



# Neural Ordinary Differential Equations with Envolutionary Weights

Lingshen He, Xingyu Xie, and Zhouchen Lin<sup>(✉)</sup>

Key Laboratory of Machine Perception, School of EECS,  
Peking University, Beijing, China  
[zlin@pku.edu.cn](mailto:zlin@pku.edu.cn)

**Abstract.** Neural networks have been very successful in many learning tasks, for their powerful ability to fit the data. Recently, to understand the success of neural networks, much attention has been paid to the relationship between differential equations and neural networks. Some research suggests that the depth of neural networks is important for their success. However, the understanding of neural networks from the differential equation perspective is still very preliminary. In this work, also connecting with the differential equation, we extend the depth of neural networks to infinity, and remove the existing constraint that parameters of every layer have to be the same by using another ordinary differential equation(ODE) to model the evolution of the weights. We prove that the ODE can model any continuous evolutionary weights and validate it by an experiment. Meanwhile, we propose a new training strategy to overcome the inefficiency of pure adjoint method. This strategy allows us to further understand the relationship between ResNet with finite layers and that with infinite layers. Our experiment indicates that the former can be a good initialization of the latter. Finally, we give a heuristic explanation on why the new training method works better than pure adjoint method. Further experiments show that our neural ODE with evolutionary weights converges faster than that with fixed weights.

**Keywords:** Neural network · ODE · Adjoint method

## 1 Introduction

Deep neural networks have become a very powerful tool in machine learning [1]. The great success of deep neural networks mainly attributes to its ability to extract useful features from data by end-to-end learning.

The performance of neural networks with different structures vary greatly even on the same dataset. For example, convolutional neural networks (CNNs)

The first author is a student.

---

**Electronic supplementary material** The online version of this chapter ([https://doi.org/10.1007/978-3-030-31654-9\\_51](https://doi.org/10.1007/978-3-030-31654-9_51)) contains supplementary material, which is available to authorized users.

achieves the state-of-the-art results on object recognition and detection [14, 24] while the fully connected networks fail. Unfortunately, the universal structure design rules remain unknown. Many researchers make efforts in order to better understand deep learning structures. The works in [26] visualized the learning procedure during training, which tried to figure out the connection between network structures and extracted features visually. However, neural networks visualization may not provide a theoretically sound designing guidance for good neural networks. Another line of works focus on the landscape of deep models. A lot of interesting theoretical results have been established to analyze the loss surface of neural networks [15, 29]. However, the current theoretical analysis is still very preliminary as it normally imposes strong assumptions on data and shallow neural networks. So the theoretical analysis cannot provide designing guidance either.

One common problem of all the above investigations is that they cannot describe the behavior of networks when the depth approaches infinity. In general, it is hard to analyze ultra deep networks, since the instabilities will cause the over-expansion or decline of the network output [6]. Another problem is that the previous theoretical works cannot perfectly match the empirical success of ResNet [10] which is one of the most successful CNN structures. To understand the success of ResNet and its variants [25, 27], the works [4, 7] try to interpret ResNet from the perspective of dynamic systems. However, their analysis is still very preliminary, and the theoretical exploration on ResNet is still ongoing [8, 11].

Recently, an interesting and theoretically sound combination of ordinary differential equation (ODE) and neural network architecture provides a new perspective on understanding ultra deep ResNet [3, 4, 22]. It is natural to adopt a differential equation to parameterize the continuous dynamics of hidden units in ResNet because the residual block of ResNet can be written as:

$$\mathbf{u}_{n+1} = \mathbf{u}_n + f_n(\mathbf{u}_n), \quad (1)$$

where  $\mathbf{u} \in \mathbb{R}^n$  and  $f_n$  is a continuous function. Equation (1) is exactly one step of forward Euler discretization in ODE. Actually, many neural networks can be viewed as numerical ODEs [25, 28, 30]. Conversely, different types of numerical schemes of ODE can also be translated as neural networks [22].

