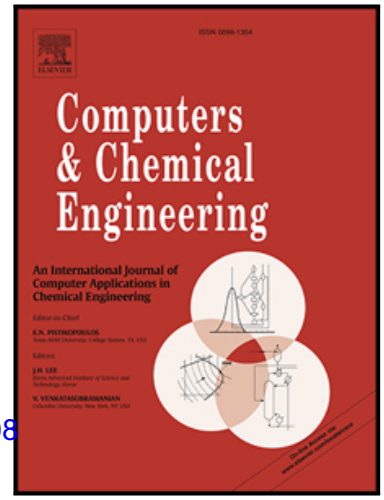
Journal Pre-proof



A Hessian-Free Gradient Flow (HFGF) Method for the Optimisation of Deep Learning Neural Networks

Sushen Zhang, Ruijuan Chen, Wenyu Du, Ye Yuan, Vassilios S. Vassiliadis

PII: S0098-1354(20)30356-2
DOI: https://doi.org/10.1016/j.compchemeng.2020.107008
Reference: CACE 107008

To appear in: *Computers and Chemical Engineering*

Received date: 20 April 2020
Revised date: 27 June 2020
Accepted date: 5 July 2020

Please cite this article as: Sushen Zhang, Ruijuan Chen, Wenyu Du, Ye Yuan, Vassilios S. Vassiliadis, A Hessian-Free Gradient Flow (HFGF) Method for the Optimisation of Deep Learning Neural Networks, *Computers and Chemical Engineering* (2020), doi: https://doi.org/10.1016/j.compchemeng.2020.107008

Highlights

- Design of a novel very large scale unconstrained local optimisation algorithm
- Gradient Flow based Quasi-Newton algorithm with full convergence proof
- Hessian-free approach, tested on nonlinear problems of up to 1 million variables
- Designed specifically for training very large-scale neural networks and datasets
- Tested successfully on industrial chemical and computer vision datasets

A Hessian-Free Gradient Flow (HFGF) Method for the Optimisation of Deep Learning Neural Networks

Sushen Zhang[a,*], Ruijuan Chen[b], Wenyu Du[c], Ye Yuan[b], Vassilios S. Vassiliadis[d]

[a]*Department of Chemical Engineering and Biotechnology, University of Cambridge, UK*
[b]*School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China*
[c]*School of Engineering, Westlake University, China; Institute of Advanced Technology, Westlake Institute for Advanced Study*
[d]*Cambridge Simulation Solutions LTD., Waterbeach, Cambridge, UK*

Abstract

This paper presents a novel optimisation method, termed Hessian-free Gradient Flow, for the optimisation of deep neural networks. The algorithm entails the design characteristics of the Truncated Newton, Conjugate Gradient and Gradient Flow method. It employs a finite difference approximation scheme to make the algorithm Hessian-free and makes use of Armijo conditions to determine the descent condition. The method is first tested on standard testing functions with a high optimisation model dimensionality. Performance on the testing functions has demonstrated the potential of the algorithm to be applied to large-scale optimisation problems. The algorithm is then tested on classification and regression tasks using real-world datasets. Comparable performance to conventional optimisers has been obtained in both cases.

Keywords: Optimisation, Neural Networks, Deep learning, Truncated Newton, Gradient Flow, Hessian-free

1. Introduction

Deep neural networks (DNN) have wide applications in many research fields including autonomous driving [1], speech recognition [2, 3], computer vision [4], natural language processing [5], and bioinformatics [6]. The performance of deep neural networks depends highly on the training process, which has been the focus of many recent research works [7, 8]. The training of neural networks is essentially the optimisation of a complicated, non-convex loss function with respect to its parameters. Due to the large dimensionality of the input data, and the complicated functional forms of the DNNs, optimizations of DNNs to high precision and computational speed poses a serious challenge in modern applications. This is because there is increasing demands in both model dimensionality (number of input and output) and model complexity internal to the DNNs.

