of elliptic partial differential equations. An extension of linear multigrid methods to nonlinear problems, called full approximation scheme (FAS), can be found in [3]. Later, several nonlinear multilevel minimization techniques have emerged, for example the multilevel line-search method (MG/OPT) [34], the recursive multilevel trust region method (RMTR) [14,15,27,28], or higher-order multilevel optimization

*Authors contributed equally.

[1] Institute of Computational Science, Università della Svizzera italiana

Article published online by EDP Sciences and available at https://www.esaim-proc.org or https://doi.org/10.1051/proc/202171131

strategies [5,6]. Our multilevel minimization method can be seen as a variant of an MG/OPT framework which is tailored for training ResNets.

The main challenge in designing an efficient multilevel minimization framework is to construct a suitable multilevel hierarchy. Here, we leverage the emerging dynamical system's viewpoint [19,40], which casts a ResNet as the discretization of an initial value problem. The training process is then formulated as the minimization of a time-dependent optimal control problem. As a consequence, we can obtain a hierarchy of ResNets with different depths by discretizing the same optimal control problem with different discretization parameters.

A dynamical system's viewpoint was first used in a multilevel context in [7], where the authors trained shallow networks to initialize parameters of a deep network. The same parameter initialization strategy was recently extended for layer-parallel training of ResNets [10]. Our method differs from the methods proposed in [7] and [10], as we take advantage of a multilevel hierarchy during the whole training process, not only in the beginning. Nevertheless, it is possible to incorporate a multilevel initialization strategy into our multilevel minimization framework. We do not exploit this possibility in the presented paper, as aim of this work is to test the proposed multilevel training framework by itself.

This work makes the following contributions:

- We present an abstract nonlinear multilevel minimization framework for training deep residual networks.
- Using our multilevel framework, we propose multilevel variants of gradient and mini-batch gradient methods.
- We numerically analyze the convergence behavior of our multilevel training strategies using two different datasets and ResNets with more than $2,000$ layers. In addition, comparisons with a standard single level methods are made, which demonstrate a speed-up of more than factor three.

2. Deep Residual Networks

This section provides a brief overview of deep residual networks (ResNets) in the context of supervised classification. Through the following, we consider a dataset $\mathcal{D} = \{(\boldsymbol{x}_j, \boldsymbol{c}_j)\}_{j=1}^{p}$ of p samples. Each sample is a pair consisting of an input feature $\boldsymbol{x}_j \in \mathbb{R}^q$ and its corresponding label $\boldsymbol{c}_j \in \mathbb{R}^m$. The size of the label vector $\boldsymbol{c}_j$ is determined by the number of output classes, i.e. m, as the i-th component of vector $\boldsymbol{c}_j$ corresponds to the probability of example $\boldsymbol{x}_j$ belonging to the i-th class.

2.1. Classification

The main idea behind supervised learning is to construct a model, which describes the relationship between input and output for a labeled dataset $\mathcal{D}$. The model function $f_m : \mathbb{R}^q \times \mathbb{R}^n \to \mathbb{R}^m$ is parametrized by a set of parameters $\boldsymbol{\Theta} \in \mathbb{R}^n$, where n denotes the number of model parameters. The process of finding suitable parameters $\boldsymbol{\Theta}$ is called training and it usually requires solving the following minimization problem:

$$\min_{\boldsymbol{\Theta}} \frac{1}{p} \sum_{j=1}^{p} \ell(f_m(\boldsymbol{x}_j, \boldsymbol{\Theta}), \boldsymbol{c}_j) + \mathcal{R}(\boldsymbol{\Theta}), \tag{1}$$

where a loss function $\ell : \mathbb{R}^m \to \mathbb{R}$ measures the deviation of the the predicted output from the known label. The regularizer $\mathcal{R} : \mathbb{R}^n \to \mathbb{R}$ in (1) is chosen such that it ensures the existence and regularity of the parameters $\boldsymbol{\Theta}$. A common choice for the regularizer is Tikhonov regularization [11], however other possibilities have also been used, see for example [35].

In the context of classification, the model function f_m is constructed by composing the forward propagation $f : \mathbb{R}^q \times \mathbb{R}^r \to \mathbb{R}^v$ with the hypothesis function $\mathcal{P} : \mathbb{R}^m \to \mathbb{R}^m$. Here, the symbols r and v denote the number of parameters and output dimension of forward propagation, respectively. The forward propagation filters input features in a nonlinear manner, while the hypothesis function predicts the class label probabilities using the

output of the forward propagation. In abstract form, the model function f_m is defined as

$$f_m(\boldsymbol{x}, \boldsymbol{\Theta}) = \mathcal{P}(\mathbf{W}_K f(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{b}_K), \tag{2}$$

where we split the model parameters $\boldsymbol{\Theta}$ into parameters of classification $\boldsymbol{\theta}_K$ and forward propagation $\boldsymbol{\theta}$, thus $\boldsymbol{\Theta} = \{\boldsymbol{\theta}, \boldsymbol{\theta}_K\}$. The classification parameters $\boldsymbol{\theta}_K = \{\mathbf{W}_K, \boldsymbol{b}_K\}$ consist of weights $\mathbf{W}_K \in \mathbb{R}^{m \times v}$ and biases $\boldsymbol{b}_K \in \mathbb{R}^m$. For multinomial classification problems, it is common to employ a cross-entropy loss function together with the softmax hypothesis function. For alternatives choices, we refer interested readers to [13].

2.1.1. *Forward propagation via ResNet*

In deep learning, the neural network constitutes a form of forward propagation function f. The parametric function f is created by concatenating many functions, called layers. Each layer k is usually composed of affine linear and point-wise nonlinear transformations, that are parametrized by the layer parameters $\boldsymbol{\theta}_k \in \mathbb{R}^d$.

In this work, we consider residual networks with identity shortcut connections [23]. The propagation of the input sample $\boldsymbol{x}$ through a network with K residual layers can be then expressed as

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \mathcal{F}(\boldsymbol{y}_k, \boldsymbol{\theta}_k), \ k \in \{0, \dots, K-1\}, \tag{3}$$

where $\boldsymbol{y}_k \in \mathbb{R}^v$ denotes the state of layer k. For simplicity, Equation (3) assumes a constant network width v. Hence, we map an input sample $\boldsymbol{x} \in \mathbb{R}^q$ into the feature space with the help of the linear operator $\mathbf{Q} \in \mathbb{R}^{v \times q}$, e.g. $\mathbf{y}_0 := \mathbf{Q}\boldsymbol{x}$. The elements of the matrix $\mathbf{Q}$ can be fixed or learned during the training process.