In practice, we observe that a deeper ResNet obtains better performance and the  $\ell_2$ -norm of  $f_n(\mathbf{u}_n)$  becomes smaller as the network goes deeper [2]. In other words, a deeper ResNet resembles an ODE numerical scheme with a smaller time step. Based on these observations, in this paper we propose an ODE system—*Neural Ordinary Differential Equations with Envolutionary Weights* (NODE-EW), which is a continuous version of a deep ResNet. Our NODE-EW makes analyzing ultra deep ResNet possible. In our system, ODE is used in two places. Besides the discrete version ODE shown in Eq. (1), we also model the  $\theta_n$  as a trajectory of ODE, where  $\theta_n$  is the parameters of the function  $f_n$ . The double using of ODE brings two advantages. Besides larger model capacity and more representation power, the practical ResNet and its variants can be more accurately modeled by our NODE-EW than by the previous ODE works [3], in which

$f_n = f$  for all  $n$ , where  $f$  is a shared function. Moreover, we rigorously prove that the ODE system proposed in [3] can approximate any continuous trajectory, which implies that NODE-EW is very general and has more expressive power than the ODE system in [3] since the latter uses fixed weights. In general, numerical methods for solving ODE consume too many iteration steps. To make a trade off between memory and efficiency, we propose a backpropagation initialized training strategy to train our NODE-EW. To be specific, we discretize the ODE system w.r.t.  $\theta_n$  and first train the weights in the corresponding neural networks by the gradient-based method. With such an initialization, we adopt the adjoint method to fine-tuning the weights. Notably, the speed of the proposed training strategy is much faster than the pure adjoint method. Our training method bridges the popular continuous and discrete training strategies and provides a new insight into solving ODEs numerically. Another notable addition of the proposed ODE system is its memory efficiency. The consumed memory of NODE-EW is independent of the depth, hence it allows the network depth to go to infinity. At last, our network is reversible, which is important for mobile devices. We summarize our contributions as follows:

1. We propose an ODE system—NODE-EW, which is a continuous version of ResNet. Specifically, we model the learnable weights of residual blocks as a trajectory of ODE, by which NODE-EW obtains larger model capacity and describes the behavior of deep ResNet more accurately than the neural ODE with fixed weights [3].
2. To address the time-consuming issue of adjoint method, we propose a new training strategy which combines the backpropagation and the adjoint method. Our strategy bridges the continuous and discrete training methods.
3. We prove that our ODE system is able to approximate any continuous trajectory. We also validate the expressive power of our ODE system empirically.

## 2 Preliminaries and Related Work

### 2.1 ResNet and Its Variants

Resnet [10] is a deep neural network consisting of multiple residual blocks. It contains skip-connections bypassing residual layers. It solves the gradient explosion/vanishing and accuracy degradation issues [9] existing in the deep neural networks. After the success of ResNet, lots of variants of ResNet appear. Some adjust the structure of residual block [25], then the single convolution becomes a multi-branch convolution. Others create novel topological structures to replace the skip-connections [28, 30].

### 2.2 Learning Partial Differential Equations

In the past few decades, partial differential equations (PDEs) have been very popular in low-level image processing. However, designing the PDE systems by hand requires a lot of tricks and prior knowledge. To deal with this problem,

[19, 20] proposed frameworks to learn PDEs from data. They use the differential invariants that are invariant under rotation and translation as bases of PDE and learn the coefficient from data using the adjoint method. [5] also indicated that PDE systems can learn good features in face recognition. [18, 21] first used learnt PDEs with learnable boundary conditions for saliency detection and object tracking. So far, the learning-based PDEs have been used in many image processing tasks, such as image denoising, color to gray and demosaicing [16, 17, 19].

## 2.3 Neural ODE

Yet, the previous systems [19, 20] discretize the differential equations first and then learn the weights at each time step by the adjoint method. [3] proposed another learning system called *Neural ODE*, which does not discretize the differential equation and updates the weights by the adjoint method directly. It reduces the memory consumption to  $O(1)$ . Neural ODE can be viewed as ResNet with infinite layers. From this perspective, the weights of ResNet at every layer should be different, however, those of Neural ODE are the same. So Neural ODE is not a continuous version of ResNet in a more rigorous sense.

### 2.3.1 Overview of Neural ODE

We introduce Neural ODE [3] in this section. Given  $\mathbf{x}_0$  and  $\mathbf{y} \in \mathbb{R}^n$  as the input and the output, respectively, Neural ODE reads as:

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t, \boldsymbol{\theta}), \quad \forall t \in [0, T],$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is a continuous function and  $\boldsymbol{\theta}$  is the parameters of  $f$ . Starting from the input layer  $\mathbf{x}(0) = \mathbf{x}_0$ , we can let the output layer  $\mathbf{x}(T) = \mathbf{y}$  be the output of our system. The above equation is solved by a black-box ODE solver with desired accuracy.

