Deep Neural Networks with Data Dependent Implicit Activation Function

Bao Wang [1] **Xiyang Luo** [1] **Zhen Li** [2] **Wei Zhu** [3] **Zuoqiang Shi** [2] **Stanley Osher** [1]

Abstract

Though deep neural networks (DNNs) achieve remarkable performances in many artificial intelligence tasks, the lack of training instances remains a notorious challenge. As the network goes deeper, the generalization accuracy decays rapidly in the situation of lacking massive amounts of training data. In this paper, we propose novel deep neural network structures that can be inherited from all existing DNNs with almost the same level of complexity, and develop simple training algorithms. We show our paradigm successfully resolves the lack of data issue. Tests on the CIFAR10 and CIFAR100 image recognition datasets show that the new paradigm leads to 20% to 30% relative error rate reduction compared to their base DNNs. The intuition of our algorithms for deep residual network stems from theories of the partial differential equation (PDE) control problems. Code will be made available.

1. Introduction

Over the last few decades, many initialization, optimization, regularization, and many other techniques have been invented (Bengio et al., 2007; Hinton et al., 2006) to make deep neural networks (DNNs) easily applicable to solving challenging artificial intelligence tasks (Lecun et al., 2015). Nevertheless, classical DNNs like VGG networks (Simonyan & Zisserman, 2014) have the problem of degradation, i.e., when the network goes deeper both training and testing errors increase even with sufficient training data (He et al., 2016a). Deep residual networks (ResNets), especially the pre-activated ones (He et al., 2016a;b), proposed by He et al. employ shortcut connections to learn residuals only and keep a clean information path which efficiently solve the aforementioned degradation problem. Furthermore, deep ResNets enable generalization accuracy improvement for networks up to 1000 layers.

Many advances have been made after the emergence of the deep ResNets. These include both theoretical analysis and algorithmic development. In the original work of He et al (He et al., 2016b), the pre-activated ResNets are formulated as discrete dynamical systems. This dynamical system point of view leads to an elegant analysis on the optimality of the identity map in the ResNets. Hardt and Ma (Hardt & Ma, 2017) use matrix factorization techniques to analyze the landscape of the linear ResNets and its representation power. E considers the deep ResNets as a control problem of a class of continuous dynamical systems (E, 2017). New network structures development is as thriving as theoretical analysis. Instead of using a single shortcut to connect two consecutive residual blocks, densely connect convolutional networks employ shortcut connections to connect all the distinct blocks (Huang et al., 2017). The wide residual networks (Zagoruyko & Komodakis, 2016) increase the width of the layers in the original ResNets. Both dense nets and wider ResNets have certain amount of improvement compared to the ResNets. Dual path networks is another family of interesting improvement over both ResNets and dense nets (Chen et al., 2017).

The accuracy of the DNN externally depends on massive amounts of training data. The lack of sufficient training data typically leads to another degradation problem. Significant accuracy reduction tends to occur as the network goes deeper, which will be further demonstrated in this paper. Many regularization techniques explored to attempt to tackle this challenge(Zhu et al., 2017; Srivastava et al., 2014; Wen et al., 2016), but satisfactory results are rare. Most existing strategies can be classified to either loss function regularization or network structure regularization. None of these considered the specificity of the data which is of critical importance in data analysis(Bishop, 2006).

In this paper, we try to solve the issue of lacking enough training data by using the information of data as a prior to train the DNNs. We will first build the connection between the deep ResNets and the partial differential equation (PDE) control problems. Well-posedness theories of PDE control problems suggest us to take data dependent activations in DNNs. To make the data dependent activation

[1]Department of Mathematics, UCLA, Los Angeles, California, USA [2]Department of Mathematical Sciences, Yau Mathematical Sciences Center, Tsinghua University, Beijing, China [3]Department of Mathematics, Duke University, Durham, North Carolina, USA. Correspondence to: Bao Wang <wangbaonj@gmail.com>.

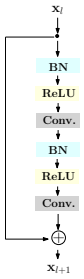

Figure 1. The building block of the pre-activated ResNets.

trainable, we propose surgeries to the existing DNNs to construct new data dependent implicitly activated DNNs. Efficient algorithms to train and test the model are investigated. Numerical results on the CIFAR10 dataset with quite limited and randomly selected instances show great success of our paradigm in solving the challenge of insufficient data. Another successful achievement of our framework is regarding the generalization error reduction. On the CIFAR10 and CIFAR100 datasets we receive 30% and 20% error reduction, respectively, compared to the base DNNs which includes VGG, ResNet, and pre-activated ResNet families. Our method provides an alternative towards model compression which is important for applications on mobile devices.

This paper is structured as follows: In section 2, we present the connection of the PDE control problems with the deep ResNets, and some improvements of deep ResNets motivated by the PDE theory. Data interpolation on a general manifold in a harmonic extension manner is reviewed in section 3. Our DNN surgeries, along with their training/testing procedures, will be presented in section 4. To validate our algorithms, a large variety of numerical results are demonstrated in section 5. The summary of this work and future directions are discussed in section 6.