The transformation $\mathcal{F} : \mathbb{R}^v \times \mathbb{R}^d \to \mathbb{R}^v$ from (3) describes the residual module, c.f. [22]. Here, we assume that $\mathcal{F}$ takes form of the simple one layer perceptron

$$\mathcal{F}(\boldsymbol{y}_k, \boldsymbol{\theta}_k) := \sigma(\mathbf{W}_k \boldsymbol{y}_k + \boldsymbol{b}_k), \tag{4}$$

where $\sigma : \mathbb{R}^v \to \mathbb{R}^v$ is the nonlinear activation function, for example the rectified linear unit (ReLU), defined as $\sigma(\boldsymbol{z}) := \max\{\mathbf{0}, \mathbf{z}\}$. For alternatives, such as logistic sigmoid, or hyperbolic tangent, see [13]. The affine transformations in (4) are defined by a set of layer parameters $\boldsymbol{\theta}_k := \{\mathbf{W}_k, \boldsymbol{b}_k\}$ consisting of weights $\mathbf{W}_k \in \mathbb{R}^{v \times v}$ and biases $\boldsymbol{b}_k \in \mathbb{R}^v$, where v denotes the network width (the number of neurons associated with a given layer). The linear operator $\mathbf{W}_k$ can be a dense matrix, or sparse, e.g. in the case of a convolutional neural network, where it expresses the convolutional operator, see [13].

2.2. **Classification as optimal control problem**

Following [19], Equation (3) can be seen as a simplification of the more generic formula for a one-step method

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \Delta_t \mathcal{F}(\boldsymbol{y}_k, \boldsymbol{\theta}_k), \tag{5}$$

with $\Delta_t = 1$. Now, the forward propagation through the network (5) can be interpreted as a forward Euler discretization of the initial value problem

$$\begin{aligned} \partial_t \boldsymbol{y}(t) &= \mathcal{F}(\boldsymbol{y}(t), \boldsymbol{\theta}(t)), \qquad \forall t \in (0, T], \\ \boldsymbol{y}(0) &= \mathbf{Q}\boldsymbol{x}. \end{aligned} \tag{6}$$

The dynamical system above then continuously transforms the initial state $\boldsymbol{y}(0)$ into the network output $\boldsymbol{y}(T)$, while the time-dependent control variables $\boldsymbol{\theta}(t)$ define the behavior of the system. The classification problem is

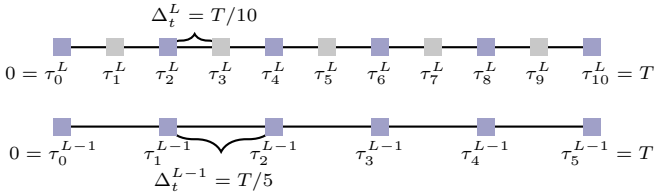

FIGURE 1. An example of time grids used for the multilevel discretization. On the fine level, we consider 10 time-steps, while on the coarse level, we use 5 larger time-steps.

now formulated as the following continuous optimal control problem [18]:

$$\begin{aligned} \min_{\boldsymbol{\theta},\boldsymbol{\theta}_K,\boldsymbol{y}} \frac{1}{p}\sum_{j=1}^{p} \ell(\mathcal{P}(\mathbf{W}_K \boldsymbol{y}_j(T) + \boldsymbol{b}_K), \boldsymbol{c}_j) + \int_0^T \mathcal{R}(\boldsymbol{\theta}(t), \boldsymbol{\theta}_K) \\ \text{subject to} \quad \partial_t \boldsymbol{y}_j(t) = \mathcal{F}(\boldsymbol{y}_j(t), \boldsymbol{\theta}(t)), \\ \boldsymbol{y}_j(0) = \mathbf{Q}\boldsymbol{x}_j, \end{aligned} \tag{7}$$

where $\boldsymbol{y}_j(T)$ denotes the output of the network for the data sample $\boldsymbol{x}_j$. The continuous formulation (7) opens the door to many new developments. For example, the design of stable network architectures [2, 18], the parallel approach to training [17, 36], or novel solution strategies [31]. In this work, we leverage the continuous formulation in order to design an efficient multilevel training strategy, see Section 3.

2.2.1. *Discretization*

To solve the continuous optimal control problem (7) numerically, we discretize (7) in time. Thus, we consider the time-grid $0 = \tau_0 < .... < \tau_K = T$ of $K+1$ uniformly distributed time points $\tau_k := \Delta_t k$, where $\Delta_t := T/K$ represents a time-step. The discretized control $\boldsymbol{\theta}_k \approx \boldsymbol{\theta}(\tau_k)$ and state $\boldsymbol{y}_{j,k} \approx \boldsymbol{y}_j(\tau_k)$ variables then correspond to the parameters and the state of the k-th layer of the ResNet, respectively.

In the discrete setting, we obtain the following constrained minimization problem:

$$\begin{aligned} \min_{\boldsymbol{\theta},\boldsymbol{\theta}_K,\boldsymbol{y}} \underbrace{\frac{1}{p}\sum_{j=1}^{p} \ell(\mathcal{P}(\mathbf{W}_K \boldsymbol{y}_{j,K} + \boldsymbol{b}_K), \boldsymbol{c}_j) + \sum_{k=0}^{K-1} \mathcal{R}(\boldsymbol{\theta}_k, \boldsymbol{\theta}_K)}_{=:\ \mathcal{L}(\boldsymbol{\theta},\boldsymbol{\theta}_K,\boldsymbol{y}_K)\ :=\mathcal{L}(\boldsymbol{\Theta},\boldsymbol{y}_K)} \\ \text{subject to} \quad \boldsymbol{y}_{j,k+1} = \boldsymbol{y}_{j,k} + \Delta_t \mathcal{F}(\boldsymbol{y}_{j,k}, \boldsymbol{\theta}_k), \\ \mathbf{y}_{j,0} = \mathbf{Q}\boldsymbol{x}_j, \end{aligned} \tag{8}$$

where we have used an explicit Euler scheme to discretize the time derivative $\partial_t \boldsymbol{y}(t)$ in (7). This choice of discretization is what imposes the particular ResNet architecture. However, other, possibly more stable discretization schemes, can be considered, see for instance [18]. Employing an explicit Euler method, we can ensure the stability of a forward propagation by ensuring that the time-step Δ_t is sufficiently small [18].

2.2.2. *Multilevel discretization*

We can discretize (7) using different discretization parameters. This allows us to construct a multilevel-hierarchy of auxiliary networks with different resolutions. We consider a hierarchy of L levels, denoted by $l = 1, \ldots, L$. The finest level, $l = L$, represents the discretization of the optimal control problem (7) with satisfactory resolution/representation capacity.