### 2.3.2 Connection with ResNet

In Neural ODE,  $f$  is chosen as a neural network. The discretized version of Neural ODE is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \cdot f(\mathbf{x}_k, k\Delta t, \boldsymbol{\theta}), \quad (2)$$

where  $\Delta t$  is the time step of the numerical scheme and  $\mathbf{x}_k$  means the numerical solution at time  $k\Delta t$ . Equation (2) is equivalent to a special type of ResNet whose weights  $\boldsymbol{\theta}$  are identical at every layer.

## 3 Neural ODE with Envolutionary Weights

In this section, we provide our ODE system which is a continuous version of the general deep ResNet. As mentioned above, although the weights of residual function  $f$  in Neural ODE are learnable, the identical  $f$  at different layers limits the expressive power of this ODE system and thus Neural ODE is actually not a continuous version of ResNet in a rigorous sense.

### 3.1 Formulation

To address the issue, we propose a new ODE system named *Neural Ordinary Differential Equations with Evolutionary Weights* (NODE-EW). In NODE-EW, the function  $f$  can vary at different time, i.e.,  $\boldsymbol{\theta}$  is a function of time  $t$ :

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t, \boldsymbol{\theta}(t)), \quad \forall t \in [0, T],$$

where  $\mathbf{x}(0) = \mathbf{x}_0$ . In order to find the optimal function  $\boldsymbol{\theta}(t)$ , we aim to solve a standard optimal control problem with a loss function  $L$ .  $L$  can be an arbitrary continuous loss such as mean squared error or cross entropy. The optimization problem is:

$$\min_{\boldsymbol{\theta}(t)} L(\mathbf{x}(T), \mathbf{y}). \quad (3)$$

In order to make the function space computable, here we use another Neural ODE to model the evolution of  $\boldsymbol{\theta}(t)$ . Given  $\boldsymbol{\theta}(0) = \boldsymbol{\theta}_0$ , we assume

$$\frac{d\boldsymbol{\theta}(t)}{dt} = g(\boldsymbol{\theta}(t), t, \boldsymbol{\theta}_1), \quad (4)$$

where  $g$  is another neural network and  $\boldsymbol{\theta}_0$  and  $\boldsymbol{\theta}_1$  are the parameters to characterize  $\boldsymbol{\theta}(t)$ .

### 3.2 Training NODE-EW

#### 3.2.1 Adjoint Method

As mentioned in [3], the relative error of the ODE solver depends on the magnitude of time step  $\Delta t$ . When there is a strict requirement on the error of the system, there will be a lot of intermediate steps which will consume a lot of memory if the system is trained with backpropagation. So [3] proposed the adjoint method. This method makes the systems require only memory of  $O(1)$ , despite its infinite layers. In contrast, an ordinary residual networks requires the memory proportional to its number of layers.

The adjoint of the system is defined as:  $\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}(t)}$  and  $\mathbf{b}(t) = \frac{\partial L}{\partial \boldsymbol{\theta}(t)}$ . The following ODEs are satisfied:

$$\begin{aligned} \frac{d\mathbf{a}(t)}{dt} &= - \left( \frac{\partial f(\mathbf{x}(t), \boldsymbol{\theta}(t))}{\partial \mathbf{x}(t)} \right)^\top \mathbf{a}(t), \\ \frac{d\mathbf{b}(t)}{dt} &= - \left( \frac{\partial g(\boldsymbol{\theta}(t), \boldsymbol{\theta}_1)}{\partial \boldsymbol{\theta}(t)} \right)^\top \mathbf{b}(t). \end{aligned} \quad (5)$$

According to the definition,  $\frac{\partial L}{\partial \mathbf{x}(0)} = \mathbf{a}(0)$  and  $\frac{\partial L}{\partial \boldsymbol{\theta}_0} = \mathbf{b}(0)$ . Computing the derivative of  $L$  with respect to  $\boldsymbol{\theta}_1$  requires evaluation of an integral

$$\frac{\partial L}{\partial \boldsymbol{\theta}_1} = \left( \int_0^T \frac{\partial g(\boldsymbol{\theta}(t), \boldsymbol{\theta}_1)}{\partial \boldsymbol{\theta}_1} \right)^\top \mathbf{b}(t).$$

