NEURAL PARTIAL DIFFERENTIAL EQUATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural ordinary differential equations (neural ODEs) introduced an approach to approximate a neural network as a system of ODEs after considering its layer as a continuous variable and discretizing its hidden dimension. While having a good characteristic that their required numbers of parameters are typically much smaller than those of conventional neural networks, neural ODEs are known to be numerically unstable and slow in solving their integral problems, resulting in the error and procrastination of the forward-pass computation. In this work, we present a novel partial differential equation (PDE)-based approach that removes the necessity of solving integral problems and considers both the layer and the hidden dimension as continuous variables. Owing to the recent advancement of learning PDEs, the presented novel concept, called *neural partial differential equations* (neural PDEs), can be implemented. Our method shows comparable (or better) accuracies in much shorter forward-pass inference time for various tasks in comparison with neural ODEs.

1 INTRODUCTION

It had been discovered that interpreting neural networks as differential equations is possible by several independent research groups (Weinan, 2017; Ruthotto & Haber, 2019; Lu et al., 2018; Ciccone et al., 2018; Chen et al., 2018; Gholami et al., 2019). Among them, the seminal neural ordinary differential equation (neural ODE) research work, which considers the general architecture in Figure 3, is to learn a neural network approximating $\frac{\partial \boldsymbol{h}(t)}{\partial t}$, where $\boldsymbol{h}(t)$ is a hidden vector at layer (or time) t (Chen et al., 2018). As such, a neural network is described by a system of ODEs, each ODE of which describes a dynamics of a hidden dimension. While neural ODEs have many good characteristics, they also have a limitation, which are listed as follows:

Pros. Neural ODEs can interpret t as a continuous variable and we can have hidden vectors at any layer l by $\boldsymbol{h}(l) = \boldsymbol{h}(0) + \int_0^l o(\boldsymbol{h}(t), t; \boldsymbol{\theta}_o)\, dt$, where $o(\boldsymbol{h}(t), t; \boldsymbol{\theta}_o) = \frac{\partial \boldsymbol{h}(t)}{\partial t}$ is basically a system of ODEs parameterized by $\boldsymbol{\theta}_o$.

Pros. Neural ODEs have smaller numbers of parameters than those of other conventional neural network designs.

Cons. Neural ODEs sometimes show numerical instability or procrastinate their forward-pass computation in solving integral problems. Several countermeasures have been proposed but it is unavoidable to solve integral problems (Zhuang et al., 2020; Finlay et al., 2020; Daulbaev et al., 2020).

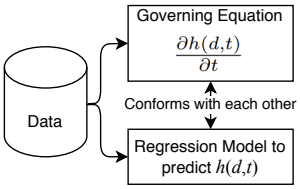


Figure 1: We avoid solving integral problems by learning a regression model that conforms with a learned governing equation.

To tackle the limitation, we propose neural partial differential equations (neural PDEs) to directly learn a hidden element, denoted $h(d, t)$ at layer (or time) $t \in [0, T]$ and dimension $d \in \mathbb{R}^m$. Under general contexts, a PDE consists of i) an initial condition at $t = 0$, ii) a boundary condition at a boundary location of the spatial domain $\mathbb{R}^m$, and iii) a governing equation describing $\frac{\partial h(d,t)}{\partial t}$. As such, learning a PDE from data can be reduced to a regression-like problem to predict $h(d, t)$ that meets its initial/boundary conditions and governing equation.

In training our proposed neural PDEs, $\boldsymbol{h}(0)$ is provided by an earlier feature extraction layer (see Figure 5). However, an appropriate governing equation is unknown. Therefore, we propose to train

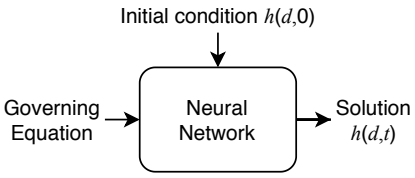


Figure 2: A neural network predicts solution values at d, t given initial condition, denoted by $h(d, 0)$, and a governing equation in our work.

