CCF Transactions on High Performance Computing (2021) 3:298–310
https://doi.org/10.1007/s42514-021-00076-7

REGULAR PAPER

Recent advance in machine learning for partial differential equation

Ka Chun Cheung[1] · Simon See[1]

Received: 5 April 2021 / Accepted: 3 August 2021 / Published online: 20 August 2021

Abstract
Machine learning method has been applied to solve different kind of problems in different areas due to the great success in several tasks such as computer vision, natural language processing and robotic in recent year. In scientific computing community, it is well-known that solving partial differential equations, which are naturally derived from physical rules that describe some of phenomena, is a challenging task in terms of computational efficiency and model accuracy. On the other hand, machine learning models are data-driven that purely reply on learning the pattern of the data distribution. Researcher recently proposed a few new frameworks to solve certain kind of partial differential equations with machine learning technique. In this paper, we discuss two newly developed machine learning based methods for solving partial differential equations.

Keywords Machine learning · Partial differential equations · Physics informed neural network · Fourier neural operator

1 Introduction

Partial differential equations arise in many fields of applied science and engineering involving physics (Sommerfeld 1949), computational fluid dynamics (Bristeau et al. 1987), computational Biology (Tang et al. 2019), molecular dynamics, earth science, weather modeling (Zhaoxia 2019), etc. Solving partial differential equation is a compute intensive task requiring large amount of computational power. Many numerical methods were developed for solving the partial differential equation in an accurate and efficient way. However, engineers usually run hundreds, even thousands of numerical simulations so as to obtain desired parameters for a particular problem. One way to accelerate the simulation is by introducing accelerators such as GPU to speed up the computation of math operations, usually matrix operation, with parallel processing technique. This common practice has been widely used in the community of scientific computing and high performance computing. On the other hand, machine learning algorithms are getting more popular for many applications in recent years. In general, it takes time to train the model but it is fast for inference task which may be good for improving the time to solution in the numerical simulation. In other words, the time to solution can be improved if the simulation is replaced by machine learning models. In this article, we discuss two recently developed methods for solving differential equations with neural network.

1.1 Motivation

In the past few decades, many numerical methods were developed for solving various kind of partial differential equations. Finite difference method, finite element method and finite volume method are the most frequently used fundamental method serving as the baseline to solve partial differential equations in the community . They are easy to implement and the theories such as convergence and stability have been well established. One may also consider spectral methods and kernel collocation method when high accuracy is of interests. Generally speaking, traditional methods are computationally intensive and the accuracy of the solution rely heavily on the fineness of the computational grid. i.e., the finer grid you use, the more accurate result you will have. There are two major concerns here. Firstly, using finer grid requires more computation and thus taking much longer time. Therefore, development of high order accurate methods which typically requires coarser grid is still a very hot topics for researcher in scientific computing. Secondly, it is

✉ Ka Chun Cheung
chcheung@nvidia.com

Simon See
ssee@nvidia.com

[1] NVIDIA AI Technology Centre, NVIDIA Corp, Shatin N.T., Hong Kong

also noticeable that grid or mesh generation is a time consuming task for complex domain. Moreover, one needs to solve the system again to obtain a solution instance when the parameters or configurations of the equations has changed. These are the barriers when engineers typically encountered when they are working on the simulation traditional numerical solvers.

1.1.1 Machine learning

Machine learning has been established more than a half century when artificial neural network was first proposed in 1943 (McCulloch and Pitts 1943). Other machine learning method such as decision tree (Breiman et al. 1984), support vector machine (Cortes and Vapnik 1995), Random Forest (Breiman 2001), K-means clustering (MacQueen 1967; Steinhaus 1957; Lloyd 1957; Forgy 1965), Autoencoder (Kramer 1991), Convolutional neural network (LeCun et al. 1989; He et al. 2016) and Long Short Term memory (Hochreiter and Schmidhuber 1997) have also been applied for wide range of applications over the years. With the recent advancement of computing hardware such as GPU and large amount of data set, deep neural network or deep learning has revolutionized the future of artificial intelligence where many complex problems that is open for many years have been solved. There are two major reasons limiting the development of deep neural network before 21st century. i.e., The computational resources and the lack of the data. Since 2007, researcher started to use graphic processing units (GPU) to solve scientific computing problems when some parallel computing frameworks such as CUDA and OpenCL were released. Several libraries for matrix operation on GPU were also developed in the coming years. This open the door for the researchers to consider using GPU to accelerate the high intensive workload of training a neural network which is basically matrix-matrix operation. Nowadays, using accelerator such as GPU has become a standard practice while training the neural network. NVIDIA also invented Tensor Core in their latest GPU architecture specializing the computation of deep neural network. Apart from the the advancement of computing hardware and the large amount of data, there are several breakthroughs in the neural network architecture that mitigate issues in training deep neural network. Convolutional neural network is well-known to be the state of the art model for the task of computer vision, a remarkable breakthrough in deep neural network architecture which is called Residual Network (ResNet) was proposed in 2015 (He et al. 2016). By considering skip connection or shortcut over some layers of the neural network, the gradient information can be passed through the layers so that the error can be sent to earlier layers of the network easily through back propagation. This gradient vanishing problem avoids researcher to build very deep neural network that can potentially have better performance has been addressed with the introduction of skip connection. Skip connection has now become a critical component in designing the state of the art deep neural network architecture.

1.1.2 From differential equations to neural network

Generally speaking, numerical method for solving differential equation and machine learning method are two different fields of research. On one hand, machine learning method such as deep neural network, a data-driven approach, uses mathematical equations to represent the relationship of data input and model output by modeling on a vast amount of data. Data quality play an very important role in the entire model development cycle for most of the data driven method. Despite of its success in many applications, it still lacks of model interpretability and robustness which are very crucial in many industries such as finance and manufacturing. On the other hand, traditional numerical method typically has rigorous mathematical deviation and mathematical proof for the convergence and stability. The model equation is always derived from the physics laws such as conservation of energy. Error estimation of numerical method is also an important indicator about how worse the approximation could be under certain conditions posed by the problem and numerical method. Although two areas of research are quite diverse in some sense, researchers recently successfully put them together. ResNet plays an important role not only in solving computer vision problem, but also one of the key to connect deep neural network and traditional numerical method for partial differential equations. Let us recall that the residual block of the ResNet can be written as

$$X_{j+1} = X_j + F_j(\boldsymbol{\theta}_j, X_j), \quad \text{for} \quad j = 0, 1, \ldots, N-1. \tag{1}$$

$$X_0 = x \tag{2}$$

where X_{j+1} is the output feature of the j th residual block, F_j is a set of layers in the j th residual block and $\boldsymbol{\theta}_j$ are the corresponding network parameters. The residual mechanism introduces skip connection directly between X_{j+1} and X_j which allows information to be passed between X_{j+1} and X_j directly. It is worth noting that the skip connection in (1) can be viewed as a forward Euler discretization for a system of ordinary differential equation. i.e., the corresponding ordinary differential equation system can be written as

$$\frac{dX}{dt} = F(\boldsymbol{\theta}, X), \quad t > 0 \tag{3}$$

$$X(t=0) = x \tag{4}$$

for some initial condition x and $X(t_j) = X_j$ for $j = 0, 1, \ldots, N$.