If we define  $\mathbf{c}(t)$  to satisfy the following equation:

$$\frac{d\mathbf{c}(t)}{dt} = - \left( \frac{\partial g(\boldsymbol{\theta}(t), \boldsymbol{\theta}_1)}{\partial \boldsymbol{\theta}_1} \right)^\top \mathbf{b}(t), \quad \mathbf{c}(T) = 0.$$

Then  $\mathbf{c}(0) = \frac{\partial L}{\partial \boldsymbol{\theta}_1}$ . We can summarize the above equations in a more concise way. Define the function *aug\_dynamics* to be:

$$\begin{aligned} & \text{aug\_dynamics}([\mathbf{x}(t), \mathbf{a}(t), \mathbf{b}(t), \mathbf{c}(t), t, \boldsymbol{\theta}_1]) \\ &= \left[ f(\mathbf{x}(t), t, \boldsymbol{\theta}), - \left( \frac{\partial f}{\partial \mathbf{x}} \right)^\top \mathbf{a}(t), - \left( \frac{\partial g}{\partial \boldsymbol{\theta}(t)} \right)^\top \mathbf{b}(t), - \left( \frac{\partial g}{\partial \boldsymbol{\theta}_1} \right)^\top \mathbf{b}(t) \right]. \end{aligned} \quad (6)$$

To compute the derivative of  $L$  respect to various quantities of the system, we need to solve the following ODE terminal value problem:

$$\begin{aligned} \frac{d}{dt}[\mathbf{x}(t), \mathbf{a}(t), \mathbf{b}(t), \mathbf{c}(t)] &= \text{aug\_dynamics}([\mathbf{x}(t), \mathbf{a}(t), \mathbf{b}(t), \mathbf{c}(t), t, \boldsymbol{\theta}_1]) \\ \mathbf{x}(T) &= \mathbf{x}(T), \quad \mathbf{a}(T) = \frac{\partial L}{\partial \mathbf{x}(T)}, \quad \mathbf{b}(T) = 0, \quad \mathbf{c}(T) = 0. \end{aligned}$$

Then we get  $\frac{\partial L}{\partial \mathbf{x}(0)} = \mathbf{a}(0)$ ,  $\frac{\partial L}{\partial \boldsymbol{\theta}_0} = \mathbf{b}(0)$ , and  $\frac{\partial L}{\partial \boldsymbol{\theta}_1} = \mathbf{c}(0)$ .

The whole process of computing the derivative is summarized in Algorithm 1, Where ODESOLVE is an ODE solver.

---

**Algorithm 1.** Computing derivative of NODE-EW

---

**Input:** parameters  $\boldsymbol{\theta}_1, \boldsymbol{\theta}(T)$ , the final state  $\mathbf{x}(T)$ , and loss gradient  $\frac{\partial L}{\partial \mathbf{x}(T)}$   
 $\mathbf{s}_0 = [\mathbf{x}(T), \boldsymbol{\theta}(T), \frac{\partial L}{\partial \mathbf{x}(T)}, \mathbf{0}, \mathbf{0}]$   
**def** aug\_dynamic( $[\mathbf{x}(t), \mathbf{a}(t), \mathbf{b}(t), \mathbf{c}(t)], t, \boldsymbol{\theta}_1$ )  
     **Return**  $[f(\mathbf{x}(t), t, \boldsymbol{\theta}), - \left( \frac{\partial f}{\partial \mathbf{x}} \right)^\top \mathbf{a}(t), - \left( \frac{\partial g}{\partial \boldsymbol{\theta}(t)} \right)^\top \mathbf{b}(t), - \left( \frac{\partial g}{\partial \boldsymbol{\theta}_1} \right)^\top \mathbf{b}(t)]$   
 $[\mathbf{x}(0), \frac{\partial L}{\partial \mathbf{x}(0)}, \frac{\partial L}{\partial \boldsymbol{\theta}_0}, \frac{\partial L}{\partial \boldsymbol{\theta}_1}] = \text{ODESOLVE}(\mathbf{s}_0, \text{aug\_dynamic}, T, 0, \boldsymbol{\theta}_1)$   
**Return**  $[\frac{\partial L}{\partial \mathbf{x}(0)}, \frac{\partial L}{\partial \boldsymbol{\theta}_0}, \frac{\partial L}{\partial \boldsymbol{\theta}_1}]$