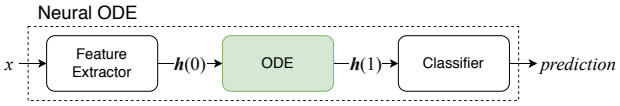


Figure 3: The general architecture of neural ODEs

Table 1: Two types of PDE problems related to our work

Type	Data	What to infer
Forward Problem	- Initial condition - Governing equation	Solution $h(d, t)$
Inverse Problem	- Solution $h(d, t)$ - Initial condition	Governing equation

a regression model for predicting $h(d, t)$ and its governing equation simultaneously (see Figure 1). In other words, neural ODEs directly learn a governing equation (i.e., $\frac{\partial \boldsymbol{h}(t)}{\partial t}$), whereas neural PDEs learn a governing equation in conjunction with a regression model that conforms with the learned governing equation. The key advantage in our approach is that we can eliminate the necessity of solving integral problems — in neural ODEs, where we learn a governing equation only, solving integral problems is mandatory.

Such forward and inverse problems (i.e., solving PDEs for $h(d, t)$ and identifying governing equations, respectively) arise in many important computational science problems and there have been many efforts applying machine learning/deep learning techniques to those problems (e.g., in earth science (Reichstein et al., 2019; Bergen et al., 2019) and climate science (Rolnick et al., 2019)). Recently, physics-informed or physics-aware approaches (Battaglia et al., 2016; Chang et al., 2017; de Bezenac et al., 2018; Raissi et al., 2019; Sanchez-Gonzalez et al., 2018; Long et al., 2018) have demonstrated that designing neural networks to incorporate prior scientific knowledge (e.g., by enforcing physical laws described in governing equations (Raissi et al., 2019)) greatly helps avoiding over-fitting and improving generalizability of the neural networks. The most similar existing work to our work is in (Long et al., 2018). However, they studied PDEs in scientific problem domains and do not consider t as a continuous variable but use a set of discretized points of t.

Compared to previous approaches, the proposed method has a distinct feature that forward and inverse problems are solved simultaneously with a continuous variable t. Due to this unique feature, the method can be applied to general machine learning downstream tasks, where we do not have a priori knowledge on governing equations, such as image classification, which we will demonstrate in the experiment section. Our proposed neural PDEs have the following characteristics:

Pros. Neural PDEs train a regression model that outputs a scalar element $h(d, t)$ (without solving any integral problems), and we can consider both d and t as continuous variables. Therefore, it is possible to construct flexible hidden dimension vectors.

Pros. Neural PDEs do not require solving integral problems. As such, there is no numerical instability and their forward-pass time is much shorter than that of neural ODEs.

Cons. Neural PDEs sometimes require a larger number of parameters than that of neural ODEs or conventional neural networks.

2 Partial Differential Equations

The key difference between ODEs and PDEs is that PDEs can have derivatives of multiple variables whereas ODEs should have only one such variable's derivative. Therefore, our PDE-based method interprets both the layer of neural network and the dimension of hidden vector as continuous variables, which cannot be done in neural ODEs. In our context, $h(d, t)$ means a scalar element of hidden vector at layer $t \in \mathbb{R}$ and dimension $d \in \mathbb{R}$ — we note that $d \in \mathbb{Z}$ is discrete in conventional neural networks and neural ODEs.

In this section, we first introduce the forward and inverse problems of PDEs in general contexts. Then, we extend them to design our proposed method in deep-learning contexts.

2.1 FORWARD PROBLEM OF PDES IN GENERAL CONTEXTS

As shown in Figure 2, the forward PDE problem in general contexts is to find a solution $h(d,t)$, where d is in a spatial domain $\mathbb{R}^m$ and t is in a time domain $[0,T]$, given i) an initial condition $h(d,0)$, ii) a boundary condition $h(d_{\text{bc}},t)$, where d_{bc} is a boundary location of the spatial domain $\mathbb{R}^m$, and iii) a governing equation g (Raissi et al., 2019) We note that the boundary condition can be missing in some cases (Kim, 2018). The governing equation is typically in the following form with particular choices of $\alpha_{i,j}$:

$$\begin{aligned} g(d,t;h) \stackrel{\text{def}}{=} h_t - \big(&\alpha_{0,0} + \alpha_{1,0}h + \alpha_{2,0}h^2 + \alpha_{3,0}h^3 \\ &+ \alpha_{0,1}h_d + \alpha_{1,1}hh_d + \alpha_{2,1}h^2h_d + \alpha_{3,1}h^3h_d \\ &+ \alpha_{0,2}h_{dd} + \alpha_{1,2}hh_{dd} + \alpha_{2,2}h^2h_{dd} + \alpha_{3,2}h^3h_{dd} \\ &+ \alpha_{0,3}h_{ddd} + \alpha_{1,3}hh_{ddd} + \alpha_{2,3}h^2h_{ddd} + \alpha_{3,3}h^3h_{ddd}\big), \end{aligned} \tag{1}$$

where $h_t = \frac{\partial h(d,t)}{\partial t}$, $h_d = \frac{\partial h(d,t)}{\partial d}$, $h_{dd} = \frac{\partial^2 h(d,t)}{\partial d^2}$, and $h_{ddd} = \frac{\partial^3 h(d,t)}{\partial d^3}$. We also note that g is always zero in all PDEs, i.e., $g(d,t;h) = 0$.