With the introduction of the continuous extension of residual network, Chen et al. (2018b) formulate the machine learning problem as a dynamical system and the residual

network is replaced by an ODE system. Such approach has already been proposed and studied in González-García et al. (1998) in 1998. Researcher recently applied ODE and numerical method techniques to design new deep neural network architecture that leads to a stable method for the corresponding ODE system (Haber and Ruthotto 2017; Lu et al. 2017; Ruthotto and Haber 2018; Chang et al. 2018). The authors were inspired by the numerical discretization of the ODE system (3) and the characteristic of the traditional numerical solver such as stability and order of accuracy to propose a family of new residual blocks and thus neural network architecture. Guo et al. (2021) apply similar technique to generate new neural network architecture by the numerical discretization of the system of partial differential equation such as reaction diffusion system, Schrodinger equation and Helmholtz equation. We see that there are a few more works trying to utilize the technique and the well developed theory in the community of traditional scientific computing to generate and design new neural network architecture. This is useful for designing new neural network with more explainability and more robustness. The bridge of conventional numerical method and deep neural network opens the door for researcher to apply technique of numerical method to solve problems in machine learning.

1.1.3 From neural network to differential equations

We discussed how deep neural network benefits from scientific computing and numerical partial differential equation. In fact, there are many works that have been done on solving partial differential equations with machine learning method. For instance, PDE Net (Long et al. 2018, 2019) has gained awareness of using neural network to solve partial differential equation when it was proposed in 2017. PDE-Net is a data-driven method aiming at training a neural network with vast amount of data in different time frame(for time dependent problems). It is standard machine learning approach different from conventional numerical method which requires the knowledge of equations, geometry of the domain and certain conditions for the problem such as initial and/or boundary conditions. Data driven method does not require any knowledge of the equation and the initial/ boundary conditions, only solution profiles such as the temperature, density or pressure are needed to build the deep neural network model. As one of the advantage of data-driven approach, the governing partial differential equations is not necessarily to be known. It is useful for those problems that are difficult to model with mathematical equations or the partial differential equations is highly nonlinear.

However, PDE-Net is not the first machine learning based method to solve partial differential equation problems using neural network. In 1998, Lagaris et al. (1998) proposed a method to solve nonlinear differential equation with neural network. A lot of works have been done in the past few decade (Flamant et al. 2020; Hasan et al. 2020; Lu 2020; Huré et al. 2020; Jiequn et al. 2018; Martin et al. 2018; Hermann et al. 2020; Zhang et al. 2020; Karumuri et al. 2020; Li et al. 2020a, 2020b; Yu et al. 2020; Tompson et al. 2017; Aarts and Van der Veer 2001; Sirignano and Spiliopoulos 2018; Stefan et al. 2020; She and Grecu 2018; Hayati and Karami 2007; Raissi 2018; Elbrächter et al. 2018; E et al. 2017). In 2018, Raissi et al. (2019) proposed physics driven neural network called physics informed neural network to solve partial differential equations and it has been successfully applied to solve various problems. It establishes the bridge between neural network and partial differential equations by using a customized multi-task loss function. The neural network is optimized to satisfy the governing partial differential equations, boundary condition and initial condition. Strictly speaking, physics informed neural network reformulate the optimization problem with the multi-task loss function integrating the partial differential equation residual, boundary condition and initial condition and then approximate the solution with a neural network. Similar setting can be seen in Meshless method or Spectral method.

2 Physics informed neural network and NVIDIA SimNet

In this section, we discuss the Physics informed neural network (Raissi et al. 2019), a neural network method for solving partial differential equations, and SimNet, a physics informed neural network based partial differential equation solver released by NVIDIA. Physics informed neural network is a framework about how to solve partial (ordinary) differential equation problem using neural network in the way of using customized multi-task loss function. The basic idea of physics informed neural network is to minimize the customized loss function so that the neural network satisfies the differential equations, initial condition, boundary conditions and other constraints. This idea was extended to solve different applications such as computational fluid dynamics, inverse problem, computational finance, etc. NVIDIA SimNet is a neural network based partial differential equation solver that employ the idea of physics informed neural network with efficient GPU implementation.

2.1 Method of physics informed

In general, the neural network approximates the solution of the partial differential equations that are subjected to any constrained conditions such as initial condition, boundary condition or interface condition. The term physics informed is referred to the differential equations and its constraints. In order to build the network network for solution approximation,

all the physics are embedded into a customized multi-task loss function. The (feed-forward fully-connected) neural network is optimized using standard deep learning optimization technique and the customized multi-task loss function will be constructed so that the neural network can satisfy both the partial differential equations and the constraints as close as possible. In Fig. 1, the schematic of the architecture of the neural network was shown. The input of the neural network could be the spatial variable of some scattered point x sampled in the physical domain Ω, realizations of time t and the realizations from the parametric space $\{p_i\}_i^n$ (if applicable). The input was fed into the (fully connected) neural network with the output as the approximated solution to the problem. Finally, the customized multi-task loss function consisting of the partial differential equations, initial condition and boundary condition will be optimized using stochastic gradient-based optimization method such as stochastic gradient decent or Adam (Kingma and Adam 2014). It should be remarked that the physics informed neural network is an unsupervised learning method that does not require label data for the model training. It is the one of major different compared to data driven deep neural network. Moreover, the loss function involves the calculation of the derivative and the differential operator of the neural network, they can be computed using auto-differentiation modulus provided in standard neural network library such as Tensorflow and Pytorch.

More precisely, let us consider the partial differential differential equation defined in the form:

$$\mathcal{L}_i u = f_i, \quad \text{in } \Omega \subset \mathbb{R}^d, \quad i = 1, \dots, N \tag{5}$$

$$\mathcal{B}_j u = g_j, \quad \text{on } \Gamma_j \subset \mathbb{R}^d, \quad j = 1, \dots, M, \tag{6}$$

where $\mathcal{L}_i$ denote the differential operators and $\mathcal{B}_i$ denote the constraint operators including the initial and boundary condition for some functions f_i and g_j defined on Ω and Γ_j, respectively. Suppose all necessary condition for the existence and the uniqueness of the solution u defined on $\Omega \subset \mathbb{R}^d$ of the partial differential Eqs. (5)–(6) are satisfied. Now, we would like to find an approximation u_{Net} represented by a neural network to the solution of (5)–(6). For the simplest form of fully connected neural network, u_{Net} can be expressed as

$$u_{Net}(x;\theta) := l_n \circ l_{n-1} \circ \dots \circ l_1(x), \quad x \in \Omega \tag{7}$$

and

$$l_i(s) := \sigma(W_i s + b_i) \tag{8}$$

denotes the i th layer and σ is the non-linear activation function applied element-wise. W_i and b_i are the matrix and bias vector parameterizing the layer l_i. Therefore, we denote $\theta = \{W_i, b_i\}_{i=1}^n$ as the collection of parameters of the neural network u_{Net} in which it is the most standard and widely used neural network architecture in general. The main contribution of physics informed neural network allows us to define customized multi-task loss function with physics information for the optimization of the neural network so that it is optimized to satisfy both set of partial differential equations $\mathcal{L}_i$ and set of constraints $\mathcal{B}_j$. Before writing down the customized multi-task loss function, let us define the residual functions of the differential equations and the constraints as follow:

