Numerical Algebra, Control and Optimization

doi:10.3934/naco.2022016

DEEP NEURAL NETS WITH FIXED BIAS CONFIGURATION

Harbir Antil* and Thomas S. Brown

Center for Mathematics and Artificial Intelligence (CMAI)
College of Science, George Mason University
Fairfax, VA 22030-4444, USA

Rainald Löhner

Center for Computational Fluid Dynamics
College of Science, George Mason University
Fairfax, VA 22030-4444, USA

Fumiya Togashi

Applied Simulations, Inc.
1211 Pine Hill Road, McLean, VA 22101, USA

Deepanshu Verma

Department of Mathematics, Emory University
Atlanta, GA 30322, USA

Abstract. For any given neural network architecture a permutation of weights and biases results in the same functional network. This implies that optimization algorithms used to 'train' or 'learn' the network are faced with a very large number (in the millions even for small networks) of equivalent optimal solutions in the parameter space. To the best of our knowledge, this observation is absent in the literature. In order to narrow down the parameter search space, a novel technique is introduced in order to fix the bias vector configurations to be monotonically increasing. This is achieved by augmenting a typical learning problem with inequality constraints on the bias vectors in each layer. A Moreau-Yosida regularization based algorithm is proposed to handle these inequality constraints and a theoretical convergence of this algorithm is established. Applications of the proposed approach to standard trigonometric functions and more challenging stiff ordinary differential equations arising in chemically reacting flows clearly illustrate the benefits of the proposed approach. Further application of the approach on the MNIST dataset within TensorFlow, illustrate that the presented approach can be incorporated in any of the existing machine learning libraries.

2020 *Mathematics Subject Classification.* 49K15, 68T99.

Key words and phrases. Deep neural network, bias ordering, chemical kinetics, convolutional neural network, Moreau-Yosida regularization.

This work is partially supported by the Defense Threat Reduction Agency (DTRA) under contract HDTRA1-15-1-0068 where Jacqueline Bell served as the technical monitor, and by NSF grants DMS-2110263, DMS-1913004, and the Air Force Office of Scientific Research under Award NO: FA9550-22-1-0248.

*Corresponding author: Harbir Antil.

1. Introduction.

Background. A typical neural network can be represented as a function $\mathcal{F} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ that consists of the composition of layer functions $\{f_\ell\}_{\ell=0}^{L-1}$ and can written as

$$\mathcal{F} = f_{L-1} \circ f_{L-2} \circ \cdots \circ f_0 \,. \tag{1}$$

Each layer function is parameterized by a weight matrix $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$, a bias vector $b_\ell \in \mathbb{R}^{n_{\ell+1}}$, and incorporates a nonlinear activation function σ, for instance, ReLU [6].

The weights W_ℓ and biases b_ℓ are determined in a process known as training the network. Training a neural network can be written in the framework of constrained optimization as follows: for training data $\{u^i, S(u^i)\}_{i=1}^N$ (input/output pairs), solve

$$\begin{aligned} &\min_{\{W_\ell\}_{\ell=0}^{L-1}, \{b_\ell\}_{\ell=0}^{L-2}} && J(\{(y_L^i, S(u^i))\}_i, \{W_\ell\}_\ell, \{b_\ell\}_\ell) \\ &\text{subject to} && y_L^i = \mathcal{F}(u^i; (\{W_\ell\}, \{b_\ell\})) \qquad i = 1, \ldots, N, \end{aligned} \tag{2}$$

where the function J, known as the loss function, measures the error of the approximation of $S(u^i)$ by the network output y_L^i in some way.

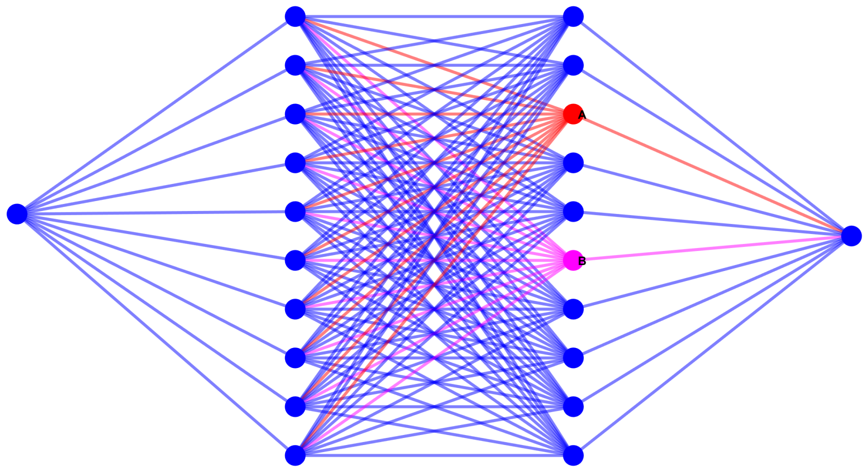

FIGURE 1. A neural network with 2 hidden layers of width 10. Switching the bias values and edges connected to node A (in red) with the those of node B (in magenta) results in a different ordering of the parameters, but the same neural network function.

It is easy to see that there are many permutations of the parameters that result in the same network. This is illustrated in Figure 1 where a network is represented graphically with the nodes of the graph representing the biases and the edges of the graph representing the weights. Switching the bias values A (red) and B (magenta) results in the same network as long as the corresponding weights (also in red and magenta) are also switched. This implies that it is possible to change the order of the parameters without changing the action of the function $\mathcal{F}$. In fact, for networks where each layer consists of n_ℓ neurons, there are $\sum_{\ell=1}^{L-2} n_\ell!$ ways to rearrange the parameters and obtain the same network. For the relatively small network displayed in Figure 1 there are 7,257,600 different ways to permute the parameters and still obtain the same network. In the context of training, this means that for this particular example there are 7,257,600 different solutions to the optimization problem (2) that will result in the same neural network function $\mathcal{F}$. This level of non-uniqueness is troubling and highly unsatisfactory. Apparently, this simple looking observation has not received any attention so far.