---

When  $g(\cdot) = 0$ , NODE-EW degenerates to the original Neural ODE. So NODE-EW is a generalization of Neural ODE. Because the computing of an ODE system is reversible, there is no need to save the intermediate variables for backpropagation. In other words, Algorithm 1 consumes  $O(1)$  memory for training. Comparing to Neural ODE, computation of  $\boldsymbol{\theta}(t)$  indeed brings some extra computation. Fortunately, neural networks are usually trained in a batch way. As the batch-size increases, the extra computation is negligible.

### 3.2.2 Acceleration of Training

The problem of the adjoint method is that we have to solve an ODE in the feed forward propagation and solve another to compute the derivative. To achieve the required errors, we need to discretize the interval into many time steps which is very time consuming. On the other hand, it is not easy to tune the error tolerance of ODE solver during training. Hence we propose a new training strategy, which can significantly accelerate the training process when the memory requirement can be kept low.

1. At the beginning of the training, we discretize the ODE system using the Euler forward scheme with constant step size. Then we train the weights of ODE system with backpropagation until the discrete system reaches a desirable loss.
2. Fine-tune the ODE system by the adjoint method at the rest of the training epochs.

Our strategy only requires to compute fixed steps of the Euler forward scheme (we choose the time steps to be small), which can significantly reduce the training time and not introduce too much memory burden. Of course, our method make a trade off between the training speed and the memory cost depending on the steps size we choose. As is shown in Sect. 4, our method not only reduces the training time, but also boosts the performance on the image classification tasks.

### 3.3 Expressive Power of NODE-EW

In this section, we prove that our ODE system has a strong expressive power as described in Theorem 1. The proof can be found in the supplementary material.

**Theorem 1.** *Define a continuous function  $s : [0, T] \rightarrow \mathbb{R}^n$  such that  $s(a) = s(b)$  if and only if  $a = b$ . Given any  $\sigma > 0$ , there always exists a Neural ODE defined on the interval  $[0, T]$  and its solution  $\mathbf{y}(t)$  satisfies:  $|\mathbf{y}(t) - s(t)| \leq \sigma, \forall t \in [0, T]$ .*

Theorem 1 states that any continuous trajectory can be approximated by a Neural ODE with an arbitrary accuracy. So it is sufficient to use a Neural ODE [3] to model the evolution of weights  $\boldsymbol{\theta}(t)$ .

## 4 Experiment Results

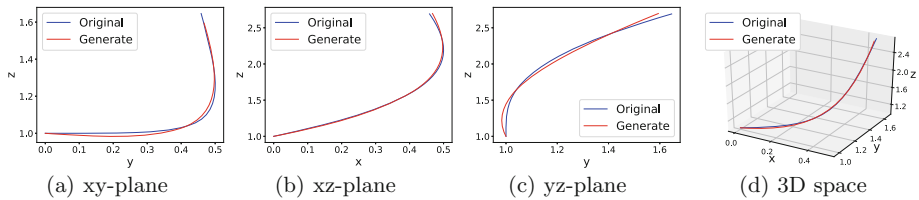
### 4.1 Verification of Theorem 1

In this section, we conduct an experiment to verify Theorem 1. In the theorem, we have proved that any continuous trajectory can be approximated by a solution of some Neural ODE. We take a continuous curve in the 3-dimensional space, such that:  $\mathbf{x}(t) = [\sin(t) \cos(t), \sin^2(t) + \cosh^2(t), \exp(t)]$ ,  $t \in [0, 1]$ .  $f$  in the experiment is a two-layer neural network with 1,000 hidden units in each layer.