This means, that the time-step Δ_t^L is sufficiently small and that the network has sufficiently many layers to ensure desirable approximation properties of the model. In order to obtain coarser level networks, we discretize

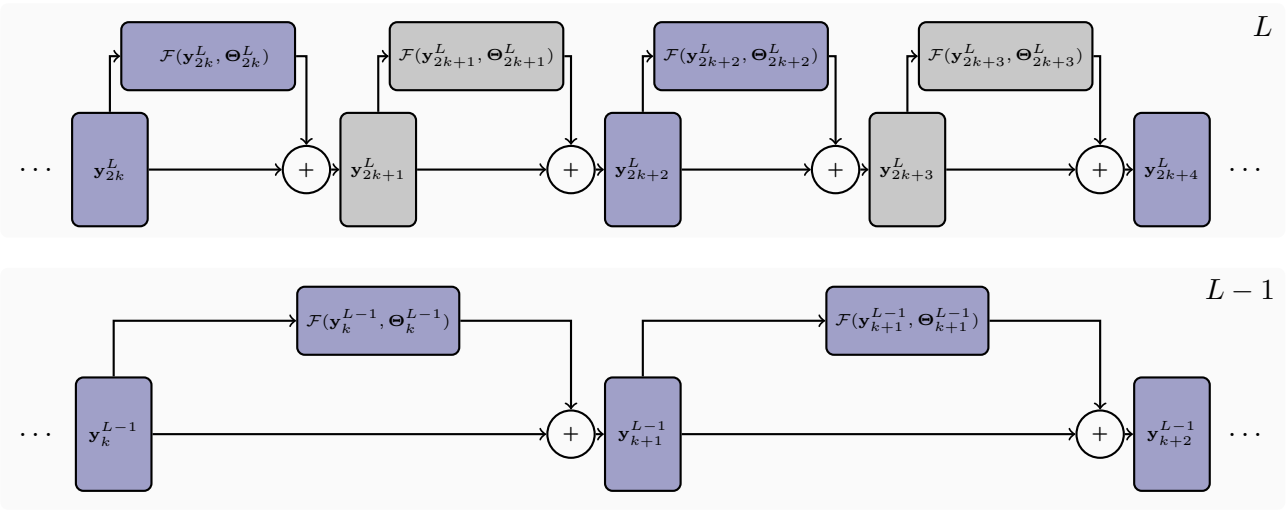

FIGURE 2. An example of a multilevel hierarchy of ResNets. The state and control variables are discretized using different time grids.

the time interval $[0, T]$ with larger time-steps. For instance, if we assume a uniform coarsening in time by a factor of two, the following relation holds for time-steps of subsequent levels $\Delta_t^{l-1} = 2\Delta_t^l$. As a consequence, the number of layers is halved between the networks on level l and $l-1$. Figure 1 demonstrates the process for a simple 2-level example. Since the networks on coarser levels of the multilevel hierarchy are constructed with fewer layers, they have less trainable parameters. Therefore, they are computationally cheaper to optimize, due to the fact that the cost of forward-backward propagation used during the training grows linearly with respect to the number of parameters [24]. As a consequence, it is roughly two–times faster to perform one forward–backward propagation on a coarser level than on the subsequent finer level.

3. MULTILEVEL TRAINING FOR RESNETS

In this section, we introduce a nonlinear multilevel minimization framework for training ResNets. The presented framework can be seen as a variant of the MG/OPT framework [34] originally developed for solving the large scale problems arising from the discretization of partial differential equations. Our variant of MG/OPT is tailored to the minimization of the discrete optimal control problem (8). In particular, we employ a hierarchy of auxiliary networks with different depths, see Figure 2, which are used to accelerate the training of the original network. Each auxiliary network is trained by approximately minimizing the associated level-dependent optimal control problem, see Section 3.2.1 for the details. The minimization of the level-dependent optimal control problem is carried out using an optimizer associated with a given level.

Through the following, we use a pair (l, μ^l) of superscripts to denote the quantities related to a given level l and iteration μ^l. If no subscript is used, we refer to quantities on all layers of the network simultaneously. Otherwise, the subscript identifies the quantities associated with a given layer. For example, $\boldsymbol{\Theta}_k^{1,\mu^1}$ denotes the parameters related to the k-th layer of the coarsest network, $l = 1$, after μ^1 update steps.

3.1. **Transfer operators**

The multilevel training framework requires to transfer data between subsequent levels of the multilevel hierarchy. For this reason, we employ two types of transfer operators. The interpolation operator $\mathbf{I}^l : \mathbb{R}^{n^l} \to \mathbb{R}^{n^{l+1}}$ transfers weights and biases from level l to level $l+1$. Here, the symbol n^l denotes the number of parameters of the model/network associated with the level l. In this work, we consider piecewise constant

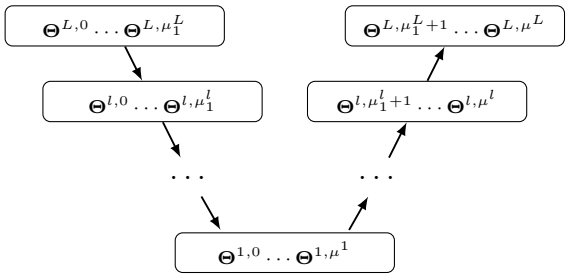

FIGURE 3. A scheme of the V-cycle used during multilevel training.

interpolation in time. Other choices of the transfer operators, such as linear interpolation, are also possible and may be even preferable. We plan to incorporate them into our multilevel training framework in future work. In addition to the interpolation operator, the multilevel method also uses a restriction operator $\mathbf{R}^l : \mathbb{R}^{n^{l+1}} \to \mathbb{R}^{n^l}$, in order to transmit data, such as gradients, from level $l+1$ to level l. As common in multgrid literature [20], we choose the restriction operator as $\mathbf{R}^l := (\mathbf{I}^l)^T$.

3.2. **Multilevel training**

The MG/OPT iteration has the form of a V–cycle, which consists of a downward and an upward phase, see Figure 3. The downward phase starts on the finest level, $l = L$, with initial weights $\boldsymbol{\Theta}^{L,0}$ and passes through all levels until the coarsest level is reached. On each level, we perform μ_1^l level-optimizer steps in order to find an approximate solution of the level-dependent optimal control problem. The approximate solution, i.e. the updated network parameters $\boldsymbol{\Theta}^{l,\mu_1^l}$ are then used to initialize weights on the subsequent coarser level, e.g. $\boldsymbol{\Theta}^{l-1,0} = \mathbf{R}^{l-1}\boldsymbol{\Theta}^{l,\mu_1^l}$. This process is repeated until we reach the coarsest level, $l = 1$.