Problem formulation. The parameter search space may be narrowed down by fixing the bias vector configurations to be monotonically increasing. This is achieved by augmenting the learning problem (2) with inequality constraints on the bias vectors in each layer. The resulting optimization problem that describes the training of the network is

$$\min_{\{W_\ell\}_{\ell=0}^{L-1}, \{b_\ell\}_{\ell=0}^{L-2}} J(\{(y_L^i, S(u^i))\}_i, \{W_\ell\}_\ell, \{b_\ell\}_\ell) \tag{3a}$$

$$\text{subject to} \quad y_L^i = \mathcal{F}(u^i; (\{W_\ell\}, \{b_\ell\})), \qquad i = 1, \dots, N, \tag{3b}$$

$$b_\ell^j \leq b_\ell^{j+1}, \qquad j = 1, \dots, n_{\ell+1} - 1, \quad \ell = 0, \dots, L-2, \tag{3c}$$

where for a fixed ℓ, the quantities b_ℓ^j are the entries of the bias vector b_ℓ. The added inequality constraints fix the configuration of the network, so that any permutation of the parameters (excluding the case when two adjacent bias values are equal) either violates the constraints or results in a different neural network.

Outline of the paper. Section 2 introduces a Moreau-Yosida regularization based algorithm to handle inequality constraints given in (3c). This is followed by a convergence result of the Moreau-Yosida regularized problem to the original problem in Section 3. Section 4 describes the ResNets used for the examples shown in Section 5. In the first example the proposed algorithm is applied to a standard trigonometric function. This is followed by an application to a realistic application in chemically reacting flows, which are governed by stiff ordinary differential equations. The final example on the MNIST dataset within TensorFlow, illustrates that the presented approach can be incorporated in any of the existing machine learning libraries.

2. Regularized problem.

The inequality constraints in (3c) can in principle be handled by a typical nonlinear programming solver. However, we are interested in an approach that allows us to implement these constraints in existing machine learning softwares which are built to tackle (3a)-(3b). Motivated by the optimization problems with partial differential equations as constraints, see [10, 14, 9, 1], we propose to handle (3c) using a penalty based method. Similarly to the aforementioned references, we call this penalty method as Moreau-Yosida regularization approach. We emphasize that we are only considering Moreau-Yosida regularization of the constraint (3c).

Before introducing the loss function with bias order regularization, a more precise definition of the loss function is given. For training data $\{u^i, S(u^i)\}_{i=1}^N$, consider

$$J := \frac{1}{2N} \sum_{i=1}^{N} \|y_L^i - S(u^i)\|_2^2 + \frac{\lambda}{2} \sum_{\ell=0}^{L-1} \left(\|W_\ell\|_1 + \|b_\ell\|_1 + \|W_\ell\|_2^2 + \|b_\ell\|_2^2 \right), \tag{4}$$

where $\lambda > 0$ is a regularization parameter. In order to fit J into the framework introduced above, b_{L-1} is taken to be the zero vector in $\mathbb{R}^{n_L}$. Even though a mean squared error term is used above to measure the approximation error of the neural network, this is easily generalizable to other terms such as cross-entropy, likelihood, etc. [6].

Using the notation above to represent the entries of each bias vector, namely $b_\ell = (b_\ell^j)_{j=1}^{n_{\ell+1}} \in \mathbb{R}^{n_{\ell+1}}$, the new loss function is defined as

$$J_\gamma := J + \frac{\gamma}{2} \sum_{\ell=0}^{L-2} \sum_{j=1}^{n_{\ell+1}-1} \| \min\{b_\ell^{j+1} - b_\ell^j, 0\}\|_2^2, \tag{5}$$

where γ is the so-called penalization parameter. The regularized optimization problem can now be written as

$$\min_{\{W_\ell\}_{\ell=0}^{L-1}, \{b_\ell\}_{\ell=0}^{L-2}} J_\gamma(\{(y_L^i, S(u^i))\}_i, \{W_\ell\}_\ell, \{b_\ell\}_\ell) \tag{6a}$$

$$\text{subject to} \quad y_L^i = \mathcal{F}(u^i; (\{W_\ell\}, \{b_\ell\})) \qquad i = 1, \dots, N. \tag{6b}$$

Note that, even though it is not explicitly written in the formulation above, all of the variables W_ℓ, b_ℓ, and y_L^i depend on the parameter γ.

In Appendix A the first order optimality conditions for this problem are derived where the DNN used is a Deep Residual Neural Net (ResNet).

3. Convergence of J_γ.

In order to show that as $\gamma \to \infty$, the minimum value of J_γ converges to the minimum value of J and the constraints (3c) are also satisfied, let θ represent the concatenation of all of the parameters which are being minimized, i.e. θ contains all of the entries of $\{W_\ell\}$ and $\{b_\ell\}$. Furthermore, assume that a fixed set of training data is being used and so the loss function J given in (4) can be rewritten as

$$J(\theta) := \frac{1}{2N} \sum_{i=1}^{N} \|\mathcal{F}(u^i;\theta) - S(u^i)\|_2^2 + \frac{\lambda}{2} \sum_{\ell=0}^{L-1} \left(\|\theta\|_1 + \|\theta\|_2^2\right).$$

Introducing the notation

$$g(\theta) := \sum_{\ell=0}^{L-2} \sum_{j=1}^{n_{\ell+1}-1} \| \min(b_\ell^{j+1} - b_\ell^j, 0)\|_2^2 \,,$$

the regularized loss function in (5) may be rewritten as

$$J_\gamma(\theta^\gamma) := J(\theta^\gamma) + \frac{\gamma}{2} g(\theta^\gamma).$$

Now, the framework of [13, Section 10.11] can be used to show the following convergence results. The proof follows exactly as in [13] after a transformation of notation.