2. Deep Residual Networks and PDE Control Problem

Deep ResNets, especially the pre-activated ResNets (Pre-ActResNets), are realized by adding shortcut connections to connect the consecutive residual blocks in the classical convolutional neural networks (CNN). This can also be regarded as a cascade of the residual block, as shown in Fig.1, followed by the final flatten and activation layers. Mathematically, a residual block is formulated as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{\mathbf{W}_i\}) + \mathbf{x}, \tag{1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output tensors of the block, the function $\mathcal{F}(\mathbf{x}, \{\mathbf{W}_i\})$ represents the residual mapping to be parametrized.

To build the connection between the deep ResNets and PDE control problems, let us consider the terminal value problem of the linear transport equation in $\mathbb{R}^d$:

$$\begin{cases} \frac{\partial u}{\partial t} + \mathbf{v}(\mathbf{x}, t) \cdot \nabla u = 0 & \mathbf{x} \in \mathbb{R}^d, t \geq 0 \\ u(\mathbf{x}, 1) = f(\mathbf{x}) & \mathbf{x} \in \mathbb{R}^d, \end{cases} \tag{2}$$

where $\mathbf{v}(\mathbf{x}, t)$ is a given velocity field, d is the dimension of the flattened input tensor, f is the composition of the activation function and the fully connected layer. If we use the softmax activation function,

$$f(\mathbf{x}) = \mathbf{softmax}(\mathbf{W}_{FC} \cdot \mathbf{x}), \tag{3}$$

where $\mathbf{W}_{FC}$ is the weight in the fully connected layer, and the softmax function is given by

$$\mathbf{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)},$$

which models the posterior probability of the instance belonging to each class.

It is well-known (Evans, 2010) that the solution at $t = 0$ can be solved along the characteristics:

$$\frac{d\mathbf{X}(t; \mathbf{x})}{dt} = \mathbf{v}\left(\mathbf{X}(t; \mathbf{x}), t\right), \quad \mathbf{X}(0; \mathbf{x}) = \mathbf{x}. \tag{4}$$

We know that along the characteristics, $\mathbf{u}$ is a constant (Set $T = 1$ below):

$$u(\mathbf{x}, 0) = u\left(\mathbf{X}(1; \mathbf{x}), T\right) = f(\mathbf{X}(1; \mathbf{x})).$$

Let $\{t_k\}_{k=0}^L$ with $t_0 = 0$ and $t_L = 1$ be a partition of $[0, 1]$. The characteristics of the transport equation Eq.(4) can be approximately solved by using the simple forward Euler discretization from $\mathbf{X}^0(\mathbf{x}) = \mathbf{x}$:

$$\mathbf{X}^{k+1}(\mathbf{x}) = \mathbf{X}^k(\mathbf{x}) + \Delta t \mathbf{v}(\mathbf{X}^k(\mathbf{x}), t_k), \tag{5}$$

where Δt is the time step. If we choose the velocity field such that

$$\Delta t\, \mathbf{v}(\mathbf{x}, t) = \mathbf{W}^{(2)}(t) \cdot \sigma\left(\mathbf{W}^{(1)}(t) \cdot \sigma(\mathbf{x})\right), \tag{6}$$

where $\mathbf{W}^{(1)}(t)$ and $\mathbf{W}^{(2)}(t)$ corresponds to the "weight" layers in the residual block, $\sigma = \mathrm{ReLU} \circ \mathrm{BN}$, one step in the forward Euler discretization Eq.(5) is equivalent to advancing one layer in the deep ResNets, (see Fig. 1). The numerical solution of the transport equation Eq.(2) at $t = 0$ is given by

$$u(\mathbf{x}, 0) = f(\mathbf{X}^L(\mathbf{x})), \tag{7}$$

which is exactly the output we get from the ResNets.

Let $\mathbf{x}$ be a point from the training data with its label $g(\mathbf{x})$. Training the ResNet is equivalent to finding the parameters in the velocity field Eq.(6) and the terminal value so that the output in Eq.(7) matches the label $g(\mathbf{x})$

In summary, the training process of the deep ResNets can be regarded as solving the following control problem of a transport equation in $\mathbb{R}^d$:

$$\begin{cases} \frac{\partial u}{\partial t} + \mathbf{v}(\mathbf{x},t)\cdot\nabla u(\mathbf{x},t) = 0 & \mathbf{x}\in\mathbb{R}^d, t\geq 0 \\ u(\mathbf{x},1) = f(\mathbf{x}) & \mathbf{x}\in\mathbb{R}^d \\ u(\mathbf{x}_i,0) = g(\mathbf{x}_i) & \mathbf{x}_i\in\mathbf{T}, \end{cases} \tag{8}$$

where $\mathbf{T}$ denotes the training set, $g(\mathbf{x}_i)$ is the label of instance $\mathbf{x}_i$. This control problem is uniquely determined by the terminal value $u(\mathbf{x},1) = f(\mathbf{x})$ and the velocity field $\mathbf{v}(\mathbf{x},t)$. Conventionally, the corresponding terminal value of the transport equation is selected to be a softmax activation function as shown in (3). From the control problem point of view, the softmax function may not be a good terminal condition, since it is pre-determined and maybe far from the real value . The ideal terminal function should be a smooth function and close to the labeled value in the training set . Based on this observation, the weighted nonlocal Laplacian (Shi et al., 2017) seems to provide a good choice for the terminal function.

