An Adaptive and Stability-Promoting Layerwise Training Approach for Sparse Deep Neural Network Architecture

C G Krishnanunni∗ and Tan Bui-Thanh †

Abstract. This work presents a two-stage adaptive framework for progressively developing deep neural network (DNN) architectures that generalize well for a given training data set. In the first stage, a layerwise training approach is adopted where a new layer is added each time and trained independently by freezing parameters in the previous layers. We impose desirable structures on the DNN by employing manifold regularization, sparsity regularization, and physics-informed terms. We introduce a $\varepsilon-\delta-$stability-promoting concept as a desirable property for a learning algorithm and show that employing manifold regularization yields a $\varepsilon-\delta$ stability-promoting algorithm. Further, we also derive the necessary conditions for the trainability of a newly added layer and investigate the training saturation problem. In the second stage of the algorithm (post-processing), a sequence of shallow networks is employed to extract information from the residual produced in the first stage, thereby improving the prediction accuracy. Numerical investigations on prototype regression and classification problems demonstrate that the proposed approach can outperform fully connected DNNs of the same size. Moreover, by equipping the physics-informed neural network (PINN) with the proposed adaptive architecture strategy to solve partial differential equations, we numerically show that adaptive PINNs not only are superior to standard PINNs but also produce interpretable hidden layers with provable stability. We also apply our architecture design strategy to solve inverse problems governed by elliptic partial differential equations.

Key words. Architecture adaptation, Manifold regularization, Physics-informed neural network, Sequential learning, Interpretable machine learning.

AMS subject classifications. 68T07, 68T05

1. Introduction. Deep neural networks (DNNs) have been prominent in learning representative hierarchical features from image, video, speech, and audio inputs [15, 23, 27, 38]. Some of the problems associated with training such deep networks include i) a possible large training set is necessary to overcome the over-fitting issue; ii) the architecture adaptability problem, e.g., any amendments to a pre-trained DNN require retraining even with transfer learning; and iii) GPU employment is almost mandatory due to massive network and data sizes. In particular, it is often unclear on the choice of depth and width of a network suitable for a specific problem. Existing neural architecture search algorithms rely on metaheuristic optimization, or combinatorial optimization, or reinforcement learning strategy to hopefully arrive at a reasonable architecture [61, 48, 50, 16, 43, 1, 37, 31]. However, these strategies involve training and evaluating many candidate architectures (possibly deep) in the process and are thus computationally expensive. Therefore, there is a need to develop a computationally tractable procedure for adaptively training/growing a DNN by leveraging information on the input data manifold and the underlying structure of the problems under consideration.

1.1. Related work. Layerwise training of neural networks is an approach that addresses the issue of the choice of depth of a neural network and the computational complexity involved
Many efforts have been proposed for growing neural architecture using this approach [22, 26, 9, 32, 55, 39, 44]. Hettinger et al. [22] showed that layers can be trained one at a time and the resulting DNN can generalize better. Belilovsky et al. [2] have further studied this approach for different convolutional neural networks. Kulkarni et al. [26] employed a kernel analysis of the trained layerwise deep networks. The weights for each layer are obtained by solving an optimization problem aiming at a better representation where a subsequent layer builds its representation on top of the features produced by a previous layer. Lengelle et al. [29] conducted a layer-by-layer training of multilayer perceptron by optimizing an objective function for internal representations while avoiding any computation of the network’s outputs. Bengio et al. [5] proposed a greedy layerwise unsupervised learning algorithm by initializing weights for each layer in a region near a good local minimum, giving rise to internal distributed representations that are high-level abstractions of the input, thereby bringing better generalization. Nguyen et al. [39] developed an analytic layerwise deep learning framework where an inverse layerwise training and a forward progressive learning approach are adopted.

Alternately, researchers have also considered growing the width gradually by adding neurons for a fixed depth neural network [53, 54]. Wynne-Jones [54] considered splitting the neurons (adding neurons) based on a principal component analysis on the oscillating weight vector. Liu et al. [53] developed a simple criterion for deciding the best subset of neurons to split and a splitting gradient for optimally updating the off-springs. Chen et al. [10] showed that replacing a model with an equivalent model that is wider (has more neurons in each hidden layer) allows the equivalent model to inherit the knowledge from the existing one and can be trained to further improve the performance.

It is noteworthy that while the works cited above show promising results for image classification tasks, layerwise training procedure can introduce a large number of parameters (especially in the context of a fully connected network) due to the addition of a new layer [22]. In such cases, it is important to recognize non-important parameters in an added layer and remove them. Further, layerwise training strategies in [2, 22, 51] do not ensure that the network output remains unchanged after a new layer is added thereby losing the performance achieved by previous layers. Moreover, recent work by Trinh et al. [51] observed overfitting in the early layers while performing a layerwise training strategy. Therefore, it is imperative to consider a layerwise training strategy that takes into account the following aspects: a) achieves light-weight hidden layers, i.e., automatically detects the non-important connections in an added layer and removes them; b) ensures that the network output remains unchanged after the addition of a new layer; c) mathematical principle to guide the design of each layer that prevents overfitting in early layers.

1.2. Our contributions. In this work, we use “robustness” as a desirable property for DNNs [24, 7, 58] and use this as a criterion to devise a strategy for progressively adapting DNN architecture for a given data-set. While recent work shows that over-parametrization could be necessary for “robustness” [7, 6], it is generally unclear how to distribute the parameters in a network (number of active connections in each hidden layer, number of layers). Our algorithm progressively builds a network by training one layer at a time while promoting stability for each layer and consequently promoting “robustness” as we will show. In particular, we set forth a sequential learning strategy where a learning task is divided into sub-tasks each of
which is learned before the next. Our idea is inspired by the mechanism of how the human brain learns [11]. A layerwise training strategy (Algorithm 2.1) where each layer is constrained to learn a specific task is adopted. To that end, when training each layer we incorporate a sparsity regularization for identifying the active/inactive parameters in each layer, a manifold regularization term [4] for promoting stability, and a physics-informed term [42] aiming to respect the underlying physics (if any). The key to our approach is to stimulate $\delta$-stable neural hidden layers. Once layerwise training saturates, which we shall prove, we provide Algorithm 2.2 where a sequence of small networks is deployed to extract information from the residuals induced by the DNN generated from Algorithm 2.1. Using Algorithm 2.2 together with Algorithm 2.1 results in robustness and accuracy as we shall show. Since the procedure emphasizes training a small network/layer each time, it is computationally tractable for large-scale problems and is free from vanishing gradient problems. Numerical investigation on prototype regression problems, physics-informed neural networks (PINNs) for solving elliptic partial differential equations, and MNIST classification problems suggest that the proposed approach can outperform ad-hoc baseline networks.

2. Proposed methodology. The proposed architecture adaptation procedure has two stages: a) A layerwise training strategy for growing the depth of DNNs; and b) a sequential residual learning strategy in which a sequence of shallow networks is added for robust and accurate predictions. A brief outline of the two stages is provided below in subsection 2.1 and subsection 2.2.

2.1. Layerwise training strategy (Algorithm 2.1). One of the key aspects in our layerwise training strategy is to employ the Residual Neural Network (ResNet) [20, 21] which, as we will show, improves the training loss accuracy when adding new layers. In this work, all the matrices and vectors are represented in boldface. Notation $(\cdot)^{(i)}$ denotes the quantities (matrix, vector, scalar, set, or function) for the $i$th layer. Consider a regression/classification problem of $O$ outputs and $S$ input training features. Given inputs $x_i \in \mathbb{R}^S$ for $i \in \{1, 2, \ldots, M\}$ organized column-wise into a matrix $X \in \mathbb{R}^{S \times M}$, we denote the corresponding network outputs as $y_i \in \mathbb{R}^O$ which can also be organized column-wise into a matrix $Y \in \mathbb{R}^{O \times M}$. The corresponding true labels are denoted as $c_i \in \mathbb{R}^O$ and in stacked column-wise as $C \in \mathbb{R}^{O \times M}$. We denote $Y^{(1)} = h^{(1)}(W^{(1)}X + b^{(1)})$ as the output of the first (optional) upsampling/downsampling layer, where $W^{(1)}$ and $b^{(1)}$ represents the weight matrix and bias vector for this layer. $Y^{(1)}$ is then propagated forward through a $L$ layers of ResNet as:

\[
Y^{(1)} = h^{(1)}(W^{(1)}X + b^{(1)}),
\]
(2.1)

\[
Y^{(i+1)} = R^{(i)}(Y^{(i)}) + h^{(i+1)}(W^{(i+1)}R^{(i)}(Y^{(i)}) + b^{(i+1)}), \quad \text{for } i = 1, \ldots, L - 1,
\]

\[
Y^{(L+1)} = h_{\text{pred}}(W_{\text{pred}}R^{(l)}(Y^{(L)}) + b_{\text{pred}}),
\]

where $h^{(i+1)}, W^{(i+1)}, b^{(i+1)}$ are the activation function, weight matrix, and bias vector of the $(l + 1)$th layer, while $h_{\text{pred}}, W_{\text{pred}}, b_{\text{pred}}$ are the activation function, weight matrix, and bias vector of the output layer, and $R^{(l)}$ represents an optional linear transformation for feature scaling on each hidden layer (we only need this for theoretical purpose in Proposition 3.9).
In order to impose desirable properties for the layerwise training strategy, we incorporate there different regularization terms, namely, (a) Manifold regularization, (b) Physics-informed regularization, and (c) Sparsity regularization.

1. **Manifold Regularization (Data-dependent regularization):** Manifold regularization exploits the geometry of marginal distribution where we assume that the input data is not drawn uniformly from input space \( \mathbb{R}^S \) but lies on a submanifold \( \mathcal{M} \subset \mathbb{R}^S \) [4, 17]. Let \( \mu \) be a probability measure with support on \( \mathcal{M} \). Let \( \nabla_{\mathcal{M}} f(x) \) denote the gradient of \( f \) along the manifold \( \mathcal{M} \) (see, e.g., Do Carmo [14] for an introduction to differential geometry and definition of gradient along the manifold). To force the gradient of the learned function (each hidden layer in our case) to be small whenever the probability of drawing a sample is large, one can define the following \( H^1 \)-like regularization term [4, 24, 45, 28]:

\[
(2.2) \quad \Phi_m = \int_{\mathcal{M}} \left\| \nabla_{\mathcal{M}} y_{\theta}^{(l)}(x) \right\|_2^2 \, d\mu(x) \approx \frac{1}{M^2} \sum_{i,j} \beta_{ij} \left\| y_{\theta}^{(l)}(x_i) - y_{\theta}^{(l)}(x_j) \right\|_2^2,
\]

where \( y_{\theta}^{(l)}(x) \) is the output of the \( l \)th hidden layer for an input \( x \), \( \theta \) represents the trainable network parameters up to layer \( l \), and the indices \( i \) and \( j \) on the right hand side of (2.2) varies from 1 to \( M \) to represent the pairwise comparison between all training samples. Note that the convergence of the right-hand side to the left-hand side of (2.2) is valid under certain conditions and when choosing exponential weights for the similarity matrix \( \beta_{ij} \) [4, 3]. However, (2.2) is expensive to compute when \( M \) is large due to the computational complexity of \( \mathcal{O}(M^2) \). In order to reduce the computational complexity, the similarity matrix \( \beta_{ij} \) in (2.2) is computed based on the pairwise must-link constraint set [34]:

\[
(2.3) \quad \beta_{ij} = \begin{cases} 
\beta_p & \text{for } c_i^{(p)} = c_j^{(p)}, \\
0 & \text{otherwise},
\end{cases}
\]

where \( c_i^{(p)} \) denotes the label assigned to training input \( x_i \) belonging to cluster/class \( C_p \) for a classification task, and \( \beta_p \geq 0 \) and bounded such that \( \beta_p \neq 0 \) for at least one \( p \). For regression problems, one may use clustering algorithms such as K-means clustering for computing \( \beta_{ij} \) in (2.3), where data points in the same clusters are assigned the same labels, and different clusters have different labels. Inspired by this choice of similarity matrix \( \beta_{ij} \) in practice, we consider a new form of manifold regularization \( \Phi_m \) as follows:

\[
(2.4) \quad \Phi_m = \sum_{p=1}^{K} \beta_p \int_{\mathcal{M}_p} \int_{\mathcal{M}_p} \left\| y_{\theta}^{(l)}(x) - y_{\theta}^{(l)}(y) \right\|_2^2 \, d\mu_p(x) \, d\mu_p(y) \\
\approx \sum_{p=1}^{K} \frac{\beta_p}{m_p^2} \sum_{x_i, x_j \in \mathcal{C}_p} \left\| y_{\theta}^{(l)}(x_i) - y_{\theta}^{(l)}(x_j) \right\|_2^2,
\]

where \( \mu_p \) is the probability measure with support \( \mathcal{M}_p \) and \( \cup_p \mathcal{M}_p = \mathcal{M} \), \( K \) denotes the total number of clusters, and \( m_p \) denotes the number of samples in \( \mathcal{C}_p \). In Propo-
sition 3.9, we will show that penalizing (2.4) helps in promoting stability for hidden layers during the learning process.

2. Physics-informed regularization: In addition to the training data, if one is also provided with information on the underlying physics of the problem, then each hidden layer in our proposed approach can be interpretable (see section 4.2). For example, given a forward operator $G$ of algebraic/ differential equation type:

\begin{equation}
G(\hat{x}, \hat{y}) = 0, \quad \hat{x} \in \mathbb{R}^S, \quad \hat{y} \in \mathbb{R}^O,
\end{equation}

a physics-informed regularization term can be defined as

\begin{equation}
\Phi_p = f \left[ G(\hat{x}_1, N(\hat{x}_1, \theta)), G(\hat{x}_2, N(\hat{x}_2, \theta)), \ldots G(\hat{x}_r, N(\hat{x}_r, \theta)) \right] \in \mathbb{R},
\end{equation}

where $N$ represents the output of the ResNet in (2.1) with parameters $\theta$, $\{\hat{x}_1 \ldots \hat{x}_r\}$ is the set of collocation points, and $f$ is a user-defined loss.

3. In addition, we also employ the $L_1$ regularization (sparsity promoting regularization) denoted as $\Phi_s(\theta)$ to promote learning only the important weights/biases in a layer.

Putting all the regularizations together, our layerwise training process is shown in Algorithm 2.1. Algorithm 2.1 starts with training (minimizing the loss function (2.5)) a two-hidden layer network where the first layer is the (optional) upsampling/downsampling layer. Once, this network is trained, a new layer is added with the weights and biases initialized as zero (line 6 in Algorithm 2.1). Keeping parameters in the previous layers fixed, the newly added layer is trained (minimizing the cost function (2.6)). The procedure is repeated until a termination criteria is satisfied (line 4 in Algorithm 2.1). Other details are provided in Algorithm 2.1.