Proposition 1. *Let J_0 be the minimum value attained from solving* (3). *For each γ, let θ^γ be a minimizer of J_γ. The following hold*

(a) $J_\gamma(\theta^\gamma) \geq J_{\tilde\gamma}(\theta^{\tilde\gamma})$, *for* $\gamma \geq \tilde\gamma$;
(b) $J_0 \geq J_\gamma(\theta^\gamma)$ *for each* γ;
(c) $\lim_{\gamma\to\infty} \frac{\gamma}{2} g(\theta^\gamma) = 0$.

In particular, part (c) shows that as $\gamma \to \infty$ the inequalities in (3c) are satisfied. As pointed out in [13], the above result does not require any continuity, convergence, or convexity assumptions. To establish a convergence of the sequence $\{\theta^\gamma\}$ to a limit point, additional assumptions are needed. In particular, we need J and g to be lower semicontinuous. See [13, Section 10.11 (Theorem 1)] for the details. It is unclear if these assumptions hold for the highly nonconvex J. In view of this, the Proposition 1 maybe of limited scope in practice. We further emphasize that

Lemma 2 and Theorem 2 in [13] are not applicable in our setting, as they require convexity of J, which we may not have.

Remark 1. Typically, when a Moreau-Yosida regularization is implemented, a path-following technique is used to increase the size of γ gradually. This means that a sequence of optimization problems is solved for subsequently larger values of γ. The initial γ value is taken to be small, and the solution to the problem, θ^γ, is used as the initial guess for the next optimization problem with a larger value of γ. This path-following process continues until γ is sufficiently large. For the numerical examples presented below, we first tested the path-following strategy and also ran the experiment for a fixed γ. For these examples, the results were almost identical and therefore we chose to present results for a fixed γ. Nevertheless, we emphasize that this choice is problem dependent and one may have to apply a path-following strategy.

4. ResNets.

As some of the examples shown below are obtained for Deep Residual Neural Networks (ResNets), a small description follows. Recall the definition of $\mathcal{F}$ in (1). In the sequel, the layer functions will be denoted as $f_\ell = f_\ell(y_\ell; (W_\ell, b_\ell)) : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_{\ell+1}}$, where the dependence of f_ℓ on σ is not explicitly written. With this representation the neural network can be viewed as an iterative progression of updating the output of each layer as

$$y_{\ell+1} = f_\ell(y_\ell; (W_\ell, b_\ell)) \qquad \ell = 0, \dots, L-1,$$

with initial input y_0 and final output y_L. As before, in order to preserve consistency, b_{L-1} is taken to be the zero vector. One way to define the layer functions is

$$\begin{aligned} f_0(y_0) &:= \sigma(W_0 y_0 + b_0), \\ f_\ell(y_\ell) &:= P_\ell y_\ell + \tau\sigma(W_\ell y_\ell + b_\ell) \qquad \ell = 1, \dots, L-2, \\ f_{L-1} &:= W_{L-1} y_{L-1}, \end{aligned}$$

for matrices $P_\ell \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$ and a scalar $\tau > 0$. With this definition and for $L > 2$, $\mathcal{F}$ is termed as a Deep Residual Neural Network (ResNet). If each P_ℓ is taken to be the identity matrix, which requires that the hidden layers have a uniform width, then this network can be viewed as a forward Euler discretization of an ODE. For more on these kinds of networks see [8, 17, 3, 2, 7], among others. Notice that if $\tau = 1$ and $\{P_\ell\}_{\ell=0}^{L-1}$ contains only zero entries, then $\mathcal{F}$ is a standard feedforward deep neural network [6].

5. Numerical Results.

In this section several examples are given that not only show the efficacy of this method, but also the advantages of using the method. In Section 5.1, a first example compares a single ResNet to learn the function $\sin(x)$ with and without bias ordering. This simple example shows that the method performs as desired, and in fact outperforms the same network trained without bias ordering. Following this, Section 5.2, a more complicated experiment is reported that use parallel ResNets to learn a model related to chemically reacting flows [5]. This example shows that the proposed method is a useful technique for practical problems in machine learning. Finally, in Section 5.3, an example is described where the bias ordering regularization is applied to a classification problem using MNIST data and implemented in Keras. This example shows that the proposed method is also suitable for Convolutional Neural Networks.

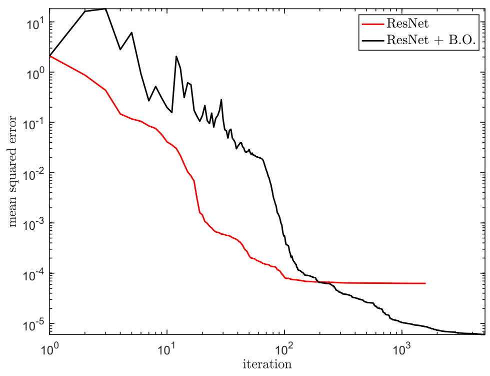

FIGURE 2. The panel shows the mean squared error corresponding to the the first term in the definition of J (cf. (4)) as a function of the number of optimization iterations during the training process. We clearly notice that the standard ResNet stalls, but the bias order (B.O.) approach leads to a much better reduction.

