HoD-Net: High-order Differentiable Deep Neural Networks and Applications

Siyuan Shen[1], Tianjia Shao[1*], Kun Zhou[1], Chenfanfu Jiang[2], Feng Luo[3], Yin Yang[3]
[1]State Key Lab of CAD&CG, Zhejiang University
[2]Department of Mathematics, University of California, Los Angeles
[3]School of Computing, Clemson University
{shensiyuan|tjshao|kunzhou}@zju.edu.cn, cffjiang@math.ucla.edu, {luofeng|yin5}@clemson.edu

Abstract

We introduce a deep architecture named HoD-Net to enable high-order differentiability for deep learning. HoD-Net is based on and generalizes the complex-step finite difference (CSFD) method. While similar to classic finite difference, CSFD approaches the derivative of a function from a higher-dimension complex domain, leading to highly accurate and robust differentiation computation without numerical stability issues. This method can be coupled with backpropagation and adjoint perturbation methods for an efficient calculation of high-order derivatives. We show how this numerical scheme can be leveraged in challenging deep learning problems, such as high-order network training, deep learning-based physics simulation, and neural differential equations.

Introduction

The prosperity of deep learning (DL) is unlikely without the development of underlying differentiation methods. Indeed, the concept of neural networks has a long history dated back to WW2, which was originally designed as a computational modality for logical calculus (McCulloch and Pitts 1943). However, the actual deployment of the deep neural network or DNN for machine learning is neither common nor feasible until differentiation techniques, i.e., the automatic differentiation (AD) method (Baydin et al. 2018; Rall and Corliss 1996; Bücker et al. 2006), become fledged. A multi-layer net is then treated as a compositional function, and we can practically calculate its gradient via the backpropagation (BP) (Hecht-Nielsen 1992; Rumelhart et al. 1995), which is a dedicated implementation of reverse AD.

AD computes the *analytic derivative* of a function, using *"exact formulas along with floating-point values"* (Neidinger 2010). When higher-order differentiation is needed, the computation complexity along the chain rule escalates at a super-polynomial rate (w.r.t. the order of differentiation) bringing practical difficulties in AD implementations (Margossian 2018). As a result, many existing AD packages (e.g., `Adept` (Hogan 2014)) only deals with the first-order derivative. While one may perform first-order differentiation multiple times to obtain a high-order derivative, it has been argued that recursive AD leads to inefficient and numerically unstable code (Margossian 2018; Betancourt 2018). High-order AD is often not well supported or could be much slower (like `JAX` (Schoenholz and Cubuk 2019)), which stands as a major technical obstacle in many applications.

We propose a new set of differentiation toolkit named HoD-Net to close the gap of high-order differentiability of DL. Our method is based on complex-step finite difference method or CSFD (Squire and Trapp 1998). Unlike the classic finite difference method (Morton and Mayers 2005), CSFD lifts differentiation computations to the complex domain. This strategy avoids the notorious numerical issue of *subtraction cancellation* (Muller et al. 2018). As a result CSFD computes differentiations highly accurately albeit being a numerical method. Based on CSFD, we approach high-order differentiable deep networks by generalizing regular complex arithmetic to multicomplex domains, designing an efficient adjoint perturbation scheme, and showing its applications in several learning problems.

To make the paper self-contained, we start with a brief review of classic finite difference method, its numerical issue, and CSFD alternatives. Related literatures will be discussed when introducing the relevant topics.

Background

Evaluating the differentiation of a function is often through inferring the exact form of its derivative. While it is possible to automatically obtain the analytic derivative formulation with symbolic differentiation packages like `Mathematica` (Wolfram et al. 1996) and `Maple` (Maple 1994), AD is more widely used in practice. AD decomposes a general computation procedure into multiple basic steps, and each step corresponds to a node on its computation graph. AD *analytically* computes the derivative of each step either using source code transformation (Utke et al. 2008; van Merrienboer, Moldovan, and Wiltschko 2018) or operator overloading (Phipps and Pawlowski 2012). The latter option is natively supported by most modern programming languages like `C++` and `Python` (e.g., see (Hogan 2014)). Following the differentiation chain rule, it is possible to accumulate the final result in two directions, namely the forward mode and backward/reverse mode (Linnainmaa 1976; Griewank 2012). Being widely used in DL, BP is essentially AD of the reverse mode.

Numerical differentiation is another differentiation tech-

*Corresponding author

nique, where one needs to neither derive the analytic formula of the differentiation nor to apply the chain rule over the computation graph. Given a function f with a small perturbation h applied, the forward finite difference (FFD) approximates the derivative as the ratio between the function variation and perturbation size:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0+h) - f(x_0)}{h} = \frac{f(x_0+h) - f(x_0)}{h} + \mathbf{O}(h). \quad (1)$$

It appears that the smaller h is, the better approximation Eq. (1) delivers. However, we are not allowed to make h arbitrarily small to improve the precision of Eq. (1). This is because the subtraction between two nearly equal numbers (i.e., $f(x_0+h) - f(x_0)$) eliminates many of their significant digits and contaminates the result. This numerical issue is known as the subtraction cancellation (Muller et al. 2018). For instance, with a simple four-digit decimal floating-point system, a real number $a = 1999.99$ is encoded as $\widetilde{a} = 1.999 \times 10^3$ i.e., only a four-digit mantissa. Here, we use $\widetilde{(\cdot)}$ to denote a digitized number in this floating-point system. We simply choose the round-by-chop rule that discards all the out-of-precision digits. The corresponding round-off error is $E_{round} = |a - \widetilde{a}|/|a| = |1999.99 - 1.999 \times 10^3|/|1999.99| \approx 4.95 \times 10^{-4}$. Next, let $b = 1998.88$, which is represented as $\widetilde{b} = 1.998 \times 10^3$. The error of calculating $a - b$ becomes $E_{subtraction} = |(\widetilde{a} - \widetilde{b}) - (a-b)|/|a-b| \approx 0.1$, a thousandfold increase! Clearly, the rounding loses the least important significant digit, and it only yields an error at the order of the floating-point precision (10^{-4}). However, the subtraction between $\widetilde{a}$ and $\widetilde{b}$ eliminates three leading significant digits, which yields a much more substantial error. This is why the cancellation of subtracting numbers of similar magnitude is also called *catastrophic cancellation*. Some numerical literature (e.g., see (Nocedal and Wright 2006) considers that the central finite difference method (CFD) in the form of: $f(x_0) = 2(f(x_0+h) - f(x_0-h))/h + \mathbf{O}(h^2)$, has higher accuracy. This conclusion is only valid when the subtractive cancellation does not occur. In reality, CFD could be even more sensitive to a smaller h because of its faster convergent rate (Fig. 1).