The optimisation methods of neural networks can be divided into two categories: stochastic methods and deterministic methods [9]. Stochastic methods have been heavily adopted in industrial applications due to their lower computational cost and easy implementation [10]. Research into stochastic methods is still ongoing, since their inception six decades ago [11–14]. From the most basic Stochastic Gradient Descent (SGD), a

*Corresponding author
Email address: sz299@cam.ac.uk (Sushen Zhang)

multitude of methods including SGD with momentum [15], and SGD with adaptive learning rates [16], have been developed. Stochastic optimisation methods are widely used in the hope of locating the global minimum by identifying the globally optimal tunings for its internal parameters. Since DNN models as objective functions are always non-convex in nature, finding the location of the global solution with increasing dimensionality in their parameter space is a combinatorially hard computational problem. The implementation of stochastic methods carry with it the advantage of simplicity. However, the disadvantage of stochastic methods is that its search direction often zig-zags and the minimum point reached is often not exact. Moreover, they offer no guarantee of global optimality, and cannot even determine if the point they converge to is a local minimum at the very least.

With recent developments in computational capability of the hardware, deterministic methods are rising in significance. The mainstream deterministic optimisation method used extensively in the training of DNNs is the L-BFGS method [17] and its variants [18]. They have been widely used in current research [19, 20]. The disadvantage is that in the optimisation of a DNN's objective function, these methods cannot guarantee global optimality due to its non-convexity [21]. However, they are able to guarantee a local minimum of the cost function of least squares fitting.

Stochastic methods have been widely researched in the past. Since [11] proved in theory the effectiveness of SGD, many variants of the stochastic methods have become the centre of attention. [22] justified the convergence of adaptive learning rate methods in a convex topology. Later on, [23] used mathematical proofs to demonstrate that one of the adaptive learning rate methods, AdaGrad-Norm, can converge to a stationary point in a non-convex topology. Variant optimisers belonging to the adaptive learning rate family have been extensively researched, including RMSprop [24], AdaDelta [25], Adam [26], AdaFTRL [27], SGD-BB [28], AdaBatch [29], SC-Adagrad [30], AMSGRAD [31], and Padam [32]. Apart from demonstrating the convergence of these stochastic methods, there is also research such as [33] that served to prove the global optimality of the converged point, but only in a theoretical context.

Research into deterministic models has been limited due to the associated computational cost in storing second-order information. Although [34] made use of the Hessian matrix of the loss function to understand the dynamics of neural network optimization, and deeply researched into the eigenvalues of the Hessian, the research results are not yet implemented in a real-world case study. However, this research accentuated the importance of understanding Hessian information to achieve optimality, and how the Hessian matrix determines the speed of convergence and the generalisation properties. The focus on the Hessian matrix has inspired research into second-order methods to be applied to the training of neural networks [34].

This paper proposes a new method adopting approximated second-order information following a quasi-Newton scheme to optimise DNNs. The method is derived based on a linearised version of the Gradient Flow method and makes use of finite differences to approximate values of a Hessian matrix. We test the effectiveness

of the optimiser on the MNIST and OILDROPLET dataset. The former adopts a deep convolutional neural network (CNN) and the latter adopts a conventional deep neural network (DNN). The architecture of the CNNs used are cutting-edge and the DNN is fine-tuned but the focus is on the optimiser performance compared conventional optimisers such as SGD, Adam and L-BFGS.

In this paper, Section 2 derives the proposed quasi-Newton method named HFGF. Section 3 provides a proof of convergence for the novel method. Section 4 evaluates and analyses the performance of the method on testing functions. Section 5 applies the method to real-life datasets of MNIST and OILDROPLET, adopting optimised DNN architecture in each case.

2. Algorithm Overview

We propose a method that adopts approximated second-order information to perform an optimisation task. To derive the optimisation method, we first write the general update rule as follows:

$$x_{k+1} = x_k + \Delta t \cdot \Delta x_k. \tag{1}$$

The gradient descent method uses the negative gradient vector as search direction:

$$x_{k+1} = x_k - \Delta t \cdot \nabla f(x_k). \tag{2}$$

The limit $\Delta t \to 0$ gives the smooth trajectory of the gradient flow method:

$$\frac{dx}{dt} = -\nabla f(x_k). \tag{3}$$

Linearizing the right-hand side, we obtain:

$$\frac{dx}{dt} \approx -[\nabla f(x_k) + \nabla^2 f(x_k) \cdot (x_{k+1} - x_k)]. \tag{4}$$