In the first two examples, we have taken a fixed parameter $\lambda = 10^{-7}$, in the definition of loss function J and λ is set to zero in the final example. For the first two examples, the choice choice of λ was made after a careful experimental study, in the non-bias ordering setting, i.e., $\gamma = 0$. We observed almost no change in the training and testing accuracies for λ in the range of 10^{-6} to 10^{-10}. Since this paper is about 'bias-ordering' and its subsequent effects, therefore we have taken λ to be fixed in these experiments. Notice that it maybe possible to also choose λ via traditional approaches such as validation.

In Sections 5.1 and 5.2 a BFGS optimization routine with Armijo line search is used during training to solve the optimization problem with or without bias ordering. In order to avoid overfitting, validation data is used with a patience of 400 iterations. This means that the training data is separated into two sets: training and validation data. The validation data is not used to update the weights and biases, rather it is used to measure the error of the network on unseen data (the validation data) during training. If the validation error increases, the patience iterations provide a buffer during which training continues. If during these iterations, the validation error reaches a new minimum, then training continues as before, otherwise the training routine is terminated.

5.1. ResNet to learn $\sin(x)$.

For the first example a simple ResNet with 2 hidden layers of width 50 is used to learn the function $\sin(x)$. The skip parameter τ is taken to be 1. For data, 1000 evenly spaced points from the interval $[0, 2\pi]$ are generated and then randomly split into training data (400 points), testing data (400 points), and validation data (200 points).

The same experiment was performed twice, once using the loss function J, and once with the loss function J_γ with $\gamma = 100$. The least square training error corresponding to the the first term in the definition of J (cf. (4)) is shown in Figure 2. We notice that a standard ResNet stagnates, while the ResNet with bias ordering continues to make progress. The resulting trained networks are shown in Figure 3, where the bias values are represented by the neurons of the network, and the weights are represented by the connections between neurons. The color for the input and output are set to zero. In the case where the network was trained with loss function J_γ, the resulting biases were perfectly ordered and so the inequality constraints were satisfied. This can also be seen in the bottom panel in Figure 3. Note that for the network with unconstrained bias values, there are on the order of 10^{64} permutations of the parameters that will give the same ResNet approximation. For the ResNet with ordered bias values, however, any permutation of the parameters will either result in a different ResNet approximation, or violate the bias ordering.

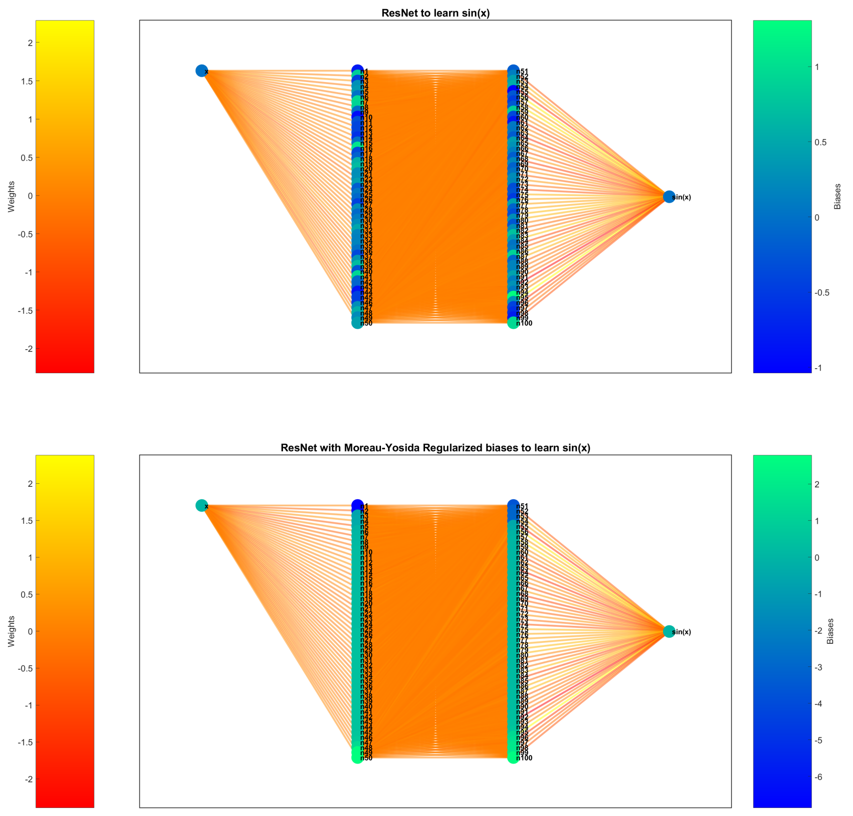
ResNet to learn sin(x)
Weights
Biases
sin(x)
ResNet with Moreau-Yosida Regularized biases to learn sin(x)
Weights
Biases
sin(x)

FIGURE 3. Network visualizations for the ResNets trained with loss function J (top) and J_γ (bottom). The network consists of 2 hidden layers with 50 neurons in each layer to learn $\sin(x)$. It is clear from the panels that the ordering of the bias values is only enforced in the bottom panel.

In Figure 4 the output of the two networks is compared on the 400 test points, with exact values in blue and ResNet output in red. The left plot of Figure 4 shows the results for a network trained with a standard loss function J, while the plot on the right shows results from a network trained with the augmented loss function J_γ which orders the biases. It is evident from the plots that the ResNet that implements bias order produced more accurate results. This is quantified above the plots with the relative error measured in the 2-norm.