We take ReLu to be the activation function. We discretize the interval  $[0,1]$  into 100 pieces uniformly and try to fit the curve at the 100 points. Hence we adopt the following loss function:

$$loss = \sum_{i=1}^{100} \|\mathbf{x}(i\Delta t) - \boldsymbol{\theta}(i\Delta t)\|^2,$$

where  $\Delta t = \frac{1}{100}$ , and  $\boldsymbol{\theta}(t)$  is the solution generated by Neural ODE. In our experiment, if we use the black-box solver to solve the Neural ODE and train the weights in  $f$  using the adjoint method, the whole process will be very time consuming. So we adopt the training strategy introduced in Sect. 3.2.2. We first train the weights in  $f$  using the backpropagation as initialization at first 5k iterations, then use the adjoint method to fine tune at the rest 100 iterations. The training time is reduced to 30 times shorter compared to the original pure adjoint method. The experiment results are shown in the Fig. 1.



**Fig. 1.** The three Figs. 1(a), (c) and (b) display the projection of the curve in the 3 orthogonal planes. In the 3D space 1(d), we can see the curve generated by our ODE system approximates well to the original curve.

## 4.2 Supervised Learning on MNIST

To compare the accuracy of Neural ODE and NODE-EW on MNIST, we utilize the structure mentioned in [3]. It first downsamples the input twice and then applies the kernel model. The figure of NODE-EW is presented in the supplementary material.

We train the two models under the same setting. We train both of them 300 epochs. The learning rate is set as 0.1 and it reduces by 0.1 after each 100 epochs. No data augmentation is used. The error rate on the test set for our NODE-EW is 0.35% which is lower than that of Neural ODE, 0.38%.

Adjoint method costs too much time for solving the ODE equations during training NODE-EW. Hence, we apply our speed-up training method proposed in Sect. 3.2.2 to train NODE-EW in this experiment. We discretize NODE-EW to 10 discrete steps, and we train the model by gradient-based backpropagation method for 80 epochs with learning rate 0.01. Then, we fine-tune the backpropagation initialized weights with learning rate 0.001 by adjoint method for 80 epochs. For comparison, we train another model by pure adjoint method for 160 epochs.

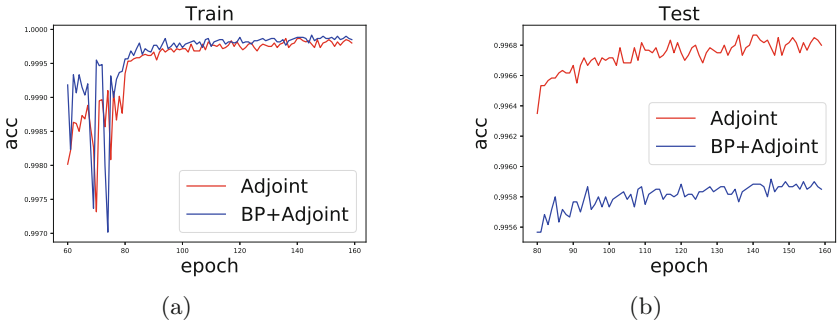


The result is summarized in Table 1 and the training process is displayed in Fig. 2. We can see that our NODE-EW finishes the training in less time and obtains better accuracy on both training and testing data sets. The speed-up training method consumes 23.18s each epoch for the first 80 epochs, while that of the adjoint method is 88.19s. The peak of the accuracy appears after 80 epochs, so we compare the mean accuracy for the last 80 epochs. The speed-up method achieves 99.57% mean accuracy on the training set while pure adjoint method only achieves 99.65%.

As is shown Fig. 2, our speed-up method boosts the accuracy during the training. The training accuracy oscillates at the first 80 epochs for our training method. We attribute this phenomenon to that direct discretization will introduce more error compared to the original ODE solver. Meanwhile, it can also have more chance to escape the saddle points, which heuristically explain why the speed-up method can boost accuracy. Note that at the 80th epoch, the speed-up training method does not have accuracy degradation, which demonstrates that discretization + backpropagation can indeed train the parameters in a continuous ODE system. So, for efficiency, it is reasonable to replace the adjoint method with backpropagation at the beginning of the training. Our observation agrees with the results in [2] which demonstrate that shallow ResNet can be a good initialization of deep ResNet. In our case, we can see that ResNet with finite layers can be a good initialization of ResNet with infinite layers.