2.2. Sequential residual learning strategy (Algorithm 2.2). As we shall show (see Corollary 3.16), even though Algorithm 2.1 proposed in this work prevents overfitting in early layers, training saturates after a certain critical layer. To overcome this issue, we add a post-processing stage for Algorithm 2.1 in order to further decrease the training loss without overfitting the data. This stage is named the “sequential residual learning strategy” where one trains a sequence of neural networks to extract information from the residual resulting from Algorithm 2.1. The procedure is shown in Algorithm 2.2. Algorithm 2.2 starts by generating new training labels (residuals) using a pre-trained network $N$ as demonstrated in line 4 of Algorithm 2.2. A shallow network $Q_2$ is then trained for a limited number of epochs as shown in line 6 of Algorithm 2.2. Note that this is essential for preventing overfitting on the residuals and promoting $\delta$—stability as will be discussed later in subsection 3.2. The procedure is then repeated until a termination criteria is satisfied (line 2 in Algorithm 2.2).

3. An analysis of Algorithm 2.1 and Algorithm 2.2. In this section, we introduce key concepts—$\delta$—stability and approximate $\delta$—robustness—that drive the development of our approach. In particular, we prove how the design of Algorithm 2.1 promotes $\delta$—stability for each hidden layer (subsection 3.1.1) while also analyzing training saturation problems that might arise from certain hyperparameter settings (subsection 3.1.2). Lastly, subsection 3.2.1 analyzes the role of Algorithm 2.2 in promoting robustness.

3.1. Layerwise training Algorithm 2.1. In Algorithm 2.1, the input features are transformed through layers with previously trained weights and biases to create a new learning
Algorithm 2.1 Layerwise training Algorithm

**Input:** Training data \( X \), labels \( C \), validation data \( X_1 \), validation labels \( C_1 \), loss function \( \Phi_d \), loss tolerance \( \varepsilon_\eta \) for addition of layers, node tolerance \( \rho \) for thresholding of nodes, initial regularization parameters \((\alpha, \tau, \gamma)\), similarity matrix \( \beta_{ij} \), the number of neurons in each hidden layer \( N_o \).

**Initialize:** \( U, b^{(1)}, W^{(2)}, b^{(2)}, W_{\text{pred}}, b_{\text{pred}} \)

1: set \( i = 2, \alpha^{(2)} = \alpha, \tau^{(2)} = \tau, \gamma^{(2)} = \gamma \).

2: Minimize the loss function (2.5):

\[
\min_{W^{(1)}, b^{(1)}; W^{(2)}, b^{(2)}; W_{\text{pred}}, b_{\text{pred}}} \Phi_d(C, h_{\text{pred}}(W_{\text{pred}}R^{(2)}(Y^{(2)}) + b_{\text{pred}})) + \alpha^{(2)} \Phi_s(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})
\]

\[
+ \tau^{(2)} \Phi_p(h_{\text{pred}}(W_{\text{pred}}R^{(2)}(Y^{(2)}) + b_{\text{pred}})) + \gamma^{(2)} \Phi_m(Y^{(2)}),
\]

subject to
\[
Y^{(2)} = R^{(1)}(Y^{(1)}) + h^{(2)}(W^{(2)}R^{(1)}(Y^{(1)}) + b^{(2)}),
\]

\[
Y^{(1)} = h^{(1)}(W^{(1)}X + b^{(1)}).
\]

3: Define: \( \eta^{(2)} \) = data loss after minimizing (2.5), \( \varepsilon_v^{(2)} \) = val. loss, set \( \eta^{(1)} \approx 0, \varepsilon_v^{(1)} > \varepsilon_v^{(2)} \).

4: while \( \left| \frac{\eta^{(i-1)} - \eta^{(i)}}{\eta^{(i-1)}} \right| > \varepsilon_\eta \) and \( \varepsilon_v^{(i)} < \varepsilon_v^{(i-1)} \) do

5: \( i = i + 1 \)

6: Update \((\alpha^{(i)}, \tau^{(i)}, \gamma^{(i)})\) based on the previous values \((\alpha^{(i-1)}, \tau^{(i-1)}, \gamma^{(i-1)})\).

7: Extend the neural network by one layer with weights \( W^{(i)} \) and \( b^{(i)} \) initialized as zero.

8: Initialize \( W_{\text{pred}} \) and \( b_{\text{pred}} \) with values inherited from the previously trained network.

9: Freeze the weights \( W^{(i)}, b^{(i)}, \{W^{(l)}\}_{l=1}^{i-1}, \{b^{(l)}\}_{l=1}^{i-1} \).

10: Minimize the loss function (2.6):

\[
\min_{W^{(i)}, b^{(i)}; W_{\text{pred}}, b_{\text{pred}}} \Phi_d(C, h_{\text{pred}}(W_{\text{pred}}R^{(i)}(Y^{(i)}) + b_{\text{pred}})) + \alpha^{(i)} \times \Phi_s(W^{(i)}, b^{(i)})
\]

\[
+ \tau^{(i)} \times \Phi_p(h_{\text{pred}}(W_{\text{pred}}R^{(i)}(Y^{(i)}) + b_{\text{pred}})) + \gamma^{(i)} \times \Phi_m(Y^{(i)}),
\]

subject to
\[
Y^{(l+1)} = R^{(l)}(Y^{(l)}) + h^{(l+1)}(W^{(l+1)}R^{(l)}(Y^{(l)}) + b^{(l+1)}), \quad l = 1, \ldots, i - 1
\]

\[
Y^{(1)} = h^{(1)}(W^{(1)}X + b^{(1)}).
\]

11: Threshold weights, biases: if \( |w_{jk}| < \rho \) then set \( w_{jk} = 0 \); if \( |b_j| < \rho \) then set \( b_j = 0 \).

12: Store the training loss \( \eta^{(i)} \) and corresponding best validation loss \( \varepsilon_v^{(i)} \).

13: end while
**Algorithm 2.2** Sequential residual learning

**Input**: Training data $\mathbf{X}$, labels $\mathbf{C}$, validation data $\mathbf{X}_1$, validation labels $\mathbf{C}_1$, trained network $\mathcal{N}$ from Algorithm 2.1 and the corresponding validation loss $\epsilon_v^{(1)}$, loss function $\Phi_d$, training epoch $E_e$.

**Initialize**: Maximum number of networks: $N_n$, initialize each network: $Q_r$ with $\leq 2$ hidden layers.

1: set $r = 1$, set $\epsilon_v^{(0)} > \epsilon_v^{(1)}$, $Q_1 = \mathcal{N}$
2: while $r \leq N_n$ and $[\epsilon_v^{(r)} < \epsilon_v^{(r-1)}]$ do
3: Load trained network $Q_r$
4: $\mathbf{C} = \mathbf{C} - Q_r(\mathbf{X})$
5: Initialize network $Q_{r+1}$ with output layer parameters as 0.
6: Train network $Q_{r+1}$ for $E_e$ epochs with data $\mathbf{X}$, labels $\mathbf{C}$ and loss function $\Phi_d$.
7: Restore the best network $Q_{r+1}$, and validation loss $\epsilon_v^{(r+1)}$.
8: $r = r + 1$
9: end while
10: Compute the net output as: $Y = \mathcal{N}(\mathbf{X}) + Q(\mathbf{X})$, where $Q(\mathbf{X}) = Q_2(\mathbf{X}) + \ldots Q_{r-1}(\mathbf{X})$.

A key aspect of the algorithm is to learn transfer maps $\mathcal{N}^{(i)}$ that allows for effective information transfer through the network. In this work, we focus on creating transfer maps $\mathcal{N}^{(i)}$ that are...
δ− stable for a given input \( x \) through the use of manifold regularization [24, 7, 58]. A schematic of Algorithm 2.1 is given in Figure 1.

\[
Y(i) = (i - 1) \text{ (ResNet Block)} \\
Y(i) = i \text{ Output layer} \\
Y(i+1) = (i + 1) \text{ (ResNet Block)} \\
\]

(a) Trained Hidden Layer, \( L = i \)  
(b) Training Hidden Layer, \( L = (i + 1) \)

**Figure 1. Schematic of Algorithm 2.1: Training the \((i + 1)\)th hidden layer.**

### 3.1.1. δ-stability in layerwise training Algorithm 2.1.

In this section, we investigate the relevance of manifold regularization in our framework. The main motivation of manifold regularization is to promote \( \delta \)-stability. Here, stability means if two data points are “similar” to each other in some sense, then the network predictions on the two data points must be close to each other. Thus, stability in this sense is closely related to continuity. We now provide the details.

**Definition 3.2 (\( \varepsilon - \delta \) stability promoting algorithm).** Let \( \mathcal{X} = \bigcup_{j=1}^{K} \mathcal{X}_j \) denote the input space, where each \( \mathcal{X}_j \) contains similar data points based on some assumed similarity measure. Consider a parameterized function \( f_\theta : \mathcal{X} \subset \mathbb{R}^T \rightarrow Y \) and a learning algorithm to find an “optimal” parameter \( \theta^* \). Let \( \zeta \) be a stability parameter characteristic of the learning algorithm and let \( \varepsilon \) be a given constant. We call a learning algorithm to be \( \varepsilon - \delta \) stability promoting algorithm for \( f_\theta \) if there exists a stability function \( \delta_j(\zeta) : [\zeta^l, \zeta^u] \rightarrow \mathbb{R} \), such that:

1. \( \delta_j(\zeta^u) = \varepsilon, \delta_j(\zeta^u) < \delta_j(\zeta^l) \),
2. \( \delta_j(\zeta) \) is continuous with respect to \( \zeta \),
3. For each cluster \( \mathcal{X}_j \), there exists \( \theta^* = \theta^*(\zeta) \) such that:

\[
\|f_{\theta^*}(x) - f_{\theta^*}(x')\|_2 \leq \delta_j(\zeta), \quad \forall x, x' \in \mathcal{X}_j.
\]

**Remark 3.3.** Conditions 1, and 2 implies that the algorithm can promote any desired stability \( \delta_{\text{desired}} \in (\varepsilon, \delta_j(\zeta^l)) \) by varying parameter \( \zeta \), i.e \( \exists \zeta^* \in (\zeta^l, \zeta^u) \) such that \( \delta_j(\zeta^*) = \delta_{\text{desired}} \) (Intermediate value theorem). Note that \( \varepsilon \) is the best stability that can be achieved. However,
if the domain of $\delta_j(\zeta)$ is discrete, i.e., the hyperparameter $\zeta$ of the learning algorithm can only take discrete values in $[\zeta^1, \zeta^n]$, then we consider a relaxation of condition 2 in Definition 3.2. To that end we define “discrete $\varepsilon - \delta$ stability promoting algorithm” in Definition 3.4.

**Definition 3.4 (Discrete $\varepsilon - \delta$ stability promoting algorithm).** Consider Definition 3.2 and assume that the parameter $\zeta$ takes only discrete values, i.e $\zeta \in \{\zeta^1, \zeta_2, \ldots, \zeta_{n-1}, \zeta_n = \zeta^n\}$ and let the discrete domain of $\delta_j(\zeta)$ be denoted as $D$. Then, we call a learning algorithm to be discrete $\varepsilon - \delta$ stability promoting algorithm for $f_{\theta}$ if $\delta_j(\zeta) : D \rightarrow \mathbb{R}$ satisfies:
1. $\delta_j(\zeta^u) = \varepsilon$, $\delta_j(\zeta^u) < \delta_j(\zeta^1)$,
2. $\delta_j(\zeta)$ is monotonically decreasing with respect to $\zeta$,
3. For each cluster $\mathcal{X}_j$, there exists $\theta^* = \theta^*(\zeta)$ such that:

$$
\|f_{\theta^*}(x) - f_{\theta^*}(x')\|_2 \leq \delta_j(\zeta), \quad \forall x, x' \in \mathcal{X}_j.
$$

**Remark 3.5.** Conditions 3 in Definition 3.4 implies that the learning algorithm can continuously promote/improve the stability in a discrete manner by increasing $\zeta$. However, the notion of discrete stability promoting algorithm in Definition 3.4 is weaker than the one in Definition 3.2 since it promotes only desired stability $\delta_{\text{desired}} \in \{\delta_j(\zeta_1), \delta_j(\zeta_2), \ldots, \delta_j(\zeta_{n-1}), \delta_j(\zeta_n)\}$.

**Definition 3.6 ($\delta-$stable function).** Consider the parameterized function $f_{\theta}$ and an $\varepsilon - \delta$ stability promoting algorithm as defined in Definition 3.2 or Definition 3.4. Let $\zeta^*$ be the optimal stability parameter chosen based on a predefined criteria. Then, we call the function $f_{\theta^*}$ to be $\delta_j(\zeta^*)-$stable on $\mathcal{X}_j$ (or in short $\delta-$stable) if it satisfies (3.3).

**Remark 3.7.** In this work, the optimal $\zeta^*$ is defined as the one that minimizes the validation loss. That is, one looks at the performance on a validation data-set (exploring the balance between over-stabilization and under-stabilization) to pick $\zeta^*$. A numerical demonstration in the context of choosing the best manifold regularization parameter is provided in subsection 4.1, Figure 2 (middle figure). Here, one plots the ($\zeta$, validation loss) curve and chooses $\zeta^*$ that gives the lowest validation loss. More details are provided in subsection 4.1.

For learning a $\delta-$stable function in the sense of Definition 3.6, we first need to create an $\varepsilon - \delta$ stability promoting algorithm through the use of manifold regularization. For analysis purpose, let us first consider a re-normalization of the loss (2.6) as follows:

$$
\mathcal{L}_r = \left(\gamma^u - \gamma^{(i)}\right) \times \Phi_d + \left(\gamma^u - \gamma^{(i)}\right) \alpha^{(i)} \times \Phi_s + \left(\gamma^u - \gamma^{(i)}\right) \tau^{(i)} \times \Phi_p + \gamma^{(i)} \times \Phi_m,
$$

where $\gamma^u$ is a predefined upper bound such that when $\gamma^{(i)} = \gamma^u$, $\Phi_m$ (manifold regularization) is the sole component in the total loss $\mathcal{L}_r$, and when $\gamma^{(i)} = 0$, the manifold regularization is absent. Now, we are in the position to show the existence of a stability function when employing manifold regularization. To that end we will first prove the following lemma.