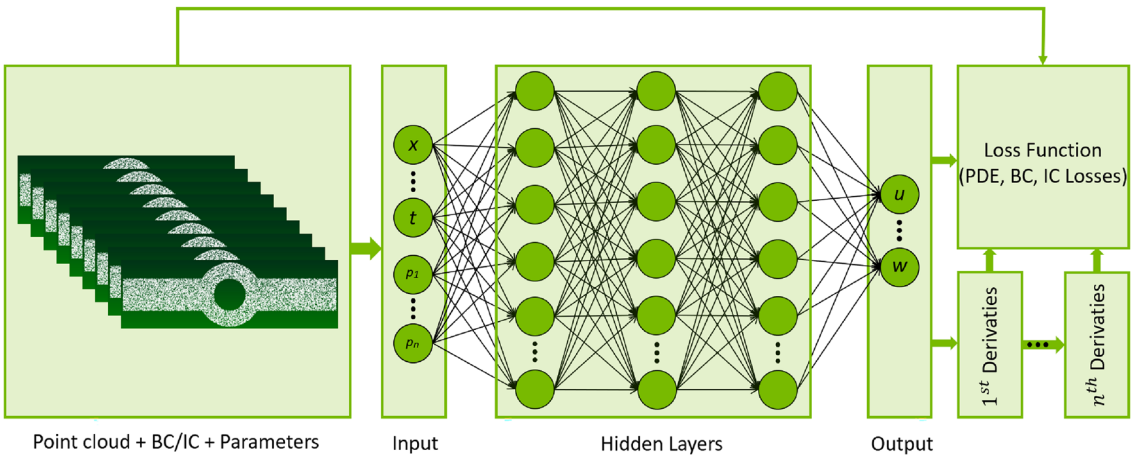

Fig. 1 A schematic of the structure of a neural network solver of SimNet. The inputs to the network are the spatial coordinates of a point cloud, realizations of time (if applicable), and realizations from the parametric space (if applicable). The inputs are mapped to the quantities of interest via a fully-connected network with nonlinear activation functions. To train this network, a loss function is considered which consists of the derivatives of the output w.r.t inputs (computed using automatic differentiation), and initial and boundary condition information. The figure was borrowed from SimNet (Hennigh et al. 2020)

$$\mathcal{R}_{\mathcal{L}}^{(i)} u_{Net}(x;\theta) := \mathcal{L}_i u_{Net}(x;\theta) - f_i(x), \quad i = 1, \ldots, N, \quad x \in \Omega \tag{9}$$

$$\mathcal{R}_{\mathcal{B}}^{(j)} u_{Net}(y;\theta) := \mathcal{B}_j u_{Net}(y;\theta) - g_j(y), \quad j = 1, \ldots, M, \quad y \in \Gamma_j \tag{10}$$

where $\mathcal{L}_i$ and $\mathcal{B}_j$ are the differential operator and the constraint operator defined in (5) and (6). The customized multi-task loss function can be defined in terms of the residual functions as follows

$$L(\theta;x) = \sum_{i=1}^{N} \int_{\Omega} \lambda_{\mathcal{L}}^{(i)} \| \mathcal{R}_{\mathcal{L}}^{(i)} u_{Net}(x;\theta) \|_p dx + \sum_{j=1}^{M} \int_{\Gamma} \lambda_{\mathcal{B}}^{(j)} \| \mathcal{R}_{\mathcal{B}}^{(j)} u_{Net}(x;\theta) \|_p ds, \tag{11}$$

where $\| \cdot \|_p$ denotes l_p norm, $\lambda_{\mathcal{L}}^{(i)}$ and $\lambda_{\mathcal{B}}^{(j)}$ are weights in order to balance the residual terms of the differential operator and the constraints operator in the customized multi-task loss function. Obtaining the optimal weighting ratio for the problems is still an current research topics, but most of the studies suggest dynamic weighting (Wang et al. 2020a, 2020b; Jin et al. 2020; Chen et al. 2018a). Similar research on the selection of weighting ratio for the residual of partial differential equations and the residual of the boundary condition can be found in conventional numerical methods (Cheung et al. 2018; Cheung and Ling 2018). SimNet currently supports three kinds of weighting strategies: global learning rate annealing as in Wang et al. (2020b) and its local variant, and signed distanced weighting. Note that in order to calculate the customized multi-task loss function defined in (11), it is necessary to compute the derivative of u_{Net} as defined in the differential operator $\mathcal{L}_i$. However, the derivative can be easily obtained using the auto-differentiation technique in Tensorflow. On the other hand, the integral in the loss function can be approximated by Monte Carlo methods or quasi-Monte Carlo methods on Ω and Γ_j, respectively. The neural network u_{Net} is eventually solved by standard numerical optimization method such as stochastic gradient decent to the customized multi-task loss function (11) with respect to the set of network parameter θ. i.e., the optimal parameter was denoted by

$$\theta^* := \arg\min_{\theta} L(\theta;x), \tag{12}$$

and the optimized neural network is given by

$$u_{Net} *= u_{Net}(x;\theta^*). \tag{13}$$

2.2 NVIDIA SimNet

Physics informed neural network is a systematic framework on how to build the network network with the physics embedded loss function to solve certain kinds of partial differential equation and it is still a hot topics in the community. NVIDIA SimNet is a physics informed neural network solver based on the deep learning framework Tensorflow for solving partial differential equations. SimNet has several modules which provides APIs for the researcher and engineer to build the neural network for their own problems efficiently.

2.2.1 Geometry module

SimNet allows researcher or model developer work on partial differential equations with complex domain. Geometry module allows one to generate and parameterize the domain of the problem (i.e., Ω and Γ_j) so that data can be sampled for the Monte Carlo method during the neural network training. It is common for the case that the complex geometries are stored in standard format such as STL, OBJ, etc. Therefore, SimNet also allows importing STL, OBJ, and other tessellated geometries to work with complex geometries. Apart from the geometry module for defining the problem domain, it has the Signed Distance function API for the multi-task loss weighting calculation. As mentioned in the previous section, selection of the weighting ratio for the PDE residual and the constraint residual is a challenging problem and it is still and active research topics. SimNet currently supports three kinds of strategies for the weighting calculation and they are included in the Geometry modules.

2.2.2 Partial differential equation module

Secondly, besides the geometry module for the domain geometry definition, SimNet has a PDE module where includes a variety of common partial differential equations such as Navier-Stokes eqation, advection-diffusion, wave equations, and linear elasticity equations. PDE module allows user to define the problems with the appropriate partial different equation. One could use the sett of partial differential equation in the PDE module or define their own equations using the symbolic mathematics library in the PDE module using SimPy (Meurer et al. 2017).

2.2.3 Network architecture modules

Although feedforward fully connected neural network architecture discussed in the last section is the most basic neural network architecture for most of the problems, SimNet in fact provides a few more options for the neural network architectures and more will be added in future release. In the latest release, SimNet supports fully connected network,

Fourier feature network (Tancik et al. 2020), modified Fourier feature network, a new architecture proposed by SimNet, and SiReN (Sitzmann et al. 2020) in the network architecture module.