In many cases, it is hard to solve the forward problem and hence general purpose PDE solvers do not exist. Nevertheless, one can use the following optimization to train a neural network $f(d,t;\boldsymbol{\theta})$ to approximate the solution function $h(d,t)$ (Raissi et al., 2019):

$$\arg\min_{\boldsymbol{\theta}} \; L_C + L_B + L_G, \tag{2}$$

$$L_C \stackrel{\text{def}}{=} \frac{1}{N_C}\sum_{d}\big(f(d,0;\boldsymbol{\theta}) - h(d,0)\big)^2, \tag{3}$$

$$L_B \stackrel{\text{def}}{=} \frac{1}{N_B}\sum_{(d_{\text{bc}},t)}\big(f(d_{\text{bc}},t;\boldsymbol{\theta}) - h(d_{\text{bc}},t)\big)^2, \tag{4}$$

$$L_G \stackrel{\text{def}}{=} \frac{1}{N_G}\sum_{(d,t)} g(d,t;f,\boldsymbol{\theta})^2, \tag{5}$$

where N_C, N_B, N_G are the numbers of training samples, L_C is to train $\boldsymbol{\theta}$ for the initial condition, L_B is for the boundary condition, and L_G is for the governing equation. Because the governing equation is always zero, we simply minimize its squared term. Note that i) $f_t, f_d, f_{dd}, f_{ddd}$ can be easily constructed using the automatic differentiation implemented in TensorFlow or PyTorch, and ii) we only need $h(d,0), h(d_{\text{bc}},t)$, which are known a priori, to train the parameters $\boldsymbol{\theta}$.

Importance of Learning Governing Equation. Here, we demonstrate the importance of learning governing equation in solving forward problems with an example, Allen–Cahn equation. The Allen–Cahn equation is a nonlinear reaction-diffusion problem, which describes the process of phase separation in alloys:

$$g(d,t) = h_t - 0.0001h_{dd} + 5h^3 - 5h = 0, d \in [-1,1],\ t \in [0,1], \tag{6}$$

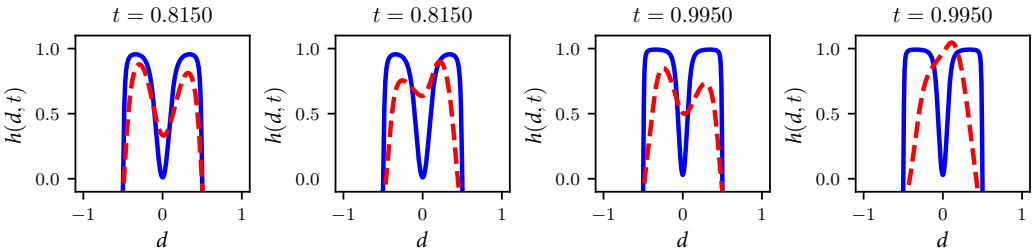
with the initial condition $h(d,0) = d^2\cos(\pi d), \forall d \in [-1,1]$, and the periodic boundary conditions $h(-1,t) = h(1,t)$ and $h_d(-1,t) = h_d(1,t), \forall t \in [0,1]$. We note that $m = 1$ and $d_{\text{bc}} \in \{-1,1\}$ in this PDE. For computing its reference solutions, a spectral Fourier discretization with 512 modes and a fourth-order explicit Runge–Kutta temporal integrator with time-step 10^{-6} is used.

To show the efficacy of the training method in Eqs. 2 through 5, we compare the method with the following naïve training method with the computed reference solutions:

$$\arg\min_{\boldsymbol{\theta}} \; L_C + L_B + L_R,$$

$$L_R \stackrel{\text{def}}{=} \frac{1}{N_R}\sum_{(d,t)}\big(f(d,t;\boldsymbol{\theta}) - h(d,t)\big)^2,$$

where L_R is to train $\boldsymbol{\theta}$ with the reference solutions of the Allen–Cahn equation with $h(d,t)$ ($t \leq 0.8$). We note that the naïve model does not learn the governing equation but learn through the supervision with the reference solutions.