**Lemma 3.8 (Set of minimizers $\theta^* (\gamma^{(i)})$ is upper hemicontinuous with respect to $\gamma^{(i)}$).** Consider the loss (3.5) associated with training the layer $\mathcal{N}^{(i)}$. For a given $\{\alpha^{(i)}, \tau^{(i)}\}$ we define the loss function (3.5) as $\Phi (\theta, \gamma^{(i)})$, where $\theta$ represents the weights and biases to be optimized.
Suppose the set of global minimizers, $S(\gamma^{(i)}) = \{\theta^* : \theta^* = \arg \min_{\theta} \Phi(\theta, \gamma^{(i)})\}$ corresponding to the loss function (2.6) is non-empty and commonly bounded\(^1\) for all $\gamma^{(i)} \in [0, \gamma^u]$, where $\gamma^u$ is a predefined upper bound. Then, $S(\gamma^{(i)})$ is upper hemicontinuous in $\gamma^{(i)}$.

**Proof.** We shall use the Berge maximum theorem (refer to Chapter 17 in [8]) to show that the set of minimizers $S(\gamma^{(i)})$ is upper hemicontinuous with respect to $\gamma^{(i)}$. To that end the following observations can be made:

1. $\Phi(\theta, \gamma^{(i)})$ is continuous with respect to both $\theta$ and $\gamma^{(i)}$, since each component of the total loss (3.5) is continuous.
2. Let $\mathbb{R}^c$ be the Euclidean space with $c$ being the total number of parameters and $\mathcal{P}(\mathbb{R}^s)$ be the set of all subsets of $\mathbb{R}^s$. There exists a compact-valued correspondence $C : [0, \gamma^u] \rightarrow \mathcal{P}(\mathbb{R}^s)$ with $C(\gamma^{(i)}) \neq \emptyset$, $\forall \gamma^{(i)} \in [0, \gamma^u]$ such that:

$$
\theta^*(\gamma^{(i)}) = \arg \min_{\theta \in C(\gamma^{(i)})} \Phi(\theta, \gamma^{(i)}) = \arg \min_{\theta \in \mathbb{R}^c} \Phi(\theta, \gamma^{(i)}).
$$

Indeed, since the set of global minimizers is non-empty $\forall \gamma^{(i)} \in [0, \gamma^u]$, the existence of a compact-valued correspondence is straightforward to verify by defining the constraint set to be the same for each $\gamma^{(i)} \in [0, \gamma^u]$, i.e $C(\gamma^{(i)}) = \bigcup_{\gamma^{(i)} \in [0, \gamma^u]} S(\gamma^{(j)})$, where $\overline{\{\}}$ denotes the closure of the set. Further, $C(\gamma^{(i)})$ is closed and bounded, and hence compact, due to the assumption that the set of global minimizers is commonly bounded $\forall \gamma^{(i)} \in [0, \gamma^u]$.

3. Since the compact constraint set $C(\gamma^{(i)})$ defined above is constant independent of $\gamma^{(i)}$, it is continuous (refer to Chapter 17 in [8]).

Thus, by Berge maximum theorem the correspondence $\theta^*(\gamma^{(i)})$ is upper hemicontinuous with respect to $\gamma^{(i)}$.

**Proposition 3.9** ($\varepsilon - \delta$ stability promoting algorithm via manifold regularization). Consider the input space and neural transfer map introduced in Definition 3.1. Further assume that:

1. The samples in cluster $C_p$\(^2\) are independent draws from $\mu_p(0)$ with support $\mathcal{M}_p(0) \subset \mathcal{M}(0)$ where $\mathcal{M}_p(0)$ is compact $\forall p$. Let $\mu_p(i) = (\mathcal{R}_p^i \mathcal{N}(i)) \# \mu_p^{(i-1)}$ be the push forward measure associated with the map (3.2) for $i = 1, \ldots$.
2. The linear transformation $\mathcal{R}^{(i)}$ in (3.2) recales the features to $[-M, 0]$, where $M$ is a positive number and the activation $h^{(i)}$ is $\in \{\text{ELU, ReLU, LeakyReLU, SELU}\}$ for $i = 2, \ldots L$.
3. The set $S(\gamma^{(i)})$ in lemma 3.8 is singleton $\forall \gamma^{(i)} \in [0, \gamma^u]$. That is, the global minimizer is unique $\forall \gamma^{(i)} \in [0, \gamma^u]$ and for any choice of $\beta_p$ in (2.3).

Then, employing manifold regularization $\Phi_m$ of the form (2.4) with $\beta_p = 1$, $\forall p$ yields a probabilistic $\varepsilon - \delta$ stability promoting algorithm for $\mathcal{N}^{(i+1)}$. In particular, there exists a stability function $\tilde{\Phi}_p(\gamma^{(i)}, \xi^{(i)}) : [0, \gamma^u] \times [0, 1) \rightarrow \mathbb{R}$, that satisfies conditions 1 and 2 for every given

\(^1\)Commonly bounded means that $\exists M < \infty$, independent of $\gamma^{(i)}$, such that $\sup_{\gamma^{(i)} \in [0, \gamma^u]} |\theta^*(\gamma^{(i)})| \leq M$.

\(^2\)Note that we do not assume that labels are available for all the inputs.
\(\xi(1)\), such that \(\forall x, x' \in M_p^{(i-1)}\):

\[
\|\mathcal{N}^{(i)}(x) - \mathcal{N}^{(i)}(x')\|_2^2 \leq \delta_{p, i}(\gamma(1), \xi(1)), \quad i = 2, \ldots, L,
\]

holds with probability at-least \(1 - \xi(1) - \epsilon(1)(m_p, \xi(1), \epsilon)\), where \(M_p^{(i-1)}\) is defined in \((3.2)\), \(\gamma(1)\) is the weight associated with the manifold regularization term in \((2.6)\), \(\epsilon(1)(m_p, \epsilon) = 2m_p \exp\left(-\omega(1)p m_p \epsilon^2\right) + 2 \exp\left(-\psi(1)p m_p \epsilon^2\right)\), and \(\omega(1), \psi(1)\) are positive constants that depends only on the cluster \(C_p\) and the layer \(i\).

**Proof.** The proof follows directly from analyzing the effect of manifold regularization applied to each neural transfer map. Consider the trained neural transfer map \(\mathcal{N}^{(2)}(x)\) which is the hidden layer of the ResNet architecture after the upsampling/downsampling layer. From \((2.4)\) with \(\beta_p = 1, \forall p\), a sample average approximation of the manifold regularization term \(\Phi_m\) can be written as:

\[
\sum_{p=1}^{K} \frac{1}{m_p^2} \sum_{x_i, x_j \in C_p} \left\|\mathcal{N}^{(2)}(x_i) - \mathcal{N}^{(2)}(x_j)\right\|_2^2,
\]

where, \(m_p\) denotes the number of data points in cluster \(C_p\), and \(C_1, C_2 \ldots C_k\) denotes the data clusters. Now for each cluster \(C_p\), one can show the following non-asymptotic bound (proof is provided in Appendix A):

\[
\mathbb{E}_{\mu(1)} \left[ \left\|\mathcal{N}^{(2)}(x) - \mathcal{N}^{(2)}(y)\right\|_2^2 \right] \leq \epsilon + \frac{1}{m_p^2} \sum_{x_i, x_j \in C_p} \left\|\mathcal{N}^{(2)}(x_i) - \mathcal{N}^{(2)}(x_j)\right\|_2^2 \geq \epsilon(2)(m_p, \epsilon),
\]

\[
\mathbb{E}_{\mu(1)} \left[ \left\|\mathcal{N}^{(2)}(x) - \mathcal{N}^{(2)}(y)\right\|_2^2 \right] = \int_{\mathcal{M}_p^{(1)}} \int_{\mathcal{M}_p^{(1)}} \left\|\mathcal{N}^{(2)}(x) - \mathcal{N}^{(2)}(y)\right\|_2^2 \, d\mu(1)(x) \, d\mu(1)(y).
\]

where \(\epsilon(2)(m_p, \epsilon) = 1 - 2m_p \exp\left(-\omega(2)p m_p \epsilon^2\right) - 2 \exp\left(-\psi(2)p m_p \epsilon^2\right)\), \(\omega(2), \psi(2)\) are positive constants that depends only on the cluster \(C_p\) and the hidden layer, and \(\mathbb{E}_{\mu_p}\) is the expectation defined in \((3.9)\). Now, note that the random variable \(\left\|\mathcal{N}^{(2)}(x) - \mathcal{N}^{(2)}(y)\right\|_2^2\) in \((3.9)\) is non-negative. Therefore, by Markov inequality [52] we have:

\[
\left\|\mathcal{N}^{(2)}(x) - \mathcal{N}^{(2)}(x')\right\|_2^2 \leq \frac{\mathbb{E}_{\mu(1)} \left[ \left\|\mathcal{N}^{(2)}(x) - \mathcal{N}^{(2)}(y)\right\|_2^2 \right]}{\xi(2)},
\]

holds \(\forall x, x' \in M_p^{(1)}\) with probability at-least \((1 - \xi(2))\). Now using the non-asymptotic bound \((3.8)\) in \((3.10)\) and applying the union bound, we have:

\[
\left\|\mathcal{N}^{(2)}(x) - \mathcal{N}^{(2)}(x')\right\|_2^2 \leq \frac{1}{\xi(2) m_p^2} \sum_{x_i, x_j \in C_p} \left\|\mathcal{N}^{(2)}(x_i) - \mathcal{N}^{(2)}(x_j)\right\|_2^2 + \epsilon,
\]
holds $\forall \mathbf{x}, \mathbf{x}'$ on $\mathcal{M}_p^{(1)}$ with probability at-least $(1 - \xi^{(2)} - \mathcal{E}_p^{(2)}(m_p, \xi^{(2)}\varepsilon))$. Therefore, from (3.11) the stability function $\hat{\delta}_p(\gamma^{(2)}, \xi^{(2)})$ is defined as follows:

\[
\hat{\delta}_p(\gamma^{(2)}, \xi^{(2)}) = \frac{1}{\xi^{(2)} \times m_p^2} \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_p} \left\| \mathcal{N}^2_{\mathbf{x}^{(2)}}(\mathbf{x}_i) - \mathcal{N}^2_{\mathbf{x}^{(2)}}(\mathbf{x}_j) \right\|_2^2 + \varepsilon,
\]

where we have introduced the subscript $\theta^*(\gamma^{(2)})$ to denote the optimal weights and biases of the trained layer $\mathcal{N}^{(2)}$ and $\xi^{(2)} \in (0, 1)$. We will first prove that (3.12) is a valid stability function that satisfies condition 1 and condition 2 in Definition 3.2 for every choice of $\xi^{(2)}$. Condition 1 is satisfied by setting $\gamma^{(2)} = \gamma^u$ in (3.5), since manifold regularization is the sole component in the total loss and consequently (3.7) is minimized. The minimizers in this case can be identified by noticing that:

\[
\left\| \mathcal{N}^2_{\theta^*(\gamma^{(2)})}(\mathbf{x}_i) - \mathcal{N}^2_{\theta^*(\gamma^{(2)})}(\mathbf{x}_j) \right\|_2^2 = \left\| \mathbf{x}_i - \mathbf{x}_j + h^2(\mathbf{W}^{(2)*}\mathbf{x}_i + \mathbf{b}^{(2)*}) - h^2(\mathbf{W}^{(2)*}\mathbf{x}_j + \mathbf{b}^{(2)*}) \right\|_2^2,
\]

which follows from (3.1). Based on assumption 2, we immediately notice that the minimizers are $\mathbf{b}^{(2)*} = 0$ and $\mathbf{W}^{(2)*} = (-\lambda) \times I$, where $I$ is the identity matrix and $\lambda$ is a constant depending on the activation function employed. In this case, the minimizers $\mathbf{b}^{(2)*}$, $\mathbf{W}^{(2)*}$ leads to the network learning a zero function and $\hat{\delta}_p(\gamma^u, \xi^{(2)}) = \varepsilon$, $\forall p$ in (3.12). Now since the minimizers $\mathbf{b}^{(2)*}$, $\mathbf{W}^{(2)*}$ is unique for any choice of similarity matrix $\mathbf{B}$ in (2.3) (assumption 3), we also note that $\hat{\delta}_p(\gamma^u, \xi^{(2)}) = \varepsilon$ for any $p$ if and only if the network learns a zero function. Therefore, by contraposition it follows that $\forall p$, $\hat{\delta}_p(0, \xi^{(2)}) > \varepsilon$ since in the absence of manifold regularization term (i.e $\gamma^{(2)} = 0$), it is necessary that the hidden layers learns a non-zero function to fit the training data. In conclusion, we have proved the following properties for $\hat{\delta}_p(\gamma^{(2)}, \xi^{(2)})$:

\[
\hat{\delta}_p(\gamma^u, \xi^{(2)}) = \varepsilon, \quad \hat{\delta}_p(0, \xi^{(2)}) > \varepsilon, \quad \forall p.
\]

Now, in order to prove condition 2 of Definition 3.2, it is sufficient to show that $\hat{\delta}_p(\gamma^{(2)}, \xi^{(2)})$ is continuous with respect to $\gamma^{(2)}$. Note that, $\forall \gamma^{(2)} \in [0, \gamma^u]$ we have:

\[
\lim_{r \to 0} \hat{\delta}_p(r + \gamma^{(2)}, \xi^{(2)}) = \lim_{r \to 0} \left[ \frac{1}{\xi^{(2)} \times m_p^2} \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_p} \left\| \mathcal{N}^2_{\theta^*(r + \gamma^{(2)})}(\mathbf{x}_i) - \mathcal{N}^2_{\theta^*(r + \gamma^{(2)})}(\mathbf{x}_j) \right\|_2^2 \right] + \varepsilon
\]

(3.15a)

\[
= \frac{1}{\xi^{(2)} \times m_p^2} \left[ \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_p} \lim_{r \to 0} \left\| \mathcal{N}^2_{\theta^*(r + \gamma^{(2)})}(\mathbf{x}_i) - \mathcal{N}^2_{\theta^*(r + \gamma^{(2)})}(\mathbf{x}_j) \right\|_2^2 \right] + \varepsilon
\]

(3.15b)

\[
= \frac{1}{\xi^{(2)} \times m_p^2} \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_p} \left\| \mathcal{N}^2_{\theta^*(\gamma^{(2)})}(\mathbf{x}_i) - \mathcal{N}^2_{\theta^*(\gamma^{(2)})}(\mathbf{x}_j) \right\|_2^2 + \varepsilon
\]

(3.15c)

\[
= \hat{\delta}_p(\gamma^{(2)}, \xi^{(2)})
\]

(3.15d)
where we have used the fact that $\theta^*(\gamma^{(2)})$ is a continuous function in (3.15c) based on assumption 3 and using the results of lemma 3.8. Therefore, condition 2 of Definition 3.2 is satisfied. Now, applying the above arguments for $\delta$–stability recursively for subsequent neural transfer maps $\mathcal{N}^{(i)}$, it follows that property (3.6) holds for any subsequent neural transfer maps $\mathcal{N}^{(i)}$, thereby concluding the proof.