3. Manifold Interpolation-A Harmonic Extension Approach

In this section, we will briefly discuss smooth interpolation on a general smooth manifold, and give a sufficient condition on the number of samples needed to make sure this interpolation has enough representation diversity. Consider the following interpolation problem: Let $P = \{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n\}$ be a set of points in a manifold $\mathcal{M}\subset\mathbf{R}^d$ and $S = \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\}$ be a subset of P. Suppose we have the labels for the data in S, and we want to extend the label function u to the entire dataset P. Harmonic extension is a natural approach, which minimizes the following Dirichlet energy functional:

$$\mathcal{E}(u) = \frac{1}{2}\sum_{\mathbf{x},\mathbf{y}\in P} w(\mathbf{x},\mathbf{y})\left(u(\mathbf{x}) - u(\mathbf{y})\right)^2, \tag{9}$$

with the boundary condition:

$$u(\mathbf{x}) = g(\mathbf{x}),\ \mathbf{x}\in S,$$

where $w(\mathbf{x},\mathbf{y})$ is a weight function, typically chosen to be the Gaussian weight $w(\mathbf{x},\mathbf{y}) = \exp(-\frac{||\mathbf{x}-\mathbf{y}||^2}{\sigma^2})$, and σ is a scaling parameter.

The Euler-Lagrange equation for Eq.(9) is:

$$\begin{cases} \sum_{\mathbf{y}\in P}\left(w(\mathbf{x},\mathbf{y}) + w(\mathbf{y},\mathbf{x})\right)\left(u(\mathbf{x}) - u(\mathbf{y})\right) = 0 & \mathbf{x}\in P/S \\ u(\mathbf{x}) = g(\mathbf{x}) & \mathbf{x}\in S, \end{cases} \tag{10}$$

which indeed is a boundary value problem for the graph Laplacian (GL). It is observed in the work (Shi et al., 2017), that adding a scale on the GL effectively tames the issue of highly unbalanced data, which leads to the following weighted nonlocal Laplacian (WNLL):

$$\begin{cases} \sum_{\mathbf{y}\in P}\left(w(\mathbf{x},\mathbf{y}) + w(\mathbf{y},\mathbf{x})\right)\left(u(\mathbf{x}) - u(\mathbf{y})\right) + \\ \left(\frac{|P|}{|S|} - 1\right)\sum_{\mathbf{y}\in S} w(\mathbf{y},\mathbf{x})\left(u(\mathbf{x}) - u(\mathbf{y})\right) = 0 & \mathbf{x}\in P/S \\ u(\mathbf{x}) = g(\mathbf{x}) & \mathbf{x}\in S, \end{cases} \tag{11}$$

In order to guarantee the representability of the WNLL, the labeled data should cover all types of instances in the data pool. For this, we give a necessary condition in Theorem 1.

Theorem 1. *(**Representability**) Suppose we have a data pool formed by N classes of data uniformly, with the number of instances of each class to be sufficiently large. If we want all classes of data to be sampled at least once, on average at least $N\left(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{N}\right)$ data need to be sampled from the data pool. In this case, the number of data sampled for each class is $1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{N}$, in expectation.*

Proof. Let $X_i, i = 1, 2, \cdots, N$, be the number of additional data needed to obtain the i-type after $(i-1)$ distinct types have been sampled. The total number of instances needed is:

$$X = X_1 + X_2 + \cdots + X_N = \sum_{i=1}^{N} X_i.$$

For any i, $i-1$ distinct types of instances have already been sampled. It follows that the probability of a new instance being of a different type is $1 - \frac{i-1}{N} = \frac{N-i+1}{N}$. Essentially, to obtain the i-th distinct type, the random variable X follows a geometric distribution with $p = \frac{N-i+1}{N}$ and $E[X_i] = \frac{N}{N-i+1}$. Thus, we have

$$E[X] = \sum_{i=1}^{N} E[X_i] = \sum_{i=1}^{N}\frac{N}{N-i+1}.$$

Asymptotically, $E[X] \approx N\ln N$ for sufficiently large N. □

4. Network Structure & Training Algorithms

Deep ResNets enable hierarchical representation learning which leads to fabulous performance on many artificial intelligence tasks. WNLL is a harmonic extension approach for manifold extension analytically and adaptable even when the labeled instances are extremely scarce. As illustrated in the connection between deep ResNets and PDE control problems, a data dependent terminal value in the control problem, i.e., data dependent activation function in the deep ResNets, should be better than the ad hoc activation functions, e.g., softmax or linear activations. In this section, we

will discuss how to efficiently perform label extension in a harmonic extension manner and put the WNLL into the deep ResNets. The new framework has the following benefits: the deep ResNet to learn optimal representations for the WNLL interpolation; simultaneously with the WNLL as the activation layer, the learned deep representations can be better utilized than the classical activation functions. We will show that this on-the-fly coupling via information feed-forward and error back propagation solves the lack of data issue and achieves great accuracy improvement compared to the existing DNNs.