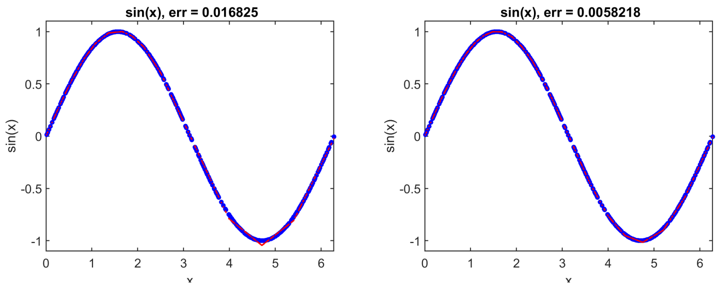

FIGURE 4. Results from training ResNets to learn the function $\sin(x)$. Exact values in blue, ResNet output in red. The left panel shows results for the standard ResNet with J and the right panel shows results where bias ordering is enforced using J_γ. Both visual inspection and quantitative inspection of the error confirms that the proposed approach works better in this example.

5.2. **Applications to chemically reacting flows.** In this section experiments are presented involving a ResNet approximation of a system of stiff ODEs that model a reduced H_2-O_2 reaction. This reduced model (see [15]) tracks 8 species and temperature as they interact over time and is completely separated from any advection and diffusion in space. For more information on this problem and more experiments using a parallel ResNet approximation see [5].

For the experiments included in this work, nine parallel ResNets with input dimension 10 and output dimension 1 (9 total output quantities) and 8 hidden layers of width 30 were trained on H_2-O_2 reaction data created by solving the stiff ODE system using CHEMKIN [11]. Given an input vector representing the data at time t_k, each ResNet is trained to learn a single quantity (temperature, for example) at time t_{k+1}. More details can be found in [5]. The generated data used to train and test these networks consists of thirteen subsets corresponding to initial conditions with a fixed equivalence ratio of 1 and different temperatures varying from 1200K to 2400K in increments of 100K. The parallel ResNets were trained on the data sets corresponding to initial temperatures 1200K, 1500K, 1800K, 2100K, and 2400K.

The experiment above was performed twice, once with loss functions J (one for each parallel ResNet) as described in Section 2, and once with regularized loss functions J_γ with $\gamma = 1000$. All other network hyperparameters including the initial values of the weights and biases prior to training are kept the same for the two experiments. In Table 1 the number of BFGS iterations used in training the parallel networks for both experiments described above is compared. The network with Moreau-Yosida regularization to order the bias values used fewer iterations to train in five of the nine networks. Nevertheless, the overall iteration count in case of bias ordering approach is smaller.

Recall from Proposition 1, that the Moreau-Yosida regularization approach will be enforcing the ordering (3c) approximately.

The loss functions J_γ penalize the violation of the ordering, but do not strictly implement the order itself. Even so, the biases in eight of the nine parallel networks were ordered perfectly for the networks that trained with the loss functions J_γ. The only violation of the bias ordering occured in Network 9, the network used to learn H_2. In the first hidden layer of this network neurons six and seven were ordered

TABLE 1. A comparision of the number of BFGS iterations used during training for the two sets of parallel ResNets. The quantity that the ResNet is learning (output) is written in parentheses.

Summary of BFGS iterations used during training		
	with loss functions J	with loss functions J_γ
Network 1 (temperature)	2638	786
Network 2 (O)	2506	5125
Network 3 (H)	2772	1123
Network 4 (OH)	5957	4444
Network 5 (HO_2)	230	540
Network 6 (H_2O_2)	886	1738
Network 7 (H_2O)	7267	583
Network 8 (O_2)	8872	2547
Network 9 (H_2)	3719	4570

incorrectly. In the trained network, the value of neuron 6 was approximately -0.0175652 and the value of neuron 7 was approximately -0.0175660, and therefore the size of the order violation was -8e-7, which is negligible. To reiterate, in each of the 9 networks, there were 240 bias values (30 per hidden layer) for a total of 2,160 different biases. In all of these biases only a single pair (negligibly) violated the monotonic ordering using the proposed method.

Figure 5 compares the results of the two sets of parallel ResNets tested with initial conditions with initial temperature 1400K (left set of plots) and 2000K (right set of plots). Note that the networks were not trained on this data. To test the networks only the initial condition comes from the CHEMKIN data. The output of the ResNets from the initial condition are then combined and used as the input to the parallel ResNets for the next timestep. This process is repeated for the duration of the reaction. To compare the results, the known CHEMKIN data are represented in blue, the results from the networks trained with J are represented with dashed red lines and the results for the networks trained with J_γ are represented with dash/dotted black lines. It can clearly be seen that the results of the parallel ResNets trained with J_γ match the data more closely. Furthermore, note that the $x-$axis (time axis) is scaled logarithmically in order to display the details of H_2O_2 which happen quickly and early in the reaction. Therefore, while the results in black anticipate the reaction, they only do so slightly, on the order of 10^{-6} seconds.

5.3. **A Convolutional Neural Network classification problem.** For this example, a convolutional neural network is constructed using Keras to solve a classification problem using the MNIST data set. The purpose of this example is to show that the flexibility of the bias ordering technique and that it can be incorporated into existing neural network software. Unlike the other examples presented, the neural network is not a ResNet. Instead the network consists of 4 convolutional layers followed by two dense layers. The diagram in Figure 1 can also be used to describe a convolutional layer, where the nodes of the graph still represent the bias value, but now the edges represent a convolutional filter or convolutional kernel. In this way, the bias ordering technique easily applies to these networks as well.