Remark 3.10. In order to reduce the number of hyperparameters, we consider a fixed growth/decay rate for the manifold regularization weight, i.e. $\gamma^{(i+1)} = \kappa \gamma^{(i)}$ in (3.5). When $\kappa \in (0, 1)$, more emphasis is placed on minimizing the data loss $\Phi_d$ in the later layers while the initial layers focus on stability. This strategy is suitable for regression/classification task with sufficient number of data points as demonstrated in subsection 4.1 and subsection 4.4. However, when the data set is sparse we consider choosing a small value for $\gamma^{(2)}$ and $\kappa > 1$ such that the initial layers over-fits on the sparse data set and stability is imparted in the later layers. The feasibility of such a training procedure for sparse data sets is discussed in subsection 4.3.1. In either case, this leads to tuning only parameter $\gamma^{(2)}$ in our procedure.

Corollary 3.11. Assume that conditions in Proposition 3.9 are satisfied and each cluster contains infinitely many samples. Further, assume that the activation function employed is Lipschitz continuous and assume $\gamma^{(i+1)} = \kappa \gamma^{(i)}$ in (3.5), where $\kappa$ is a fixed growth/decay rate for the manifold regularization weight. Then, employing manifold regularization of the form (2.4) with $\beta_p = 1$, $\forall p$ yields a probabilistic $\varepsilon - \delta$ stability promoting algorithm for network $\mathcal{N}$ with $\varepsilon = 0$ in the sense of Definition 3.2. That is, there exists a stability function $\delta^N_p(\gamma, \xi^{(2)}, \ldots \xi^{(L)}) : [0, \gamma^u] \times (0, 1)^{L-1} \rightarrow \mathbb{R}$ that satisfies conditions 1 and 2 for any fixed $\xi^{(2)}, \ldots \xi^{(L)}$, such that in the limit $m_p \rightarrow \infty$, $\forall x, \ x' \in \mathcal{M}^{(0)}_p \subset \mathcal{M}^{(0)}$:

$$
(3.16) \quad \|\mathcal{N}(x) - \mathcal{N}(x')\|_2 \leq \delta^N_p(\gamma, \xi^{(2)}, \ldots \xi^{(L)}),
$$

holds with probability at-least $\left(1 - \sum_{i=2}^{L} \xi^{(i)}\right)$, where $\gamma = \gamma^{(2)}$ is the initial weight chosen for the manifold regularization term in the loss (3.5).

Proof. Proof is provided in Appendix B.

Remark 3.12. Note that for a given $\xi^{(2)}, \ldots \xi^{(L)}$, the stability function $\delta^N_p(\gamma, \xi^{(2)}, \ldots \xi^{(L)})$ can be made as small as one wishes by varying the parameter $\gamma$ (also see remark 3.3). A numerical demonstration of controlling the stability function by varying $\gamma$ is provided in Figure 7 of subsection 4.3. Further, by choosing the optimal stability parameter $\gamma^*$ in (3.16), we achieve a $\delta$–stable network $\mathcal{N}$ in the sense of Definition 3.6.

3 Layerwise training saturation problem. Even though we have ensured that Algorithm 2.1 promotes $\varepsilon - \delta$ stability, it is imperative to analyse any training saturation problem

3 Inequality (3.16) is an asymptotic property here. For more details, refer to the proof in Appendix B.
that might arise from certain hyperparameter settings. In this section, we consider the case when \( \kappa \in (0, 1) \) in Corollary 3.11 and derive the necessary conditions for trainability of a newly added layer (see Proposition 3.15). To that end, let us define the following:

**Definition 3.13. Layerwise training promoting property (LTP)**

A non-linear activation function \( h(\cdot) \) possesses the layerwise training promoting property (LTP) if \( h'(0) \neq 0 \).

**Definition 3.14. Trainability of an added layer**

A newly added \((i + 1)\)th layer with parameters \( W^{(i+1)} \) and \( b^{(i+1)} \) initialized as \( W_0 \) and \( b_0 \), respectively, is trainable with respect to the \( j \)th training pair \( \{Y^{(i)}_j, C_j\} \) if

\[
\begin{bmatrix}
\frac{\partial L_j}{\partial W^{(i+1)}} \\
\frac{\partial L_j}{\partial b^{(i+1)}}
\end{bmatrix}_{W^{(i+1)}=W_0, b^{(i+1)}=b_0} \neq 0,
\]

where \( L_j \) is the Lagrangian defined for the \( j \)th training sample in (C.3) in Appendix C.

**Proposition 3.15 (Necessary condition for trainability of newly added layer).** Consider training the \((i + 1)\)th hidden layer with the loss (2.6) and parameters \( W^{(i+1)} \) and \( b^{(i+1)} \) initialized as zero and assume that \( h^{(i+1)}(0) = 0 \) and \( \alpha^{(i)} = 0 \) (sparsity regularization). Further, assume that the \( j \)th hidden layer is trained to a local minimum with respect to the \( j \)th training sample such that \( (h^{(i)})' W^{(i)} \cdot R^{(i-1)} (Y^{(i-1)}_j + b^{(i)*}) \neq 0 \), where \( W^{(i)} \) and \( b^{(i)*} \) denote the optimal weights and biases of the \( i \)th layer and \( R^{(i)} \) follows assumption (2). Then, the necessary conditions for trainability of an added layer are:

1. \( \gamma^{(i+1)} \neq \gamma^{(i)} \) or \( \tau^{(i+1)} \neq \tau^{(i)} \);
2. Activation \( h^{(i+1)} \) should satisfy the LTP property.

**Proof.** The ResNet forward propagation (see (2.1)) for the newly added layer with weights and biases initialized as zero gives:

\[
Y^{(i+1)} = R^{(i)} (Y^{(i)}) + h^{(i+1)} (0 \times R^{(i)} (Y^{(i)}) + 0).
\]

Since \( h^{(i+1)}(0) = 0 \), we have \( Y^{(i+1)} = R^{(i)} (Y^{(i)}) \) on adding the new layer at the beginning of training and the network output \( h_{\text{pred}} (W_{\text{pred}} \cdot R^{(i+1)} (Y^{(i+1)}) + b_{\text{pred}}) \) remains unchanged since \( R^{(i+1)} R^{(i)} (Y^{(i)}) = R^{(i)} (Y^{(i)}) \), which follows from assumption (2). Now, using (3.17), the gradient for the newly added layer evaluated at \( 0 \) for a particular \( j \)th training sample can be written as (see Appendix C):

\[
\frac{\partial L_j}{\partial b^{(i+1)}} = -\left( R^{(i)} (W_{\text{pred}}^{*})^T \left[ h_{\text{pred}}' \left( W_{\text{pred}}^{*} \cdot R^{(i)} (Y^{(i)}_j) + b^{*}_{\text{pred}} \right) \circ \lambda^{(i+1)}_2 \right] - \frac{\partial \Phi^{(i+1)}_{m}}{\partial Y^{(i)}_j} \right) \circ (h^{(i+1)})'(0),
\]

and,

\[
\frac{\partial L_j}{\partial W^{(i+1)}} = \frac{\partial \Phi^{(i+1)}_{m}}{\partial Y^{(i)}_j} \left( R^{(i)} (Y^{(i)}_j) \right)^T,
\]
where $\mathcal{L}$ denotes the Lagrangian defined in Appendix C, (C.3), $W_{\text{pred}}^*$, $b_{\text{pred}}^*$ represents the optimal parameters for the output layer obtained after training the $i^{th}$ layer, and $\lambda_2^{(i+1)}$ is defined as:

$$\lambda_2^{(i+1)} = - \left( \frac{\partial \Phi_p}{\partial Y_j^{(o)}} + \frac{\partial \Phi_p^{(i+1)}}{\partial Y_j^{(o)}} \right).$$

where, $Y_j^{(o)}$ represents the output of the network. Therefore, it is clear from (3.18) that $(h_i^{(i+1)})' = 0$ (LTP property) is a necessary condition for trainability of the newly added layer. Further, since the previous layer, i.e the $r^{th}$ layer is trained to a local minima, then one also has the following condition (see Appendix C equation (C.6)):

$$\left( R^{(i)}(W_{\text{pred}}^*)^T \left[ h_i^{(i)}(W_{\text{pred}}^* \mathcal{R}^{(i)}(Y_j^{(i)}) + b_{\text{pred}}^*) \circ \lambda_2^{(i)} \right] - \frac{\partial \Phi_i^m}{\partial Y_j^{(i)}} \right) \circ z = 0,$n

where, $W(i)^*$, $b(i)^*$ denotes the trained parameters of the $i^{th}$ layer and the vector $z$ given by:

$$z = (h_i^{(i)})' \left( W(i)^* \mathcal{R}^{(i-1)}(Y_j^{(i-1)}) + b(i)^* \right) \neq 0,$n

by assumption in Proposition 3.15. Therefore, condition (3.21) translates to the following:

$$\frac{\partial \Phi_i^m}{\partial Y_j^{(i)}} = R^{(i)}(W_{\text{pred}}^*)^T \left[ h_i^{(i)}(W_{\text{pred}}^* \mathcal{R}^{(i)}(Y_j^{(i)}) + b_{\text{pred}}^*) \circ \lambda_2^{(i)} \right].$$

Further from (3.20), we have:

$$\lambda_2^{(i+1)} = \lambda_2^{(i)} + \frac{\partial \Phi_i}{\partial Y_j^{(o)}} - \frac{\partial \Phi_i^{(i+1)}}{\partial Y_j^{(o)}}.$n

In addition from (3.18), for trainability of the newly added layer it is necessary that:

$$R^{(i)}(W_{\text{pred}}^*)^T \left[ h_i^{(i)}(W_{\text{pred}}^* \mathcal{R}^{(i)}(Y_j^{(i)}) + b_{\text{pred}}^*) \circ \lambda_2^{(i+1)} \right] - \frac{\partial \Phi_i^{(i+1)}}{\partial Y_j^{(i)}} \neq 0.$n

The above condition can be simplified using (3.22), and (3.23) as:

$$\frac{\partial \Phi_i^m}{\partial Y_j^{(i)}} - \frac{\partial \Phi_i^{(i+1)}}{\partial Y_j^{(i)}} + R^{(i)}(W_{\text{pred}}^*)^T \left[ h_i^{(i)}(W_{\text{pred}}^* \mathcal{R}^{(i)}(Y_j^{(i)}) + b_{\text{pred}}^*) \circ \left( \frac{\partial \Phi_i}{\partial Y_j^{(o)}} - \frac{\partial \Phi_i^{(i+1)}}{\partial Y_j^{(o)}} \right) \right] \neq 0.$n

Using the (C.1) in Appendix C, the above condition translates to:

$$\left( \gamma^{(i)} - \gamma^{(i+1)} \right) \frac{\partial \Phi_i^m}{\partial Y_j^{(i)}} + \left( \tau^{(i)} - \tau^{(i+1)} \right) (W_{\text{pred}}^*)^T \left[ h_i^{(i)}(Y_j^{(i)} W_{\text{pred}}^* + b_{\text{pred}}^*) \circ \frac{\partial \Phi_i}{\partial Y_j^{(o)} \right] \neq 0.$$n

Thus, it is necessary that $\gamma^{(i+1)} \neq \gamma^{(i)}$ or $\tau^{(i+1)} \neq \tau^{(i)}$ for trainability.
Corollary 3.16 (Training saturation). Assume that the necessary conditions of Proposition 3.15 are satisfied, by choosing \( \gamma^{(i+1)} = \kappa_1 \gamma^{(i)} \) and \( \tau^{(i+1)} = \kappa_2 \tau^{(i)} \), where \( \kappa_1, \kappa_2 \in (0, 1) \). Then
\[
\lim_{i \to \infty} \frac{\partial L_j}{\partial b^{(i+1)}} = 0, \quad \text{and} \quad \lim_{i \to \infty} \frac{\partial L_j}{\partial W^{(i+1)}} = 0,
\]
for all \( j = 1, \ldots M \).

**Proof.** Since the \( i^{th} \) layer is supposed to be trained to a local minimum with respect to each training sample, we have:
\[
\left[ \frac{\partial L_j}{\partial b^{(i)}} \right]_{b^{(i)} = b^{(i)*}} = 0, \quad \left[ \frac{\partial L_j}{\partial W^{(i)}} \right]_{W^{(i)} = W^{(i)*}} = 0, \quad \forall j = 1, \ldots\]
where, \( W^{(i)*} \) and \( b^{(i)*} \) denote the optimal parameters of the \( i^{th} \) layer. Since \( \lim_{i \to \infty} |\gamma^{(i+1)} - \gamma^{(i)}| = 0, \lim_{i \to \infty} |\tau^{(i+1)} - \tau^{(i)}| = 0 \), it follows from (3.25) and (3.18) that:
\[
\left[ \frac{\partial L_j}{\partial W^{(i+1)}} \right]_{W^{(i+1)} = 0} \to 0, \quad \left[ \frac{\partial L_j}{\partial b^{(i+1)}} \right]_{b^{(i+1)} = 0} \to 0, \quad \forall j = 1, \ldots M.
\]
as \( i \) approaches \( \infty \).

Remark 3.17. The results of Corollary 3.16 implies that for sufficiently large \( i \), both \( \frac{\partial L_j}{\partial b^{(i+1)}} \) and \( \frac{\partial L_j}{\partial W^{(i+1)}} \) are small. In other words, we face the problem of vanishing gradient when adding new layer and the training saturates, i.e., either little or no decrease in the total loss given by (2.6) for sufficiently large \( i \).

Remark 3.18. Note that we have assumed \( h^{(i)}(0) = 0 \) in Proposition 3.15. Further, the LTP property is a necessary condition for trainability of an added layer. It is easy to see that activation function such as ‘ELU’ and ‘Tanh’ satisfy these conditions. In addition, activation functions such as \{ReLU[12], LeakyReLU, SELU\} can also be used in practice if one considers a non-zero subgradient at 0. In this work, the activation functions \( h^{(i)} \) in Algorithm 2.1 are selected to be ELU.