**Table 1.** Training results for the pure adjoint method and the out new training strategy.

	Adjoint method	BP+Adjoint method
Mean Test Acc	99.57%	99.65%
Mean Train Acc	99.97%	99.98%
Time/epoch	89.76s	62.54s



**Fig. 2.** Accuracies of supervised learning on the training set (left) and the testing set (right) of MNIST.

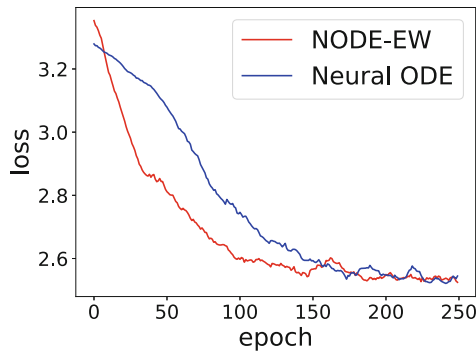
### 4.3 Normalizing Flow

The application of Neural ODE in the normalizing flow is very attractive, for it can bring down the computational cost from computing the determinant of Jacobian whose complexity is  $O(n^3)$  to only solving a differential equation whose computational cost is  $O(n)$ .

Normalizing flow stacks many layers of invertible transformation to attain a complex distribution from very simple distribution such as the Gaussian distribution. The method has been used in variational inference [23] and generative model [13] recently.

For a discrete model, it relies on the *Change of Variable Theorem*: Let  $h$  be an invertible smooth mapping from  $\mathbb{R}^n$  to  $\mathbb{R}^n$ ,  $\mathbf{x}$  be a random variable with distribution  $p(\mathbf{x})$  and  $\mathbf{x}_1 = h(\mathbf{x})$ , then

$$\log(p(\mathbf{x}_1)) = \log(p(\mathbf{x})) + \log \left( \det \left( \frac{\partial h}{\partial \mathbf{x}} \right) \right).$$



**Fig. 3.** The training curve of the Neural ODE and the NODE-EW (where the vertical axis represents loss and the horizontal one represents number of iterations): We can see that the new model descent the loss faster than the original one.

If the transformation is placed by an ODE, the following theorem is proposed by [3].

**Instantaneous Change of Variables Theorem:** Let  $\mathbf{x}(t)$  is a continuous random variable depending on time which is described by a differential equation:  $\frac{d\mathbf{x}}{dt} = f(\mathbf{x}(t), t)$ , then the log of its density function follows:

$$\frac{\partial \log(p(\mathbf{x}(t)))}{\partial t} = -\text{tr} \left( \frac{\partial f}{\partial \mathbf{x}(t)} \right).$$

The proof can be found in [3].

By an easy derivation, the above rule is also true for our NODE-EW. We conduct an experiment to generate a target 2D distribution to compare the performance of the two models. Here, we consider the maximum likelihood training, that is to maximize  $E(p(x))$ , where the  $p$  is the density of the model,  $x$  is the sample and  $E$  is the expectation operator. We adopt the Adam [12] optimizer and train the model 1,000 epochs, taking 100 data points in every epoch.

We found that the training of two models can achieve the almost the same loss ultimately. However, NODE-EW can achieve the minimum value in less epochs, which is shown in Fig. 3.

## 5 Conclusions

In this work, we demonstrate that the weights of continuous ResNet can be modeled by an ODE system. It provides us with a new tool for the further research about the neural networks with infinite layers. ResNet of finite layers is a good initialization of ResNet of infinite layers, which indicates a close relationship between them. This helps us to accelerate the pure adjoint training method. As the experiment indicates, equipped with evolutionary weights, our NODE-EW acquires more capacity and expressive power and it can arrive at the minimum loss in less epochs. So our NODE-EW is a more accurate and better continuous ODE model of the discrete ResNet.

**Acknowledgments.** The work of Zhouchen Lin is supported in part by 973 Program of China under Grant 2015CB352502, in part by NSF of China under Grants 61625301 and 61731018, and in part by Beijing Academy of Artificial Intelligence (BAAI) and Microsoft Research Asia.