Let us first discuss the numerical approach to solve the WNLL interpolation given in Eq.(11). The numerical approach is straight forward with two computational burdens involved: finding the weights $w(\mathbf{x}, \mathbf{y})$ for any $\mathbf{x}, \mathbf{y} \in P$, and solving the resulting linear system. To find the pairwise weights, we need to perform the nearest neighbor searching. A brute-force approach is of quadratic scaling, however, there are many fast algorithms with sub-linear scaling for this purpose, e.g., KD-Tree, Ball-Tree, etc. Here we adopt the approximate nearest neighbor (ANN) searching algorithm which is scalable to extremely large scale and high dimensional data(Muja & Lowe, 2014). The resulted linear system is sparse and positive definite which is efficiently solved by the conjugate gradient method in this work. It is worth emphasizing that in order to guarantee the WNLL interpolation is suitable to represent all classes of instances, the labeled instances should be at least around $N \ln N$ with N be the number of classes.

The most important component of our algorithm is to put the WNLL activation layer into the DNNs, and design efficient algorithms for information feed-forward propagation and error back-propagation. Our information and error propagation paths are demonstrated in Fig.2. The standard DNN, e.g., VGG, ResNet, is plotted in Fig.2 (a), the 'DNN' block represents all the layers except the last softmax activation function. A naive approach to place the WNLL into the DNN is to simply replace the softmax function by WNLL. However, in this case though the information can be feed-forwarded, the error cannot be back-propagated, since WNLL implicitly defines an activation function on the learned representation, and the gradient is not explicitly available. To efficiently train the network with WNLL activation, we introduce a new structure inherited from the standard DNN as depicted in chart (c) of Fig.2. Our structure is quite flexible which can be inherited from any existing DNN. We equip two new blocks to the standard DNN, a buffer block and a WNLL activation function, where the buffer block is simply chosen to be a composition of a fully connected layer which preserves the dimension of the input tensor followed by a ReLU function(Nair & Hinton, 2010). The buffer block can be made more complicated. After the buffer block, the tensor is passed into two activations, a linear activation and a WNLL function, in parallel. Our training algorithm for the network in Fig.2(c) is an iterative procedure among the following three steps:

- **Step 1.** Train the network with only the linear activation functions to steady state. For this purpose, we do not feed the data to the WNLL activation.
- **Step 2.** Run a few training epochs on the network which we freeze the "DNN" and "Linear Activation" blocks, and only fine tune the 'Buffer Block'. In order to back-propagate the error between the ground-truth and the WNLL interpolated results, we feed the data into the pre-trained linear activation function, and use the corresponding computational graph to perform error back-propagation.
- **Step 3.** Unfreeze the entire network, and train the network with data only feeding to the linear activation to the steady state again.

With the trained network, during the generalization step, we use the WNLL activation to get the final inference. Our algorithm is designed in a greedy fashion. The following numerical results validate the efficiency of our training algorithm and the superiority of our network structure. The training and testing procedures of our proposed network are summarized in Algorithms 1 and 2, respectively.

Remark 1. *During the back-propagation, we use the computational graph of the linear function to approximate that of the WNLL function since the linear functions are the simplest nontrivial harmonic functions. Mixed Gaussian seems to be a more appealing approximation, since it is compatible with the WNLL. We will continue to explore better approximations in our subsequent work.*

5. Numerical Results

To validate the accuracy, efficiency, and robustness of the proposed model, we present the numerical results of different tests on the CIFAR10 and CIFAR100 (Krizhevsky, 2009), MNIST(LeCun, 1998) and SVHN dataset(Netzer et al., 2011). It is generally believe that the difficulty of these datasets are ranked as CIFAR100 followed by CIFAR10, then SVHN, and the easiest one is MNIST. In all of our numerical experiments, we take the standard data augmentation that is widely used for both CIFAR datasets (He et al., 2016a; Huang et al., 2017; Zagoruyko & Komodakis, 2016). For MNIST and SVHN, we use the raw data without any data augmentation. In order to use the computational graph of the linear function to approximate that of WNLL, we need the dynamical computational graph. For this purpose, we implement our algorithm on the PyTorch platform (Paszke et al., 2017), where automatic differentiation is used

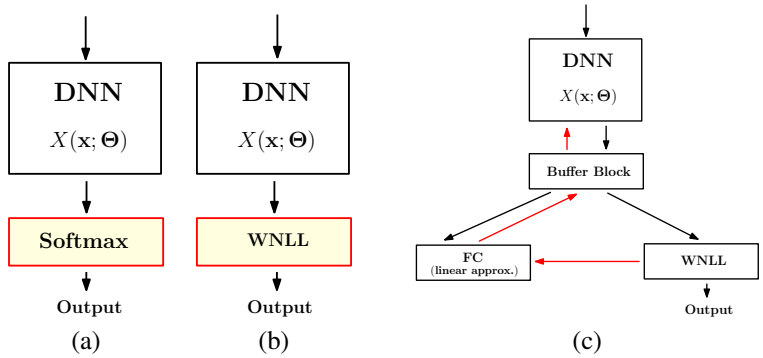

Figure 2. Illustration of the DNN architectures. Panel (a) depicts the standard DNN, where the "DNN" block represents the layers except the last activation layer in the network. Panel (b) plots the standard network with the last layer replaced by the WNLL layer. Panel (c) shows the network structure used in our work, detailed explanation is presented in the paper.

instead of the symbolic computation used in TensorFlow (Abadi et al., 2016) and many other systems. All computations are done on a machine using only a single Nvidia Titan Xp graphics card.