Once the coarsest level is reached and we have performed μ^1 level-1-optimizer step, yielding the parameters $\boldsymbol{\Theta}^{1,\mu^1}$, we can initiate the upward phase. During the upward phase, we return to the finest level, while passing through all levels of the multilevel hierarchy. Starting on the coarsest level, we compute the coarse grid correction $\boldsymbol{e}^l = \boldsymbol{\Theta}^{l,\mu^l} - \boldsymbol{\Theta}^{l,0}$, which characterizes the difference between the initial and the updated parameters on a given level. This correction is then transferred to the next finer level using the interpolation operator as $\boldsymbol{e}^{l+1} = \mathbf{I}^l\boldsymbol{e}^l$. Once we have the interpolated correction, we use it to update the parameters of the finer network, thus $\boldsymbol{\Theta}^{l+1,\mu_1^{l+1}+1} = \boldsymbol{\Theta}^{l+1,\mu_1^l} + \boldsymbol{e}^{l+1}$. Finally, we perform μ_2^l steps of the level-optimizer in order to improve the current approximation of the parameters on level l. The whole process is summarized in Algorithm 1.

3.2.1. *Level-dependent minimization problems*

On each level of the multilevel hierarchy, we look for an approximate solution of some level-dependent optimal control problem. As common for nonlinear multilevel (minimization) schemes, such as FAS [3], or RMTR [14], we define the level-dependent optimal control problems as

$$\begin{aligned} \min_{\boldsymbol{\Theta}^l, \boldsymbol{y}^l} \quad & \mathcal{H}^l(\boldsymbol{\Theta}^l, \boldsymbol{y}_K^l) := \mathcal{L}^l(\boldsymbol{\Theta}^l, \boldsymbol{y}_K^l) + \langle \delta \mathbf{g}^l, \boldsymbol{\Theta}^l \rangle \\ \text{subject to} \quad & \boldsymbol{y}_{k+1}^l = \boldsymbol{y}_k^l + \Delta_t^l \mathcal{F}(\boldsymbol{y}_k^l, \boldsymbol{\Theta}_k^l), \\ & \boldsymbol{y}_0^l = \mathbf{Q}\boldsymbol{x}, \end{aligned} \tag{9}$$

where $\delta \boldsymbol{g}^l$ is given by

$$\delta \boldsymbol{g}^l := \mathbf{R}^l \nabla \mathcal{H}^{l+1}(\boldsymbol{\Theta}^{l+1,\mu_1^l}, \boldsymbol{y}_K^{l+1,\mu_1^l}) - \nabla \mathcal{L}^l(\boldsymbol{\Theta}^{l,0}, \boldsymbol{y}_K^{l,0}),$$

for all levels $l < L$. For the finest level, $l = L$, we assume that $\delta \boldsymbol{g}^L := \mathbf{0}$, and therefore the functional $\mathcal{H}^l : \mathbb{R}^{n^l} \to \mathbb{R}$ coincides with the loss functional $\mathcal{L}^L$ defined in (8).

On the coarser levels, the functional $\mathcal{H}^l$ consists of two terms: the loss functional $\mathcal{L}^l$ and the so-called coupling term $\langle \delta \boldsymbol{g}^l, \boldsymbol{\Theta}^l \rangle$. The coupling term creates a connection between two subsequent levels of the multilevel hierarchy. This is accomplished using the $\delta \boldsymbol{g}^l$ term, which measures the deviation between the restricted fine-level gradient $\nabla \mathcal{H}^{l+1}(\boldsymbol{\Theta}^{l+1,\mu_1^l}, \boldsymbol{y}_K^{l+1,\mu_1^l})$, and the initial coarse-level gradient $\nabla \mathcal{L}^l(\boldsymbol{\Theta}^{l,0}, \boldsymbol{y}_K^{l,0})$. The use of this coupling term is of major importance, as it enforces the following relationship:

$$\nabla \mathcal{H}^l(\boldsymbol{\Theta}^{l,0}, \boldsymbol{y}_K^{l,0}) = \mathbf{R}^l \nabla \mathcal{H}^{l+1}(\boldsymbol{\Theta}^{l+1,\mu_1^l}, \boldsymbol{y}_K^{l+1,\mu_1^l})$$

for the first optimizer step on a given level. In addition, it guarantees, that the minimization on the coarse level is guided by the restricted fine level gradient and that the prolongated coarse level correction will be a descent direction on the fine level [34].

3.2.2. *Multilevel gradient-based methods*

Algorithm 1 employs an auxiliary optimizer on every level of the multilevel hierarchy. By design, we are conveniently independent in the choice of the optimizer on each level. Our multilevel framework does not even require to employ the same type of optimizer on all levels. One can, for example, utilize computationally expensive optimizers on the coarser levels, while employing computationally cheaper optimizers on the finer levels. In the multilevel community, it is quite popular to employ a second-order optimizer on the coarsest level and gradient-based optimizers on all finer levels.

The easiest way to construct a multilevel training algorithm is to employ a gradient method on all levels. One level-optimizer iteration then consists of a simple gradient step computed using the whole dataset, see Algorithm 3. Although, the choice of other gradient-based algorithms, such as LBFGS [29], or RMSprop [39], might be more beneficial, using a vanilla gradient descent method allows for plain testing of our multilevel framework without introducing additional hyper-parameters.

Algorithm 1 V-cycle of MG/OPT($\mathcal{L}^l, l, \boldsymbol{\Theta}^{l,0}, \delta \mathbf{g}^l$)

Constants: $\mu_1^l, \mu_2^l, \mu^1 \in \mathbb{N}$
1. Downward phase
Construct $\mathcal{H}^l$ by means of (9)
$[\boldsymbol{\Theta}^{l,\mu_1^l}]$ = LevelOptimizer($\mathcal{H}^l$, $\boldsymbol{\Theta}^{l,0}$, μ_1^l)
$\boldsymbol{\Theta}^{l-1,0} \leftarrow \mathbf{R}^{l-1} \boldsymbol{\Theta}^{l,\mu_1^l}$
Evaluate $\delta \mathbf{g}^{l-1}$
2. Recursion or call to optimizer on the coarsest level
if $l = 2$ **then**
$[\boldsymbol{\Theta}^{l-1,\mu^l}]$ = LevelOptimizer($\mathcal{H}^1$, $\boldsymbol{\Theta}^{l-1,0}$, μ^1)
else
$[\boldsymbol{\Theta}^{l-1,\mu^l}]$ = MG/OPT($l-1, \boldsymbol{\Theta}^{l-1,0}, \delta \mathbf{g}^{l-1}$)
end if
3. Upward phase
$\boldsymbol{e}^{l-1} \leftarrow \boldsymbol{\Theta}^{l-1,\mu^l} - \boldsymbol{\Theta}^{l-1,0}$
$\boldsymbol{e}^l \leftarrow \mathbf{I}^{l-1} \boldsymbol{e}^{l-1}$
$\boldsymbol{\Theta}^{l,\mu_1^l+1} \leftarrow \boldsymbol{\Theta}^{l,\mu_1^l} + \boldsymbol{e}^l$
$[\boldsymbol{\Theta}^{l,\mu^l}]$ = LevelOptimizer($\mathcal{H}^l$, $\boldsymbol{\Theta}^{l,\mu^l+1}$, μ_2^l)
return $\boldsymbol{\Theta}^{l,\mu^l}$