CSFD and Generalization

The subtractive cancellation can be avoided by CSFD (Squire and Trapp 1998; Lyness 1968; Martins, Sturdza, and Alonso 2003). Let $(\cdot)^*$ denote a complex variable, and suppose $f^* : \mathbb{C} \to \mathbb{C}$ is differentiable around $x_0 + 0i$. Setting the perturbation hi imaginary, f^* can be expanded via the complex Taylor expansion as:

$$f^*(x_0+hi) = f^*(x_0) + f^{*'} \cdot hi + \frac{1}{2} f^{*''} \cdot (hi)^2 + \mathbf{O}(h^3). \quad (2)$$

Any elementary function can be "promoted" to be a complex-value one by allowing complex inputs while retaining its original computation. As long as the input of f^* is real, f^* overlaps with f such that $f^*(x_0) = f(x_0) \in \mathbb{R}$ and $f^{*'}(x_0) = f'(x_0) \in \mathbb{R}$, etc. Extracting imaginary parts of both sides of Eq. (2) yields: $\mathtt{Im}(f^*(x_0+hi)) = \mathtt{Im}\left(f^*(x_0) + f^{*'} \cdot hi + \frac{1}{2} f^{*''} \cdot (hi)^2 + \mathbf{O}(h^3)\right)$, and we then have the first-order CSFD approximation:

$$f'(x_0) = \frac{\mathtt{Im}(f^*(x_0+hi))}{h} + \mathbf{O}(h^2). \quad (3)$$

Comparing with Eq. (1), Eq. (3) does not have a subtractive numerator suggesting it only has the round-off error regardless of the size of the perturbation. In addition, the operation of $\mathtt{Im}(\cdot)$ removes the $(hi)^2$ term in the complex Taylor expansion suppressing the approximation error to $\mathbf{O}(h^2)$. We know that any floating-point computation could induce a rounding error ϵ a.k.a. *machine epsilon*. For the double precision under `IEEE 754` (IEEE 1985), $\epsilon \approx 1.11 \times 10^{-16}$. CSFD approximation error reaches the order of ϵ if $h \leq \sqrt{\epsilon}$.

An example is plotted in Fig. 1, where we compare the relative error of numerical derivatives of $f(x) = e^x/(x^2+1)$ using FFD, CFD, and CSFD with its analytic derivative at $x = 10$. The numerical behavior of FFD and CFD is consistent with our analysis: when h decreases, CFD converges faster than FFD initially. Both soon hit the threshold of subtractive cancellation. After that, the relative error bounces back. CSFD lowers the error as quickly as CFD does, and the relative error stably remains at the order of ϵ.

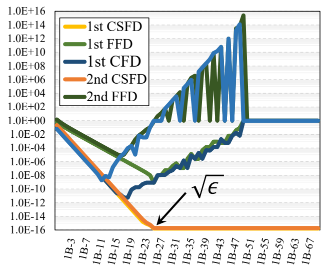

Figure 1: Relative error of different numerical differentiation schemes for $f(x) = e^x/(x^2+1)$ at $x = 10$.

It is mentionable that analytic differentiation of basic operators can also be obtained via *dual number* (Revels, Lubin, and Papamarkou 2016). The concept of dual number is similar to complex number but the imaginary part of a dual number vanishes at the second order (i.e., $i^2 = 0$ for dual numbers whereas $i^2 = -1$ for complex numbers). This specification automatically eliminates high-order terms in the Taylor expansion of f^* (Eq. (2)), and dual number thus gives the exact differentiation formulation. From this perspective, dual number is more a man-crafted algebraic rule. Unlike complex numbers, generalizing dual number arithmetic to trigonometric, exponential, or logarithmic functions is ill-defined.

High-order complex-step finite difference

There are several generalizations of complex arithmetics. A well-known one is quaternion (Hamilton 1848), which is often used to represent 3D rotations. Another extension is commonly named as *multicomplex* number. It has been studied in detail numerical textbooks such as (Price 2018) and could be regarded as an implementation of Clifford algebra (Garling 2011).

The multicomplex number is defined recursively: its base cases are the real set $\mathbb{C}^0 = \mathbb{R}$, and the regular complex set

$\mathbb{C}^1 = \mathbb{C}$. $\mathbb{C}^1$ extends the real set ($\mathbb{C}^0$) by adding an imaginary unit i as: $\mathbb{C}^1 = \{x + yi | x, y \in \mathbb{C}^0\}$. The multicomplex number up to an order of n is defined as: $\mathbb{C}^n = \{z_1 + z_2 i_n | z_1, z_2 \in \mathbb{C}^{n-1}\}$. Under this generalization, the multicomplex Taylor expansion becomes:

$$f^\star(x_0 + hi_1 + \cdots + hi_n) = f^\star(x_0) + f^{\star\prime} h \sum_{j=1}^{n} i_j + \frac{f^{\star\prime\prime}}{2} h^2 (\sum_{j=1}^{n} i_j)^2 + \cdots + \frac{f^{\star(k)}}{k!} h^k (\sum_{j=1}^{n} i_j)^k \cdots. \quad (4)$$

Here, $(\sum i_j)^k$ can be computed following the *multinomial theorem* (Bolton 1968), and it contains products of mixed k imaginary directions for k-order terms. Based on Eq. (4), derivative of an arbitrary order can be conveniently obtained by extracting the coefficient at the corresponding imaginary direction. For instance, the second-order CSFD formulation can be obtained from Eq. (4) as:

$$\begin{aligned} \frac{\partial^2 f(x,y)}{\partial x^2} &\approx \frac{\texttt{Im}^{(1,2)}(f(x + hi_1 + hi_2, y))}{h^2}, \\ \frac{\partial^2 f(x,y)}{\partial x \partial y} &\approx \frac{\texttt{Im}^{(1,2)}(f(x + hi_1, y + hi_2))}{h^2}, \end{aligned} \quad (5)$$

where $\texttt{Im}^{(1,2)}$ picks the mixed imaginary direction of $i_1 i_2$. One can easily tell from Eq. (5) that high-order CSFD is also free of subtractive cancellation making it as robust/accurate as the first-order case (e.g., see Fig. 1). Its recursive definition also greatly eases the implementation.

HoD-Net: High-order Differentiable DNNs

With CSFD, it becomes straightforward to build HoD-Nets. Basically, one can promote network inputs or parameters to the complex domain and obtain its high-order differentiation by extracting the imaginary part of the output. Sometimes, the network already involves complex or multi-value neurons (Hirose and Yoshida 2012; Aizenberg, Aizenberg, and Krivosheev 1996). Its differentiability should be enabled with higher-order CSFD. Unfortunately, such naïve strategy could be inefficient making HoD-Net less useful in practice. Recently, Shen and colleagues (2021) give a paradigm of differentiating network input with CSFD for finite element simulation. We show that this idea could be generalized to enable efficient, scalable, and convenient HoD-Net scheme.

CSFD-BP for more effective perturbation

For a function $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$, CSFD is more effective if $M > N$. In this setting, an argument perturbation yields an M-dimension differentiation vector (i.e., one column of $\nabla f \in \mathbb{R}^{M \times N}$). We suspect this is why CSFD is somewhat overlooked, since many DL applications map high-dimension inputs (e.g., an image) into a lower-dimension space (e.g., a classification label). When $N \gg M$, BP is a more efficient solution for the first-order derivative. However, if high-order derivatives are needed, how can we leverage CSFD with maximized efficiency?