Rewriting the gradient vector and the Hessian matrix as g_k and H_k, and applying a linearly implicit Euler scheme gives:

$$x_{k+1} - x_k = -\Delta t \cdot [g_k + H_k \cdot (x_{k+1} - x_k)]. \tag{5}$$

The step length Δt_k is replaced with the step-size h_k and Equation 5 is rearranged:

$$\left[H_k + \frac{1}{h_k} \cdot I\right](x_{k+1} - x_k) = -g_k. \tag{6}$$

The search direction is equal to Δx_k which takes the form p_k, similar to Newton's equation. To give Brown and Bartholomew-Bigg's equation [35] and the iteration update for x_{k+1}:

$$\begin{aligned} \left[H_k + \frac{1}{h_k} \cdot I\right] p_k &= -g_k, \\ x_{k+1} &= x_k + p_k. \end{aligned} \tag{7}$$

This equation can be rewritten in the form of a linear equation, where the Hessian matrix and the product of step-size and identity matrix have been grouped together into a new matrix, Q_k:

$$\begin{aligned} Q_k &= H_k + \frac{1}{h_k} \cdot I, \\ Q_k \cdot p_k &= -g_k. \end{aligned} \tag{8}$$

To solve for this equation, we follow a schema that is similar to the truncated Newton method, where the outer loop solves for the optimisation problem and the inner loop solves for the search direction p_k. The inner loop adopts the Conjugate Gradient method with a solution for a linear system $Ax = b$ demonstrated as [36]:

$$\begin{aligned} \alpha_k &= \frac{r_k^T r_k}{d_k^T A d_k}, \\ x_{k+1} &= x_k + \alpha_k d_k, \\ r_{k+1} &= r_k + \alpha_k A d_k, \\ \beta_{k+1} &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}, \\ d_{k+1} &= -r_{k+1} + \beta_{k+1} d_k. \end{aligned} \tag{9}$$

Instead of solving for $Ax = b$, we iteratively solve for $Qp = -g$ using the Conjugate Gradient algorithm. The inner loop does not solve for the exact solution. Instead, the number of iterations is truncated, i.e. we stop after a finite number of iterations. This reduces the computational complexity of each iteration, generating an algorithm that converges faster to the optimum.

The combination of the truncated Newton and gradient flow methods creates a large-scale solution method for unconstrained optimisation problem with improved properties. These properties include better navigation of non-convex regions through the substitution of Newton's equation with the gradient flow equation. The manipulation of step-size (h_k) within the gradient flow method also supports a faster and more accurate convergence than Newton's method. This should result in better convergence qualities for the truncated Newton method when combined with gradient flow.

Another important improvement of the algorithm is that it is Hessian-free. The product of the Hessian matrix and the conjugate direction vector product is approximated using finite difference such that the algorithm only evaluates first-order derivatives. The scheme is outlined below in the vector finite difference equation (Equation 10), where ϵ is a user-defined small number to approximate an infinitesimal value.

$$Q_k d_i = \left[\frac{\nabla f(x_k + \epsilon d_i) - \nabla f(x_k)}{\epsilon} \right] + \frac{1}{h_k} I d_i. \tag{10}$$

The proposed HFGF algorithm is outlined in Algorithm 1. The value of h_0 is initialised to any value between 0 and 1, and p_0 is the zero vector commonly used in truncated Netwon methods. From the pseudocode, it is observable that the inner loop is effectively a CG iteration and the values of $Q_k p_k$ and $Q_k d_i$ are solved by finite differences.

Algorithm 1 : HFGF Algorithm

Initialize: x_0, $p_0 = \overrightarrow{0}$ and $h_0 = 10^{-3}$

while $\|g_k\| > tol$ **do**

$Q_k p_k = \frac{g(x_k+\epsilon p_k)-g(x_k)}{\epsilon} + \frac{1}{h_k} I \cdot p_k$

Initialize: $r_0 = Q_k p_k + g_k$, $d_0 = -r_0$ and $p_i = p_k$

for $i = 0, 1, 2, ..., i_{max}$ **do**

$Q_k d_i = \frac{g(x_k+\epsilon d_i)-g(x_k)}{\epsilon} + \frac{1}{h_k} I \cdot d_i$