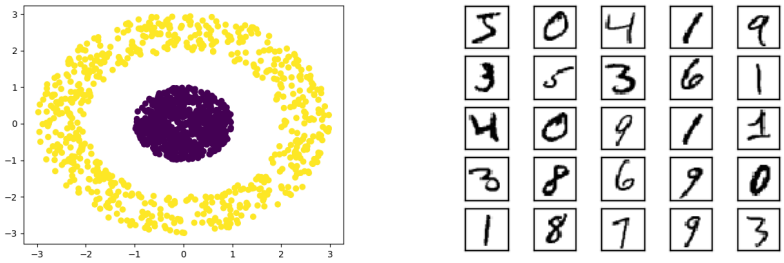

FIGURE 4. Left: Co-centric dataset consisting of 2 classes (illustrated by different colors.) Right: Example of samples contained in MNIST dataset.

3.2.3. *Multilevel mini-batch gradient descent*

Since mini-batch gradient descent (SGD) is typically the algorithm of choice when training a neural network, here we propose its multilevel variant, Algorithm 2. Similarly to the single level SGD algorithm, we split the dataset into nb mini-batches. The algorithm then iterates through all mini-batches. For each mini-batch, the algorithm invokes a multilevel gradient descent step, thus a V-cycle of MG/OPT configured with a gradient descent optimizer on all levels.

Algorithm 2 One epoch of multilevel SGD

Constants: $nb, L \in \mathbb{N}$
for $b = 1, \ldots, nb$ **do**
  Construct $\mathcal{L}^L$ using mini-batch $\mathcal{D}_b$
  $[\boldsymbol{\Theta}^{L,b}] = \text{MG/OPT}(\mathcal{L}^L, L, \boldsymbol{\Theta}^{L,b-1}, \mathbf{0})$
end for
return $\boldsymbol{\Theta}^{L,nb}$

Algorithm 3 LevelOptimizer($\mathcal{H}^l$, $\boldsymbol{\Theta}^{l,0}$, max_it)

Constants: $\alpha \in \mathbb{R}^+$
for $i = 1, \ldots, \text{max\_it}$ **do**
  $\boldsymbol{\Theta}^{l,i} = \boldsymbol{\Theta}^{l,i-1} - \alpha \nabla \mathcal{H}^l(\boldsymbol{\Theta}^{l,i-1}, \boldsymbol{y}_K^{l,i-1})$
end for
return $\boldsymbol{\Theta}^{l,\text{max\_it}}$

3.2.4. *Computational complexity*

One V-cycle of the multilevel training strategy is computationally more expensive than one iteration of a single level optimizer. For the gradient-based optimizers, the computational cost is associated with the evaluation of the gradient, thus with the cost of a forward-backward pass. To provide a fair comparison between multilevel and single level methods, we introduce the notation of work units. One work unit $\mathcal{U}^L$ represents the cost of a gradient evaluation on the finest level. Assuming a coarsening factor of two, the cost related to the gradient evaluation on the coarser levels is $\mathcal{U}^l = 2^{l-L}\mathcal{U}^L$. The computational cost of one V-cycle, denoted $\mathcal{U}_c$, can be obtained by summing over the cost required on each level, thus

$$\mathcal{U}_c \approx \big(2^{1-L}\mu^0 + \sum_{l=2}^{L}(\mu_1^l + \mu_2^l + 1)2^{l-L}\big)\mathcal{U}^L. \tag{10}$$

The cost on the coarsest level is related to μ^0 level-optimizer steps. On all other levels, we have to take into account the gradient evaluation that is required for computing the coupling term $\delta \boldsymbol{g}^l$, in addition to μ_1^l and μ_2^l level-optimizer steps. The overall computational cost of the multilevel training $\mathcal{U}$ is then simply computed as $\mathcal{U} = (\#\text{V-cycles})\ \mathcal{U}_c$, where (#V-cycles) denotes the number of V-cycles required to achieve a prescribed tolerance.

4. NUMERICAL EXPERIMENTS

We analyze the performance of the proposed multilevel optimizers using two classification problems:

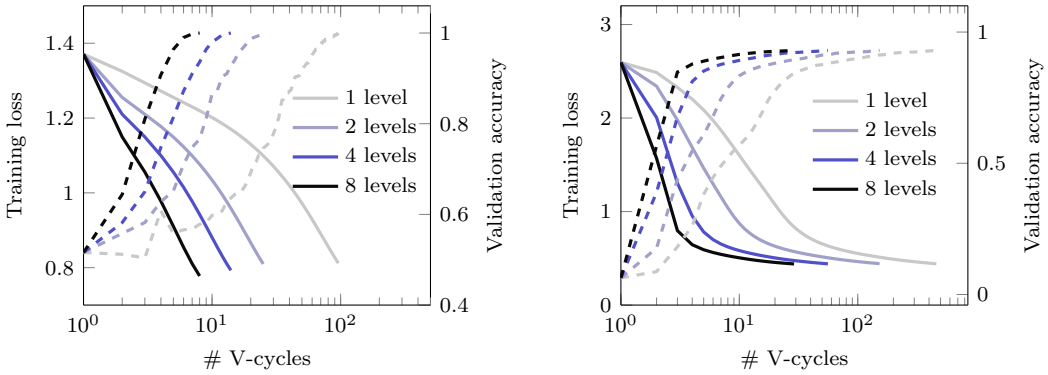

FIGURE 5. The loss function (solid lines) and validation accuracy (dashed lines) as a function of V-cycles. Results obtained for varying numbers of levels. Left: Co-centric circles. Right: MNIST.