Before diving into the performance of the DNNs, we first compare the performance of WNLL interpolation and other shallow classifiers on different datasets. Table 1 lists the performance of k-nearest neighbors (KNN) (the optimal k is listed in the table), support vector machine (SVM) with RBF kernel, softmax regression and WNLL interpolation function. For the WNLL interpolation, in order to speed up the computation, we only keep 15 nearest neighbors, and the 8th neighbor's distance is used to normalize the weight matrix. Both KNN and WNLL can be regarded as nonparametric approaches. WNLL outperforms the other methods except SVM. KNN, in general, is better than softmax regression which demonstrates the importance of the manifold structure of the data. These results show the potential of using WNLL instead of softmax as the activation function in the DNNs.

Table 1. Accuracy of simple classifiers over different datasets

Dataset	KNN	SVM	Softmax	WNLL
Cifar10	32.77% (k=5)	57.14%	39.91%	40.73%
MNIST	96.40% (k=1)	97.79%	92.65%	97.74%
SVHN	41.47% (k=1)	70.45%	24.66%	56.17%

We run 400 epochs when training the vanilla DNN, i.e., standard DNN, with the initial learning rate being 0.05 and halved after every 50 epochs for Cifar10 and Cifar100 datasets. We also train 5 epochs of the WNLL DNN, i.e., WNLL activated DNN. Since at this stage, the DNN is already well trained, we use a smaller learning rate 0.0005. For the Cifar datasets, these hyper-parameters are chosen based on cross validation. We keep alternating the above three steps with the learning rate being one fifth of that in the previous stages. The batch size in training the vanilla DNN is set to 128 for all experiments. In the SVHN experiments, we use the same hyperparameters as reported in (Huang et al., 2016). The batch size is set to 2000 when training the WNLL DNN. By Theorem 1, this number is big enough to sample all types of instances for even CIFAR100 with 100 classes of images. All the optimizations are carried out by a simple SGD solver with the default Nesterov momentum acceleration.

5.1. Resolving the Challenge of the Lack of Training Data

When we do not have sufficient training data, the generalization accuracy typically decays as the network goes deeper. This phenomenon is illustrated in Fig.3. The left and right panels plot the cases when the first 1000 and 10000 data in the training set of CIFAR10 are involved in training the vanilla and WNLL DNNs. We believe the increase of the generalization error is due to the data not being sufficient to parametrize the deep networks. With suitable regularization techniques, the deep networks can be better parametrized by a small amount of training data. WNLL activation which involves the information of the data's geometric structures is one of such regularizers. With the WNLL activation, the generalization error rate decays persistently as the network goes deeper. The generalization accuracy between the vanilla and WNLL DNN can differ up to 10 percent within our testing regime.

Even though we have only built the connection between the deep ResNets and the PDE control problems, we also test the performance of our surgeries and algorithms on other base DNNs, e.g., VGG networks. In table 2 we list the generalization error rates of 15 different DNNs from VGG, ResNet, Pre-activated ResNet families on the entire, first 10000, 5000, and 1000 instances in the CIFAR10 training set. It is easy to see that WNLL activated DNNs typically has much more accuracy improvement for ResNets or pre-

Table 2. Generalization error rate over the test set of the vanilla DNNs and the WNLL activated ones trained over the entire, the first 10000, 5000, and 1000 instances of the training set of CIFAR10. (Median of 5 independent trials)

Network	Whole		10000		5000		1000	
	Vanilla	**WNLL**	**Vanilla**	**WNLL**	**Vanilla**	**WNLL**	**Vanilla**	**WNLL**
VGG11	9.23%	7.35%	10.37%	8.88%	12.36%	10.49%	26.75%	24.10%
VGG13	6.66%	5.58%	9.12%	7.64%	10.89%	9.02%	24.85%	22.56%
VGG16	6.72%	5.69%	9.01%	7.54%	11.25%	9.13%	25.41%	22.23%
VGG19	6.95%	5.92%	9.62%	8.09%	11.76%	9.22%	25.70%	22.87%
ResNet20	9.06%	7.09%	12.83%	9.96%	14.30%	11.24%	34.90%	29.91%
ResNet32	7.99%	5.95%	11.18%	8.15%	12.75%	10.63%	33.41%	28.78%
ResNet44	7.31%	5.70%	10.66%	7.96%	11.84%	10.14%	34.58%	27.94%
ResNet56	7.24%	5.61%	9.83%	7.61%	12.39%	10.17%	37.83%	28.18%
ResNet110	6.41%	4.98%	8.91%	7.13%	13.45%	10.05%	42.94%	28.29%
ResNet18	6.16%	4.65%	8.26%	6.29%	10.38%	8.53%	27.02%	22.48%
ResNet34	5.93%	4.26%	8.31%	6.11%	10.75%	8.65%	26.47%	20.27%
ResNet50	6.24%	4.17%	9.64%	6.49%	12.96%	8.76%	29.69%	20.19%
PreActResNet18	6.21%	4.74%	8.20%	6.61%	10.64%	8.18%	27.36%	21.88%
PreActResNet34	6.08%	4.40%	8.52%	6.34%	10.85%	8.44%	23.56%	19.02%
PreActResNet50	6.05%	4.27%	9.18%	6.05%	10.64%	8.35%	25.05%	18.61%