(a) G.E. Aware (b) G.E. Ignorant (c) G.E. Aware (d) G.E. Ignorant

Figure 4: Extrapolation results by two neural networks (with identical architectures) that are aware or ignorant of the governing equation of the Allen–Cahn equation. The blue solid lines are reference solutions and the red dotted lines are extrapolation predictions. In all cases, better results are obtained when a neural network is aware of the governing equation, i.e., trained with L_G.

We also set $N_G = N_R$ and $t \leq 0.8$ to construct L_G for the fair comparison with the naïve model. We adopt the neural network architecture used in (Raissi et al., 2019) and train it with the two different training methods. As a result, one is aware of the governing equation because we use L_G and the other is ignorant of it because we use L_R instead of L_G. Figure 4 shows the extrapolation results for $t = \{0.8150, 0.9950\}$ obtained by using the two neural networks and we clearly see the governing-equation-aware neural network outperforms the other. In particular, the two figures at $t = 0.9950$ shows the efficacy of learning the governing equation: the prediction of the naïve model in Figure 4 (d) is not conforming to the underlying physical laws, considering that the Allen–Cahn equation is about the separation process of alloy. On the other hand, the model in Figure 4 (c) is aware of the existence of the valley around $x = 0$. This simple example demonstrates that the governing-equation-aware regression model generalizes much better for samples with unseen characteristics, e.g., extrapolation in the example. Thus, it is of our particular interest to make neural networks aware of governing equations in this work.

2.2 Inverse Problem of PDEs in General Contexts

The inverse problem is to find a governing equation given i) an initial condition $h(d, 0)$ and ii) a solution function $h(d, t)$ (Raissi, 2018). It learns $\alpha_{i,j}$ in Eq. 1 with the following loss (if possible, they use reference solutions as well):

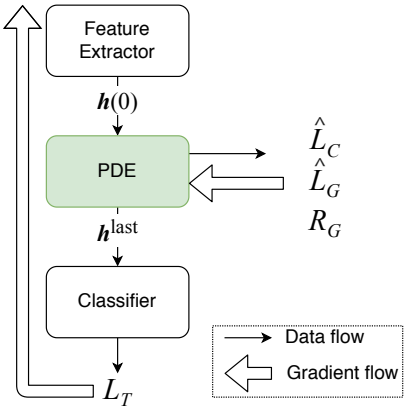
$$\arg\min_{\alpha_{i,j}} \frac{1}{N_G} \sum_{(d,t)} g(d, t; h)^2.$$

Given a solution function h and its partial derivative terms, we train $\alpha_{i,j}$ by minimizing the objective loss. Note that we know h in this case. Therefore, the objective loss is defined with h rather than with f, unlike Eq. 5.

The optimal solution of $\alpha_{i,j}$ is not unique sometimes. However, we note that no trivial solutions, e.g., $\alpha_{i,j} = 0$ for all i, j, exist for the inverse problem.

3 Neural Partial Differential Equations

Our goal in this work is to replace the ODE in Figure 3 with a PDE. To achieve this, we should know a governing equation optimized for a target task. Then, what we want to do in this work becomes a forward PDE problem because i) the feature extractor will extract an initial condition $\boldsymbol{h}(0) = [h(0, 0), h(1, 0), \cdots, h(\dim(\boldsymbol{h}) - 1, 0)]$, where $m = 1$ and $\dim(\boldsymbol{h})$ is the dimensionality of the initial condition (hidden vector), from a sample x, and ii) the governing equation is known. However, we do not know such governing equations in real-world environments but just know that governing equations are typically in the form of Eq. 1. Therefore, we propose to assume the general governing equation form and solve the forward and the inverse problems at the same time. Specifically, we alternately solve the forward and the inverse problems. When solving the forward (resp. inverse) problem, we fix its governing equation g (resp. its approximating neural network f).