Our answer is to integrate CSFD into BP to differentiate along the optimal direction (i.e., forward or backward). Specifically, we consider BP as a generic function which maps network inputs to its first-order derivative. As a result, the second- or higher-differentiation can be obtained by a CSFD-perturbed BP procedure (CSFD-BP). The perturbation is applied at the beginning of the forward pass. Along with the forward pass, the perturbation leads to imaginary values to all the neurons it influences. As h can be understood as the numerical differential of the input parameter, the imaginary parts at other neurons represent the corresponding (partial) differentials induced by h. During the backward pass, BP calculates how such differential is changed along the network, which becomes the second-order derivative of the parameter.

The choice of CSFD or CSFD-BP should be made based on the dimensionality of the network input/output i.e., N and M. If a network yields a higher-dimension output (e.g., image super-resolution (Zhang et al. 2018; Ledig et al. 2017)), CSFD should be preferred over CSFD-BP. Both CSFD and CSFD-BP readily generalize to a higher order with multicomplex perturbations.

Parallelization using Cauchy-Riemann formula

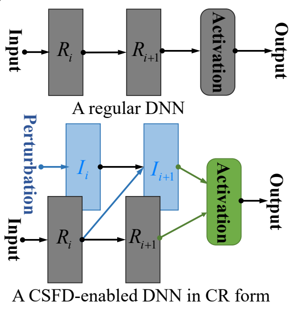

Figure 2: CR formula arranges a network with dedicated imaginary layers to avoid complex overloading at FC/CNN layers, and massive parallelization on GPU becomes straightforward under this configuration.

The dominant network modules of DNNs are often fully connected (FC) layers and convolutional (CNN) layers, where the inter-layer computation mainly consists of BLAS operations and can be parallelized on GPU (e.g., via `cuBLAS` (Barrachina et al. 2008)). The recent version of `PyTorch` (Paszke et al. 2019) and `TensorFlow` (Abadi et al. 2016) does include the support of regular complex operations. However, the complex-based parallelization in BLAS form still lacks, no to mention the multicomplex generalization. We show that CSFD and CSFD-BP can be significantly accelerated by using a complex-free formulation, namely the Cauchy-Riemann (CR) formula (Lang 2013).

In CR formula, a multicomplex number can be represented in the form of a *real matrix*. Specifically, the recursive definition of $\mathbb{C}^n$ allows us to express a rank-n complex number in terms of a matrix of lower-rank complex quantities:

$$z^n = z_0^{n-1} + z_1^{n-1} i_n \in \mathbb{C}^n := \begin{bmatrix} z_0^{n-1} & -z_1^{n-1} \\ z_1^{n-1} & z_0^{n-1} \end{bmatrix}. \quad (6)$$

Here, we use the superscript $(\cdot)^n$ to denote the order of a complex number. Each of the 2×2 blocks in Eq. (6) is a $(n-1)$-order complex number, which can be further expanded with $(n-2)$-order multicomplex numbers and so on until all the matrix elements are real.

Based on CR forms, we re-organize the network into a real and (multiple) imaginary layers so that the imaginary parts of all the neurons are stored in dedicated layers, and all the neurons, including the ones in imaginary layers are all real-value. As shown in Fig. 2, the original network is "duplicated" with an imaginary copy, which is used to house the imaginary values of all the neurons. If a high-order differentiation is needed, more of such duplicates will be created, corresponding to high-order mixed imaginary directions. The real-to-real computation is identical to the original net (i.e., black arrows in the figure). The CR-based net also handles real-to-imaginary and imaginary-to-imaginary propagations. They are just Eq. (6) in the matrix form. For instance, assume R_i and R_{i+1} is related by a weight matrix W as $R_{i+1} = WR_i$ originally. With CR formula, R_{i+1} and I_{i+1} are computed via:

$$\begin{bmatrix} R_{i+1} & -I_{i+1} \\ I_{i+1} & R_{i+1} \end{bmatrix} = \begin{bmatrix} W & -V \\ V & W \end{bmatrix} \begin{bmatrix} R_i & -I_i \\ I_i & R_i \end{bmatrix} \quad (7)$$
$$\Rightarrow R_{i+1} = WR_i, I_{i+1} = VR_i + WI_i.$$

V stores possible imaginary perturbations at network parameters. Therefore, VI_i is $\mathbf{O}(h^2)$ and truncated in Eq. (7). In other words, the real portion of HoD-Net is always identical to the original network. If $V = 0$, computations at R and I are fully independent, indicating the linearity between R_i and R_{i+1}. As all the computations are now in real values, the parallelization is convenient. As reported in our experiment, CR-based implementation outperforms built-in BP method from existing DL frameworks including `PyTorch` and `TensorFlow`.

The activation and some other network modules like batch normalization (BN) (Ioffe and Szegedy 2015) need to be taken care of separately. In general, the activation is applied at each neuron individually, making the parallelization straightforward. We overload nonlinear activation with CSFD to output necessary imaginary information. BN is also treated as a special batch-wise activation with CSFD.

Adjoint Perturbation in HoD-Net

The n-order derivative of $f : \mathbb{R}^N \to \mathbb{R}^M$ yields an $M \times N^n$ tensor with a prohibitive complexity of $\mathbf{O}(N^n)$. Thus, high-order differentiation of a deep network appears infeasible at the first sight. A closer look however, reveals otherwise. This $M \times N^n$ tensor is seldom needed in its original high-rank form. Its rank is often reduced via follow-up contractions, and all the way down to a matrix or a vector. A good example is the adjoint analysis (Giles and Pierce 2000), which avoids an explicit assembly of the Jacobian matrix w.r.t. a large number of parameters. We show that this idea can also be exploited in CSFD as the real-part computations of the network remain unchanged for differentiations under different perturbations. Following the naming of the adjoint method, we refer to our strategy as *adjoint perturbation*.

Adjoint perturbation for right contraction

We now elaborate our method with $M = 1$ or $f : \mathbb{R}^N \to \mathbb{R}$. Computing its Hessian ($\nabla^2 f$) will need N^2 perturbations with second CSFD. However, if $\nabla^2 f$ is (single) contracted with a right vector a, $\nabla^2 \cdot a$ can actually be evaluated as:

$$\begin{aligned}[\nabla^2 f \cdot a]_k &= \sum_{l=0}^{N-1} \lim_{h\to 0} \frac{[\nabla f(x + he_l) - \nabla f(x)]_k}{h} \cdot [a]_l \\ &= \sum_{l=0}^{N-1} \lim_{h\to 0} \frac{[\nabla f(x + [a]_l he_l) - \nabla f(x)]_k}{h}, \end{aligned} \quad (8)$$

where $[\cdot]_k$ gives k-th element of the vector, and e_l is the l-th canonical basis. Here, we substitute h with $[a]_l h$ to cancel the multiplication of $[a]_l$. In the vector form, we have

$$\begin{aligned}\nabla^2 f \cdot a &= \lim_{h\to 0} \frac{\nabla f(x + ha) - \nabla f(x)}{h} \\ &= \frac{\mathtt{Im}(\nabla f(x + hi \cdot a))}{h} + \mathbf{O}(h^2). \end{aligned} \quad (9)$$

One may immediately recognize that $\nabla^2 f \cdot a$ is essentially the *directional derivative* of $\nabla_a f$. The observation of Eq. (9) is not new, and it leads to a collection of so-called Jacobian-free or Hessian-free algorithms (Knoll and Keyes 2004). Some of them have also been used for DL training (Pearlmutter 1994; Martens 2010; Martens and Sutskever 2011).