2.2.4 Training and data modules

Figure 2 depicted the overview of the SimNet structure. Different modules in SimNet are clearly presented so that the user can modified specific module so as to build their own neural network for their problems. Apart from the modules mentioned above, SimNet still have DataSets and Training parameters modules for the neural network training handling. The training module allows users to select various optimizer for the network training. The DataSet module handles different training data such as training domain, inference domain, validation domain, etc. Similar to standard neural network training, training neural network for partial differential equation is a compute intensive workload. It is essential to accelerate the whole workflow with external accelerator such as NVIDIA GPU for building efficient neural network solver. SimNet support XLA (Accelerated Linear Algebra) in which the kernel fusion functionality allows some mathematical operation to be combined into one single kernel can reduce the training time. (Hennigh et al. 2020) stated that a performance boost of 3.3X can be seen when using XLA in single precision training. Using multi-GPU training is another way to accelerate the training time especially when the scale of the problem is very large. SimNet also supports data parallelism training for neural network using multi-GPU. Data parallelism based multi-GPU training with NCCL (NVIDIA Communication Collective Library) which leverage NVIDIA NVSwitch can attain nearly linear scaling. SimNet also support TF32 arithmetic which is a new math mode available on NVIDIA Ampere architecture GPU for the mixed precision training. Finally, several uses case such as Turbulent and multi-physics simulations, Blood flow in an Intracranial Aneurysm, inverse problems are also discussed in Hennigh et al. (2020).

2.3 Solving 2D advection diffusion equation using SimNet

In this section, we demonstrate SimNet by solving an advection-duffusion equation along with the continuity and the Navier-Stoke equation to model the heat transfer in a 2D flow. We show the workflow on how to train the neural network step by step in SimNet.

2.3.1 Problem statement

In this experiment, we solve the heat transfer from a 3-fin heat sink as shown in Fig. 3. The problem describes a hypothetical scenario wherein a 2D slice of the heat sink simulated as shown in the figure. The heat sinks are maintained at a constant temperature of 350 K and the inlet is at 293.489 K. The channel walls are treated as adiabatic and the inlet is assumed to be a parabolic velocity profile with 1.5 m/s as the peak velocity. The kinematic viscosity ν is set to $0.01m^2/s$ and the Prandtl number is 5. The zero equation turbulence model is kept on athough the flow is laminar.

2.3.2 Implementation

We list the step for the setup below:

– *Creating geometry*: Firstly, we will use the geometry modules of SimNet to define the problem domain. We

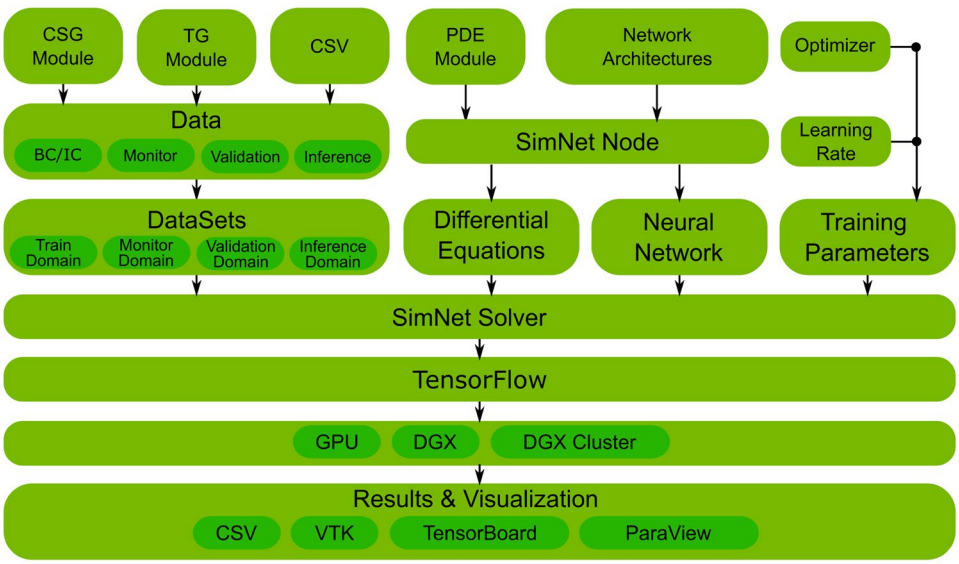

Fig. 2 SimNet structure

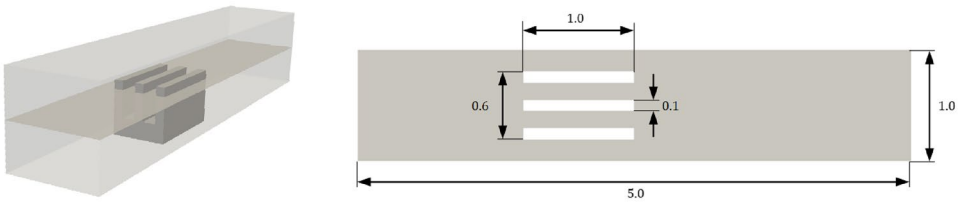

Fig. 3 2D slice of three fin heat sink geometry

use the Channel2D API and Rectangle API for generating duct and heat sink, respectively
- *Defining the boundary conditions and equations*: Secondly, we have to define the Neuamman type boundary conditions as describe in the problem statement. In addition to the continuity equation and the Navier Stoke equations, the advection-diffusion equation with no source term will also be solved in the interior. The thermal diffusivity D for this experiment will be set to $0.002m^2/s$ for $D\Delta c = V \cdot \nabla c$, where $V = (u, v)^T$ is the velocity field.
- Monitor, Inference and Validation domains: different pipelines for validation, Inference and monitor in which the validation data comes from a 2D simulation computed using OpenFOAM.
- *Making the neural network solver*: Putting all the equations and boundary conditions together, the multi-task loss function is then defined.
- Start the neural network training by the SimNet Solver
- Getting result and post-processing

As you can see from the Fig. 4, the neural network solution closes to the solution solved by OpenFoam. This means that SimNet solver can achieve similar level of accuracy of OpenFoam. However, SimNet takes less time for the solution computation. For example, OpenFOAM takes 4099 hours for the three fin heat sink design optimization while SimNet requires 120 hours only. SimNet and Physics informed neural network open the door for having fast simulation.