3.2. Sequential residual learning Algorithm 2.2. Recall that promoting \( \varepsilon - \delta \) stability (Corollary 3.11) does not guarantee that the network classifies/regress correctly since training saturates for certain hyperparameter settings in Algorithm 2.1 (see Corollary 3.16). To overcome this issue, we incorporate a post-processing stage termed as the “sequential residual learning” step for further improving the predictions. The idea of sequential residual learning is to train a sequence of shallow neural networks to learn the residual (i.e lower the data loss \( \Phi_d \) while generalizing well) from Algorithm 2.1 where each network \( \mathcal{Q}_i \) is trained for a limited epochs to prevent overfitting on the residuals. Algorithm 2.2 creates new training problem for each network \( \mathcal{Q}_{i+1} \) by generating data sets of the form:

\[
K_{II,i+1} = \{X, C_i\}, \quad C_i = C_{i-1} - \mathcal{Q}_i(X), \quad i = 1, 2, \ldots
\]

where, \( C_0 = C \) and \( \mathcal{Q}_1 = \mathcal{N} \) is the neural network produced by Algorithm 2.1. For a given test data point \( x \), the output \( f(x) \) of our algorithm is computed as:

\[
f(x) = \mathcal{N}(x) + \mathcal{Q}(x), \quad \text{where} \quad \mathcal{Q}(x) = \mathcal{Q}_2(x) + \ldots \mathcal{Q}_{r-1}(x).
\]
3.2.1. $\varepsilon - \delta$ stability and approximate $\delta$-robustness. Note that while manifold regularization allows us to promote $\varepsilon - \delta$ stability for the network $\mathcal{N}$ (final network from Algorithm 2.1) (see Corollary 3.11), it is imperative to have a mechanism that promotes $\varepsilon - \delta$ stability for $\mathcal{Q}$ given in (3.27) during the sequential adaptation process. Further, the main motivation of Algorithm 2.2 is to promote robustness. Robustness is the property that if a testing sample is “similar” to a training sample, then the testing error is close to the training error along with the condition that the algorithm fits the training data “well-enough” [57, 24]. If the prediction on the training sample is accurate, then the prediction on a similar testing sample will also be accurate. It is therefore immediately clear that learning a $\delta$-stable function (Definition 3.6) is necessary for robustness. Additionally, by using a sequence of networks to learn the residuals in Algorithm 2.2, one ensures that the algorithm fits the training data “well-enough”. Since in practice, one fits the training data points to at most $\iota$ tolerance (to prevent overfitting), we define approximate $\delta$-robustness as follows:

Definition 3.19 (Approximate $\delta$-robustness). The function $f(x) = \mathcal{N}(x) + \mathcal{Q}(x)$ in (3.27) is called approximately $\delta$-robust at an input $x \in \mathcal{M}_p^{(0)}$ if each of the networks $\mathcal{N}$ and $\mathcal{Q}$ are $\delta$-stable on $\mathcal{M}_p^{(0)}$ as per Definition 3.6 and $\|f(x) - c\|_2 \leq \iota$, where $c$ is the correct label for $x$ and, $\iota$ is a small positive constant chosen to prevent overfitting.

Note that we refrain from the use of any regularizers in Algorithm 2.2 due to the small magnitude of residuals $\mathcal{C}_i$ in (3.26) which inhibits learning in the presence of explicit regularizers. Therefore, one is naturally left with the question on how to design a $\varepsilon - \delta$ stability promoting algorithm for $\mathcal{Q}$ in Algorithm 2.2. Proposition 3.20 provides a practical implicit way of achieving this avoiding the use of any regularizers. In particular, proposition 3.20 explores how early stopping (a form of implicit regularization in machine learning where a network is trained for a limited number of iterations) can be used to design a $\varepsilon - \delta$ stability promoting algorithm as defined in Definition 3.4. In the below Proposition 3.20, the term “iterations” refer to the number of gradient updates made during the training process.

Proposition 3.20 ($\varepsilon - \delta$ stability promoting algorithm via early stopping). Consider the function $\mathcal{Q}(x)$ in Algorithm 2.2 with the input space given in Definition 3.1 and assume that the activation function employed is 1-Lipschitz continuous $^4$ for each network $\mathcal{Q}_i$. Further, assume that the input set $\mathcal{M}_p^{(0)}$ is compact $\forall p$. Then, employing early stopping criteria yields a $\varepsilon - \delta$ stability promoting algorithm for network $\mathcal{Q}$ in (3.27) with $\varepsilon = 0$. That is, there exists a stability function $\delta_p^Q(\zeta) : \left\{ \frac{1}{m^{p+1}}, \frac{1}{m^p}, \ldots, 1 \right\} \mapsto \mathbb{R}$ satisfying conditions in Definition 3.4 where $m^p \in \mathbb{Z}^+$ is some chosen upper bound on the training iterations. In particular, $\forall x, x' \in \mathcal{M}_p^{(0)} \subset \mathcal{M}^{(0)}$:

\begin{equation}
\|\mathcal{Q}(x) - \mathcal{Q}(x')\|_2 \leq \sum_{i=2}^{r-1} \ell^3 \times \left[ 3 \prod_{k=1}^{(1/\zeta) - 1} \left( \sum_{i=0}^1 \| (g_i^{(k)})_i \|_2 \right) \right] \times \varepsilon_p = \delta_p^Q(\zeta),
\end{equation}

where $\varepsilon_p = \sup_{x, x' \in \mathcal{M}_p^{(0)}} \|x - x'|_2$, $(g_i^{(k)})_i$ represents the gradients for $k^{th}$ layer weight matrix

\footnote{We call a function, $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$, 1-Lipschitz continuous if $\forall x, y \in \mathbb{R}^n$, $\|h(x) - h(y)\|_2 \leq \|x - y\|_2$.}
at the \( l \)th iteration of gradient descent for network \( Q_l \), \( \ell \) denotes a constant learning rate.

\textbf{Proof.} Proof is provided in Appendix D.

\textbf{Remark 3.21 (Approximate \( \delta \)–robustness by Algorithm 2.2).} Recall Definition 3.6 for \( \delta \)–stable function. By tuning parameter \( \gamma \) in (3.16), and parameter \( \zeta \) in (3.28), each network \( N \) and \( Q \) is \( \delta \)–stable. More information on tuning the parameters for \( \delta \)–stability are provided in Appendix E.2. Lastly, for an input \( x \) in the training data set, the residual \( \| f(x) - c \|_2 \leq \iota \) in Definition 3.19 is achieved after adding sufficient number of networks leading to approximate \( \delta \)–robustness.

\textbf{Remark 3.22 (Manifold regularization vs. early stopping).} Note that even though both manifold regularization and early stopping helps one design a \( \varepsilon \)–\( \delta \)–stability promoting algorithm, it is important to note the subtle difference between the stability functions \( \delta^N_p(\gamma, \xi^{(2)} \ldots \xi^{(L)}) \) in (3.16) and \( \delta^Q_p(\zeta) \) in (3.28). While \( \delta^N_p(\gamma, \xi^{(2)} \ldots \xi^{(L)}) \) explicitly depends on the geometry (similarity) of data points, the stability function \( \delta^Q_p(\zeta) \) is a discrete function that does not directly depend on the geometry of the dataset. In our numerical examples (see results in subsection 4.1), we demonstrate that the \( \delta \)–stability enforced by manifold regularization in Algorithm 2.1 is indispensable for achieving better generalization performance in comparison to a baseline trained with a pure early stopping criteria.

4. Numerical Experiments. In this section, we numerically demonstrate the proposed approach on both simulated data sets and real world data sets as follows:

1. Regression task: Boston house price prediction problem.
2. Physics informed adaptive neural network (PIANN).
3. Adaptive learning of inverse maps from sparse data.
4. Image classification problem: MNIST data set.

General experimental settings for all the problems and descriptions of methods adopted for comparison are described in Appendix E. We will show in our numerical experiments that the use of Algorithm 2.2 is inevitable for problems 1 and 4 to deal with the training saturation problem as described in Corollary 3.16. However, the nature of problems 2 and 3 allows us to devise a different hyperparameter settings that evades the use of Algorithm 2.2 (by mitigating the training saturation problem).

4.1. Regression task: Boston house price prediction problem. This regression problem aims to predict the housing prices in Boston and is a common prototype problem for testing ML regression algorithms [9, 19, 46]. The input setting for Algorithm 2.1 and Algorithm 2.2 as well as other details are provided in Appendix F and Appendix J.

The layerwise training curve is shown in Figure 2 which shows that, subsequent layers are able to learn on top of the representations of previous layers. The percentage of active and inactive parameters (consequence of sparsity regularization) in each layer is shown in Figure 2. From Figure 2 it is clear that initial layers matter more in a deep neural network requiring more number of parameters [41]. We also note from Figure 2 that Algorithm 2.1 is only capable of reducing the training loss to 24.2 and training saturates (negligible decrease in the loss) as predicted in Corollary 3.16.

\(^5\left( g_0^{(4)} \right)_i\) represents the initial weights.
Further, Figure 2 (middle figure) shows how to choose the optimal stability parameter (manifold regularization parameter $\gamma$) for $\delta-$stability as defined in Definition 3.6. The optimal parameter ($\gamma^* = 0.06$) is the one that gives the lowest validation loss. From Figure 2 (middle figure), it is evident that in the absence of manifold regularization (i.e $\gamma = 0$), stability is not promoted and consequently the final validation loss achieved by Algorithm 2.1 is considerably higher in this case. Note that even though we do not have infinitely many training samples as dictated by Proposition 3.9, the effect of manifold regularization is quite pronounced in this case. In order to further decrease the loss (promote approximate $\delta-$robustness), we resort to Algorithm 2.2. The inputs for Algorithm 2.2 are provided in Appendix J.

| Method          | Test loss | Params. trained simultaneously | Total params. |
|-----------------|-----------|---------------------------------|---------------|
| Proposed        | 7.11      | 11,601                          | 48,474        |
| Baseline        | 11.2      | 82,301                          | 82,301        |
| Baseline + Algo. 2.2 | 10.5      | 82,301                          | 82,905        |
| Forward-Thinking [22] | 28.0      | 11,601                          | 82,301        |
| Approach [22] + Algo. 2.2 | 11.5      | 11,601                          | 82,905        |

**Figure 3.** Boston house price prediction problem. Comparison between the proposed approach and other methods. As can be seen, the proposed two-stage approach is the most accurate (by a large margin) with the least number of network parameters.

The adaptation results and performance comparison with other methods (the baseline and the approach from [22]) are summarized in Figure 3. From Figure 3, it is clear that our proposed approach outperformed all the other methods by a good margin. The number of trained parameters in each step of Algorithm 2.2 and from other methods is provided as a Table in Figure 3. As can be seen, the proposed two-stage approach is the most accurate (by a large margin) with the least number of network parameters. Further, a random architec-
ture search based on the approach in [30] produced a best test loss of 9.28 which was also considerably higher than our approach. It is noteworthy that using Algorithm 2.2 on top of the baseline network trained with early stopping criteria (shown in Figure 3) did not provide much improvements on the result since the baseline network lacks the $\delta$-stability imposed by the manifold regularization. Similar conclusion is observed when applying Algorithm 2.2 on top of forward-thinking approach [22] (see Table 3).

4.2. Physics informed adaptive neural network (PIANN). Consider partial differential equation (PDE) of the form:

\begin{align}
G(x, y, a) &= 0, \quad \text{in } \Omega, \\
y &= g \quad \text{in } \partial \Omega,
\end{align}

where, $a \in A$ is the PDE coefficient, $A$ is the parameter space, $y \in B$ is the unknown solution, $B$ is the solution space and the operator $G : B \times A \to F$ is in general a non-linear partial differential operator. Physics informed neural networks (PINNs) [60, 33] aims to solve the PDE (4.1) with the prescribed boundary condition (4.2). A lot of skepticism exists regarding the solution time and accuracy of PINNs in comparison to traditional numerical methods [18]. Nevertheless several attempts have been made to accelerate the training of PINNs [36]. In this section we show how a layerwise training process can lead to faster and more accurate solution in comparison to traditional-PINNs. The key is to adapt the neural network architecture with PINN-type loss terms. If one denotes the discretized equations for (4.1) arising out of a weak form as $\hat{G}_i(y(x_1), y(x_2), \ldots y(x_n))$, $\ i = 1, \ldots n$, the physics loss function is defined as:

\begin{align}
\Phi_p(\theta) = \sum_{i=1}^{n} \left( \hat{G}_i(y(\theta, x_1), y(\theta, x_2), \ldots, y(\theta, x_n)) \right)^2,
\end{align}

where, $\{x_1 \ldots x_n\}$ is the set of collocation points in $\Omega$. Further, the neural network learns the boundary condition (4.2) by considering training data points on $\partial \Omega$. It may be noted that, if one requires to update the network for a new parameter field $a \in A$ or a modified operator $G$, it is often unclear on the best layers to re-train [49]. Recent work by Subel et al. [49] shows that common wisdom guided transfer learning in machine learning literature that involves retraining only the last few layers of the network is often not the best option and that interpretable hidden layers are necessary to devise meaningful transfer learning strategy. Our approach (PIANN), attempts to create interpretable hidden layers in a deep network with superior performance compared to traditional PINNs.

For the present approach, $\gamma = 0$ (manifold regularization) since collocation points are uniformly distributed over the domain and the input data clusters does not exist. The training data set contains 4000 data points on the boundary. Further, $n = 961$.

Inspired from the continuation method in optimization [47], we propose the layerwise continuation approach where each layer is tasked with learning a simple function that can be minimized efficiently, and gradually transform it to more complicated cost function upon adding new layers. For achieving this, we choose $\tau^{(i+1)} > \tau^{(i)}$ ($\tau$ is the weight on the physics loss), and thereby also ensuring that a newly added layer is trainable (Proposition 3.15).
Also, since $|\tau^{(i+1)} - \tau^{(i)}| \neq 0$ for any $i$, training saturation problem does not exist in this case (Corollary 3.16) and Algorithm 2.2 is not required. For demonstration of our approach, we consider prototype problems considered by Weinan et al. [60].