Algorithm 1 How to train neural PDEs

Input: training data X, validating data V, max iteration number max_iter
Output: $\boldsymbol{\theta}$, $(d,t) \in H$, and $\alpha_{i,j}$ for all i,j

1 Initialize $\boldsymbol{\theta}$, $(d,t) \in H$, and $\alpha_{i,j}$ for all i,j;
2 $k \leftarrow 0$;
3 $L_{\text{sum}} \leftarrow \infty$;
4 **while** L_{sum} *is not converged and* $k < max_iter$ **do**
5 Train $\boldsymbol{\theta}$, Feature Extractor, and Classifier with L_T;
6 Train $\boldsymbol{\theta}$ with $L_T + \hat{L}_C + \hat{L}_G$;
7 Train $(d,t) \in H$ with L_T ;
8 Train $\alpha_{i,j}$ for all i,j with $\hat{L}_G + R_G$;
9 Validate with V and update the best model;
10 $L_{\text{sum}} \leftarrow L_T + \hat{L}_C + \hat{L}_G + R_G$;
11 $k \leftarrow k + 1$;
12 **return** $\boldsymbol{\theta}$, $(d,t) \in H$, and $\alpha_{i,j}$ for all i,j;

Figure 5: The general architecture and the training algorithm of neural PDEs

How to Solve Forward Problem. We customize the method presented in Section 2.1 by i) adding a task-specific loss, e.g., cross-entropy loss for image classification, ii) parameterizing it by the initial condition $\boldsymbol{h}(0)$, and iii) dropping the boundary condition. Let $f(\boldsymbol{h}(0), d, t; \boldsymbol{\theta})$ be our neural network to approximate $h(d,t)$ given the varying initial condition $\boldsymbol{h}(0)$[1]. The definition of the governing equation is also extended to $g(d, t; f, \boldsymbol{h}(0), \boldsymbol{\theta})$. We use the following loss definition to train $\boldsymbol{\theta}$:

$$\arg\min_{\boldsymbol{\theta}} \quad L_T + \hat{L}_C + \hat{L}_G, \tag{7}$$

$$\hat{L}_C \stackrel{\text{def}}{=} \frac{1}{N_X} \sum_{x \in X} \Big(\frac{1}{\dim(\boldsymbol{h})} \sum_{d} \big(f(\boldsymbol{h}(0), d, 0; \boldsymbol{\theta}) - h(d,0) \big)^2 \Big), \tag{8}$$

$$\hat{L}_G \stackrel{\text{def}}{=} \frac{1}{N_X} \sum_{x \in X} \Big(\frac{1}{N_H} \sum_{(d,t) \in H} g(d, t; f, \boldsymbol{h}(0), \boldsymbol{\theta})^2 \Big), \tag{9}$$

where L_T is a task-specific loss, X is a training set, and H is a set of (d,t) pairs, where $d \in \mathbb{R}_{\geq 0}, t \in \mathbb{R}_{\geq 0}$, with which we construct the last hidden vector, denoted by $\boldsymbol{h}^{\text{last}}$ (See Figure 5).

We query $f(\boldsymbol{h}(0), d, t; \boldsymbol{\theta})$ with the (d,t) pairs in H to construct $\boldsymbol{h}^{\text{last}}$. One more important point to note is that in order to better construct $\boldsymbol{h}^{\text{last}}$, we can train even the pairs in H as follows: $\arg\min_{(d,t) \in H} L_T$ (line 7 in Alg. 1). Thus, the elements of $\boldsymbol{h}^{\text{last}}$ can be collected from different dimensions and layers. A similar approach to optimize the end time of integral was attempted for neural ODEs in (Massaroli et al., 2020).

How to Solve Inverse Problem. After fixing $\boldsymbol{\theta}$, we train $\alpha_{i,j}$ for all i,j by using the following L_1 regularized loss with an importance parameter w:

$$\arg\min_{\alpha_{i,j}} \quad \hat{L}_G + R_G, \tag{10}$$

$$R_G \stackrel{\text{def}}{=} w \sum_{i,j} |\alpha_{i,j}|. \tag{11}$$

We minimize the sum of $|\alpha_{i,j}|$ to induce a sparse governing equation according to Occam's razor and since in many PDEs, their governing equations are sparse. This optimization allows us to choose a

[1]Therefore, one can consider that our neural network f approximates a general solution rather than a particular solution. A general solution means a solution of PDE with no initial conditions and a particular solution means a solution of PDE given an initial condition. Both neural ODEs and neural PDEs approximate general solutions because initial conditions are varied.

Table 2: Image classification in MNIST and SVHN. The inference time is the time in seconds to classify a batch of 1,000 images. In general, PDE-Net shows the best efficiency.

Name	# Params	MNIST		SVHN	
		Test Accuracy	Inference Time	Test Accuracy	Inference Time
ResNet	0.60M	0.9966	7.6447	**0.9660**	**8.6721**
RK-Net	0.22M	0.9970	7.4774	0.9652	13.5139
ODE-Net	0.22M	0.9964	24.8355	0.9599	37.6776
PDE-Net	0.21M	**0.9972**	**6.5023**	0.9615	9.8263

sparse solution among many possible governing equations. In many cases, therefore, our regularized inverse problem can be uniquely solved.