3 Fourier neural operator

Physics informed Neural network based partial differential equation solver has become very popular and many extended work has been done in the community since the original work published in 2018. Recently, a new neural network based partial differential equation solver called Neural Operator (Li et al. 2003, 2006, 2010) has been proposed for solving parametric partial differential equations. PINNs is a kind of unsupervised learning method in which the neural network is trained using the physics from the partial differential equation and the constraints and data sampled from the problem domain. It relies on sampling technology in the domain Ω and Γ_j for the integral evaluation by Monte Carlo

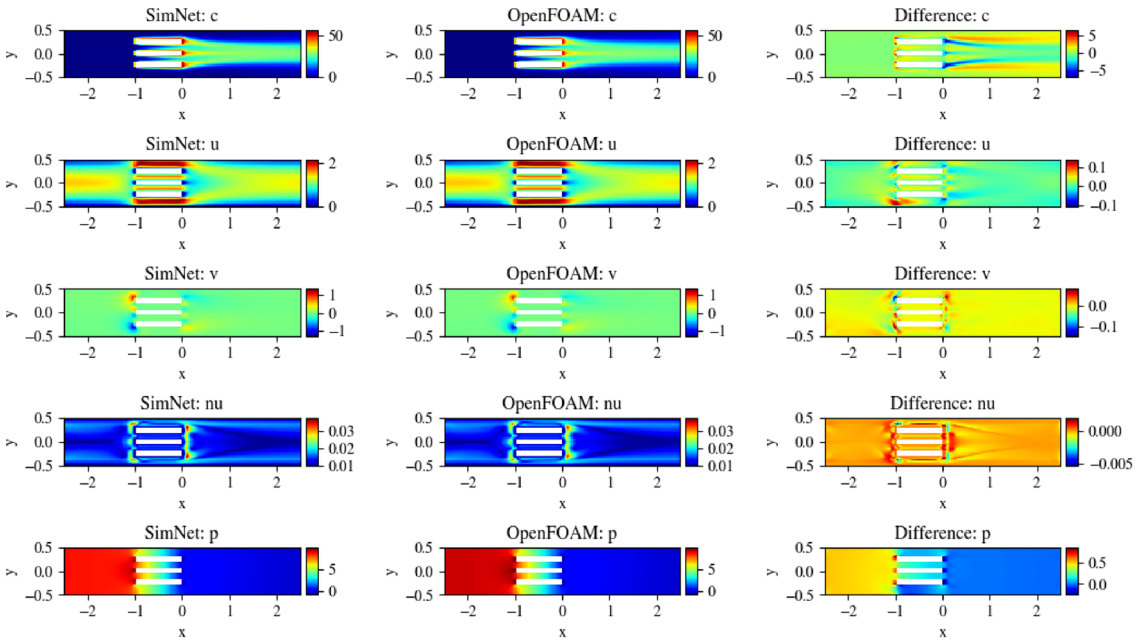

Fig. 4 Left: SimNet. Center: OpenFoam. Right: Difference

Method. No solution profile is needed to be prepared for the network training. What it needs is to define the domain geometry and partial differential equations and its constraint operators clearly, then the neural network solver such as SimNet can train the neural network automatically.

3.1 As a data driven method

Different from Physics informed neural network, Fourier Neural Operator is a data-driven method that aims at learning the mesh free, infinite dimensional operators with neural network. Firstly, it is mesh-free compared with traditional solver such as finite difference and finite element which requires computation for the discretization of the domain. Secondly, it does not require to re-train the neural network when the parameter of the partial differential equations has changed. The neural network learns the differential operator in the partial differential equations together with the set of parameters instead of one single solution instance. This is extremely useful for the problem such as inverse problem or optimal parameter recovery problem which require running thousand times of forward pass to solve for the optimal parameters. Thirdly, it does not require to know the exact formulation of the partial differential equation which is very common for data driven method such as the PDE-Net (Long et al. 2018, 2019). It is useful when sometime it is difficult to model the problem mathematically.

3.1.1 Mathematics of Fourier neural operator

Unlike physics informed neural network and SimNet, Neural operator learns a mapping between two infinite dimensional spaces. Let Ω be the spatial domain as usual and let $\mathcal{A}$ and $\mathcal{U}$ be the parameter space and solution space on Ω, respectively. We use the neural operator to approximate the solution operator of the partial differential equations and the neural operator is defined as follows:

$$u_F : \mathcal{A} \times \Theta \to \mathcal{U}, \tag{14}$$

where Θ is the space of all trainable parameters of the neural network. The neural network takes the pair of data $\{a_j \in \mathcal{A}, u_j \in \mathcal{U}\}_{j=1}^N$ for neural network learning where a_j is any function in the parameter space $\mathcal{A}$ and u_j is the solution of the partial differential equations corresponds to the parameter a_j.

The network architecture of the neural operator is made of a chain of Fourier layers. Let us consider u_F to be the Fourier neural operator defined as:

$$u_F(a;\theta) := l_n \circ l_{n-1} \circ \ldots \circ l_1(a), \quad a(x) \in \mathcal{A}, x \in \Omega \tag{15}$$

and the Fourier layer is defined as

$$l_{i+1}(v)(x) := \sigma(Wv(x) + (\mathcal{K}(a;\phi)v)(x)), \quad x \in \Omega \tag{16}$$

where v is the function denoting the output of the *ith* Fourier layer l_i. W is the matrix of parameter as usual and the kernel integral operator

$$(\mathcal{K}(a;\phi)v)(x) := \int_\Omega \kappa(x, y, a(x), a(y);\phi)v(y)dy, \quad x \in \Omega \tag{17}$$

is a linear transformation and κ is a neural network parameterized by $\phi \in \Theta_K$. σ is the activation function of the Fourier layer applied element-wise. In order to evaluate (17) efficiently, Fast Fourier transform was used to parameterize κ in the Fourier space. Let $\mathcal{F}$ be the operator of Fourier transform of $f : \Omega \to \mathbb{R}^{d_v}$ as

$$(\mathcal{F}f)_j(k) = \int_\Omega f_j(x)e^{-2i\pi\langle x,k\rangle}dx \tag{18}$$

and its inverse transform $\mathcal{F}^{-1}$:

$$(\mathcal{F}^{-1}f)_j(x) = \int_\Omega f_j(x)e^{2i\pi\langle x,k\rangle}dk \tag{19}$$

for $j = 1, \ldots, d_v$. By setting $\kappa(x, y, a(x), a(y)) = \kappa(x - y)$, the Fourier integral operator is defined as

$$(\mathcal{K}(\phi)v)(x) = \mathcal{F}^{-1}(R_\phi \cdot (\mathcal{F}v))(x), \quad x \in \Omega \tag{20}$$

where R_ϕ is the Fourier transform of a periodic function κ parameterized by $\phi \in \Theta_K$ as shown in Fig. 5. It remains to discuss the parameterization of R_ϕ in the Fourier layer. In general, R_ϕ can be defined in many ways. After some empirical study, the author of (Li et al. 2010) suggest to use direct parameterization so that R_ϕ is treated as a weight tensor in $\mathbb{R}^{k_{max} \times d_v \times d_v}$ where k_{max} is the number of truncated Fourier modes throughout the Fourier transform.