- **Co-centric circles**: This simple example was proposed in [32] and requires the classification of particles into two distinct classes. The input features $\boldsymbol{x}_j \in [-3,3]^2$ describe the position of a particle in a two-dimensional plane, while the output vector $\boldsymbol{c}_j \in \mathbb{R}^2$ prescribes an affiliation to a given class. In particular, the elements of label $\boldsymbol{c}_j$ are defined as follows:

$$(\boldsymbol{c}_j)_1 = \begin{cases} 1 & \text{if} \quad \|\boldsymbol{x}_j\| \leq 1, \\ 0 & \text{otherwise.} \end{cases} \qquad (\boldsymbol{c}_j)_2 = \begin{cases} 1 & \text{if} \quad 2 \leq \|\boldsymbol{x}_j\| < 3, \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

  The dataset consists of $3,000$ samples, where $2,000$ are used for training and $1,000$ for validation.
- **MNIST**: Our second classification task considers the database of handwritten digits [30]. The dataset contains greyscale images of size 28×28 pixels that are uniformly divided into ten classes. As a preprocessing, we standardize the images, so pixel values lie in the range $[0,1]$, and perform centering by subtracting the mean from each pixel. The data is split into $60,000$ samples for training and $10,000$ samples for validation.

Figure 4 illustrates both datasets.

4.1. **Implementation and testing environment**

Our implementation of deep residual networks and nonlinear multilevel training framework uses the Keras [9] and Tensorflow [1] library.

The classification is performed using a ResNet architecture as described in (3). We employ a simple variant of residual blocks, i.e. a one layer perceptron with ReLU activation function, see (4). Each layer consists of 3 nodes i.e., $v = 3$ for the co-centric circles example and 10 nodes, i.e., $v = 10$ for the MNIST example. The operator $\mathbf{Q}$, which maps input features into network width, is learned during training. All layers are fully-connected, thus the operator $\boldsymbol{W}_k$ is a dense matrix, for all layers $k = 0, \ldots, K-1$.

Unless specified differently, the deep residual network consists of $2,048$ residual blocks. In the case of multilevel training, the multilevel-hierarchy of auxiliary networks with different resolutions is created by coarsening in time with a factor of two, see Section 2.2.1 for more details. On each level of the multilevel hierarchy, we consider the final time $T = 1$. As commonly used, we employ Tikhonov regularization, thus $\mathcal{R}(\cdot) := \beta\|\cdot\|_F^2$, where $\|\cdot\|_F^2$ denotes the Frobenius norm. On the finest level (deepest network), we prescribe the regularization parameters $\beta = 10^{-4}$ and $\beta = 10^{-5}$ for co-centric circles and MNIST, respectively. On the coarser levels, the value of the regularization parameter β is scaled by a coarsening factor 2^{L-l}, where l denotes a given level.

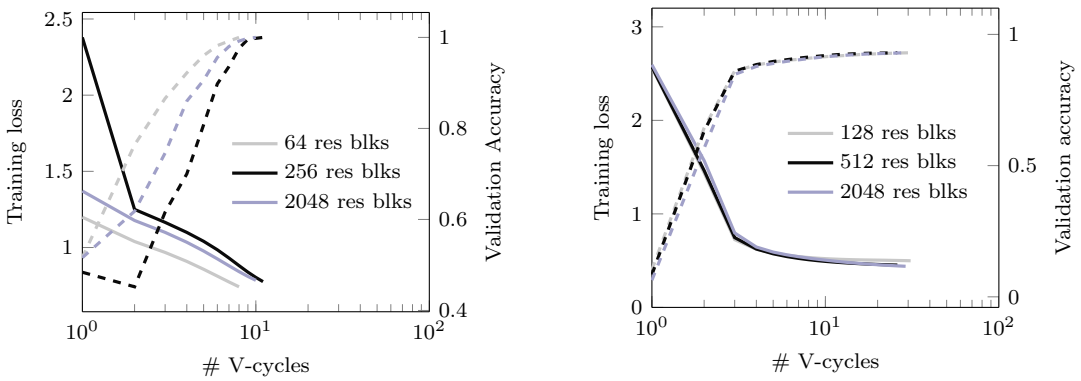

FIGURE 6. Loss function (solid lines) and validation accuracy (dashed lines) over number of V-cycles. Results obtained for varying numbers of residual blocks (res blks). Left: 6-level method trained on co-centric circles. Right: 8-level method trained on MNIST.

All presented experiments were performed on our local cluster consisting of 42 compute nodes, each equipped with 2 Intel R E5-2650 v3 processor with a clock frequency of 2.60 GHz. The memory per node is 64 GB.

4.2. **Algorithmic setup**

We trained both test examples using gradient-based multilevel optimizers with a constant learning rate of 0.1 in the co-centric circle example and 0.01 in case of the MNIST dataset. During all numerical tests, the weights are initialized randomly, while biases are set to zero. The performance of the training methods is assessed using validation accuracy, defined as

$$\text{validation accuracy} = \frac{\text{number of correctly classified samples from the validation dataset}}{\text{total number of samples from the validation dataset}}. \tag{12}$$

The co-centric circles example is trained using the full dataset, giving rise to a multilevel gradient descent. We terminate training if a validation accuracy, of 1 is achieved. The training of the MNIST example is performed using a multilevel mini-batch gradient descent with a mini-batch size of $1,000$. As a termination criterion, we require the validation accuracy to be higher than 0.93.

During multilevel training, level-optimizers perform several steps while completing the downward and upward phase of the V-cycle. To describe our particular level-optimizer setup, we introduce a list notation, such as $[(1), 2, 1, 3, \{4\}]$ for a 5-level training strategy. Each entry of the list indicates a number of optimizer steps used on a given level. The list is ordered from the finest to the coarsest level. If no bracket is used, we assume $\mu_1^l = \mu_2^l$. The use of a parentheses implies that the level-optimizer was not called during the upward phase, thus $\mu_2^l = 0$. The curly bracket indicates the number of optimizer steps on the coarsest grid, i.e. μ^1.

4.3. **Numerical results**

We analyze the convergence behavior of the proposed multilevel training methods with respect to varying numbers of levels. During the experiment, the depth of the finest level network is fixed. The presented results also include the single-level version of the algorithms, thus vanilla gradient descent and mini-batch gradient descent. We employ one iteration of level-optimizer for all levels $l \in \{L/2 + 1, \ldots, L\}$. On the coarser levels, that is $l \in \{1, \ldots, L/2\}$, we perform 2 level-optimizer iterations. The described setup is used for both the downward and the upward phase of a V-cycle, except on the finest level, where we skip the call to the optimizer during the upward phase. Thus, for 6-level training, we use following setup $[(1), 1, 1, 2, 2, \{2\}]$.

Figure 5 demonstrates the obtained results for both test problems. As we can see, adding more levels reduces the number of required V-cycles significantly. In particular, using the 2-level method already leads to a decrease

TABLE 1. Computational cost of multilevel training with respect to varying numbers of levels. The symbol $\mathcal{U}_c$ denotes the cost per V-cycle and $\mathcal{U}$ stands for the total cost.