Training Algorithm. Our overall training algorithm is in Alg. 1. We alternately train $\boldsymbol{\theta}$, $(d, t) \in H$, and $\alpha_{i,j}$ for all i, j. The forward problem to train $\boldsymbol{\theta}$ becomes a well-posed problem (i.e., its solution always exists and is unique) if the neural network f is analytical or equivalently, uniformly Lipschitz continuous (Chen et al., 2018). Many neural network operators are analytical, such as softplus, fully-connected, and exponential. Under the mild condition of analytical neural networks, therefore, the well-posedness can be fulfilled. The inverse problem can also be uniquely solved in many cases due to the sparseness requirement. As a result, our proposed training algorithm can converge to a cooperative equilibrium. Note that $\boldsymbol{\theta}$, $(d, t) \in H$, and $\alpha_{i,j}$ for all i, j cooperate to minimize $L_T + \hat{L}_C + \hat{L}_G + R_G$. Therefore, the proposed training method can be seen as a cooperative game (Mas-Colell, 1989). After finishing the training process, $\alpha_{i,j}$, for all i, j, are not needed any more (because $\boldsymbol{\theta}$ already conforms with the learned governing equation at this point) and can be discarded during testing.

Theoretical Analyses. The adjoint sensitivity method of neural ODEs enables the space complexity of $\mathcal{O}(1)$ while calculating gradients. However, its forward-pass inference time is $\mathcal{O}(\frac{1}{s})$, where s is the (average) step size of an underlying ODE solver. Because s can sometimes be very small, its inference via forward-pass can be procrastinated.

Our neural PDEs use the standard backpropagation method to train and its gradient computation complexity is the same as that in conventional neural networks. In addition, the forward-pass inference time is $\mathcal{O}(1)$, given a fixed network f, because we do not solve integral problems.

4 Experiments

In this section, we introduce our experimental evaluations with various tasks. All experiments were conducted in the following software and hardware environments: Ubuntu 18.04 LTS, Python 3.6.6, Numpy 1.18.5, Scipy 1.5, Matplotlib 3.3.1, PyTorch 1.2.0, CUDA 10.0, and NVIDIA Driver 417.22, i9 CPU, and NVIDIA RTX Titan.

4.1 Image Classification with MNIST and SVHN

We reuse the convolutional neural network, called ODE-Net, in the work by Chen et al. (2018) to classify MNIST and SVHN and replace its ODE part with our proposed PDE, denoted PDE-Net in Table 2. See Appendix for the architecture and the hyperparameters of the network f in PDE-Net for this experiment. We reuse their codes and strictly follow their experimental environments.

Its detailed results are summarized in Table 2. We compare with ResNet, RK-Net and ODE-Net. In ResNet, we have a downsampling layer followed by 6 standard residual blocks (He et al., 2016). For RK-Net and ODE-Net, we replace the residual blocks with an ODE but they differ at the choice of ODE solvers. RK-Net uses the fourth-order Runge–Kutta method and ODE-Net uses the adaptive Dormand–Prince method for their forward-pass inference — both of them are trained with the adjoint sensitivity method which is a standard backward-pass gradient computation method. Our PDE-Net, which does not require solving integral problems, shows the best performance in all aspects for MNIST. In particular, PDE-Net shows much better efficiency than ResNet, considering

Table 3: Image classification in Tiny ImageNet. PDE-Net shows better efficiency than ODE-Net.

Name	M.Net V3	ODE-Net	PDE-Net	M.Net V3	ODE-Net	PDE-Net
Width Multiplier	1	1	1	2	2	2
Mobile Blocks	4	3	3	4	3	3
ODE Blocks	N/A	1	N/A	N/A	1	N/A
PDE Blocks	N/A	N/A	1	N/A	N/A	1
Accuracy (top-1)	0.5809	0.5547	**0.5972**	0.6076	0.5672	**0.6157**
Accuracy (top-5)	0.8049	0.7946	**0.8166**	0.8115	0.7911	**0.8357**
# Params	1.21M	1.36M	1.36M	4.30M	4.90M	4.56M
Inference Time	**4.14**	5.26	5.23	**5.21**	8.3	6.25
Out-of-distribution Robustness (top-1 accuracy)						
Gaussian Noise	0.4495	0.4165	**0.4685**	0.4757	0.4474	**0.4878**
Random Crop & Resize	0.4636	0.4305	**0.4841**	0.4814	0.4419	**0.4965**
Random Rotation	0.3961	0.3667	**0.4267**	0.4256	0.3901	**0.4381**
Color Jittering	0.4206	0.3812	**0.4429**	0.4555	0.4108	**0.4693**
Out-of-distribution Robustness (top-5 accuracy)						
Gaussian Noise	0.68	0.6619	**0.7064**	0.7025	0.6757	**0.7205**
Random Crop & Resize	0.7106	0.6935	**0.7357**	0.7215	0.6936	**0.7442**
Random Rotation	0.6372	0.6216	**0.6627**	0.6546	0.6319	**0.6778**
Color Jittering	0.6742	0.6396	**0.6878**	0.6874	0.6506	**0.713**