The network used for this example consists of four convolutional layers followed by two dense layers. The first two convolutional layers consist of 32 convolutional

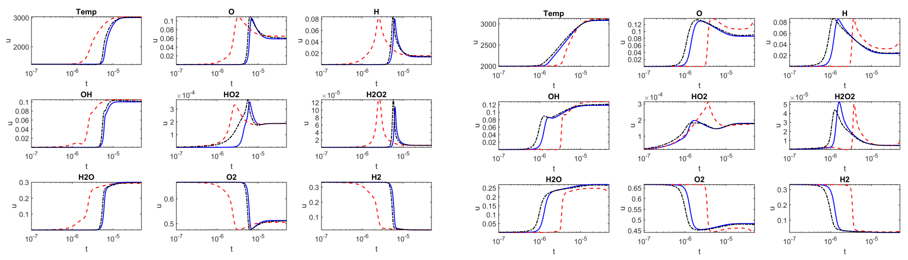

FIGURE 5. Results from training parallel ResNets on a reduced H_2-O_2 reaction model. Values from known data in blue, results from ResNets trained with traditional loss functions in red, and results from ResNets trained with loss functions with a Moreau-Yosida penalization term to order the biases in black. Clearly, the proposed approach outperforms the existing one.

filters of width 3. Convolutional layers three and four both consist of 64 convolutional filters of width 3. The first dense layer consists of 512 neurons, and the second dense layer (the output layer) consists of 10 neurons since there are 10 digits (or classes) in the MNIST data. Unlike the other examples, bias values are included on the output layer of this network. A softmax activation function is used for the output layer, while a ReLU activation function is used for the other five layers. Max pooling is used after layers 2 and 4. In order to avoid overfitting, dropout is used in the network.

Two versions of the network described above are constructed using Keras with a Tensorflow backend. Both of the networks use a categorical cross entropy loss function, and the difference between the two is that one of the networks also implements the Moreau-Yosida regularization term through a custom bias regularizer. The regularizer is defined using the following lines in Python

```
def MY_regularizer(bias, gamma=100):
    bias_length = tf.size(bias)
    bias_diff = bias[1:bias_length]-bias[0:bias_length -1]
    bias_min = tf.math.minimum(bias_diff,0)
    bias_norm = tf.math.reduce_sum(tf.math.square(bias_min))
    return 0.5*gamma*bias_norm
```

In the above code , the value of the regularization parameter is taken to be 100, but this is easily customized. Once the regularizer is defined, it can be implemented into any layer with the `bias_regularizer` option.

Both of the networks were trained on 60,000 samples from the MNIST data set. The loss functions were minimized using stochastic gradient descent with a learning rate of 0.01. Batch normalization was used during training with a batch size of 32. Each network trained for 5 epochs before the networks were tested on 10,000 MNIST samples. In Figure 6 the bias values for each layer of the two networks are plotted. It can clearly be seen that the regularization resulted in successfully ordering the bias values in each layer. The accuracy of the two trained networks were comparable. For the networks corresponding to the plots in Figure 6, the network with the bias regularization had an accuracy of 98.71% and the network without regularization had an accuracy of 98.56%. The plot in Figure 7 represents the bias values from the same two networks described above, and shown in Figure 6, except now the bias values for the unregularized network have also been ordered for comparison. From these plots, one can easily see that not only is the Moreau-Yosida regularization term ordering the bias values during training, it is also causing the magnitudes of the biases to have a smaller range. Since the focus of this current work is the implementation and efficacy of the bias ordering, the study of this other effect is saved for future investigations.

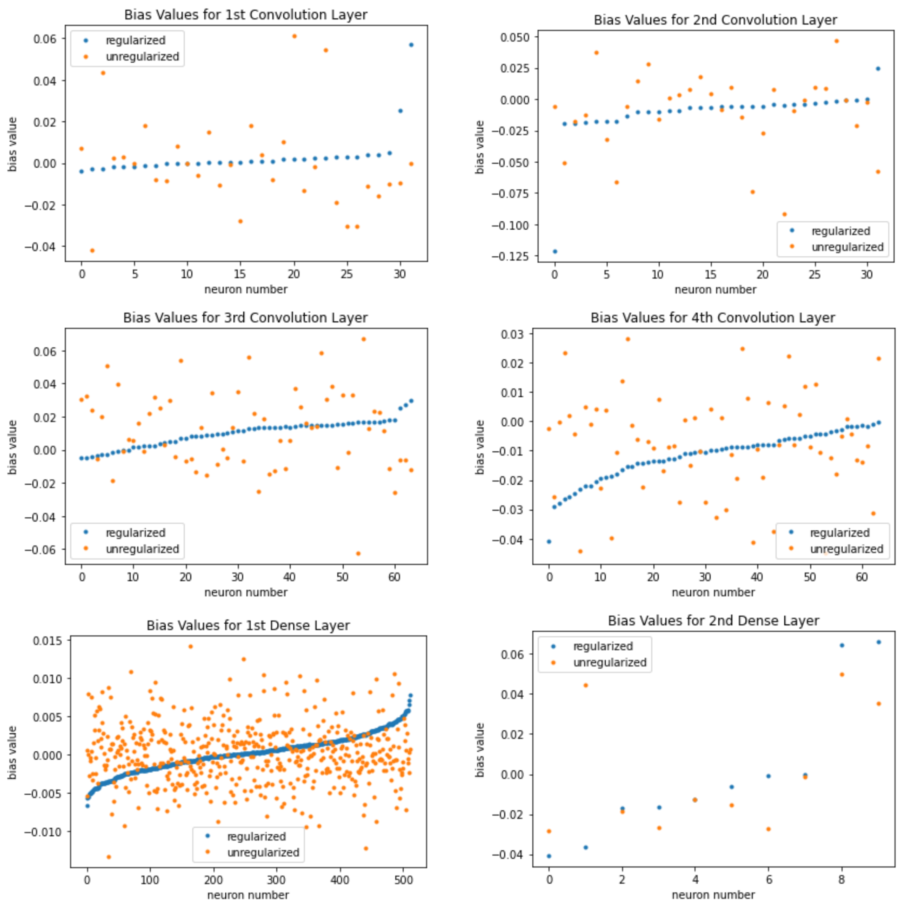

FIGURE 6. The bias values by layer for the two neural networks described in Section 5.3. These plots show the successful implementation of the Moreau-Yosida bias order regularization in Keras with a Tensorflow backend.