$\alpha_i = \frac{r_i^T r_i}{d_i^T Q_k d_i}$

$p_{i+1} = p_i + \alpha_i d_i$

$r_{i+1} = r_i + \alpha_i d_i$

$\beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$

$d_{i+1} = -r_{i+1} + \beta_{i+1} d_i$

$x_{try} = x_k + p_{i+1}$

if $f(x_{try}) < f(x_k)$ **then**

if $f(x_{try}) < f(x_k) + \mu(g_k^T p_k)$ [Armijo First Order Conditions (of descent)] **then**

$x_{k+1} = x_{try}$ and $p_{k+1} = p_{i+1}$

$h_{k+1} = 2h_k$

break [Minor iteration]

else if $i = i_{max}$ **then**

$x_{k+1} = x_{try}$ and $p_{k+1} = p_{i+1}$

$h_{k+1} = \frac{1}{2} h_k$

break [Minor iteration]

end if

else if $i = i_{max}$ **then**

$h_{k+1} = \frac{1}{2} h_k$ and x_{try} not accepted

break [Minor iteration]

end if

end for

end while

3. Convergence Analysis

Training ANNs and DNNs can be viewed as the equivalent to minimizing a large-scale optimization problem of the form:

$$\min f(x), \tag{11}$$

where $x \in R^n$ is a real valued n-dimensional vector of system variables that are to be optimized to minimize the scalar function $f(x) : R^n \to R$. We will adopt the inexact Newton Method to derive the proof of convergence.

In this paper, we assume that f has an optimal value $f(x^*)$ at x^*. We will use the following assumption about the objective function for the rest of this article.

Assumption 1. *Assume that f is L-smooth, that is, f is differentiable and the gradient is L-Lipschitz continuous, i.e., $\forall x, y \in R^n$, $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$.*

A study focused on variational partial differential equations by Botsaris [37] led to the development of gradient flow (GF) methods. The solution of an unconstrained optimization problem shown in Equation 11 is calculated by solving this coupled set of ordinary differential equations [37, 35, 38]:

$$\frac{dx(t)}{dt} = -\nabla f(x). \tag{12}$$

It is worth considering the linearized form of Equation 12, namely:

$$\frac{dx(t)}{dt} \approx -\nabla f(x_k) - \nabla^2 f(x_k)[x(t) - x_k]. \tag{13}$$

Rewriting the gradient vector and the Hessian matrix as g_k and H_k, respectively, and applying a linearly implicit Euler scheme gives the following:

$$x_{k+1} - x_k = -\Delta t_k[\nabla f(x_k) - \nabla^2 f(x_k)(x_{k+1} - x_k)]. \tag{14}$$

The step length Δt_k is replaced with the step-size h_k and Equation 14 is rearranged:

$$\left[\nabla^2 f(x_k) + \frac{1}{h_k} I\right](x_{k+1} - x_k) = -\nabla f(x_k). \tag{15}$$

The search direction is equal to Δt_k which takes the form p_k, similar to Newton's equation, to give Brown and Bartholomew-Bigg's equation [19] and the iteration update for x_{k+1}:

$$\begin{gathered}\left[\nabla^2 f(x_k) + \frac{1}{h_k} I\right] p_k = -\nabla f(x_k), \\ x_{k+1} = x_k + p_k.\end{gathered} \tag{16}$$

Theorem 1. *Consider HFGF algorithm (equation [10]). Under the convex assumption (Assumption 1), when $0 < h_k \leq \frac{1}{L}$, we have*

$$\|x_k - x^*\|^2 \leq \sigma^k \|x_0 - x^*\|^2, \tag{17}$$

where σ is a constant satisfy that $1 - \frac{1}{L} \leq \sigma < 1$. Furthermore,

$$\|f(x(t)) - f(x^*)\| \leq \frac{L}{2}\sigma^k \|x_0 - x^*\|^2,$$

holds.