their numbers of parameters, i.e., 0.60M of ResNet and 0.21M of PDE-Net. Comparing ODE-Net and PDE-Net for the inference time, our method shows much faster performance, i.e., 24.8355 seconds of ODE-Net vs. 6.5023 seconds of PDE-Net to classify a batch of 1,000 images. Considering its short inference time, in SVHN we can say that its efficiency is still better than that of ODE-Net. One interesting point is that using the fourth-order Runge–Kutta method in RK-Net produces better accuracy and inferente time than ODE-Net in our experiments, which is slightly different from the original neural ODE paper (Chen et al., 2018). We tested more hyperparameters for them.

4.2 Image classification with Tiny ImageNet

We use one more larger convolutional neural network than the one in the previous subsection to test with Tiny ImageNet. Tiny ImageNet is the modified subset of ImageNet with downscaled image resolution 64×64. It consists of 200 different classes with 100,000 training images and 10,000 validation images. Our baseline model is Isometric MobileNet V3 (Sandler et al., 2019). Their isometric architecture maintains constant resolution throughout all layers. Therefore, pooling layers are not needed and computation efficiency is high, according to their experiments. In addition, neural ODEs require an isometric architecture, i.e., the dimensionality of $\boldsymbol{h}(t)$, $t \geq 0$, cannot be varied. In our neural PDEs, we do not have such restrictions. For fair comparison, however, we have decided to use Isometric MobileNet V3. We replace some of its MobileNet V3 blocks with neural ODEs or neural PDEs, denoted ODE-Net and PDE-Net in Table 3, respectively. We train our models from scratch without using any pretrained network, with a synchronous training setup.

Table 3 summarizes their results. We report both of the top-1 and the top-5 accuracy, which is a common practice for (Tiny) ImageNet. In general, our PDE-Net shows the best accuracy. PDE-Net achieves an top-1 accuracy of 0.6157 with 4.56M parameters. The full Isometric MobileNet V3 marks an top-1 accuracy of 0.6578 with 20M parameters and the reduced Isometric MobileNet V3 with 4.30M parameters shows an top-1 accuracy of 0.6076. Considering the large difference on the number of parameters, PDE-Net's efficiency is high. In particular, it outperforms others in the top-5 accuracy by non-trivial margins, e.g., 0.7911 of ODE-Net vs. 0.8115 of Isometric MobileNet V3 vs. 0.8357 of PDE-Net. In addition, PDE-Net shows faster forward-pass inference time in comparison with ODE-Net. The inference time is to classify a batch of 1,000 images.

4.3 Out-of-Distribution Image classification with Tiny ImageNet

To check the efficacy of learning a governing equation, we conduct one more additional experiment with Tiny ImageNet. We apply many augmentation (perturbation) techniques to generate out-of-

(a) Original (b) Noise (c) Crop (d) Rotation (e) Jittering

Figure 6: Out-of-distribution examples

distribution images and check how each model responses to them. According to our hypothesis, PDE-Net which knows the governing equation for classifying Tiny ImageNet should show better robustness than others (similarly as seen in Figure 4).

We use four image augmentation methods: i) adding a Gaussian noise of $\mathcal{N}(0, 0.1)$, ii) cropping a ceter area by size 56×56 and resizing to the original size, iii) rotating into a random direction for 30 degree, and iv) perturbing colors through randomly jittering the brightness, contrast, saturation, and hue with a strength coefficient of 0.2. All these are popular out-of-distribution augmentation methods (Shen et al., 2016; Azulay & Weiss, 2019; Engstrom et al., 2019). Figure 6 shows augmentation examples.