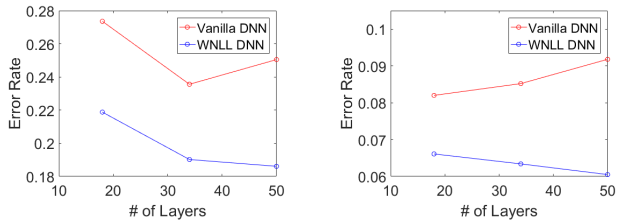

Figure 3. Taming of the degeneration problem of vanilla DNN by WNLL activated DNN. Panels (a) and (b) plot the generation error when 1000 and 10000 training data are used to train the vanilla and the WNLL activated DNN, respectively. In each plot, we test three different networks: PreActResNet18, PreActResNet34, and PreActResNet50. It is easy to see that when the vanilla network becomes deeper, the generation error does not decayed, while WNLL activation resolves this degeneracy. All tests are done on the Cifar10 dataset.

activated ResNets. Significant accuracy improvement can still be observed when our surgeries and training algorithms are applied to the VGG networks. Except for VGG networks, we can achieve relatively 20% to 30% testing error rate reduction. All the results presented here and in the rest of this paper are the median of 5 independent trials to reduce the influence of stochasticity.

5.2. Error Rate Reduction in Base DNNs

We next present the superiority of our deep network in terms of the generalization accuracy when compared to its base network. Figure. 4 plots the generalization accuracy evolution during the training procedure. Panels (a) and (b) plot the cases for the ResNet50 and WNLL activated ResNet50 when only the first 1000 CIFAR10 training data are utilized. Charts (c) and (d) are for the cases when the first 10000 CIFAR10 training instances are used to train the vanilla pre-activated ResNet50 and WNLL activated version. After around 300 epochs, the accuracies of the vanilla DNNs plateaued and cannot improve any more. However, in stage 2, once we use the WNLL activation, there is a jump in the generalization accuracy; during stage 3, even though initially there is an accuracy reduction, with the training continuing, the accuracy keeps climbing for a while. The generalization accuracy increases if finally we use the WNLL as the activation function.

For the street view house number recognition (SVHN) task, we simply test the performance when the full training data are used. Here we only test the performance of the ResNets and pre-activated ResNets. There is a relatively 7%-10% error rate reduction for all these DNNs. There is relatively more improvement on pre-activated ResNets than on ResNets, this is consistent with our basic PDE control problem ansatz.

Tables 2 and 4 list the error rate of 15 different vanilla networks and the WNLL activated networks. On CIFAR10, the WNLL activated DNNs outperformed the vanilla ones with around 1.5% to 2.0% absolute, or 20% to 30% relative error rate reduction. We reproduced the results of the vanilla DNNs on both datasets. Our results are consistent with the original reports and other researchers' reproductions (He et al., 2016a;b; Huang et al., 2017). Interestingly, when the task become harder, our improvement becomes more

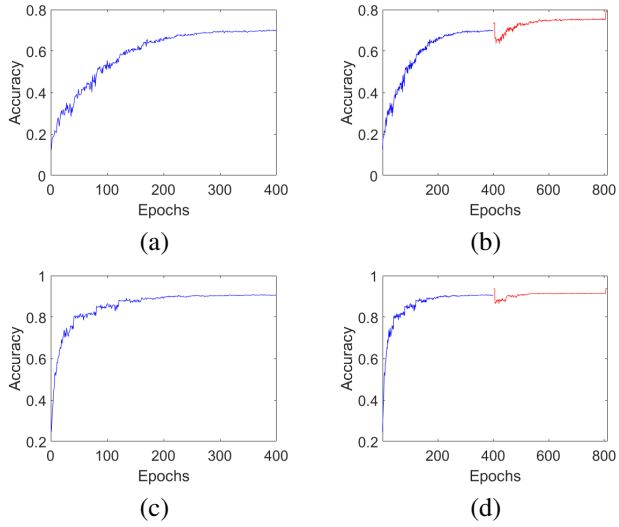

Figure 4. The evolution of the generation accuracy over the training procedure. Charts (a) and (b) are the accuracy plots for ResNet50 with 1000 training data, where (a) and (b) are plots for the epoch v.s. accuracy of the vanilla and the WNLL activated DNN. Panels (c) and (d) correspond to the case of 10000 training data for PreActResNet50. All tests are done on the Cifar10 dataset.