Due to the linearity of differential, Eq. (9) can be generalized to high-order cases and high-rank tensor contractions. For instance, consider the third-order differentiation $\nabla^3 f = \partial^3 f / \partial x^3 \in \mathbb{R}^{N\times N\times N}$, followed by a right double contraction of a matrix $A \in \mathbb{R}^{N\times N}$. $\nabla^3 f : A$ is a vector, and it can be efficiently computed by a single adjoint perturbation – a third-order complex perturbation over all the N elements of promoted function input $x^\star$. The second and the third imaginary directions of this adjoint perturbation is scaled by the corresponding element in A. The same strategy can be used if $\nabla^3 f$ is right contracted twice by vectors a and b since $(\nabla^3 f \cdot a) \cdot b = \nabla^3 f : (a \otimes b)$. If f has a vector output with $M > 1$, we arrange M adjoint perturbations into batches, which can be parallelized on GPU with ease.

Adjoint perturbation for left contraction

Now, we consider left contractions like $a \cdot \nabla^3 f$. Unlike right contractions, a left contraction occurs at the dimension which is not expanded by the differentiation. Hence, the strategy of adjoint perturbation does not apply directly. We carry out our computation with an auxiliary function $g(x) = a \cdot f(x) \in \mathbb{R}$. This auxiliary function can also be viewed as appending an FC layer at the end of the net reducing its M-dimension output to a single scalar. Because a is independent on f, $a \cdot \nabla^3 = \nabla^3 g$. Knowing g is a scalar-value network ($N > M = 1$) suggests CSFD-BP more effective for the differentiation. $a \cdot \nabla^3 f$ is then computed via a single CSBP-BP pass using third-order complex perturbations.

Adjoint perturbation can be applied with both left and right contractions simultaneously. Using this technique, *the time complexity no longer depends on the differentiation order*. This makes the high-order differentiation of DNNs tangible in practice.

Results and Applications

We implement HoD-Net with `PyTorch` in CR form on a workstation computer with `Intel i9-10980XE` CPU and an nVidia 3090 GPU with 24G G-memory. In time critical applications, we port the trained network to CUDA, and use cuBLAS (Nvidia 2008) for CR-based network pass.

Time performance comparison

We first investigate the time performance of HoD-Net with off-the-shelf AD routines in PyTorch (autograd) and TensorFlow (GradientTape) for gradient computation. We also compare HoD-Net with JAX (Schoenholz and Cubuk 2019) in Hessian computation. In this experiment, we use three networks with three representative input/output configurations: 1) $N = 40$, $M = 400$; 2) $N = 200$, $M = 200$; 3) $N = 400$, $M = 40$. All the nets have 8 FC layers, and each layer is activated by ELU.

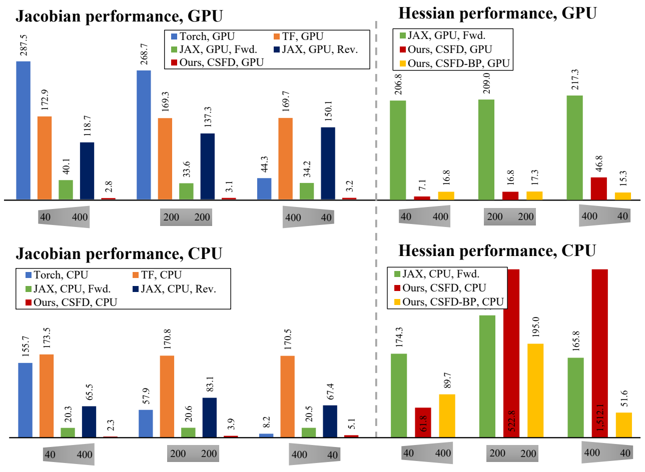

Figure 3: Time performance comparison among HoD-Net, built-in AD routines from PyTorch (Torch) and TensorFlow (TF), and JAX. We also compare HoD-Net and JAX in Hessian calculation. The networks tested have 8 FC layers with ELU activations. All the timing statistics are in millisecond.

The result is reported in Fig. 3. In the first-order case, CSFD-BP is not an option, and we only use CSFD-based HoD-Net implementation. This is a forward process. As a result, $40 \to 400$ is faster than $400 \to 40$ on both CPU and GPU. In general, HoD-Net significantly outperforms AD-based packages. This is because AD needs to build the computation graph, which is not required with HoD-Net. In JAX and TensorFlow, it is possible to pre-build this graph in the forward AD mode. If this option is on, the performance of HoD-Net is comparable to JAX and TensorFlow. However, the computation graph is always needed in reverse AD mode in current version of JAX. In Hessian computation, HoD-Net outperforms JAX when the differentiation direction is probably set. For instance, if the net is $40 \to 400$, CSFD-based HoD-Net should be adopted; and if the net is $400 \to 40$, CSFD-BP should be used. JAX recursively uses first-order AD, and it is not sensitive to the differentiation order in Hessian calculation (as it will always need one reverse accumulation and one forward accumulation). Higher-order differentiation with JAX is orders of magnitude slower than HoD-Net (even using the Taylor-mode (Bettencourt, Johnson, and Duvenaud 2019)). This is because HoD-Net does not need to perform high-order chain rule as in (Bettencourt, Johnson, and Duvenaud 2019).

Applications of HoD-Net

We now showcase how to leverage high-order differentiability of HoD-Net in a few concrete applications. It is not our intention to claim that HoD-Net should always be used in *any* differentiation scenarios. However, HoD-Net provides novel and orthogonal perspectives to high-order differentiation and enables a wide range of applications.

Application I: High-order DL finetuning It has been noticed that training deep networks with gradient-based algorithms exhibits distinct behaviors at early and late stages (usually empirically divided by the validation error plateau, or by the learning-rate switch points) (Li, Wei, and Ma 2019; Frankle, Schwab, and Morcos 2029; Leclerc and Madry 2020). Many observations of "simplicity bias" by SGD and variants (Arpit et al. 2017; Kalimeris et al. 2019; Valle-Perez, Camargo, and Louis 2019) endorse their roles to contribute to the early stage. High-order optimization on the other hand, could benefit the second stage (a.k.a. network finetuning). To this end, we design a Newton-Krylov-based (Knoll and Keyes 2004) training algorithm with HoD-Net. Unlike pseudo second-order methods (e.g., (Gupta, Koren, and Singer 2018; Martens and Grosse 2015; Osawa et al. 2019)), where low-rank Hessian approximations are used, HoD-Net allows us to access the full information of $\nabla^2 f$ during the training.