Proof. Using a finite difference scheme to approximate Hessian vector-product, there exists $\epsilon > 0$, such that:

$$H_k p_k = \frac{\nabla f(x_k + \epsilon p_k) - \nabla f(x_k)}{\epsilon} + \frac{1}{h_k} I p_k. \tag{18}$$

Then we get the iterate, as following:

$$\begin{aligned} -\nabla f(x_k) &= \frac{\nabla f(x_k + \epsilon p_k) - \nabla f(x_k)}{\epsilon} + \frac{1}{h_k} I p_k, \\ x_{k+1} &= x_k + p_k. \end{aligned} \tag{19}$$

Let $\tilde{x}_{k+1} := x_k + \epsilon p_k$ and $p_k = x_{k+1} - x_k$, then rewrite Equation 19 as:

$$\begin{aligned} x_{k+1} &= x_k - h_k \nabla f(x_k) - \frac{h_k}{\epsilon}(\nabla f(\tilde{x}_{k+1}) - \nabla f(x_k)), \\ \tilde{x}_{k+1} &= x_k + \epsilon(x_{k+1} - x_k). \end{aligned} \tag{20}$$

By substituting the second equation in Equation 20 into the first equation, the following implicit iterative form can be obtained:

$$x_{k+1} = x_k - h_k \nabla f(x_k) - \frac{h_k}{\epsilon}(\nabla f(x_k + \epsilon(x_{k+1} - x_k)) - \nabla f(x_k)). \tag{21}$$

According to the hypothesis function f second order differentiable, and Hessian matrix $\nabla^2 f$ is positive definite. The first-order Taylor expansion of $\nabla f(x_k + \epsilon(x_{k+1} - x_k))$ in the above equation at x_k can be obtained as follows:

$$x_{k+1} = x_k - h_k \nabla f(x_k) - \frac{h_k}{\epsilon}\Big(\nabla f(x_k) + \nabla^2 f(z_k) \cdot \epsilon(x_{k+1} - x_k) - \nabla f(x_k)\Big), \tag{22}$$

where $z_k \in (x_k, x_k + \epsilon(x_{k+1} - x_k))$, i.e., there exists some $\delta \in (0, 1)$ such that $z_k = x_k + \epsilon \cdot \delta(x_{k+1} - x_k)$. Rewrite Equation 22 as follows:

$$x_{k+1} - h_k \nabla^2 f(z_k)(x_{k+1} - x_k) = x_k - h_k \nabla f(x_k), \tag{23}$$

then:

$$\left\| x_{k+1} - x^* - h_k \nabla^2 f(z_k)(x_{k+1} - x_k) \right\|^2 = \left\| x_k - x^* - h_k \nabla f(x_k) \right\|^2. \tag{24}$$

For the left side of Equation 24:

$$\begin{aligned} & \left\| x_{k+1} - x^* - h_k \nabla^2 f(z_k)(x_{k+1} - x_k) \right\|^2 \\ = \; & \|x_{k+1} - x^*\|^2 + h_k^2 \left\| \nabla^2 f(z_k) \right\|^2 \|x_{k+1} - x_k\|^2 - 2h_k \langle x_{k+1} - x^*, \nabla^2 f(z_k)(x_{k+1} - x_k) \rangle \\ \geq \; & \|x_{k+1} - x^*\|^2 + h_k^2 \left\| \nabla^2 f(z_k) \right\|^2 \|x_{k+1} - x_k\|^2 - 2h_k \|x_{k+1} - x^*\| \cdot \left\| \nabla^2 f(z_k) \right\| \cdot \|x_{k+1} - x_k\|. \end{aligned} \tag{25}$$

Assume that $h_k \leq \dfrac{2}{\|\nabla^2 f(z_k)\|} \dfrac{\|x_{k+1} - x^*\|}{\|x_{k+1} - x_k\|}$, then:

$$\left\| x_{k+1} - x^* - h_k \nabla^2 f(z_k)(x_{k+1} - x_k) \right\|^2 \geq \|x_{k+1} - x^*\|^2. \tag{26}$$

For the right side of Equation 24:

$$\begin{aligned} & \left\|x_k - x^* - h_k \nabla f(x_k)\right\|^2 \\ = \; & \|x_k - x^*\|^2 + h_k^2 \left\|\nabla f(x_k)\right\|^2 - 2h_k \langle x_k - x^*, \nabla f(x_k)\rangle \\ \leq \; & \|x_k - x^*\|^2 - h_k\left(\frac{2}{L} - h_k\right)\left\|\nabla f(x_k)\right\|^2, \end{aligned} \tag{27}$$

where we have used the following inequality:

$$\frac{1}{L}\|\nabla f(x) - \nabla f(y)\|^2 \leq \langle x - y, \nabla f(x) - \nabla f(y)\rangle,$$

for all $x, y \in R^n$, and $\nabla f(x^*) = 0$.