Table 3. Error rate of the vanilla DNN v.s. the WNLL activated DNN over the whole SVHN dataset. (Median of 5 independent trials)

Network	Vanilla DNN	WNLL DNN
ResNet20	3.76%	3.44%
ResNet32	3.28%	2.96%
ResNet44	2.84%	2.56%
ResNet56	2.64%	2.32%
ResNet110	2.55%	2.26%
ResNet18	3.96%	3.65%
ResNet34	3.81%	3.54%
PreActResNet18	4.03%	3.70%
PreActResNet34	3.66%	3.32%

significant. This builds our confident in trying harder tasks in the future. Reducing the sizes of DNN models is an important direction to make the DNN applicable for generalize purpose, e.g., auto-drive, mobile intelligence, etc. So far the most successful attempt is DNN weights quantization(Bengio & David, 2015). Our approach is a new direction for reducing the size of the model: to achieve the same level of accuracy, compared to the vanilla networks, our model's size can be tens of times smaller.

Table 4. Error rate of the vanilla DNN v.s. the WNLL activated DNN over the whole CIFAR100 dataset. (Median of 5 independent trials)

Network	Vanilla DNN	WNLL DNN
VGG11	32.68%	28.80%
VGG13	29.03%	25.21%
VGG16	28.59%	25.72%
VGG19	28.55%	25.07%
ResNet20	35.79%	31.53%
ResNet32	32.01%	28.04%
ResNet44	31.07%	26.32%
ResNet56	30.03%	25.36%
ResNet110	28.86%	23.74%
ResNet18	27.57%	22.89%
ResNet34	25.55%	20.78%
ResNet50	25.09%	20.45%
PreActResNet18	28.62%	23.45%
PreActResNet34	26.84%	21.97%
PreActResNet50	25.95%	21.51%

6. Concluding Remarks

Motivated by the connection between deep ResNets and PDE control problems, we propose a novel DNN structure which can be inherited from any existing DNN. An end-to-end greedy styled, multi-staged training algorithm is proposed to train the novel networks. In order to efficiently back propagate the errors, we utilized the computational graph of a linear function dynamically to approximate that for the manifold interpolation function. On one hand, our new framework resolves the issue of the lack of big training data, on the other hand, it provides great accuracy improvement compared to the base DNNs. This improvement is consistent for networks with different depths. Utilizing our structure, it is very easy to get near state-of-the-art results with very small model, which has great potential for the mobile device applications. Nevertheless, there are many directions for improvement: the current manifold interpolation is still one of the computational bottlenecks, according to the representability theorem for the data with many classes. For instance, the batch size need to be very large for the ImageNet dataset (J. et al., 2009), which poses memory challenges. Another important issue is the approximation of the gradient of the WNLL activation function. Linear function is an option but it is far from optimal. We believe a better harmonic function approximation can further lift the model's performance.

7. Acknowledgement

This material is based, in part, upon work supported by the U.S. Department of Energy, Office of Science and by National Science Foundation, and National Science Foundation of China, under Grant Numbers DOE-SC0013838 and DMS-1554564, (STROBE), NSFC 11671005.

References

Abadi, M., Agarwal, A., and et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv:1603.04467*, 2016.

Bengio, M. Courbariaux Y. and David, J. Binaryconnet: Training deep neural networks with binary weights. *NIPS*, 2015.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. *NIPS*, 2007.

Bishop, C. M. Pattern recognition and machine learning. *Springer*, 2006.

Chen, Yunpeng, Li, Jianan, Xiao, Huaxin, Jin, Xiaojie, Yan, Shuicheng, and Feng, Jiashi. Dual path networks. *NIPS*, 2017.

E, Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

Evans, L.C. Partial differential equations. *American Mathematical Soc*, 2010.

Hardt, Moritz and Ma, Teng Yu. Identity matters in deep learning. *ICLR*, 2017.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CVPR*, 2016a.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Identity mappings in deep residual networks. *ECCV*, 2016b.

Hinton, G. E., Osindero, S., and Teh, T. W. A fast learning algorithm for deep belief nets. *Neural Computation*, 18 (7):1527–1554, 2006.

Huang, Gao, Sun, Yu, Liu, Zhuang, Sedra, Daniel, and WeinBerger, Kilian. Deep networks with stochastic depth. *ECCV*, 2016.

Huang, Gao, Liu, Zhuang, Weinberger, K. Q., and van der Maaten, Laurens. Densely connected convolutional networks. *CVPR*, 2017.

J., Deng, W., Dong, R., Socher, L.-J., Li, K., Li, and L., Fei Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Krizhevsky, Alex. Learning multiple layers of features from tiny images. 2009.

Langley, P. Crafting papers on machine learning. In Langley, Pat (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

LeCun, Yann. The mnist database of handwritten digits. 1998.

Lecun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521:436–444, 2015.