3.1.2 How to implement

To summarize the Fourier neural operator, the full architecture of the Fourier neural operator is shown in Fig. 5a. The Fourier neural operator takes $a(x)$ from the parameter space as the input and an embedding layer P is used to convert the input a to a higher dimensional space. The parameter space $\mathcal{A}$ depends on the problem and it can be any parameter of the partial differential equation such as the velocity of the transport equation or initial condition in initial value problems. The main component of the Fourier neural operator is its Fourier layer which will be discussed shortly. The output of the Fourier neural operator is obtained by projection layer projecting the output of a sequence of Fourier layers to the solution space with target dimension. The details of Fourier layer is shown in Fig. 5b). The input of the Fourier layer is transformed to the Fourier space via Fourier transform. A

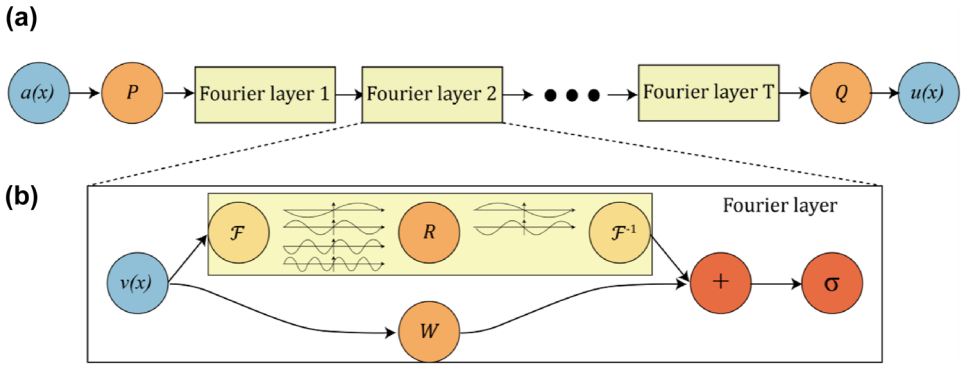

Fig. 5 **a** Full architecture of Fourier neural operator: For any input a, (i) Lift it to a higher dimension channel space by a neural network P. (ii) Apply a chain of Fourier layers. (iii) Project back to the target dimension by a neural network Q with output u. **b** Fourier layers: For any function v, we apply Fourier transform $\mathcal{F}$; a linear transform R on lower Fourier modes and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. Adding the linear transform W and activation function lead to the output of the Fourier layers. The figure was copied from (Li et al. 2010)

linear transformation R_ϕ which is trainable and parameterized by ϕ is then applied. Finally it is transformed back to the space domain via inverse Fourier transform. Activation function is applied element-wise to the sum of the output of the inverse Fourier transform and a linear transformation of the input of the Fourier layer. The linear transformation can be seen as a skip connection inside the Fourier layer. The original paper of Fourier neural operator (Li et al. 2010) demonstrates a few examples of partial differential equation such as Burger's equation, Darcy Flow and Navier-Stoke equations using the Fourier neural operator. For the data generation, all the data in that three examples were generated by traditional numerical solver such as finite difference method with very fine grid. Numerical experiment showed that the result is promising and outperform other neural network based solver.

3.2 Experiments of Fourier neural operators

Fourier neural operator has shown impressive empirical advantage against other machine learning methods and conventional solvers. Li et al. (2010) tests Fourier neural operator on the 2-d Navier-Stokes equation in the following form

$$\begin{aligned} \partial_t w(x,t) + u(x,t)\cdot\nabla w(x,t) &= \nu\Delta w(x,t) + f(x), \\ & \qquad x\in(0,1)^2, t\in(0,T] \\ \nabla\cdot u(x,t) &= 0, \qquad x\in(0,1)^2, t\in[0,T] \\ w(x,0) &= w_0(x), \qquad x\in(0,1)^2 \end{aligned} \tag{21}$$

where $u\in C([0,T];H^r_{\text{per}}((0,1)^2;\mathbb{R}^2))$ for any $r>0$ is the velocity field, $w=\nabla\times u$ is the vorticity, $w_0\in L^2_{\text{per}}((0,1)^2;\mathbb{R})$ is the initial vorticity, $\nu\in\mathbb{R}_+$ is the viscosity coefficient, and $f\in L^2_{\text{per}}((0,1)^2;\mathbb{R})$ is the forcing function. The operator maps the vorticity from the initial time 10 to the later time $T>10, u_F : w|_{(0,1)^2\times[0,10]}\mapsto w|_{(0,1)^2\times(10,T]}$.

As shown in Table 1, the FNO-3D has the best performance when there is sufficient data ($\nu=$ 1e−3, $N=1000$ and $\nu=$ 1e−4, $N=10000$). For the configurations where the amount of data is insufficient ($\nu=$ 1e−4, $N=1000$ and $\nu=$ 1e−5, $N=1000$), all methods have > 15% error with FNO-2D achieving the lowest. The neural operator is mesh-invariant, so it can be trained on a lower resolution and evaluated at a higher resolution, without seeing any higher resolution data (zero-shot super-resolution). Figure 6 shows an example where the FNO-3D model is trained

Table 1 Benchmarks on Navier Stokes

Config	Parameters	Time per epoch	$\nu=$ 1e−3 $T=50$ $N=1000$	$\nu=$ 1e−4 $T=30$ $N=1000$	$\nu=$ 1e−4 $T=30$ $N=10000$	$\nu=$ 1e−5 $T=20$ $N=1000$
FNO-3D	6, 558, 537	38.99s	**0.0086**	0.1918	**0.0820**	0.1893
FNO-2D	414, 517	127.80s	0.0128	**0.1559**	0.0834	**0.1556**
U-Net	24, 950, 491	48.67s	0.0245	0.2051	0.1190	0.1982
TF-Net	7, 451, 724	47.21s	0.0225	0.2253	0.1168	0.2268
ResNet	266, 641	78.47s	0.0701	0.2871	0.2311	0.2753

The number in bold are indicating the smallest inference time

on $64 \times 64 \times 20$ resolution data in the setting above with ($\nu = 1e{-}4, N = 10000$) and transferred to $256 \times 256 \times 80$ resolution, demonstrating super-resolution in space-time.

Li et. al. compare the Fourier neural operator against the pseudo-spectral solver in the Bayesian inverse problem. A function space Markov chain Monte Carlo (MCMC) method is used to draw samples from the posterior distribution of the initial vorticity in Navier-Stokes given sparse, noisy observations at time $T = 50$. The Fourier neural operator is used acting as a surrogate model with the traditional solvers used to generate our train-test data (both run on GPU). It generates 25,000 samples from the posterior (with a 5000 sample burn-in period), requiring 30,000 evaluations of the forward operator. FNO and the traditional solver recover almost the same posterior mean which, when pushed forward, recovers well the late-time dynamic of Navier Stokes. In sharp contrast, FNO takes 0.005s to evaluate a single instance while the traditional solver, after being optimized to use the largest possible internal time-step which does not lead to blow-up, takes 2.2s. This amounts to 2.5 minutes for the MCMC using FNO and over 18 hours for the traditional solver. Even if including the data generation and training time (offline steps) which take 12 hours, using FNO is still faster! Once trained, FNO can be used to quickly perform multiple MCMC runs for different initial conditions and observations, while the traditional solver will take 18 hours for every instance.