6. **Conclusions.** A method to reduce the very large search space of equivalently optimal neural nets has been introduced. The key idea is to try to enforce the biases to be monotonically increasing in each layer of neurons. This is accomplished by a Moreau-Yosida regularization-based approach to solve the resulting optimization problem. The convergence of the regularized problem has been proven. The benefit of this new approach has been demonstrated not only on simple approximation cases

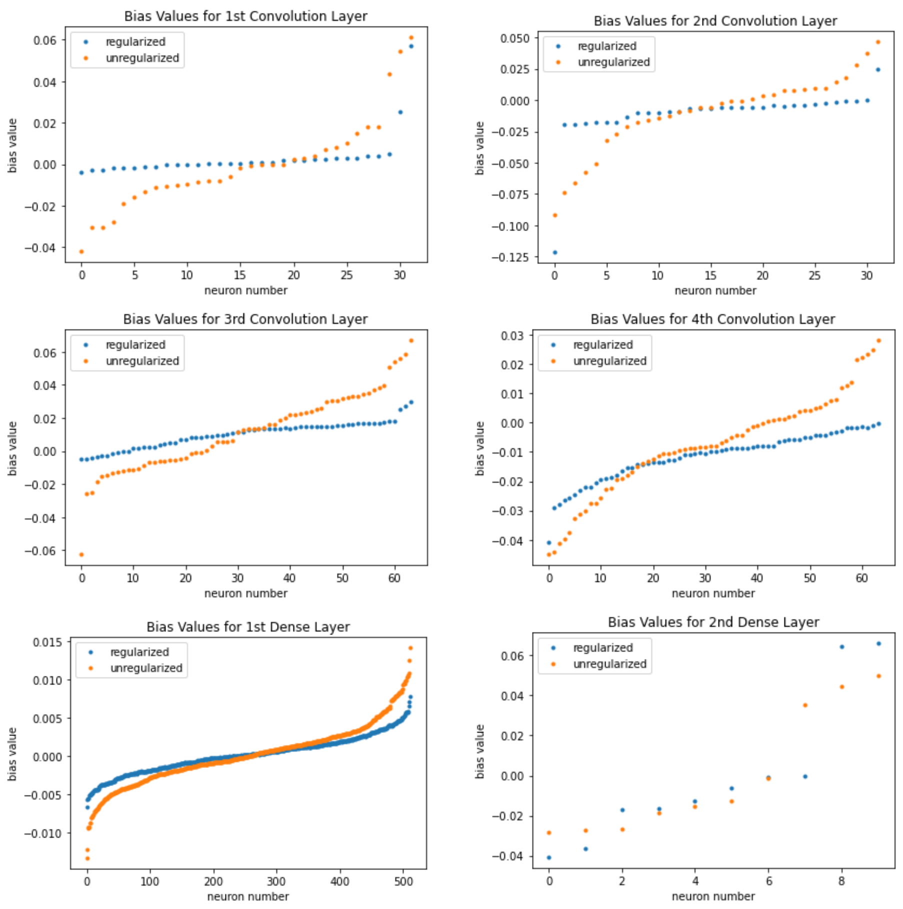

FIGURE 7. The bias values by layer for the two neural networks described in Section 5.3 and shown in Figure 6. The bias values for the unregularized network have been ordered after training for comparison.

but also on a realistic problem arising in chemically reacting flows. The numerical experiments presented also show that the regularization technique is effective for relatively small values of the penalization parameter γ.

Appendix A. **Derivation of the first order optimality conditions.** In the context of constrained optimization, the problem (6) is typically solved by using a gradient based method [16, 12, 18, 4, 16]. Indeed this is the approach taken in the numerical experiments. Each evaluation of the gradient requires solving the state equation or forward problem and solving the adjoint equation problem. In what follows, the state and adjoint equations will be derived for the problem in (6) where the neural network being used is a Deep Residual Neural Network (ResNet), as introduced in Section 4.

The state and adjoint equations as well as the gradient will be derived by using the Lagrangian approach. For brevity, the following will be written for a single input u, rather than for a set of training data. Similar to Section 3, θ is used to represent the concatenation of the weights and bias, i.e., the parameters being optimized. For appropriate adjoint variables $\psi = (\psi_j)_{j=1}^{L}$ the Lagrangian functional corresponding to (6) is

$$\mathcal{L}(u,\theta,\psi) := J_\gamma(\theta) + \langle y_1 - \tau\sigma(W_0 u + b_0), \psi_1\rangle$$
$$+ \sum_{j=2}^{L-1} \left\langle y_j - P_{j-1}y_{j-1} - \tau\sigma(W_{j-1}y_{j-1} + b_{j-1}), \psi_j \right\rangle + \langle y_L - W_{L-1}y_{L-1}, \psi_L\rangle,$$

From here, the state and adjoint equations result from evaluating the derivatives of the Lagrangian with respect to y_j and ψ_j at a stationary point. Furthermore, the gradient is derived by taking the derivatives of the Lagrangian with respect to θ.