For all augmentation types, our PDE-Net shows the best accuracy (robustness) in all cases. In comparison with ODE-Net, it shows much better robustness, e.g., 0.3812 of ODE-Net vs. 0.4429 of PDE-Net for the color jittering augmentation. One interesting point is that all methods are commonly more vulnerable to the random rotation and the color jittering augmentations than the other two augmentations.

It is well known than neural networks are vulnerable to out-of-distribution samples (Shen et al., 2016; Azulay & Weiss, 2019; Engstrom et al., 2019). As being more fitted to training data, they typically show lower robustness to out-of-distribution samples. However, PDE-Net's processing them should follow its learned governing equation. Therefore, one way to understand learning a governing equation is a sort of regularization which prevents overfitting and implanting knowledge governing the classification process.

5 Discussions & Conclusions

It recently became popular to design neural networks based on differential equations. In most cases, ODEs are used to approximate neural networks. In this work, on the other hand, we presented a PDE-based approach to design neural networks. Our method simultaneously learns a regression model and a governing equation that conform with each other. Therefore, the internal processing mechanism of the learned regression model should follow the learned governing equation. One can consider that this mechanism is a sort of implanting domain knowledge into the regression model.

In some previous works, an appropriate governing equation was given and reliable training was made possible from it. However, a challenge in our problem definition is that we need to discover a governing equation from data while training a regression model. Therefore, we adopt a joint training method of the regression model and the governing equation.

To show the efficacy of the presented concept, we conducted three experiments: i) MNIST/SVHN classification, ii) Tiny ImageNet classification, and iii) Out-of-distribution Tiny ImageNet classification. Our method shows the best accuracy (or close to the best) only except SVHN. In particular, the challenging out-of-distribution classification task (and the simple example in Figure 4) empirically proves why learning an appropriate governing equation is important.

One limitation on this method is that it is sometimes hard to achieve a good trade-off between all different loss and regularization terms. Our method intrinsically involves various terms and we found that it is important to set hyperparameters carefully (especially for various coefficients and learning rates) in order to achieve reliable performance. We list all our detailed hyperparameter configurations in Appendix.

REFERENCES

Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *J. Mach. Learn. Res.*, 20:184:1–184:25, 2019.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *NeurIPS*, pp. 4502–4510, 2016.

Karianne J Bergen, Paul A Johnson, V Maarten, and Gregory C Beroza. Machine learning for data-driven discovery in solid earth geoscience. *Science*, 363(6433), 2019.

Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. In *ICLR*, 2017.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NeurIPS*. 2018.

Marco Ciccone, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino Gomez. Naisnet: Stable deep networks from non-autonomous differential equations. In *Advances in Neural Information Processing Systems*, pp. 3025–3035, 2018.

Talgat Daulbaev, Alexandr Katrutsa, Larisa Markeeva, Julia Gusak, Andrzej Cichocki, and Ivan Oseledets. Interpolated Adjoint Method for Neural ODEs. *arXiv:2003.05271*, 2020.

Emmanuel de Bezenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. In *ICLR*, 2018.

Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling CNNs with simple transformations. In *ICLR*, 2019.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *ICML*, 2020.

Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

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

Andrew Howard, Mark Sandler, Grace Chu, Lian-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V Le, and Hartwig Adam. Searching for mobilenetv3. In *ICCV*, 2019.

Hwajoon Kim. The solution of the heat equation without boundary conditions. *Dynamic Systems and Applications*, 27:653–662, 08 2018.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-net: Learning PDEs from data. ICML, 2018.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pp. 3276–3285, 2018.

A. Mas-Colell. *Cooperative Equilibrium*, pp. 95–102. Palgrave Macmillan UK, London, 1989.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes, 2020.

Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *Journal of Machine Learning Research*, 19(25):1–24, 2018.

Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.

Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, Nuno Carvalhais, et al. Deep learning and process understanding for data-driven earth system science. *Nature*, 566(7743):195–204, 2019.

David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *arXiv preprint arXiv:1906.05433*, 2019.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pp. 1–13, 2019.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *ICML*, 2018.

Mark Sandler, Jonathan Baccash, Andrey Zhmoginov, and Andrew Howard. Non-discriminative data or weak model? on the relative importance of data and model resolution. In *ICCV Workshops*, 2019.

Xu Shen, Xinmei Tian, Anfeng He, Shaoyan Sun, and Dacheng Tao. Transform-invariant convolutional neural networks for image classification and search. In *MM*, 2016.

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

Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ode. In *ICML*, 2020.