3.3 Discussion, summary and future work

3.3.1 Physics informed neural network helps on HPC simulation

We discussed physics informed neural network in Sect. 2. It is an unsupervised learning method for solving partial differential equations. Unlike most of the data driven approach that only train the neural network with a set of data and serve it as a black-box model for the prediction, physics informed neural network includes the physics information such as the governing equations, initial condition, boundary condition, etc during the network optimization. With the help of defining a customized multi-task loss function with all physics included, the neural network as the approximation of the solution will be optimized to satisfy all the equations in some sense as close as possible to the analytical solution. It allows people to train the neural network with physics-driven optimization. It is quite different from the data-driven approach for training machine learning model. For the data driven approach, people need to generate the data by other solvers such as conventional method like finite difference or finite element method. Data can also be collected from some sensors before starting to build the machine learning model. Physics informed neural network or SimNet requires to define the problem domains and the set of equations only. No solution profile is needed. The network will be trained by data sampling from the problem domain. The Pros of PINNs based solver is that it is physics driven, therefore it is very similar to conventional numerical solver. It is replacing the approximation space by the space span by layers of neural network. Secondly, it is unsupervised. Data preparation is an important step in the entire model development. The quality of the data affect the model performance a lot. Thus, without the need to play attention to data cleaning, data manipulation and data prepossessing make the life easier. However, like many standard deep neural network, training the neural network for convergent is not a easy and trivial task as physics informed neural network does not converge all the time. One may need to spend time on hyper-parameter tuning including learning rate, network architecture, loss weighting, etc. On the other hand, Physics informed neural network like many conventional numerical methods, the solution is solved for a particular parameter of the problems such as diffusion coefficient in diffusion equations or velocity in the advection equation. One may need to re-train another neural network if any parameters were changed.

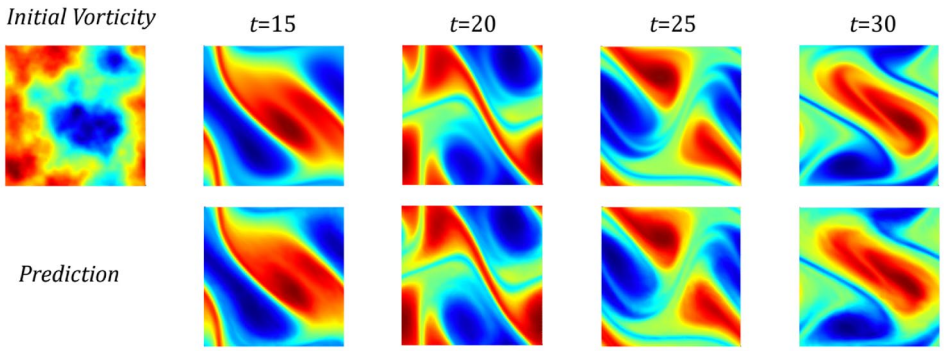

Fig. 6 **top:** Fourier neural operator on the Navier-Stokes equation

Zero-shot super-resolution: Navier-Stokes Equation with viscosity $\nu = 1e{-}4$; Ground truth on top and prediction on bottom; trained on $64 \times 64 \times 20$ dataset; evaluated on $256 \times 256 \times 80$.

3.3.2 From physics driven to data driven

On the other hand, we also discuss another neural network based solver for solving partial differential equation. The Fourier neural operator is quite a bit different from physics informed neural network. Both physics informed neural network and Fourier neural operator use neural network to approximate the solution of the partial differential equations. However, Fourier neural operator requires to prepare solution profile to train the neural network in the supervised learning setting. In other words, we need to generate many solutions with different parameters, different location input by other solvers such as conventional numerical solver like high order finite difference method. The advantage of this Fourier neural operator is obvious. It learns the operator mapping that can generalize the result for unseen location input or unseen parameters. It will be very useful for dealing with parametric partial differential equations and the problems involving parameter recovery. We do not need to re-train the neural network when the parameters or the initial condition were changed. Secondly, it is fast. It takes 0.005s to get the solution of the Navier Stoke equations on a 256×256 grid compared to 2.2s of pseudo-spectral method. Thirdly, Fourier neural network does not use customerized multi-task loss for the neural optimization. Using standard loss function such as MSE make the network training easier for convergence. However, there are a few Cons for Fourier neural operator. Firstly, it is data-driven and lack of physics information during the neural network optimization. It is a black-box approach to learn the differential operator. Secondly, as it is a data-driven approach, it requires a lot of training data in advanced of the model training. It takes time for the solution generation with conventional numerical solver.

3.3.3 Future direction

As we discussed, both physics informed neural network and Fourier neural operator have Pros and Cons. More and more researchers are working on physics informed neural network as there are still many works has to be done so that PINNs become more stable and robust. Developers and researchers now can use SimNet to solve various physics problems in a way that using GPU to accelerate the simulation and research. On the other hand, physics informed neural network is not limited to fully connected neural network. It does support other network architecture such as Fourier feature network or Recurrent neural network. We foresee that in some day physics informed neural network can integrate with the Fourier neural operator to address the limitation of each standalone method.

3.3.4 Conclusion

Machine learning method become very popular and is applied to solve many complex problems successfully in recent year. Machine learning method is known to be data driven and lack of robustness and interpretability while numerical partial differential equations have many theoretical foundations for convergence and stability. Several new machine learning based methods have been proposed for solving partial differential equations. By designing customized multi task loss function, physics informed neural network allows one to model the problem with neural network satisfying both the partial differential equation and the constraints such as initial conditions, boundary conditions, etc. We also discussed another new data driven machine learning method for solving partial differential equations with Fourier analysis. The Fourier Neural operator aims at modeling an infinite dimensional operator from the space of parameter functions such as initial conditions or parameter of the equations to the space of solution of the partial differential equations. On the other hand, partial differential equations and various numerical method have also been used to design and develop more stable and reliable neural network architecture.

Acknowledgements We gratefully thank the Dr. Sanjay Choudhry and Zongyi Li for providing support and comment. We also thank the anonymous reviewer for their useful suggestions.

References

Aarts, L.P., Van der Veer, P.: Solving nonlinear differential equations by a neural network method. In: Computational Science—ICCS (2001)

Breiman, L.: Random forests. Mach. Learn. (2001)

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and regression trees. Wadsworth and Brooks Cole Advanced Books and Software, Monterey (1984)

Bristeau, M.O., Glowinski, R., Periaux, J.: Numerical methods for the Navier-stokes equations. Applications to the simulation of compressible and incompressible viscous flows. Comput. Phys. Rep. **6**, 73–187 (1987)

Chang, B. Meng, L. Haber, E., Ruthotto, L., Begert, D., Holtham, E.: Reversible architectures for arbitrarily deep residual neural networks. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence. New Orleans, LA, pp 2811–2818 (2018)

Chen, Z., Badrinarayanan, V., Lee, C.-Y., Rabinovich, A.: Gradnorm: gradient normalization for adaptive loss balancing in deep multitask networks. In: International Conference on Machine Learning, pp 794–803. PMLR, (2018a)

Chen, T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations. In: Advances in Neural Information Processing Systems 31 (NeurIPS 2018), Montreal, Canada (2018b)

Cheung, K.C., Ling, L.: A kernel-based embedding method and convergence analysis for surfaces PDEs. SIAM J. Sci. Comput. **40**(1), A266–A287 (2018)