Co-centric circle

L	Optimizers setup	# V-cycles	$\mathcal{U}_c$	$\mathcal{U}$
1	[{1}]	95	1.00	95
2	[(1),{2}]	32	3.00	96
4	[(1),1,2,{2}]	12	5.00	60
6	[(1),1,1,2,2,{2}]	7	5.25	37
8	[(1),1,1,1,2,2,2,{2}]	5	5.19	26

MNIST

L	Optimizers setup	# V-cycles	$\mathcal{U}_c$	$\mathcal{U}$
1	[{1}]	460	1.00	460
2	[(1),{2}]	152	3.00	456
4	[(1),1,2,{2}]	55	5.00	275
6	[(1),1,1,2,2,{2}	33	5.25	173
8	[(1),1,1,1,2,2,2,{2}]	28	5.19	145

in the number of iterations by a factor of 3, compared to the single level method. For the 4-level method, we obtain a reduction in the number of V-cycles by a factor of 8, while for the 8-level method, the number of V-cycles is approximately reduced by a factor of 15. Obtained results also demonstrate that the proposed multilevel method achieves the same validation accuracy, independently on the number of levels. Thus, the multilevel method allows for an enhancement of the training speed, while retaining the same validation accuracy as achieved by the standard/single level optimizer.

We compare the total computational cost of the multilevel training methods. Following the analysis presented in Section 3.2.2, we show the computational cost in Table 1 for both datasets. For more than four levels, the computational cost does not increase substantially anymore, as the cost of numerical operations on those levels is negligible compared to the cost of the same operations performed on the finest level. The results also demonstrate, that the total computational cost $\mathcal{U}$ decreases as the number of levels increases. This is not surprising, as the number of V-cycles required for convergence decreased. In particular, the 8-level method is approximately 3.4 times computationally more efficient than its single level counterpart.

Results reported in Table 1 also indicate that the number of required V-cycles is roughly proportional to the sum of corrections taken over all levels. The observed behavior is related to the fact, that our multilevel method employs the first-order optimizer on all levels of multilevel hierarchy, including the coarsest one. The complexity bound of the first-order multilevel method is of the same order as that of the single level first-order method. Thus, if a number of corrections taken across all levels stays constant, we can expect savings in terms of the computational cost. In other words, the cheap iterations provided by the coarse levels can decrease the number of required iterations on the more expensive/finer levels. This phenomenon was theoretically analyzed for example in [14], in the context of the multilevel trust-region methods.

The fact that the presented method is not level independent is also related to the following observation: The multilevel methods typically accelerate the convergence by removing the components of the error associated with different levels of the multilevel hierarchy [4]. More precisely, a few steps of the basic iterative method (smoother) remove high-frequency components of the error associated with a given level, while the coarsest level solver (traditionally a direct solver) eliminates the remaining low-frequency components of the error completely. Our multilevel method employs the first-order optimizer on all levels of the multilevel hierarchy. In this particular case, introducing more levels helps to reduce the number of required V-cycles, as it allows to capture the larger part of the spectrum. Note, this behavior can be also observed in the context of the multigrid methods for elliptic problems, if an inexact iterative method is employed on the coarsest level.

4.4. **Convergence behavior with respect to number of residual blocks**

Further, we analyze the convergence behavior of the proposed multilevel training strategy for varying numbers of residual blocks. We keep the same parameters as described in the beginning of Section 4, but alter the number of residual blocks. Figure 6 illustrates the obtained results for both datasets. As we can see, the method exhibits the same asymptotic convergence behavior independently on the number of residual blocks. These results are very promising, as they suggest that the convergence rate of our multilevel training strategy does not deteriorate with network depth.

TABLE 2. Computational cost as a function of optimizer steps for co-centric circles example with the 8-level method. The symbol $\mathcal{U}_c$ denotes the cost per V-cycle and $\mathcal{U}$ stands for the total cost.

Optimizers setup	# V-cycles	$\mathcal{U}_c$	$\mathcal{U}$
$[1,1,1,1,1,1,1,\{1\}]$	7	5.97	42
$[1,1,1,1,1,1,1,\{2\}]$	7	5.96	42
$[1,1,1,1,1,1,1,\{5\}]$	6	5.99	36
$[1,1,1,1,1,1,1,\{10\}]$	5	6.03	30
$[1,1,1,1,2,2,2,\{2\}]$	5	6.19	31
$[(1),1,1,1,2,2,2,\{2\}]$	5	5.19	26

4.5. **Influence of hyper-parameters**

In the end, we investigate the sensitivity of multilevel methods with respect to the choice of hyper-parameters. Firstly, we demonstrate how different setups of level-optimizers influence the convergence properties of the multilevel training strategy. Table 2 reports the results obtained for different setups of the 8-level method trained on the co-centric circle example. As contemplated, increasing the number of optimizer calls on the lower levels decreases the total computational cost of the multilevel method. This is due to the fact, that additional calls to coarse level optimizers lower the number of required V-cycles. But at the same time, those additional calls do not considerably increase the cost of one V-cycle. The most expensive part of the V-cycle is the optimizer call on the finest level. Therefore, it is beneficial to skip it during the upward phase. This has no substantial impact on the performance of the method, as the upward optimizer step is immediately followed by the downward optimizer step of the next V-cycle.

Secondly, we study the sensitivity with respect to the choice of learning rate and regularization parameter. Here, we consider the co-centric circle example and three different values of learning rate, $\alpha = \{0.05; 0.1; 0.5\}$, and regularization parameter $\beta = \{10^{-3}; 10^{-4}; 10^{-5}\}$. This yields nine different hyper-parameter setups, which we tested using single level and 8-level ([(1),1,1,1,2,2,2,{2}]) methods. On average, the computational cost of the 8-level method is 3.54 lower than the cost required by a single level method. The relative standard deviation of our results is 8.02%.

5. CONCLUSION

In this work, we proposed a nonlinear multilevel minimization framework for training the deep residual networks. Our multilevel framework is based on MG/OPT framework [34] and utilizes a hierarchy of auxiliary networks with different depths to speed up the training process of the original network. Using our novel training framework, we proposed multilevel gradient and mini-batch gradient methods. The performed numerical experiments demonstrated the convergence behavior of multilevel training methods and showed significant decrease in the computational cost compared to its single level variants.