Muja, Marius and Lowe, David G. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence (PAMI)*, 36, 2014.

Nair, Vinod and Hinton, Geoffrey. Rectified linear units improve restricted boltzmann machines. *ICML*, 2010.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised features learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. Automatic differentiation in pytorch. 2017.

Shi, Z., Osher, S., and Zhu, W. Weighted nonlocal Laplacian on interpolation from sparse data. *Journal of Scientific Computing*, pp. to appear, 2017.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *Arxiv:1409.1556*, 2014.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Wen, Yandong, Zhang, Kaipeng, Li, Zhifeng, and Qian, Yu. A discriminative feature learning approach for deep face recognition. *ECCV*, 2016.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *BMVC*, 2016.

Zhu, Wei, Qiu, Qiang, Huang, Jiaji, Carderbank, Robert, Sapiro, Guillermo, and Daubechies, Ingrid. Low dimensional manifold regularized neural networks. *UCLA CAM Report: 17-66*, 2017.

Algorithm 1 DNN with Data Dependent Activation: Training Procedure.

Input: Training set (T, l_T), where T is the set of features, l_T is the label set.
Output: An optimized DNN with our surgeries, denoted as DNN_s.
for iter $= 1,\ldots, N$ (where we let $N = 2$ for the entire data, $N = 5$ for the selected subset of the training set. **do**
Train the DNN with linear activation starting from the previous iteration. In the first step use the default initialized one. Denote the temporary model as DNN_1.
Split T into training and validation parts, $T \doteq T^T \bigcup T^V$. We will treat instances in T^V as unlabeled data when training the model.
Partition T^T and T^V into nbatch1 and nbatch2 mini-batches, denoted as $\{T_i^T\}_{i=1}^{\mathrm{nbatch1}}$ and $\{T_i^V\}_{i=1}^{\mathrm{nbatch2}}$, where nbatch1 and nbatch2 are given integers.
for $i = 1, 2, \cdots, \mathrm{nbatch2}$ **do**
for $j = 1, 2, \cdots, \mathrm{nbatch1}$ **do**
Apply the model DNN_1 to $T_j^T \bigcup T_i^V$ to get $\hat{T}_j^T \bigcup \hat{T}_i^V = \mathrm{DNN}_1(T_j^T \bigcup T_i^V)$.
Apply the WNLL interpolation to $\hat{T}_j^T \bigcup \hat{T}_i^V$ by solving the linear system

$$\sum_{\mathbf{x}_n \in T_j^T \bigcup T_i^V} (w_{mn} + w_{nm})(u_m - u_n) + \frac{|T_j^T \bigcup T_i^V|}{|T_i^V|} \sum_{\mathbf{x}_n \in T_i^V} w_{nm}(u_m - l_n) = 0, \quad \forall \mathbf{x}_m \in T_j^T \bigcup T_i^V.$$

to obtain the inferred label $u^j_{T_i^V}$
end for
Voting on $\{u^j_{T_i^V}\}_{j=1}^{\mathrm{nbatch1}}$ to get the inferred label $u_{T_i^V}$.
Back-propagate the $\mathrm{loss}(l_{T_i^V}, u_{T_i^V})$ by using the computational graph of the linear activation function, and update DNN_1. For generally classification tasks, the loss is selected to be cross entropy between the exact and predicted labels.
end for
end for

Algorithm 2 DNN with Data Dependent Activation: Testing Procedure.

Input: Testing set W and training set (T, l_T), where W and T are features, l_T are the labels of the instances in T. And the trained DNN with our surgeries, denoted as DNN_s.
Output: The predicted labels for the test set W.
Partition T and W into nbatch1 and nbatch2 number of mini-batches, denoted as $\{T_i\}_{i=1}^{\mathrm{nbatch1}}$ and $\{W_i\}_{i=1}^{\mathrm{nbatch2}}$, where nbatch1 and nbatch2 are given integers.
for $i = 1, 2, \cdots, \mathrm{nbatch2}$ **do**
for $j = 1, 2, \cdots, \mathrm{nbatch1}$ **do**
Apply the model DNN_s to $T_j \bigcup W_i$ to get $\hat{T}_j \bigcup \hat{W}_i = \mathrm{DNN}_s(T_j \bigcup W_i)$
Apply the WNLL interpolation to $\hat{T}_j \bigcup \hat{W}_i$ by solving the linear system

$$\sum_{\mathbf{x}_n \in T_j \bigcup W_i} (w_{mn} + w_{nm})(u_m - u_n) + \frac{|T_j \bigcup W_i|}{|W_i|} \sum_{\mathbf{x}_n \in W_i} w_{nm}(u_m - l_n) = 0 \quad \forall \mathbf{x}_m \in T_j \bigcup W_i.$$

to obtain the inferred label $u^j_{W_i}$
end for
Voting on $\{u^j_{W_i}\}_{j=1}^{\mathrm{nbatch1}}$ to get the inferred label u_{W_i}.
Output the predicted label $u_W = \bigcup_{i=1}^{\mathrm{nbatch2}} u_{W_i}$.
end for