4.2.1. Learning the Poisson equation. For demonstration, we consider solving the Poisson equation where the operator $G(x, y, a)$ in (4.1) is given as:

\begin{align}
G(x, y, a) &= -\nabla \cdot (a(x_1, x_2)\nabla y) - f(x_1, x_2) \quad \text{in} \quad \Omega \subset \mathbb{R}^2, \\
y &= 0 \quad \text{in} \quad \partial \Omega,
\end{align}

with $a(x_1, x_2) = 1$ and $f(x_1, x_2) = 200$. Further, we experiments are conducted for two different domains:

1. Case (a): $\Omega = (0, 1) \times (0, 1)$;
2. Case (b): $\Omega = (0, 1) \times (0, 1) - (0.5, 1) \times \{0.5\}$ [59]

Additional details on training process are given in Appendix G. The evolution of PDE solution $y(x)$ as more layers are added is shown in Figure 4 and 5: with 5 layers the solution looks identical to the finite element solution. Note that learning the solution progressively by our proposed approach (trained for a total of 8000 epochs) outperformed a baseline (traditional PINNs trained for 15000 epochs) trained under similar settings as seen from Figure 6. It can be seen that layerwise training procedure by Hettinger et al. [22] performed poorly in this case since by this method the network output is not preserved after adding a new layer. The
summary of results and parameter efficiency is provided in Appendix G. Further, creating interpretable hidden layers in a deep network helps one in devising efficient transfer learning strategies and make informed decisions on which layers to re-train. An example is provided in Appendix H where we demonstrate the superiority of our approach over traditional transfer learning strategies.

4.3. Adaptive learning of inverse maps from sparse data. Inverse problems are usually ill-posed and involves learning a map from low-dimensional space (observation space) to a high dimensional space (parameter space). This pose a challenge for learning the inverse map especially in the low data regime. Further, it is desirable for the inverse map to be well-posed where the solution’s behaviour should change continuously with respect to input. Therefore, it is clear that the notion of $\delta$—stability defined in Definition 3.2 is closely related to the concept of well-posedness. Therefore, the developed procedure serves as a natural candidate for this problem and we show how our approach can be applied to this regression task characterized by low availability of data. In this section, we demonstrate the approach for conductivity coefficient field inversion in a 2D heat equation written as:

$$
-\nabla \cdot (e^u \nabla y) = 20 \quad \text{in } \Omega = [0, 1]^2, \\
y = 0 \quad \text{on } \Gamma_{\text{ext}}, \\
n \cdot (e^u \nabla y) = 0 \quad \text{on } \Gamma_{\text{root}},
$$

(4.6)

where $n$ is normal vector on the equivalent boundary surfaces. The parameter of interest is denoted as $u$ and the observables are 10 pointwise values of the heat state (measurement locations are shown in Figure 8), $y$ at arbitrary location. The training data is generated by drawing parameter samples as:

$$
u(x) = \sum_{i=1}^{n} \sqrt{\lambda_i} \phi_i \ x_i
$$

(4.7)

where $\lambda_i$, $\phi_i$ is the eigen-pair of an exponential two point correlation function and $x = (x_1, \ldots, x_n)$ is drawn from standard Gaussian distribution [13]. Note that $x$ represents the output of the network. For the present study, we choose the output dimension $n = 12$. The
Table 1

| Training data size | Equivalent baseline | Proposed method | Forward thinking [22] | Architecture search (AS) [30] |
|-------------------|---------------------|-----------------|----------------------|--------------------------------|
| 20                | 0.64                | 0.585           | 1.41                 | 0.625                          |
| 50                | 0.44                | 0.42            | 0.84                 | 0.427                          |

observation vector (input of the network) is generated by solving (4.6). 5% additive Gaussian noise is added to the observations to represent field condition.

We consider experiments with two different training data size; 20 and 50. The validation data set consists of 20 data points and the test set contains 500 data points. Since the number of training data points is too low, forming clusters with the available dataset is impractical.

4.3.1. Forming artificial data clusters and training procedure. In order to compute the manifold regularization (2.4), we generate artificial clusters by first assuming that each input data point (observation) lies on a different cluster, i.e. \( x_m \in M_m \subset \mathbb{R}^{10} \). Each set \( M_m \) is then populated with 100 artificial perturbations of the input \( x_m \) (assuming 1% additive Gaussian noise) and we further assume that each perturbed data is a valid observation data point.

![Summary of adaptive inversion (trained with 20 data set).](image1.png)

**Figure 7.** Summary of adaptive inversion (trained with 20 data set).

Left to right: Summary of Algorithm 2.1; Active and inactive parameters in each hidden layer; Promoting \( \delta \)–stability through manifold regularization.

In addition, we consider increasing the manifold regularization weight, \( \gamma^{(i+1)} = 2 \times \gamma^{(i)} \) while adding new layers where stability is more strongly enforced in the later layers. This strategy is adopted due to the small data-set size of the problem where initial layers are weakly regularized to allow for maximum information transfer through the initial layers (note that we freeze the parameters of these layers later). Further, for this problem setup we have numerically observed that the data loss \( \Phi_d \) in (2.6) only increases marginally while optimizing the later layers (where \( \gamma^{(i)} \) is too large) making this strategy feasible. Most importantly, note that by Proposition 3.15, training does not saturate since \( \gamma^{(i+1)} = 2 \times \gamma^{(i)} \) (satisfying the necessary condition for trainability) and hence one does not require Algorithm 2.2.

The relative error achieved by different methods is shown in Table 1 and Figure 7 which
clearly shows that Algorithm 2.1 outperforms traditional methods. Further, Figure 8 also shows that the parameter field predicted by the proposed method is better than the one produced by baseline network.

Since for this problem, data can be artificially generated from each $\mathcal{M}_m$, one could investigate the effect of manifold regularization numerically through a sampling based approach. We study the nature of the stability function $\delta^N_j(\gamma)$ in (3.16) by training three networks exactly in the same way but with different $\gamma^{(2)}$. Once the network is trained, 5000 different points $\{\mathbf{x}_1, \ldots, \mathbf{x}_{5000}\} \in \mathcal{M}_m$ is artificially generated. Then, based on (3.16), an upper bound corresponding to the set $\mathcal{M}_m$ is approximately computed as:

$$
\delta^N_m(\gamma^{(2)}) = \max_{i,j} \left\{ \| \mathcal{N}(\mathbf{x}_j) - \mathcal{N}(\mathbf{x}_i) \|_2 \right\}, \quad \mathbf{x}_i, \mathbf{x}_j \in \mathcal{M}_m
$$

Figure 7 shows that as $\gamma$ is increased, the numerically computed $\delta^N_m(\gamma^{(2)})$ decreases almost surely $\forall m$ and provides a feasible way for controlling stability via manifold regularization. In addition, the evolution of solution across the hidden layers for a particular test observation sample is shown in Figure 9. It is clear from Figure 9 that injecting stability in later layers allows the network to recover fine details in the parameter field when the baseline network fails to do so.

**4.4. Image classification problem: MNIST data set.** Finally, we consider the MNIST handwritten digit classification problem using a fully connected residual neural network. It is noteworthy that, even though MNIST problem is easier to solve with convolutional architectures, achieving high classification accuracy remains challenging with fully connected networks. In this section we examine how the design of $\varepsilon - \delta$ stability promoting algorithm allows us to achieve a high accuracy using fully-connected networks. We demonstrate the adaptation procedure for two different scenarios: a) Case with 20 neurons in each hidden layer; b) Case with 500 neurons in each hidden layer. Figure 10 shows that the proposed methodology outperforms the baseline network and other methods by noticeable margins and even exhibiting performance on par to that of a LeNet-5 architecture (a basic CNN architecture). Further, Figure 11 shows the schematic of Algorithm 2.1 applied to the MNIST dataset. Note that the inactive parameters can be removed after training each hidden layer.
Figure 9. Evolution of parameter field across the hidden layers for a particular test observation sample. First row left to right: Solution after training layer $N^{(3)}$; Solution after training layer $N^{(5)}$; Solution after training layer $N^{(7)}$. Second row left to right: Solution after training layer $N^{(9)}$; Solution after training layer $N^{(12)}$; Exact solution.

as demonstrated in Figure 11 thereby leading to a partially connected network. Additional details on experiments carried out is provided in Appendix I.

Figure 10. MNIST classification problem. Left figure is with 20 neurons in each hidden layer: our approach provides the best results with 96.9% testing accuracy. Right figure is with 500 neurons in each hidden layer: our approach provides the best results with 98.7% testing accuracy.

5. Concluding remarks. In this paper, we presented a two-stage procedure for adaptive learning that promotes “robustness” and thereby generalizing well for a given data-set. The first stage (Algorithm 2.1) is designed to grow a neural network along the depth while promoting $\delta$-stability for each hidden layer through the use of manifold regularization. In addition, non-important parameters are removed using a sparsity promoting regularization. We have proved that for certain hyperparameter settings of Algorithm 2.1, one faces with the training
Figure 11. Schematic of Algorithm 2.1 demonstrated on MNIST dataset (20 neurons in each hidden layer) for the first few layers: Note that the input layer and ResNet connections are not shown here.

saturation problem where the newly added layer does not learn (Corollary 3.16). In order to further improve the prediction accuracy, we designed Algorithm 2.2 as a post-processing stage where the main goal is to promote robustness (Definition 3.19). Besides several theoretical results, numerical results on prototype regression and classification tasks, including solving forward and inverse problems governed by elliptic PDEs, suggest that the proposed approach can outperform an ad-hoc baseline and other adaptation strategies in terms of generalization error. The proposed PIANN is perhaps the first PINN approaches that is capable of adapting...
the network architecture with theoretical results on stabilities.

Acknowledgement. This research is partially funded by the National Science Foundation awards NSF-OAC-2212442, NSF-2108320, and NSF-CAREER-1845799; and by the Department of Energy award DE-SC0018147 and DE-SC0022211. The authors would also like to thank Dr. Hwan Goh, Dr. Jonathan Wittmer, Hai Van Nguyen, and Jau-Uei Chen for fruitful discussions. The authors also acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC, visualization, database, or grid resources that have contributed to some of the results reported within this paper. URL: http://www.tacc.utexas.edu.

REFERENCES

[1] P. Balaprakash, M. Salim, T. D. Uram, V. Vishwanath, and S. M. Wild, Deephyper: Asynchronous hyperparameter search for deep neural networks, in 2018 IEEE 25th international conference on high performance computing (HiPC), IEEE, 2018, pp. 42–51.
[2] E. Belilovsky, M. Eickenberg, and E. Oyallon, Greedy layerwise learning can scale to imagenet, in International conference on machine learning, PMLR, 2019, pp. 583–593.
[3] M. Belkin and P. Niyogi, Towards a theoretical foundation for laplacian-based manifold methods, Journal of Computer and System Sciences, 74 (2008), pp. 1289–1308.
[4] M. Belkin, P. Niyogi, and V. Sindhwani, Manifold regularization: A geometric framework for learning from labeled and unlabeled examples., Journal of machine learning research, 7 (2006).
[5] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al., Greedy layer-wise training of deep networks, Advances in neural information processing systems, 19 (2007), p. 153.
[6] S. Bubeck, Y. Li, and D. M. Nagarici, A law of robustness for two-layers neural networks, in Conference on Learning Theory, PMLR, 2021, pp. 804–820.
[7] S. Bubeck and M. Sellke, A universal law of robustness via isoperimetry, Advances in Neural Information Processing Systems, 34 (2021), pp. 28811–28822.
[8] D. Charalambos and B. Aliprantis, Infinite Dimensional Analysis: A Hitchhiker’s Guide, Springer-Verlag Berlin and Heidelberg GmbH & Company KG, 2013.
[9] S. Chatterjee, A. M. Javid, M. Sadeghi, P. P. Mitra, and M. Skoglund, Progressive learning for systematic design of large neural networks, arXiv preprint arXiv:1710.08177, (2017).
[10] T. Chen, I. Goodfellow, and J. Shlens, Net2net: Accelerating learning via knowledge transfer, arXiv preprint arXiv:1511.07289, (2015).
[11] T. Elsken, J. H. Metzen, and F. Hutter, Efficient multi-objective neural architecture search via lamarckian evolution, arXiv preprint arXiv:1804.09081, (2018).
[12] C. Fefferman, S. Mitter, and H. Narayanan, Testing the manifold hypothesis, Journal of the American Mathematical Society, 29 (2016), pp. 983–1049.
[13] D. Harrison Jr and D. L. Rubinfeld, Hedonic housing prices and the demand for clean air, Journal
of environmental economics and management, 5 (1978), pp. 81–102.

[20] 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.

[21] K. He, X. Zhang, S. Ren, and J. Sun, Identity mappings in deep residual networks, in European conference on computer vision, Springer, 2016, pp. 630–645.

[22] C. Hettinger, T. Christensen, B. Ehlert, J. Humpherys, T. Jarvis, and S. Wade, Forward thinking: Building and training neural networks one layer at a time, arXiv preprint arXiv:1706.02480, (2017).

[23] G. E. Hinton, Learning multiple layers of representation, Trends in cognitive sciences, 11 (2007), pp. 428–434.

[24] C. Jin and M. Rinard, Manifold regularization for locally stable deep neural networks, arXiv preprint arXiv:2003.04286, (2020).

[25] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980, (2014).

[26] M. Kulkarni and S. Karande, Layer-wise training of deep networks using kernel similarity, arXiv preprint arXiv:1703.07115, (2017).

[27] Y. LeCun, Y. Bengio, et al., Convolutional networks for images, speech, and time series, The handbook of brain theory and neural networks, 3361 (1995), p. 1995.

[28] T. Lee, M. Choi, and S. Yoon, Manifold regularized deep neural networks using adversarial examples, arXiv preprint arXiv:1511.06381, (2015).

[29] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, On the expressive power of deep neural networks, in international conference on machine learning, PMLR, 2017, pp. 2847–2854.