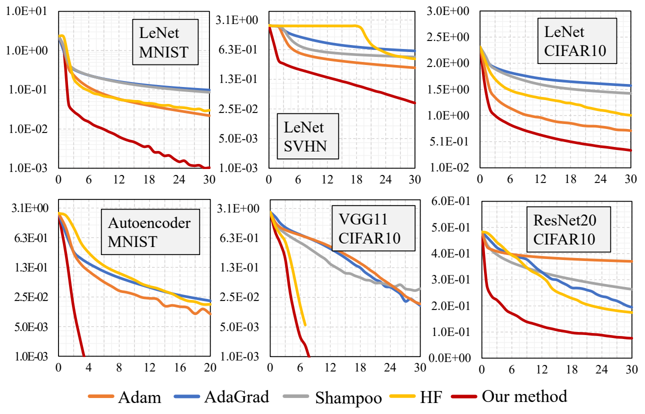

Figure 4: Training curves among different training methods. The x axis is the epoch index.

Our HoD-Net-based second-order finetuning algorithm is given in Alg. 1. Basically, our method performs the training in a Newton-Krylov way – the linear system at each Newton step is solved with a conjugate-gradient (CG)-like procedure. In DL, f exhibits high nonlinearity, and the vanilla CG is infeasible. If a problematic search direction is identified (i.e., see Fig. 5), we skip the current Hessian sample and move to the next batch.

We compare our method with several well-known algorithms including Adam (Kingma and Ba 2014), AdaGrad (Duchi, Hazan, and Singer 2011), Shampoo (Gupta, Koren, and Singer 2018), and Hessian-free method (HF) (Martens 2010) during the training of autoencoder (AE) (Kramer 1991), LeNet-5 (LeCun et al. 1998), VGG-11 (Simonyan and Zisserman 2014), and ResNet-20 (He et al. 2016). The datasets used are MNIST, SVHN, and CIFAR10. The training curves are plotted in Fig. 4. In a nutshell, HoD-Net provides an efficient approach to extract curvature information of the optimization manifold, and our method shows a superior performance in those DL tasks.

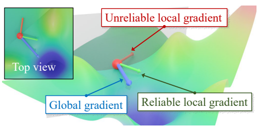

Figure 5: We skip unreliable batch by comparing local and global gradients.

Application II: High-order contractive autoencoder Contractive autoencoder or CAE (Rifai et al. 2011b) extracts more robust features by penalizing the Frobenius norm of the Jacobian matrix of an encoder (w.r.t. the input). This regularization helps to make the features more consistent in different directions around the training data. It is measured by the *contractive ratio* i.e., the ratio between the distance of two points in the feature space and the original space. By extending Jacobian-based penalty term to the second order, we show that better contractive ratios could be achieved.

A practical issue is the dimensionality of the Hessian $\nabla^2 f$ – building the Hessian explicitly consumes significant memory for large-scale CAE nets. As a compromise, we only control the trace of $\nabla^2 f \cdot \nabla^2 f$, which partially measures the mean curvature of the encoder f:

$$\mathtt{Tr}(\nabla^2 f \cdot \nabla^2 f) = \sum_{ij} \left(\frac{\partial^2 [f(x)]_j}{\partial x_i^2} \right)^2. \tag{10}$$

The final loss includes both the norm of Jacobian and Hessian trace:

$$L_{\text{CAE}} = \mathtt{MSE}(x, \widetilde{x}) + \alpha \, \|\nabla_f(x)\|_F^2 + \beta \mathtt{Tr}(\nabla^2 f \cdot \nabla^2 f), \tag{11}$$

where $\widetilde{x}$ is the reconstructed signal. α and β are two hyperparameters controlling the strength of the regularization.

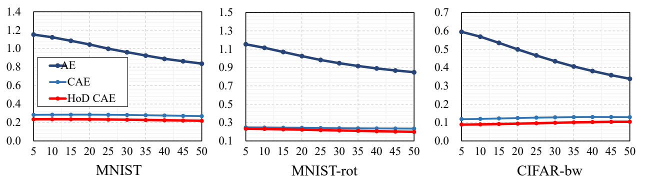

Figure 6: Contractive ratios on different datasets. The x axis represents the radius.

The results are reported in Fig. 6. We compare HoD CAE with the traditional autoencoder and Jacobian-penalized CAE (Rifai et al. 2011b) on MNIST, MNIST-rot (Vincent et al. 2010), and CIFAR-bw, a grey-scale version of CIFAR-10. The network is a 4-layer MLP activated by ELU, which

ALGORITHM 1: HoD-Net Newton-Krylov training.

```
Input: minibatch set $\{B_1, B_2, \cdots\}, \widetilde{\eta}$
compute $\nabla f$; // $\nabla f$ is the global gradient
$j \leftarrow 0$; // minibatch index
for each $B_j$ do
    compute $\nabla f_j$; // $\nabla f_j$ is the local gradient sampled
        at mini-batch $B_i$
    if $\nabla f_j \cdot \nabla f < 0$ then
        $j \leftarrow j + 1$;
        continue; // skip this loop for $B_j$
    end
    $i \leftarrow 0, \quad \Delta w \leftarrow \Delta \widetilde{w}$;
    $\langle \nabla f, h_i \rangle \leftarrow$ CSFDBP$(w, \Delta w)$;// $h_i$ is the directional
        derivative of $f(w)$ under $\Delta w$
    $r_i \leftarrow -(\nabla f + h_i), \quad p_i \leftarrow r_j$;// $r_i$ is the residual; $p_i$
        is the search direction
    while $\|r_i\|$ is not small enough do
        $\kappa = p_i \cdot \nabla^2 f \cdot p_i$; // $\kappa$ is computed via adjoint
            perturbation with left and right
            contractions
        if $\kappa < 0$ then
            break; // negative curvature, early
                termination
        end
        if $0 < \kappa < 1.0 \times 10^{-8}$ then
            $\kappa \leftarrow 0.01 \cdot \|p_j\|$; // local flatness, move
                forward with momentum
        end
        $\alpha_i \leftarrow \|r_i\|/\kappa$;
        $\Delta w \leftarrow \Delta w + \alpha_i p_i$;
        $q_i \leftarrow$ CSFDBP$(w, p_i)$; // $q_i = \nabla^2 f \cdot p_i$
        $r_{i+1} \leftarrow r_i - \alpha_i q_i$;
        $\beta_j \leftarrow \|r_{i+1}\|^2 / \|r_j\|^2$;
        $p_{i+1} \leftarrow r_{i+1} + \beta_i p_i$;
        $i \leftarrow i + 1$;
    end
    if $\Delta w \cdot \nabla f > 0$ then
        $\Delta w \leftarrow (1 + \frac{\Delta w \cdot \nabla f}{\|\Delta w\| \cdot \|\nabla f\|}) \Delta w$;
    end
    $\gamma \leftarrow 1$; // the default step size
    compute $\eta$;
    if $\eta > 0.5\widetilde{\eta}$ then
        $s \leftarrow |\widetilde{\eta}/\eta|$; // an initial test size
        while $\eta \notin [0.5\widetilde{\eta}, \widetilde{\eta}]$ do
            if $\eta > \widetilde{\eta}$ then
                $\gamma \leftarrow 0.5 \cdot \gamma$; // shrink a step
            end
            else
                $\gamma \leftarrow 1.5 \cdot \gamma$; // stretch a step
            end
            update $\eta$ with $f(w + \gamma \cdot \Delta w)$;
        end
    end
    $w \leftarrow w + \gamma \cdot \Delta w$;
    compute $g$;
    $j \leftarrow j + 1$;
end
```