Cheung, K.C., Ling, L., Schaback, R.: H^2-convergence of least-squares kernel collocation methods. SIAM J. Numer. Anal. **56**(1), 614–633 (2018)
Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**, 273–297 (1995)
Elbrächter, D., Grohs, P., Jentzen, A., Schwab, C.: DNN Expression rate analysis of high-dimensional PDEs: application to option pricing. arXiv:1809.07669. (2018)
Flamant, C., Protopapas, P., Sondak, D.: Solving differential equations using neural network solution bundles. arXiv:2006.14372 (2020)
Forgy, E.W.: Cluster analysis of multivariate data: efficiency versus interpretability of classifications. Biometrics **21**, 768–769 (1965)
González-García, R., Rico-Martínez, R., Kevrekidis, I.G.: Identification of distributed parameter systems: a neural net based approach. Comp. Chem. Eng. **22**, S965–S968 (1998)
Guo, P., Huang, K., Xu, Z.: Partial differential equations is all you need for generating neural architectures. arXiv preprint arXiv:2103.08313, (2021)
Haber, E., Ruthotto, L.: Stable architectures for deep neural networks. Inverse Prob. **34**(1), 1–22 (2017)
Hasan, A., Pereira, J. M., Ravier, R., Farsiu, S., Tarokh, V.: Learning partial differential equations from data using neural networks. In: ICASSP 2020–2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, pp. 3962–3966 (2020)
Hayati, M., Karami, B.: Feedforward neural network for solving partial differential equations. J. Appl. Sci. **7**, 2812–2817 (2007)
He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, (2016), pp 770-778, https://doi.org/10.1109/CVPR.2016.90
Hennigh, O., Narasimhan, S., Nabian, M. A., Subramaniam, A., Tangsali, K., Rietmann, M., Ferrandis, J.A., Byeon, W., Fang, Z., Choudhry, S.: NVIDIA SimNet: an AI-accelerated multi-physics simulation framework. arXiv:2012.07938. (2020)
Hermann, J., Schätzle, Z., Noé, F.: Deep-neural-network solution of the electronic Schrödinger equation. Nat. Chem. **12**, 891–897 (2020)
Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**, 1735–1780 (1997)
Huré, C., Pham, H., Warin, X.: Deep backward schemes for high-dimensional nonlinear PDEs. (2020). ffhal-02005362v2f
Jiequn, H., Arnulf, J., Weinan, E.: Solving high-dimensional partial differential equations using deep learning. Proc. Natl. Acad. Sci. **115**, 8505–8510 (2018)
Jin, X., Cai, S., Li, Hui, K., George E.: Nsfnets (navier-stokes flow nets): physics-informed neural networks for the incompressible Navier-stokes equations. arXiv preprint arXiv:2003.06496, (2020)
Karumuri, S., Tripathy, R., Bilionis, I., Panchal, J.: Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks. J. Comput. Phys. **404**, 109120 (2020)
Kingma, D. P., Adam, J.B.: A method for stochastic optimization. arXiv:1412.6980, (2014)
Kramer, Mark A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE J. **37**, 233–243 (1991)
Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw. **9**, 987–1000 (1998)
LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. ATT Bell Laboratories, (1989)
Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A.: Neural operator: graph kernel network for partial differential equations. arXiv preprint arXiv:2003.03485 (2003)
Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A.: Multipole graph neural operator for parametric partial differential equations. arXiv preprint arXiv:2006.09535 (2006)
Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A.: Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895 (2010)
Li, X., Wong, T.-K., Leonard, C., Ricky T. Q., Duvenaud, D.: Proceedings of the twenty third international conference on artificial intelligence and statistics, PMLR 108:3870–3882, (2020a)
Li, Y., Lu, J., Mao, A.: Variational training of neural network approximations of solution maps for physical models. J. Comput. Phys. **409**, 109338 (2020b)
Lloyd, S.P.: Least square quantization in PCM. Bell Telephone Laboratories Paper, (1957)
Long, Z., Lu, Y., Ma, X., Dong, B.: PDE-Net: learning PDEs from data. arXiv:1710.09668. (2018)
Long, Z., Lu, Y., Dong, B.: PDE-Net 2.0: learning PDEs from data with a numeric-symbolic hybrid deep network. arXiv:1812.04426, (2019)
Lu, Y., Zhong, A., Li, Q., Dong, B.: Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. arXiv preprint arXiv:1710.10121, (2017)
Lu, L., Meng, X., Mao, Z., Karniadakis, G. E.: DeepXDE: a deep learning library for solving differential equations. arXiv:1907.04502. (2020)
MacQueen, J. B.: Some methods for classification and analysis of multivariate observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, (1967)
Martin, M., Faisal, Q., Hendrick, de H.: Neural networks trained to solve differential equations learn general representations. Adv. Neural Inform. Process. Syst. (2018)
McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**(4), 115–133 (1943)
Meurer, A., Smith, C.P., Paprocki, M., Certík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., et al.: Sympy: symbolic computing in python. PeerJ Comput. Sci. **3**, e103 (2017)
Raissi, M.: Deep hidden physics models: deep learning of nonlinear partial differential equations. J. Mach. Learn. Res. **19**, 932–955 (2018)
Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019)
Ruthotto, L., Haber, E.: Deep neural networks motivated by partial differential equations. arXiv preprint arXiv:1804.04272, (2018)
She, J.-H., Grecu, D.: Neural network for CVA: learning future values. Papers 1811.08726, arXiv.org(2018) (2018)
Sirignano, J., Spiliopoulos, K.: DGM: a deep learning algorithm for solving partial differential equations. J. Comput. Phys. **375**, 1339–1364 (2018)
Sitzmann, V., Martel, J.N.P., Bergman, A.W., Lindell, D.B., Wetzstein, G.: Implicit neural representations with periodic activation functions. arXiv preprint arXiv:2006.09661, (2020)
Sommerfeld, A.: Partial differential equations in physics. Academic Press, New York (1949)
Stefan, K., Alexander, S., Michaela, S.: A deep neural network algorithm for semilinear elliptic PDEs with applications in insurance mathematics. Risks **8**, 136 (2020)
Steinhaus, H.: Sur la division des corps matériels en parties Bull. Acad. Polon. Sci. (1957)
Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J.T.: Ren Ng. Fourier features let networks learn high frequency functions

in low dimensional domains. arXiv preprint arXiv:2006.10739, (2020)
Tang, B., Pan, Z., Yin, K., Khateeb, A.: Recent advances of deep learning in bioinformatics and computational biology. Front. Genet. **10**, 214 (2019)
Tompson, J., Schlachter, K., Sprechmann, P., Perlin, K.: Accelerating Eulerian Fluid SimulationWith Convolutional Networks. In: Proceedings of the 34th International Conference on Machine Learning, (2017)
Wang, S., Teng, Y., Perdikaris, P.: Understanding and mitigating gradient pathologies in physics-informed neural networks. arXiv preprint arXiv:2001.04536, (2020a)
Wang, S., Yu, X., Perdikaris, P.: When and why pinns fail to train: a neural tangent kernel perspective. arXiv preprint arXiv:2007.14527, (2020b)
Weinan, E., Han, J., Jentzen, A.: Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun. Math. Stat. **5**, 349–380 (2017)
Yu, Y., Hientzsch, B., Ganesan, N.: Backward deep BSDE methods and applications to nonlinear problems. arXiv:2006.07635. (2020)
Zhang, D., Guo, L., Karniadakis, G.E.: Learning in modal space: solving time-dependent stochastic PDEs using physics-informed neural networks. SIAM J. Sci. Comput. **42**, A639–A665 (2020)
Zhaoxia, P., Eugenia, K.: Models, numerical methods, and data assimilation, numerical weather prediction basics. Springer, Berlin (2019)