[30] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, 378 (2019), pp. 686–707.
Appendix A. Proof of non-asymptotic bound in equation (3.8) Let us first define the random variable $X_i$ as follows:

$$X_i = \int_{\mathcal{M}_p^{(1)}} \left\| \mathcal{N}^{(2)}(x_i) - \mathcal{N}^{(2)}(y) \right\|_2^2 d\mu_p^{(1)}(y), \quad x_i \sim \mu_p^{(1)}(x),$$

(A.1)

$$E [X_i] = \int_{\mathcal{M}_p^{(1)}} \int_{\mathcal{M}_p^{(1)}} \left\| \mathcal{N}^{(2)}(x) - \mathcal{N}^{(2)}(y) \right\|_2^2 d\mu_p^{(1)}(x) d\mu_p^{(1)}(y).$$

Now using assumption 1 (continuous function $\mathcal{N}^{(1)}$ maps the compact set $\mathcal{M}_p^{(0)}$ to $\mathcal{M}_p^{(1)}$ which is compact) it is easy to see that $X_i$ is a bounded random variable based on the Weierstrass
theorem (Theorem 1.18.1 in [40]). Therefore, from Hoeffding’s inequality [52] we have:

\[ P \left( \left| \frac{1}{m_p} \sum_{i=1}^{m_p} X_i - \mathbb{E}[X_i] \right| \geq \varepsilon \right) \leq 2 \exp \left( -2c_1 m_p \varepsilon^2 \right), \]

where \( c_1 \) is a positive constant. Thus,

\[ (A.2) \]

\[ P \left( \mathbb{E}[X_i] \leq \varepsilon + \frac{1}{m_p} \sum_{i=1}^{m_p} X_i \right) \geq P \left( \left| \frac{1}{m_p} \sum_{i=1}^{m_p} X_i - \mathbb{E}[X_i] \right| < \varepsilon \right) \geq 1 - 2 \exp \left( -2c_1 m_p \varepsilon^2 \right), \]

Now for any given \( x_i \in \mathcal{M}_p^{(1)} \) drawn from \( \mu_p^{(1)}(x) \), let us define the following random variable:

\[ Y_j^i = \|N^{(2)}(x_i) - N^{(2)}(y_j)\|^2_2, \quad y_j \sim \mu_p^{(1)}(y), \]

\[ (A.3) \]

\[ \mathbb{E}[Y_j^i] = \int_{\mathcal{M}_p^{(1)}} \|N^{(2)}(x_i) - N^{(2)}(y)\|^2_2 d\mu_p^{(1)}(y). \]

Again, from Hoeffding’s inequality for random variable \( Y_j^i \) we have:

\[ P \left( \left| \frac{1}{m_p} \sum_{j=1}^{m_p} Y_j^i - \mathbb{E}[Y_j^i] \right| \geq \varepsilon \right) \leq 2 \exp \left( -2c_2 m_p \varepsilon^2 \right), \]

where \( c_2 \) is a positive constant. Therefore, we have:

\[ (A.4) \]

\[ P \left( \mathbb{E}[Y_j^i] \leq \varepsilon + \frac{1}{m_p} \sum_{j=1}^{m_p} Y_j^i \right) \geq P \left( \left| \frac{1}{m_p} \sum_{j=1}^{m_p} Y_j^i - \mathbb{E}[Y_j^i] \right| \leq \varepsilon \right) \geq 1 - 2 \exp \left( -2c_2 m_p \varepsilon^2 \right). \]

Now considering the concentration inequality (A.4) for all \( i \) (summing up), and using the union bound we have:

\[ (A.5) \]

\[ P \left( \frac{1}{m_p} \sum_{i=1}^{m_p} \mathbb{E}[Y_j^i] \leq \varepsilon + \frac{1}{m_p} \sum_{i=1}^{m_p} \sum_{j=1}^{m_p} Y_j^i \right) \geq 1 - 2m_p \exp \left( -2c_2 m_p \varepsilon^2 \right). \]

Now combining the result (A.2) and (A.5) and using the union bound, we have:

\[ P \left( \mathbb{E}[X_i] \leq \varepsilon + \frac{1}{m_p} \sum_{i=1}^{m_p} \sum_{j=1}^{m_p} Y_j^i \right) \geq 1 - 2m_p \exp \left( -c_3 m_p \varepsilon^2 \right) - 2 \exp \left( -c_4 m_p \varepsilon^2 \right) = 1 - \mathcal{E}_p^{(2)}(m_p, \varepsilon) \]

where, the constants \( c_3, c_4 \) depends on the cluster \( C_p \) and the hidden layer. Therefore, we have the final required result:

\[ P \left[ \mathbb{E}_{\mu_p^{(1)}} \left[ \left\| \mathcal{N}^{(2)}(x) - \mathcal{N}^{(2)}(y) \right\|^2_2 \right] \leq \varepsilon + \frac{1}{m_p^2} \sum_{x_i, y_j \in C_p} \left\| \mathcal{N}^{(2)}(x_i) - \mathcal{N}^{(2)}(y_j) \right\|^2_2 \right] \geq 1 - \mathcal{E}_p^{(2)}(m_p, \varepsilon). \]
Appendix B. Proof of Corollary 3.11. A sketch of the proof is provided here. Since $\mathcal{N}$ is a composition of hidden layers (note that $(L + 1)^{th}$ layer is the output layer), we have:

$$\mathcal{N}(x) = \mathcal{N}^{(L+1)} \circ \mathcal{R}^{(L)} \circ \mathcal{N}^{(L)} \circ \ldots \circ \mathcal{R}^{(1)} \circ \mathcal{N}^{(1)}(x).$$

Since the activation function employed is Lipschitz continuous, we have $\forall x, x' \in \mathcal{M}_p^{(L-1)}$:

\begin{align}
(B.1) & \quad \left\| \mathcal{N}^{(L)}(x) - \mathcal{N}^{(L)}(x') \right\|_2^2 \leq c^{(L)} \times \left\| x - x' \right\|_2^2, \\
(B.2) & \quad \left\| \mathcal{N}^{(L)}(x) - \mathcal{N}^{(L)}(x') \right\|_2^2 \leq \tilde{\delta}_p \left( \kappa^{L-2} \times \gamma, \xi^{(L)} \right),
\end{align}

where $c^{(L)} \geq 0$, and (B.2) holds with probability at-least $\left( 1 - \xi^{(L)} - \xi_p^{(L)}(m_p, \xi^{(L)} \varepsilon) \right)$ (from (3.6)). Now multiplying (B.1) and (B.2), we have:

\begin{align}
(B.3) & \quad \sqrt{c^{(L)}} \times \sqrt{\tilde{\delta}_p \left( \kappa^{L-2} \times \gamma, \xi^{(L)} \right)} \times \left\| \mathcal{R}^{(L-1)} \left( \mathcal{N}^{(L-1)}(y) \right) - \mathcal{R}^{(L-1)} \left( \mathcal{N}^{(L-1)}(y') \right) \right\|_2 \\
& \quad \leq c^{(L)*} \times \sqrt{\tilde{\delta}_p \left( \kappa^{L-2} \times \gamma, \xi^{(L)} \right)} \times \left\| \mathcal{N}^{(L-1)}(y) - \mathcal{N}^{(L-1)}(y') \right\|_2,
\end{align}

where $y, y' \in \mathcal{M}_p^{(L-2)}$ and we have used the fact that $\mathcal{R}^{(L-1)}$ is Lipschitz continuous. Similarly, applying (B.1) and (B.2) recursively to each $\mathcal{N}^{(L-1)}, \ldots \mathcal{N}^{(2)}$ and using the union bound we have $\forall x, x' \in \mathcal{M}_p^{(0)} \subset \mathcal{M}^{(0)}$:

\begin{align}
(B.4) & \quad \left\| \mathcal{N}(x) - \mathcal{N}(x') \right\|_2^2 \leq \epsilon \prod_{i=2}^L \left( \tilde{\delta}_p \left( \kappa^{i-2} \times \gamma, \xi^{(i)} \right) \right)^{\frac{1}{2^{(L-i+1)}}} \left\| x - x' \right\|_2^2 \\
& \quad \leq \epsilon \prod_{i=2}^L \left( \tilde{\delta}_p \left( \kappa^{i-2} \times \gamma, \xi^{(i)} \right) \right)^{\frac{1}{2^{(L-i+1)}}} \delta_p^N \left( \gamma, \xi^{(1)}, \ldots \xi^{(L)} \right),
\end{align}

holds with probability at-least $\left( 1 - \sum_{i=2}^L \xi^{(i)} - \sum_{i=2}^L \xi_p^{(i)}(m_p, \xi^{(i)} \varepsilon) \right)$, and $\epsilon_p$ is defined as $\epsilon_p = \sup_{x, x' \in \mathcal{M}_p^{(0)}} \left\| x - x' \right\|_2$. Now in the limit $m_p \to \infty$, (B.4) holds with probability at-least $\left( 1 - \sum_{i=2}^L \xi^{(i)} \right)$, where the stability function $\tilde{\delta}_p \left( \gamma, \xi^{(2)} \right)$ (for $i = 2$) from (3.11) simplifies as:

$$\tilde{\delta}_p \left( \gamma, \xi^{(2)} \right) = \frac{1}{\xi^{(2)} \times m^2} \sum_{x_i, x_j \in C_p} \left\| \mathcal{N}^{(2)}_{\theta^{(\gamma)}}(x_i) - \mathcal{N}^{(2)}_{\theta^{(\gamma)}}(x_j) \right\|_2^2,$$

and the subsequent functions $\tilde{\delta}_p \left( \kappa^{i-2} \times \gamma, \xi^{(i)} \right)$ for $i > 2$ follows a similar pattern. Further note that from (3.14) we also have $\tilde{\delta}_p \left( \gamma^{u}, \xi^{(2)} \right) = 0$ and $\tilde{\delta}_p \left( \gamma, \xi^{(2)} \right) > 0$, $\forall p$. Therefore, we have $\delta_p^N \left( \gamma^{u}, \xi^{(2)}, \ldots \xi^{(L)} \right) = 0$ and $\delta_p^N \left( \gamma, \xi^{(2)}, \ldots \xi^{(L)} \right) > 0$, $\forall p$, thereby satisfying condition 1 (with $\varepsilon = 0$) for any fixed $\xi^{(2)}, \ldots \xi^{(L)}$. In addition, since each $\tilde{\delta}_p \left( \kappa^{i-2} \times \gamma, \xi^{(i)} \right)$ is continuous
with respect to \( \gamma \) as proved in Proposition 3.9, \( \phi_p^N(\gamma, \xi^{(2)}, \ldots \xi^{(L)}) \) is also continuous with respect to \( \gamma \) thereby satisfying condition 2 of Definition 3.2.

**Appendix C. Gradients for layerwise training.** Consider training the \((i + 1)^{th}\) layer. Let us denote the different loss components for a particular \(j^{th}\) training sample belonging to cluster \(C_p\) as follows:

\[
\Phi_d\left(Y^{(o)}_j\right), \quad \text{(data loss)}
\]

\[
\Phi_{m}^{(i+1)}\left(Y^{(i+1)}_j\right) = \frac{\gamma^{(i+1)}}{n_p} \sum_m \beta_p \left\| Y^{(i+1)}_j - Y^{(i+1)}_m \right\|^2 = \gamma^{(i+1)} \times \Phi_{m} \left(Y^{(i+1)}_j\right),
\]

\[
\Phi_{p}^{(i+1)}\left(Y^{(o)}_j\right) = \tau^{(i+1)} \times \Phi_{p} \left(Y^{(o)}_j\right), \quad \text{(physics loss)}
\]

where, \(Y^{(i+1)}_j\) represents the output of the \((i + 1)^{th}\) layer for the \(j^{th}\) training sample and \(Y^{(o)}_j\) represents the output of the network. Therefore, training problem for the new \((i + 1)^{th}\) layer can be formulated as:

\[
\min_{\mathbf{b}^{(i+1)}, \mathbf{W}^{(i+1)}, \mathbf{W}_{\text{pred}}, \mathbf{b}_{\text{pred}}} \Phi_d\left(Y^{(o)}_j\right) + \Phi_{m}^{(i+1)}\left(Y^{(i+1)}_j\right) + \Phi_{p}^{(i+1)}\left(Y^{(o)}_j\right)
\]

such that:

\[
Y^{(i+1)}_j = R^{(i)}\left(Y^{(i)}_j\right) + h^{(i+1)}\left(W^{(i+1)}R^{(i)}\left(Y^{(i)}_j\right) + b^{(i+1)}\right),
\]

\[
Y^{(o)}_j = h_{\text{pred}}\left(W_{\text{pred}}R^{(i+1)}\left(Y^{(i+1)}_j\right) + b_{\text{pred}}\right).
\]

The Lagrangian for the above constrained optimization problem is defined as:

\[
\mathcal{L}_j = \Phi_d\left(Y^{(o)}_j\right) + \Phi_{m}^{(i+1)}\left(Y^{(i+1)}_j\right) + \Phi_{p}^{(i+1)}\left(Y^{(o)}_j\right) + (\lambda_2)^T \left(Y^{(o)}_j - h_{\text{pred}}\left(W_{\text{pred}}R^{(i+1)}\left(Y^{(i+1)}_j\right) + b_{\text{pred}}\right)\right) + (\lambda_1)^T \left(Y^{(i+1)}_j - R^{(i)}\left(Y^{(i)}_j\right) - h^{(i+1)}\left(W^{(i+1)}R^{(i)}\left(Y^{(i)}_j\right) + b^{(i+1)}\right)\right).
\]

The first order optimality conditions are as follows:

\[
\frac{\partial \mathcal{L}_j}{\partial Y^{(i+1)}_j} = 0, \quad \frac{\partial \mathcal{L}_j}{\partial Y^{(o)}_j} = 0, \quad \frac{\partial \mathcal{L}_j}{\partial W^{(i+1)}} = 0, \quad \frac{\partial \mathcal{L}_j}{\partial b^{(i+1)}} = 0, \quad \frac{\partial \mathcal{L}_j}{\partial W_{\text{pred}}} = 0, \quad \frac{\partial \mathcal{L}_j}{\partial b_{\text{pred}}} = 0.
\]

\[
\frac{\partial \mathcal{L}_j}{\partial Y^{(i+1)}_j} = 0 \implies \lambda_1 = R^{(i+1)}W_{\text{pred}}^T \left[h_{\text{pred}}^T\left(W_{\text{pred}}R^{(i+1)}\left(Y^{(i+1)}_j\right) + b_{\text{pred}}\right)\circ \lambda_2\right] - \frac{\partial \Phi_{m}^{(i+1)}}{\partial Y^{(i+1)}_j},
\]

where \(R^{(i+1)}\left(Y^{(i+1)}_j\right) = R^{(i+1)}Y^{(i+1)}_j + \mathbf{b}_r\) such that \(R^{(i+1)}\) is a diagonal matrix and \(\mathbf{b}_r\) is a constant vector. Similarly, other optimality conditions can be derived as follows:

\[
\frac{\partial \mathcal{L}_j}{\partial Y^{(o)}_j} = 0 \implies \lambda_2 = - \left(\frac{\partial \Phi_d}{\partial Y^{(o)}_j} + \frac{\partial \Phi_{p}^{(i+1)}}{\partial Y^{(o)}_j}\right),
\]
where $L$ is the local Lipschitz constant of $Q$. Now note that $\|Q(x) - Q(x')\|_2 = \|Q_2(x) + \ldots + Q_{r-1}(x) - (Q_2(x') + \ldots + Q_{r-1}(x'))\|_2$.