REFERENCES

[1] M. ABADI, A. AGARWAL, P. BARHAM, E. BREVDO, Z. CHEN, C. CITRO, G. S. CORRADO, A. DAVIS, J. DEAN, M. DEVIN, S. GHEMAWAT, I. GOODFELLOW, A. HARP, G. IRVING, M. ISARD, Y. JIA, R. JOZEFOWICZ, L. KAISER, M. KUDLUR, J. LEVENBERG, D. MANÉ, R. MONGA, S. MOORE, D. MURRAY, C. OLAH, M. SCHUSTER, J. SHLENS, B. STEINER, I. SUTSKEVER, K. TALWAR, P. TUCKER, V. VANHOUCKE, V. VASUDEVAN, F. VIÉGAS, O. VINYALS, P. WARDEN, M. WATTENBERG, M. WICKE, Y. YU, AND X. ZHENG, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[2] M. BENNING, E. CELLEDONI, M. J. EHRHARDT, B. OWREN, AND C.-B. SCHÖNLIEB, *Deep learning as optimal control problems: models and numerical methods*, arXiv preprint arXiv:1904.05657, (2019).

[3] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Mathematics of computation, 31 (1977), pp. 333–390.

[4] W. L. BRIGGS, V. E. HENSON, AND S. F. MCCORMICK, *A multigrid tutorial*, SIAM, second ed., 2000.

[5] H. Calandra, S. Gratton, E. Riccietti, and X. Vasseur, *On high-order multilevel optimization strategies*, arXiv preprint arXiv:1904.04692, (2019).

[6] ———, *On a multilevel Levenberg–Marquardt method for the training of artificial neural networks and its application to the solution of partial differential equations*, Optimization Methods and Software, (2020), pp. 1–26.

[7] B. Chang, L. Meng, E. Haber, F. Tung, and D. Begert, *Multi-level residual networks from dynamical systems view*, arXiv preprint arXiv:1710.10348, (2017).

[8] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs*, IEEE transactions on pattern analysis and machine intelligence, 40 (2017), pp. 834–848.

[9] F. Chollet et al., *Keras*. https://keras.io, 2015.

[10] E. C. Cyr, S. Günther, and J. B. Schroder, *Multilevel initialization for layer-parallel deep neural network training*, arXiv preprint arXiv:1912.08974, (2019).

[11] H. W. Engl, M. Hanke, and A. Neubauer, *Regularization of inverse problems*, vol. 375, Springer Science & Business Media, 1996.

[12] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, *Spatially adaptive computation time for residual networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1039–1048.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT press, 2016.

[14] S. Gratton, A. Sartenaer, and P. L. Toint, *Recursive Trust-Region Methods for Multiscale Nonlinear Optimization*, SIAM Journal on Optimization, 19 (2008), pp. 414–444.

[15] C. Gross and R. Krause, *On the Convergence of Recursive Trust-Region Methods for Multiscale Nonlinear Optimization and Applications to Nonlinear Mechanics*, SIAM Journal on Numerical Analysis, 47 (2009), pp. 3044–3069.

[16] C. Gulcehre, M. Moczulski, F. Visin, and Y. Bengio, *Mollifying networks*, arXiv preprint arXiv:1608.04980, (2016).

[17] S. Günther, L. Ruthotto, J. B. Schroder, E. Cyr, and N. R. Gauger, *Layer-parallel training of deep residual neural networks*, SIAM Journal on Mathematics of Data Science, 2 (2020), pp. 1–23.

[18] E. Haber and L. Ruthotto, *Stable architectures for deep neural networks*, Inverse Problems, 34 (2017), p. 014004.

[19] E. Haber, L. Ruthotto, E. Holtham, and S.-H. Jun, *Learning across scales—multiscale methods for convolution neural networks*, in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[20] W. Hackbusch, *Multi-grid methods and applications*, vol. 4, Springer-Verlag Berlin Heidelberg, 1985.

[21] J. Håstad and M. Goldmann, *On the power of small-depth threshold circuits*, Computational Complexity, 1 (1991), pp. 113–129.

[22] 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, 2016, pp. 770–778.

[23] ———, *Identity mappings in deep residual networks*, in European conference on computer vision, Springer, 2016, pp. 630–645.

[24] R. Hecht-Nielsen, *Theory of the backpropagation neural network*, in Neural networks for perception, Elsevier, 1992, pp. 65–93.

[25] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, *Deep networks with stochastic depth*, in European conference on computer vision, Springer, 2016, pp. 646–661.

[26] H. Jung, M.-K. Choi, J. Jung, J.-H. Lee, S. Kwon, and W. Young Jung, *Resnet-based vehicle classification and localization in traffic surveillance systems*, in Proceedings of the IEEE conference on computer vision and pattern recognition workshops, 2017, pp. 61–67.

[27] A. Kopaničáková and R. Krause, *A recursive multilevel trust region method with application to fully monolithic phase-field models of brittle fracture*, Computer Methods in Applied Mechanics and Engineering, 360 (2020), p. 112720.

[28] A. Kopaničáková, R. Krause, and R. Tamstorf, *Subdivision-based nonlinear multiscale cloth simulation*, SIAM Journal on Scientific Computing, 41 (2019), pp. S433–S461.

[29] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, *On optimization methods for deep learning*, (2011).

[30] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al., *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[31] Q. Li, L. Chen, C. Tai, and E. Weinan, *Maximum principle based algorithms for deep learning*, The Journal of Machine Learning Research, 18 (2017), pp. 5998–6026.

[32] H. Lin and S. Jegelka, *Resnet with one-neuron hidden layers is a universal approximator*, in Advances in Neural Information Processing Systems, 2018, pp. 6169–6178.

[33] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, *Microsoft coco: Common objects in context*, in European conference on computer vision, Springer, 2014, pp. 740–755.

[34] S. G. Nash, *A multigrid approach to discretized optimization problems*, Optimization Methods and Software, 14 (2000), pp. 99–116.

[35] A. Y. Ng, *Feature selection, l 1 vs. l 2 regularization, and rotational invariance*, in Proceedings of the twenty-first international conference on Machine learning, 2004, p. 78.

[36] P. Parpas and C. Muir, *Predict globally, correct locally: Parallel-in-time optimal control of neural networks*, arXiv preprint arXiv:1902.02542, (2019).
[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV), 115 (2015), pp. 211–252.
[38] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, (2014).
[39] T. Tieleman and G. Hinton, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning, 4 (2012), pp. 26–31.
[40] E. Weinan, *A proposal on machine learning via dynamical systems*, Communications in Mathematics and Statistics, 5 (2017), pp. 1–11.
[41] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., *Google's neural machine translation system: Bridging the gap between human and machine translation*, arXiv preprint arXiv:1609.08144, (2016).
[42] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, *The microsoft 2017 conversational speech recognition system*, in 2018 IEEE international conference on acoustics, speech and signal processing (ICASSP), IEEE, 2018, pp. 5934–5938.