maps the input to a hidden feature with dimension of 400. For the first-order regular CAE, HoD-Net trains this MLP 2.4 times faster than `PyTorch`. Unlike existing Hessian-based CAE (Rifai et al. 2011a), HoD CAE is not stochastic. The decoder net always produces a high-dimension output from a low-dimension hidden representation. More importantly, HoD-Net does not rely on analytic solution for the gradient of each penalty, and is capable to evaluate high-order derivatives of an encoder with any number of layers.

We generate random samples with different radius as the

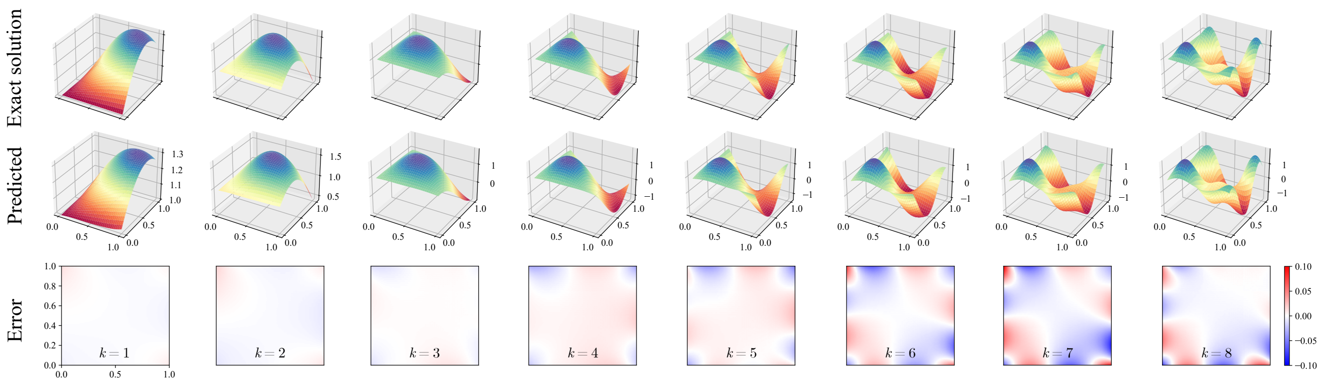

Figure 7: HoD-Net for PINN. We test the accuracy of HoD-Net for a high-dimension Poisson problems defined on the 2D domain of $[0, 1] \times [0, 1]$. The ground truth, network prediction, and the prediction errors are visualized with colormaps.

test samples, and compare the distance on the hidden representation. The lower contractive ratios show that HoD CAE is more robust to the directions orthogonal to the manifold. We take the trained encoder as a head of an MLP for the classification task on the same datasets. As shown in Table 1, HoD CAE leads to higher test accuracy than CAE. The advantage of HoD CAE is more noticeable with bigger and complex data sets.

Data set	Vanilla AE	Jacobian CAE	HoD CAE
MNIST	1.61	1.30	1.25
MNIST-rot	10.9	10.4	10.1
CIFAR-bw	54.7	50.2	49.3

Table 1: Test error on different datasets. HoD CAE makes use of both Jacobian and Hessian trace to enhance the contractive ratio.

Application III: Reduced-order finite element simulation Physics simulation has been widely used in modern science and engineering disciplines. Full-scale simulation is known to be costly and often prohibitive for interactive applications. To relieve the computation burden, reduced-order simulation (Przekop and Rizzi 2006; Sifakis and Barbic 2012) defines a compact set of *generalized coordinates* to prescribe the dynamics of the model. Classic reduced simulations use linear maps like PCA (Treuille, Lewis, and Popović 2006) and modal analysis (Hurty 1960). Recently, DL has also been exploited to establish the nonlinear map between generalized and fullspace coordinates, often in the structure of encoder-decoder (Fulton et al. 2019; Shen et al. 2021).

Clearly, HoD-Net is a powerful tool for this purpose. In Newtonian dynamics, any map of model's trajectory takes at least second-order differentiation to extract necessary inertia information (i.e., the acceleration-triggered dynamics). Unlike (Shen et al. 2021), our model reduction is fully network based (i.e., without using PCA). Under FEM discretization, the motion of a deformable solid can be described with the *Euler-Lagrange equation*: $M\ddot{u} + f_{int}(u) = f_{ext}$. Here, M is the mass matrix; u is the unknown displacement of the

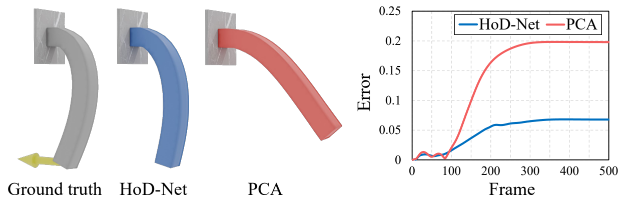

Figure 8: Bending an elastic beam. The error at the tip of the beam is plotted on the right. HoD-Net enables faster simulation (over 30×), and is more accurate than PCA with the same subspace size.

mesh; f_{int} and f_{ext} are the nonlinear internal force and the external force. We use the generalized coordinate q such that $u = f(q)$, and f is a HoD-Net trained from existing simulation poses of the model. Applying time differentiation at both sides yields:

$$\begin{aligned} \dot{u} &= \frac{\mathrm{d}f(q)}{\mathrm{d}t} = \frac{\partial f(q)}{\partial q}\dot{q} = \nabla f \cdot \dot{q}, \\ \ddot{u} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial f(q)}{\partial q}\dot{q}\right) = \nabla^2 f : (\dot{q} \otimes \dot{q}) + \nabla f \cdot \ddot{q}. \end{aligned} \tag{12}$$

Eq. (12) is then used to build the reduced equation of motion. During the simulation, we need third-order differentiation of f in order to use Newton's method to compute u, which is only possible with HoD-Net. The snapshots of a reduced-order simulation are reported in Fig 8. HoD-Net maps a 1542-dimension problem to a 6-dimension latent space and achieves a 30× speedup with lower accuracy loss than PCA.