Applying the transformation $\delta_p = \sum_{i=2}^{r-1} \ell^3 \times \left( \prod_{k=1}^{3} \left( \sum_{l=0}^{m} \left\| (g^{(k)}_l)_i \right\|_2 \right) \right) \times \varepsilon_p = \sum_{i=2}^{r-1} \delta_p^{(i)}(\zeta) = \delta_p^{Q}(\zeta).$
Now in order to ensure that $\delta_Q^j(\zeta)$ is a valid stability function as per Definition 3.4, it is enough to check if each $\delta_p^{(i)}(\zeta)$ satisfies condition 1 in Definition 3.2, and condition 3 in Definition 3.4.

Note that $\delta_j^{(i)}(\zeta)$ is now a discrete function with domain $\left\{ \frac{1}{m^u+1}, \frac{1}{m^u}, \ldots, 1 \right\}$, where $m^u \in \mathbb{Z}^+$ is some chosen upper bound for the training iterations. Note that $\delta_p^{(i)}(1) = 0$ since $\left( g_0^{(3)} \right) = 0$ due to the output layer being initialized with zeros in Algorithm 2.2. Further, $\delta_p^{(i)} \left( \frac{1}{m^u+1} \right) > 0$ since the gradients in the subsequent iterations must be non-zero for decreasing the loss. Therefore, we have $\delta_p^{(i)}(1) < \delta_p^{(i)} \left( \frac{1}{m^u+1} \right)$ satisfying condition 1 in Definition 3.2. In addition, it is clear that $\delta_p^{(i)}(\zeta)$ is monotonically decreasing with respect to $\zeta$ thereby satisfying condition 3 in Definition 3.4.

Appendix E. General setting for numerical experiments. All codes were written in Python using TensorFlow. Throughout the study, we have employed the Adam optimizer [25] for minimizing the cost function. The manifold regularization term is computed over mini-batches. We consider a decay of manifold regularization weight $\gamma^{(i+1)} = 0.5 \times \gamma^{(i)}$ on adding a new layer unless otherwise stated. We choose $\beta_p = 1$, $\forall p$ in (2.3). Our proposed approach is compared with a number of different approaches. Description of different algorithms adopted for comparison with the proposed approach

Algo. 2.1 : layerwise training algorithm described in subsection 2.1.

Algo. 2.2 : Sequential residual learning after Algo. 2.1 (subsection 2.2).

Baseline network : A fully-connected deep neural network with the same depth and width as that produced by Algorithm 2.1.

Algo. 2.2 (1 network) : Sequential residual learning after Algo. 2.1 using 1 network in Algo. 2.2 trained for a large number of epochs.

Baseline + Algo. 2.2 : Algorithm 2.2 applied on top of the residuals produced by the baseline network.

Forward Thinking : Algorithm for layerwise adaptation by Hettinger et al. [22]. This strategy is also similar to the algorithm by Belilovsky et al. [2] and Trinh et al. [51].

Approach [22] + Algo. 2.2 : Algorithm 2.2 applied on top of the residuals produced by Forward thinking [22].

Architecture Search (AS) : Random search with early stopping proposed by Liam Li and Ameet Talwalkar [30].

When training the baseline, 100 different random initializations are considered and the best result is chosen for comparison. For architecture search (AS), the search space is defined by setting the maximum width (for each hidden layer) and maximum depth the same as that produced by Algorithm 2.1.

E.1. Choice of $R^{(i)}$. For the results provided in this work, we choose $R^{(i)}$ in (2.1) as
the identity map for computational efficiency. This avoids recomputing the map $R^{(i)}$ in each training iteration. We observed numerically that the feature scaling introduced in (2) from a pure theoretical purpose did not provide improvement in results.

**E.2. Tuning parameters $\gamma$ and $\zeta$ for $\delta$-stability.** Note that the hyperparameters $\gamma$ (in Algorithm 2.1) and $E_e$ (in Algorithm 2.2) are tuned based on the validation dataset to achieve the best results. The best parameters are the ones that minimizes the validation loss such as depicted in Figure 2. Note that tuning parameter $E_e$ in Algorithm 2.2 corresponds to tuning the parameter $\zeta$ in proposition 3.20. We consider a strategy of training the first two networks in Algorithm 2.2 for $E_1^e$ number of epochs and training the remaining networks for $E_2^e$ number of epochs. This leads to tuning $\{E_1^e, E_2^e\}$ in Algorithm 2.2. That is, Algorithm 2.2 is run for different choices of $\{E_1^e, E_2^e\}$ and the best choice is the one that provides the lowest validation loss. This strategy is adopted since the residuals becomes increasingly nonlinear and subsequent networks requires more number of epochs to promote training (i.e decrease the loss).

**Appendix F. Boston house price prediction problem.** For this problem, the manifold regularization is computed by employing the K-means clustering algorithm on the input features [35] and the number of clusters $K$ is chosen as 5.

**Appendix G. Physics informed adaptive neural network (PIANN).** This section contains additional details on the PIANN problem discussed in subsection 4.2. The relative $L^2$ error achieved after adding each layer is shown in Table 2 (case a) and Table 5 (case b). The active parameters in each hidden layer is shown in Table 3 (case a) and Table 6 (case b). The results achieved by the baseline network as well as other layerwise training method [22] is also provided in Table 4 (case a) and Table 7 (case b).

| Layer | Relative $L^2$ error | Layer | % of non-zero parameters |
|-------|----------------------|-------|--------------------------|
| No.   |                      |       |                          |
| 2     | $1.723 \times 10^{-1}$ | 2     | 99.5 %                   |
| 3     | $2.254 \times 10^{-2}$ | 3     | 97.7 %                   |
| 4     | $4.974 \times 10^{-3}$ | 4     | 98.9 %                   |
| 5     | $6.649 \times 10^{-4}$ | 5     | 95.4 %                   |
| 6     | $2.660 \times 10^{-4}$ | 6     | 92.5 %                   |
| 7     | $6.720 \times 10^{-5}$ | 7     | 96.7 %                   |

**Appendix H. Interpretability of hidden layers and transfer learning strategy.** In this section, we demonstrate how the interpretability of different layers in PIANN helps one in devising efficient transfer learning strategies and make informed decisions on which layers to re-train. For demonstration, consider the operator in (4.5) with $a(x_1, x_2) = e^{x_1 x_2}$ and $f = 200$. We reuse the pre-trained model that learnt the solution in Figure 5. The objective is to correct this pre-trained model for the new coefficient $a(x_1, x_2) = e^{x_1 x_2}$. 
Table 4

Performance of Baseline Network and other layerwise training methods (Case a)

| Method                    | Relative $L^2$ error | Parameters trained simultaneously |
|---------------------------|----------------------|----------------------------------|
| Proposed method           | $6.7 \times 10^{-3}$ | 10,501                           |
| Baseline network          | $1.0 \times 10^{-4}$ | 61,001                           |
| Forward Thinking [22]     | $9.2 \times 10^{-2}$ | 10,501                           |

Table 5

Relative $L^2$ error on layer addition (Case b) % of non-zero parameters in each layer (Case b)

| Layer No. | $\tau_{(i)}$ | Relative $L^2$ error | Layer No. | % of non-zero parameters |
|-----------|-------------|----------------------|-----------|--------------------------|
| 2         | 5           | $3.59 \times 10^{-1}$ | 2         | 97.52 %                  |
| 3         | 10          | $4.75 \times 10^{-2}$ | 3         | 91.29 %                  |
| 4         | 15          | $1.13 \times 10^{-3}$ | 4         | 97.35 %                  |
| 5         | 20          | $7.86 \times 10^{-5}$ | 5         | 90.00 %                  |

Table 6

% of non-zero parameters in each layer (Case b)

| Layer No. | % of non-zero parameters |
|-----------|--------------------------|
| 2         | 97.52 %                  |
| 3         | 91.29 %                  |
| 4         | 97.35 %                  |
| 5         | 90.00 %                  |

Table 7

Performance of Baseline Network and other layerwise training methods (Case b)

| Method                    | Relative $L^2$ error | Parameters trained simultaneously |
|---------------------------|----------------------|----------------------------------|
| Proposed method           | $7.8 \times 10^{-5}$ | 10,501                           |
| Baseline network          | $1.1 \times 10^{-4}$ | 40,801                           |
| Forward Thinking [22]     | $1.6 \times 10^{-2}$ | 10,501                           |

Figure 12. Transfer learning strategy. Left to right: Transfer learning on interpretable network achieved by our approach (relative $L^2$ error $= 4.432 \times 10^{-3}$ at the end of 100 epochs); True solution; Traditional transfer learning strategy on a baseline (relative $L^2$ error $= 8.27 \times 10^{-3}$ at the end of 100 epochs).

Since in our approach (PIANN), the first few layers focus mainly on enforcing the boundary conditions (Figure 5) it is anticipated that only the last layers needs retraining for model correction. In our approach for transfer learning, we consider retraining the last two layers reinitialized with zero weights and biases. The relative $L^2$ error achieved and the learnt solution after retraining the last two layers for 100 epochs is shown in Figure 12. In addition,
we also consider the traditional transfer learning strategy of retraining the last two layers of a pre-trained baseline network that lacks the interpretability of its layers [49]. Note that in this case it is unclear on the best layers to re-train and a discussion on this aspect is provided in [49]. Comparison of efficiency between the two approaches is shown in Figure 12. It is clear from Figure 12 that our approach learns the new solution faster in comparison to the traditional transfer learning approach.

**Appendix I. Performance on MNIST dataset.**

1.1. Training with 20 neurons in each hidden layer. The inputs for this experiment is given in Appendix J. The parameter efficiency achieved for the case of using 20 neurons in the hidden layers is provided in Figure 13. Table 8 shows the comparison of results with baseline and other layerwise training method.

![Figure 13](image)

*Figure 13. Left to right: Active and inactive parameters in each hidden layer (20 neurons in each hidden layer); Active and inactive parameters in each hidden layer (500 neurons in each hidden layer)*

| Method                  | Test Accuracy | Parameters trained simultaneously |
|-------------------------|---------------|----------------------------------|
| Proposed method         | 96.90         | 16,330                           |
| Baseline network        | 96.00         | 21,370                           |
| Baseline + Algo. 2.2    | 95.90         | 21,370                           |
| Forward Thinking [22]   | 95.90         | 16,330                           |
| Approach [22] + Algo. 2.2 | 96.00      | 16,330                           |
| Architecture search (AS)| 96.47         | -                                |

Table 8

*MNIST: Performance of baseline network and other layerwise training methods (20 neuron case)*

Note that our experiments showed that, forward thinking model [22] quickly over-fits on the training data while adding the initial few layers. Further, forward thinking model is not \( \delta \) – stable which is a necessary condition for Algorithm 2.2 to achieve robustness (Remark 3.21).
Numerically, it is observed that the residual defined in line 4 of Algorithm 2.2 is very low in magnitude, and it is hardly possible to extract information from the residual and improve the results. Hence, the performance of forward thinking model + Algorithm 2.2 has been not been included in the comparison results shown in Figure 10 since the behaviour is similar to baseline + Algorithm 2.2.

1.2. Training with 500 neurons in each hidden layer. The parameter efficiency and the comparison of results with baseline is shown in Figure 13 and Table 9.

| Method                      | Test accuracy % | Params. trained simultaneously |
|-----------------------------|-----------------|--------------------------------|
| Proposed method             | 98.7            | 6,48010                        |
| Baseline network            | 98.2            | 34,03510                       |
| Baseline + Algo. 2.2        | 98.2            | 34,03510                       |
| Forward Thinking [22]       | 98.3            | 6,48010                        |
| Approach [22] + Algo. 2.2  | 98.4            | 6,48010                        |
| Architecture search (AS)    | 98.3            | -                              |

Appendix J. Details of input parameter values for different problems. The description of each problem is given below:

I : Boston house price prediction problem

II (a) : Physics informed adaptive neural network (PIANN), Symmetric boundary.

II (b) : Physics informed adaptive neural network (PIANN), Slit in the Domain.

III (a) : Adaptive learning for inverse problems (20 data-set problem).

III (b) : Adaptive learning for inverse problems (50 data-set problem).

IV (a) : MNIST classification (20 neuron in hidden layers).

IV (b) : MNIST classification (500 neuron in hidden layers).

Details of input parameters used in Algorithm 2.1 is provided in Table 10. \( \ell \) represents the learning rate, \( d_l \) represent the decay of learning rate, and \( E_e \) represents the number of epochs for which each layer is trained. Details of input parameters used in Algorithm 2.2 is provided in Table 11. In Table 11, Epoch \( \{E_e^1, E_e^2\} \) means that the first two networks in Algorithm 2.2 is trained for \( E_e^1 \) number of epochs and the remaining networks are trained for \( E_e^2 \) number of epochs.

\footnote{FC represents fully connected network.}
Table 10
Details of input parameters in Algorithm 2.1 for different problems

| Problem | $\epsilon_\eta$ | $\rho$ | $(\alpha^{(2)}, \tau^{(2)}, \gamma^{(2)})$ | $\ell$ | $d_l$ | $E_e$ | $N_o$ | Batch size |
|---------|-----------------|--------|---------------------------------|-------|------|-------|------|------------|
| I       | 0.035           | $10^{-6}$ | (0.05, 0, 0.06) | 0.001 | 0.9  | 50    | 100  | 70         |
| II (a)  | 0.8             | $10^{-6}$ | (0.001, 10, 0)  | 0.0005| 1    | 1000  | 100  | 500        |
| II (b)  | 0.5             | $10^{-6}$ | (0.001, 5, 0)   | 0.0005| 1    | 2000  | 100  | 500        |
| III (a) | 0.035           | $10^{-6}$ | (0.0001, 0, 0.001) | 0.001 | 1    | 400   | 100  | 20         |
| III (b) | 0.0015          | $10^{-6}$ | (0.0001, 0, 0.001) | 0.001 | 1    | 400   | 100  | 50         |
| IV (a)  | 0.005           | $10^{-6}$ | (0.001, 0, 0.0035) | 0.001 | 0.9  | 100   | 20   | 900        |
| IV (b)  | 0.005           | $10^{-6}$ | (0.001, 0, 0.005) | 0.001 | 0.9  | 100   | 500  | 900        |

Table 11
Details of input parameters in Algorithm 2.2 for different problems

| Problem | $N_n$ | Network $^6$ | Activation | No. of layers | No. of neurons | Epoch $E_e$ | Batch size |
|---------|------|--------------|------------|----------------|----------------|-------------|------------|
| I       | 5    | $FC$         | relu       | 1              | 10             | $\{70, 100\}$ | 70         |
| IV (a)  | 20   | $FC$         | elu        | 2              | 20             | $\{100, 200\}$ | 900        |
| IV (b)  | 4    | $FC$         | elu        | 2              | 500            | $\{200, 400\}$ | 900        |