(i) State Equation.

$$\begin{aligned} y_1 &= \tau\sigma(W_0 u + b_0), \\ y_j &= P_{j-1}y_{j-1} + \tau\sigma(W_{j-1}y_{j-1} + b_{j-1}), \qquad 2 \le j \le L-1, \\ y_L &= W_{L-1}y_{L-1}. \end{aligned} \tag{7a}$$

(ii) Adjoint Equation.

$$\begin{aligned} \psi_j &= P_j^T\psi_{j+1} - \tau\left[-W_j^T\left(\psi_{j+1}\odot\sigma'\left(W_j y_{j+1} + b_j\right)\right)\right] \qquad j = L-2,\ldots,1 \\ \psi_{L-1} &= -W_{L-1}^T\psi_L, \\ \psi_L &= -\partial_{y_L}J_\gamma(\theta). \end{aligned} \tag{7b}$$

(iii) Derivative with respect to θ.

$$\begin{aligned} \partial_{W_{L-1}}\mathcal{L} &= -\,\psi_L\, y_{L-1}^T + \partial_{W_{L-1}}J_\gamma(\theta) \\ &= \partial_{y_L}J_\gamma(\theta)\, y_{L-1}^T + \partial_{W_{L-1}}J_\gamma(\theta), \\ \partial_{W_j}\mathcal{L} &= -\,y_j\ \left(\psi_{j+1}\odot\sigma'(W_j y_j + b_j)\right)^T + \partial_{W_j}J_\gamma(\theta) \qquad j = 0,\ldots,L-2, \\ \partial_{b_j}\mathcal{L} &= -\ \psi_{j+1}^T\,\sigma'(W_j y_j + b_j) + \partial_{b_j}J_\gamma(\theta) \qquad j = 0,\ldots,L-2\,. \end{aligned} \tag{7c}$$

In the jargon of machine learning the equations in (7b) are called back propagation. The gradient is represented by the right hand side of (7c), where the contributions from the Moreau-Yosida regularization terms enter in the term $\partial_{b_j}J_\gamma(\theta)$.

REFERENCES

[1] Harbir Antil, Thomas S. Brown, Deepanshu Verma and Mahamadi Warma, Optimal control of fractional PDEs with state and control constraints, To appear in Pure and Applied Functional Analysis, 2021.

[2] Harbir Antil, Howard C Elman, Akwum Onwunta and Deepanshu Verma, Novel deep neural networks for solving bayesian statistical inverse, arXiv:2102.03974, 2021.

[3] Harbir Antil, Ratna Khatri, Rainald L Lohner and Deepanshu Verma, Fractional deep neural network via constrained optimization, *Machine Learning: Science and Technology*, 2020.

[4] Martin Benning, Elena Celledoni, Matthias J. Ehrhardt, Brynjulf Owren and Carola-Bibiane Schönlieb, Deep learning as optimal control problems: models and numerical methods, *J. Comput. Dyn.*, **6** (2019), 171–198.

[5] Thomas S Brown, Harbir Antil, Rainald Löhner, Fumiya Togashi and Deepanshu Verma, Novel dnns for stiff odes with applications to chemically reacting flows, In *High Performance Computing* (eds. H. Jagode, H. Anzt, H. Ltaief, P. Luszczek), ISC High Performance 2021, Lecture Notes in Computer Science, 12761, 2021.

[6] Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*, Adaptive Computation and Machine Learning, MIT Press, Cambridge, MA, 2016.

[7] Stefanie Günther, Lars Ruthotto, Jacob B. Schroder, Eric C. Cyr and Nicolas R. Gauger, Layer-parallel training of deep residual neural networks, *SIAM J. Math. Data Sci.*, **2** (2020), 1–23.

[8] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (2016), 770–778.

[9] Michael Hintermüller and Michael Hinze, Moreau-Yosida regularization in state constrained elliptic control problems: error estimates and parameter adjustment, *SIAM J. Numer. Anal.*, **47** (2009), 1666–1683.

[10] Kazufumi Ito and Karl Kunisch, *Lagrange Multiplier Approach to Variational Problems and Applications*, Volume 15 of *Advances in Design and Control*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2008.

[11] R. J. Kee, F. M. Rupley, J. A. Miller, M. E. Coltrin, J. F. Grcar, E. Meeks, H. K. Moffat, A. E. Lutz, G. Dixon-Lewis, M. D. Smooke, J. Warnatz, G. H. Evans, R. S. Larson, R. E. Mitchell, L. R. Petzold, W. C. Reynolds, M. Caracotsios, W. E. Stewart, P. Glarborg, C. Wang and O. Adigun, Chemkin Collection, release 3.6, 2000.

[12] Yann LeCun, D Touresky, G Hinton and T Sejnowski, A theoretical framework for back-propagation, In *Proceedings of the 1988 Connectionist Models Summer School*, **1** (1988), 21–28.

[13] David G. Luenberger, *Optimization by Vector Space Methods*, John Wiley & Sons, Inc., New York-London-Sydney, 1969.

[14] Ira Neitzel and Fredi Tröltzsch, On regularization methods for the numerical solution of parabolic control problems with pointwise state constraints, *ESAIM Control Optim. Calc. Var.*, **15** (2009), 426–453.

[15] Eric L. Petersen and Ronald K. Hanson, Reduced kinetics mechanisms for ram accelerator combustion, *Journal of Propulsion and Power*, **15** (1999), 591–600.

[16] Lorenzo Rosasco, Silvia Villa and Bang Công Vũ, Convergence of stochastic proximal gradient algorithm, *Appl. Math. Optim.*, **82** (2020), 891–917.

[17] Lars Ruthotto and Eldad Haber, Deep neural networks motivated by partial differential equations, *J. Math. Imaging Vision*, **62** (2020), 352–364.

[18] E Weinan, A proposal on machine learning via dynamical systems, *Communications in Mathematics and Statistics*, **1** (2017), 1–11.

Received August 2021; 1^{st} revision March 2022; Final revision June 2022; early access July 2022.

E-mail address: hantil@gmu.edu
E-mail address: tbrown62@gmu.edu
E-mail address: rlohner@gmu.edu
E-mail address: fumiya.togashi@gmail.com
E-mail address: deepanshu.verma@emory.edu