Application IV: Physics-informed neural network As a differentiation modality, HoD-Net is also useful for solving neural PDEs problems (Hsieh et al. 2019; Yang, Meng, and Karniadakis 2021) a.k.a. physics-informed neural networks (PINNs). In PINN, the network takes a coordinate vector as input and predicts the value of the PDE at the input coordinate, for which the network is representing. With HoD-Net, PINNs are more expressive for high-dimension and high-order PDEs.

A concrete example is shown in Fig. 7. In this experiment, we solve a high-dimension Poisson $\nabla^2\phi(x) = f$:

$$[f]_k = -2k^2\sin(kx_1)\sin(kx_2) \\ - k^2x_1^2\cos(kx_1x_2) - k^2x_2^2\cos(kx_1x_2), \quad (13)$$

for k from 1 to 8. The boundary conditions are $[\phi]_k(0, x_2) = 1$, $[\phi]_k(x_1, 0) = 1$, $[\phi]_k(1, x_2) = \sin(k)\sin(kx_2) + \cos(kx_2)$, and $[\phi]_k(x_1, 1) = \sin(k)\sin(kx_1) + \cos(kx_1)$. For this specific driving function, the Poisson problem has an analytic solution of $[\phi]_k(x_1, x_2) = \sin(kx_1)\sin(kx_2) + \cos(kx_1x_2)$, which is used as the ground truth. The network structure is $2 \to 100 \to 100 \to 100 \to 100 \to 100 \to 8$, and each hidden layer is activated by `tanh`. With HoD-Net, we can directly access the Laplacian of the network (and its derivative) to supervise the training based on the MSE loss.

Conclusion

HoD-Net is a numerical differentiation method for neural networks. It uses CSFD as primary differentiation mechanism. By extending the complex arithmetic to multicomplex realms, we can achieve network differentiation of an arbitrarily high order with a similar complex-like computation procedure. This makes HoD-Net fundamentally different from AD or BP. HoD-Net does not need to build the computation graph and does not rely on high-order chain rule, making its computation lightweight. In fact, we show that HoD-Net synergizes well with AD to maximize the information of differential obtained from each perturbation. We also present a novel scheme of adjoint perturbation, which reduces the time complexity of high-order differentiation by orders. In addition, CR-form based HoD-Net can be efficiently parallelized on GPU.

HoD-Net also has limitations. For instance, CR formula duplicates network copies for storing the perturbations at imaginary directions, which could be memory demanding for large-scale networks. Fortunately, HoD-Net also removes the need of computation graph, which saves memory and computations back. HoD-Net allows applications to better use the information hidden in the high-order differentiations such as geometry curvature (Michalkiewicz et al. 2019), motion acceleration (Shen et al. 2021), shape sensitivity (Kubilius, Bracci, and Op de Beeck 2016), light reflectance (Zhang et al. 2019) etc.

Acknowledgements

We thank anonymous reviewers and AC for their constructive comments. Tianjia Shao is partially supported by NSF of China (No. 61772462, U1736217), and the 100 Talents Program of Zhejiang University. Kun Zhou is partially supported by NSF of China (No. 61890954). Chenfanfu Jiang is supported in part by NSF (1943199, 1813624, 2023780) and DOE (4000171342). Feng Luo is supported in part by NSF (ABI-1759856 and MTM2-2025541 to FL). Yin Yang is partially supported by NSF (2016414, 2011471).

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 265–283.

Aizenberg, N. N.; Aizenberg, I. N.; and Krivosheev, G. A. 1996. Multi-valued and universal binary neurons: mathematical model, learning, networks, application to image processing and pattern recognition. In *Proceedings of 13th International Conference on Pattern Recognition*, volume 4, 185–189. IEEE.

Arpit, D.; Jastrzębski, S.; Ballas, N.; Krueger, D.; Bengio, E.; Kanwal, M. S.; Maharaj, T.; Fischer, A.; Courville, A.; and Bengio, Y. 2017. A closer look at memorization in deep networks. In *International Conference on Machine Learning*, 233–242. PMLR.

Barrachina, S.; Castillo, M.; Igual, F. D.; Mayo, R.; and Quintana-Orti, E. S. 2008. Evaluation and tuning of the level 3 CUBLAS for graphics processors. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, 1–8. IEEE.

Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; and Siskind, J. M. 2018. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18.

Betancourt, M. 2018. A Geometric Theory of Higher-Order Automatic Differentiation. *arXiv preprint arXiv:1812.11592*.

Bettencourt, J.; Johnson, M. J.; and Duvenaud, D. 2019. Taylor-Mode Automatic Differentiation for Higher-Order Derivatives in JAX.

Bolton, D. 1968. The multinomial theorem. *The Mathematical Gazette*, 336–342.

Bücker, H. M.; Corliss, G.; Hovland, P.; Naumann, U.; and Norris, B. 2006. *Automatic differentiation: applications, theory, and implementations*, volume 50. Springer Science & Business Media.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul): 2121–2159.

Frankle, J.; Schwab, D. J.; and Morcos, A. S. 2029. The Early Phase of Neural Network Training. In *International Conference on Learning Representations*.

Fulton, L.; Modi, V.; Duvenaud, D.; Levin, D. I.; and Jacobson, A. 2019. Latent-space dynamics for reduced deformable simulation. In *Computer graphics forum*, volume 38, 379–391. Wiley Online Library.

Garling, D. J. 2011. *Clifford algebras: an introduction*, volume 78. Cambridge University Press.

Giles, M. B.; and Pierce, N. A. 2000. An introduction to the adjoint approach to design. *Flow, turbulence and combustion*, 65(3): 393–415.

Griewank, A. 2012. Who invented the reverse mode of differentiation. *Documenta Mathematica, Extra Volume ISMP*, 389–400.

Gupta, V.; Koren, T.; and Singer, Y. 2018. Shampoo: Preconditioned stochastic tensor optimization. *arXiv preprint arXiv:1802.09568*.

Hamilton, W. R. 1848. Xi. on quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 33(219): 58–60.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hecht-Nielsen, R. 1992. Theory of the backpropagation neural network. In *Neural networks for perception*, 65–93. Elsevier.

Hirose, A.; and Yoshida, S. 2012. Generalization characteristics of complex-valued feedforward neural networks in relation to signal coherence. *IEEE Transactions on Neural Networks and learning systems*, 23(4): 541–551.

Hogan, R. J. 2014. Fast reverse-mode automatic differentiation using expression templates in C++. *ACM Transactions on Mathematical Software (TOMS)*, 40(4): 26.

Hsieh, J.-T.; Zhao, S.; Eismann, S.; Mirabella, L.; and Ermon, S. 2019. Learning neural PDE solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*.

Hurty, W. C. 1960. Vibrations of structural systems by component mode synthesis. *Journal of the Engineering Mechanics Division*, 86(4): 51–69.

IEEE. 1985. IEEE standard for binary floating-point arithmetic. Institute of Electrical and Electronic Engineers.