## References

1. Bengio, Y., et al.: Learning deep architectures for AI. *Found. Trends Mach. Learn.* **2**(1), 1–127 (2009)
2. Chang, B., Meng, L., Haber, E., Tung, F., Begert, D.: Multi-level residual networks from dynamical systems view. *arXiv preprint [arXiv:1710.10348](https://arxiv.org/abs/1710.10348)* (2017)
3. Chen, T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. In: *Advances in Neural Information Processing Systems*, pp. 6571–6583 (2018)
4. Weinan, E.: A proposal on machine learning via dynamical systems. *Commun. Math. Stat.* **5**(5), 1–11 (2017)
5. Fang, C., Zhao, Z., Zhou, P., Lin, Z.: Feature learning via partial differential equation with applications to face recognition. *Pattern Recogn.* **69**, 14–25 (2017)
6. Haber, E., Ruthotto, L.: Stable architectures for deep neural networks. *Inverse Prob.* **34**(1), 014004 (2017)
7. Haber, E., Ruthotto, L., Holtham, E., Jun, S-H.: Learning across scales—Multiscale methods for convolution neural networks. In: *AAAI Conference on Artificial Intelligence* (2018)
8. Hardt, M., Ma, T.: Identity matters in deep learning. *arXiv preprint [arXiv:1611.04231](https://arxiv.org/abs/1611.04231)* (2016)

9. He, K., Sun, J.: Convolutional neural networks at constrained time cost. In: Computer Vision and Pattern Recognition, pp. 5353–5360 (2015)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Computer Vision and Pattern Recognition, pp. 770–778 (2016)
11. Jastrzebski, S., Arpit, D., Ballas, N., Verma, V., Che, T., Bengio, Y.: Residual connections encourage iterative inference. arXiv preprint [arXiv:1710.04773](https://arxiv.org/abs/1710.04773) (2017)
12. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980) (2014)
13. Kingma, D.P., Dhariwal, P.: Glow: Generative flow with invertible 1x1 convolutions. In: Advances in Neural Information Processing Systems, pp. 10215–10224 (2018)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
15. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In Advances in Neural Information Processing Systems, pp. 6389–6399 (2018)
16. Lin, Z., Zhang, W., Tang, X.: Learning partial differential equations for computer vision. MSR-TR-2008-189 (2008)
17. Lin, Z., Zhang, W., Tang, X.: Designing partial differential equations for image processing by combining differential invariants. MSR-TR-2009-192 (2009)
18. Liu, R., Cao, J., Lin, Z., Shan, S.: Adaptive partial differential equation learning for visual saliency detection. In: Computer Vision and Pattern Recognition, pp. 3866–3873 (2014)
19. Liu, R., Lin, Z., Zhang, W., Su, Z.: Learning PDEs for image restoration via optimal control. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010. LNCS, vol. 6311, pp. 115–128. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-15549-9\\_9](https://doi.org/10.1007/978-3-642-15549-9_9)
20. Liu, R., Lin, Z., Zhang, W., Tang, K., Zhixun, S.: Toward designing intelligent PDEs for computer vision: an optimal control approach. Image Vis. Comput. **31**(1), 43–56 (2013)
21. Liu, R., Zhong, G., Cao, J., Lin, Z., Shan, S., Luo, Z.: Learning to diffuse: a new perspective to design pdes for visual analysis. IEEE Trans. Pattern Anal. Mach. Intell. **38**(12), 2457–2471 (2016)
22. Lu, Y., Zhong, A., Li, Q., Dong, B.: Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In International Conference on Machine Learning (2017)
23. Rezende, D.J., Mohamed, S.: Variational inference with normalizing flows. In: International Conference on Machine Learning, pp. 1530–1538. JMLR (2015)
24. Szegedy, C., et al.: Going deeper with convolutions. In: Computer Vision and Pattern Recognition, pp. 1–9 (2015)
25. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: Computer Vision and Pattern Recognition, pp. 1492–1500 (2017)
26. Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., Lipson, H.: Understanding neural networks through deep visualization. arXiv preprint [arXiv:1506.06579](https://arxiv.org/abs/1506.06579) (2015)
27. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint [arXiv:1605.07146](https://arxiv.org/abs/1605.07146) (2016)
28. Zagoruyko, S., Komodakis, N.: DiracNets: Training very deep neural networks without skip-connections. arXiv preprint [arXiv:1706.00388](https://arxiv.org/abs/1706.00388) (2017)

29. Zhang, H., Shao, J., Salakhutdinov, R.: Deep neural networks with multi-branch architectures are less non-convex. arXiv preprint [arXiv:1806.01845](https://arxiv.org/abs/1806.01845) (2018)
30. Zhang, X., Li, Z., Loy, C.C., Lin, D.: PolyNet: a pursuit of structural diversity in very deep networks. In: Computer Vision and Pattern Recognition, pp. 718–726 (2017)