Ioffe, S.; and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 448–456. PMLR.

Kalimeris, D.; Kaplun, G.; Nakkiran, P.; Edelman, B. L.; Yang, T.; Barak, B.; and Zhang, H. 2019. SGD on Neural Networks Learns Functions of Increasing Complexity. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Knoll, D. A.; and Keyes, D. E. 2004. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2): 357–397.

Kramer, M. A. 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2): 233–243.

Kubilius, J.; Bracci, S.; and Op de Beeck, H. P. 2016. Deep neural networks as a computational model for human shape sensitivity. *PLoS computational biology*, 12(4): e1004896.

Lang, S. 2013. *Complex analysis*, volume 103. Springer Science & Business Media.

Leclerc, G.; and Madry, A. 2020. The two regimes of deep network training. *arXiv preprint arXiv:2002.10376*.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Ledig, C.; Theis, L.; Huszár, F.; Caballero, J.; Cunningham, A.; Acosta, A.; Aitken, A.; Tejani, A.; Totz, J.; Wang, Z.; et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4681–4690.

Li, Y.; Wei, C.; and Ma, T. 2019. Towards explaining the regularization effect of initial large learning rate in training neural networks. *Advances in neural information processing systems*.

Linnainmaa, S. 1976. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2): 146–160.

Lyness, J. 1968. Differentiation formulas for analytic functions. *Mathematics of Computation*, 352–362.

Maple, V. 1994. Waterloo maple software. *University of Waterloo, Version*, 5.

Margossian, C. C. 2018. A Review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, e1305.

Martens, J. 2010. Deep learning via hessian-free optimization. In *ICML*, volume 27, 735–742.

Martens, J.; and Grosse, R. 2015. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, 2408–2417.

Martens, J.; and Sutskever, I. 2011. Learning recurrent neural networks with hessian-free optimization. In *ICML*.

Martins, J. R.; Sturdza, P.; and Alonso, J. J. 2003. The complex-step derivative approximation. *ACM Transactions on Mathematical Software (TOMS)*, 29(3): 245–262.

McCulloch, W. S.; and Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4): 115–133.

Michalkiewicz, M.; Pontes, J. K.; Jack, D.; Baktashmotlagh, M.; and Eriksson, A. 2019. Implicit surface representations as layers in neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 4743–4752.

Morton, K. W.; and Mayers, D. F. 2005. *Numerical solution of partial differential equations: an introduction*. Cambridge university press.

Muller, J.-M.; Brisebarre, N.; De Dinechin, F.; Jeannerod, C.-P.; Lefevre, V.; Melquiond, G.; Revol, N.; Stehlé, D.; Torres, S.; et al. 2018. *Handbook of floating-point arithmetic*, volume 1. Springer.

Neidinger, R. D. 2010. Introduction to automatic differentiation and MATLAB object-oriented programming. *SIAM review*, 52(3): 545–563.

Nocedal, J.; and Wright, S. 2006. *Numerical optimization*. Springer Science & Business Media.

Nvidia, C. 2008. Cublas library. *NVIDIA Corporation, Santa Clara, California*, 15(27): 31.

Osawa, K.; Tsuji, Y.; Ueno, Y.; Naruse, A.; Yokota, R.; and Matsuoka, S. 2019. Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12359–12367.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Pearlmutter, B. A. 1994. Fast exact multiplication by the Hessian. *Neural computation*, 6(1): 147–160.

Phipps, E.; and Pawlowski, R. 2012. Efficient expression templates for operator overloading-based automatic differentiation. In *Recent Advances in Algorithmic Differentiation*, 309–319. Springer.

Price, G. B. 2018. *An introduction to multicomplex spaces and functions*. CRC Press.

Przekop, A.; and Rizzi, S. A. 2006. Nonlinear reduced order random response analysis of structures with shallow curvature. *AIAA journal*, 44(8): 1767–1778.

Rall, L. B.; and Corliss, G. F. 1996. An introduction to automatic differentiation. *Computational Differentiation: Techniques, Applications, and Tools*, 89.

Revels, J.; Lubin, M.; and Papamarkou, T. 2016. Forward-mode automatic differentiation in Julia. *arXiv preprint arXiv:1607.07892*.

Rifai, S.; Mesnil, G.; Vincent, P.; Muller, X.; Bengio, Y.; Dauphin, Y.; and Glorot, X. 2011a. Higher order contractive auto-encoder. In *Joint European conference on machine learning and knowledge discovery in databases*, 645–660. Springer.

Rifai, S.; Vincent, P.; Muller, X.; Glorot, X.; and Bengio, Y. 2011b. Contractive auto-encoders: Explicit invariance during feature extraction. In *Icml*.

Rumelhart, D. E.; Durbin, R.; Golden, R.; and Chauvin, Y. 1995. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, 1–34.

Schoenholz, S. S.; and Cubuk, E. D. 2019. Jax md: End-to-end differentiable, hardware accelerated, molecular dynamics in pure python.

Shen, S.; Yang, Y.; Shao, T.; Wang, H.; Jiang, C.; Lan, L.; and Zhou, K. 2021. High-Order Differentiable Autoencoder for Nonlinear Model Reduction. *ACM Trans. Graph.*, 40(4).

Sifakis, E.; and Barbic, J. 2012. FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction. In *Acm siggraph 2012 courses*, 1–50.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Squire, W.; and Trapp, G. 1998. Using complex variables to estimate derivatives of real functions. *SIAM review*, 40(1): 110–112.

Treuille, A.; Lewis, A.; and Popović, Z. 2006. Model reduction for real-time fluids. *ACM Transactions on Graphics (TOG)*, 25(3): 826–834.

Utke, J.; Naumann, U.; Fagan, M.; Tallent, N.; Strout, M.; Heimbach, P.; Hill, C.; and Wunsch, C. 2008. OpenAD/F: A modular open-source tool for automatic differentiation of Fortran codes. *ACM Transactions on Mathematical Software (TOMS)*, 34(4): 1–36.

Valle-Perez, G.; Camargo, C. Q.; and Louis, A. A. 2019. Deep learning generalizes because the parameter-function map is biased towards simple functions. In *International Conference on Learning Representations*.

van Merrienboer, B.; Moldovan, D.; and Wiltschko, A. B. 2018. Tangent: Automatic differentiation using source-code transformation for dynamically typed array programming. *arXiv preprint arXiv:1809.09569*.

Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; Manzagol, P.-A.; and Bottou, L. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12).

Wolfram, S.; et al. 1996. *Mathematica*. Cambridge university press Cambridge.

Yang, L.; Meng, X.; and Karniadakis, G. E. 2021. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425: 109913.

Zhang, C.; Wu, L.; Zheng, C.; Gkioulekas, I.; Ramamoorthi, R.; and Zhao, S. 2019. A differential theory of radiative transfer. *ACM Transactions on Graphics (TOG)*, 38(6): 1–16.

Zhang, Y.; Tian, Y.; Kong, Y.; Zhong, B.; and Fu, Y. 2018. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2472–2481.