According to Equation 24, the following inequality can be obtained from Equation 26 and Equation 27:

$$\begin{aligned} \|x_{k+1} - x^*\|^2 \; & \leq \; \|x_k - x^*\|^2 - h_k\left(\frac{2}{L} - h_k\right)\left\|\nabla f(x_k)\right\|^2 \\ & \leq \; \|x_k - x^*\|^2 - h_k\left(\frac{2}{L} - h_k\right) L\|x_k - x^*\|^2 \\ & \leq \; (1 - h_k(2 - h_k L))\|x_k - x^*\|^2. \end{aligned} \tag{28}$$

Thus when $0 \leq h_k \leq \dfrac{1}{L}$, there exist some $1 - \frac{1}{L} \leq \sigma < 1$, such that

$$\|x_k - x^*\|^2 \leq \sigma\|x_k - x^*\|^2 \leq \sigma^k\|x_0 - x^*\|^2.$$

Furthermore, we arrive at the simplest analysis: $f(x_k) - f(x^*) \leq \frac{L}{2}\|x_k - x^*\|^2 \leq \frac{L}{2}\sigma^k\|x_0 - x^*\|^2$. □

4. Analysis of the Algorithm

This section undertakes the analysis of the cost and performance of the HFGF algorithm. To test the performance, we introduce several testing functions that is highly complex and non-convex in nature. These functions are introduced to simulate the highly complicated shape of the objective function of neural networks. The results are generated using a 2.3 GHz Intel Core i5 processor with a memory of 8 GB 2133 MHz LPDDR3.

4.1. Performance on Testing Functions

The standard testing functions adopted are shown in Table 1. The functions range from the simplest case of a two-dimensional unimodal convex function to the more complex case of a multidimensional, multimodal function. The latter is the focus since we expect the objective function of a neural network to be multimodal and multidimensional.

Table 1: Standard testing function types and details.

Test Functions	Types of Function	Types of Mimima	Reference
1. Rosenbrock, 2. Chung Reynolds, 3. De Jong	Unimodal, convex, multidimensional	Global	Jamil & Yang [39], Molga & Smutnicki [40]
4. Rastrigin, 5. Ackley	Multimodal, multidimensional	Many local & Global	Jamil & Yang [39], Molga & Smutnicki [40]
6. Booth	Unimodal, convex, two-dimensional	Global	Jamil & Yang [39]
7. Dropwave, 8. Shubert	Multimodal, two-dimensional	Many local & Global	Molga & Smutnicki [40]

4.1.1. Two-Dimensional Test Functions

The test on two-dimensional functions serve as a testament to the convergence behaviour of the optimisers. By applying the optimiser on two-dimensional functions we observe that the proposed HFGF method outperforms other traditional optimisers including Truncated Newton, Newton Conjugate Gradient, and L-BFGS. The results are summarised in Table 2. These results are generated from 10 independent runs in each equation.

The two-dimensional test functions have been used to test the basic convergence ability of the proposed method. The CPU time required for each method was negligible for all 2-D test cases. The Dropwave and Shubert functions have many local minima which require the optimisers to navigate through a highly non-convex region to find the global minima. At the global minima, the Dropwave and Shubert functions have values of $f(x^*) = -1$ and $f(x^*) = 0$ respectively. From the Table 2, it is clear that the proposed method performed the best as it was the only method that converged to values closer to the global minima with a high success rate.

4.1.2. Multidimensional Test Functions

We perform analysis on the multidimensional testing functions, both unimodal and multimodal. High dimensions of 100,000 and 1,000,000 are selected to simulate the high dimension of data we input to a deep neural network. The results are displayed in Table 3-4. At this scale, the other optimisers calculated using SciPy were not able to converge in a short time frame, except where initialisation of x values were very close to the global optimum. The CPU times extended to hours in some cases. Details of a comparison between the HFGF and the other quasi-Newton optimisers defined in SciPy are tabulated in the Supplementary Material. The HFGF method was able to handle problems ranging from 50,000 to 200,000 inputs with less than 5 minutes

Table 2: Results of performing different classic second-order optimisation algorithms on 2-D Test Functions. The algorithms include Truncated Newton method (TN), Nonlinear Conjugate Gradient (NCG), L-BFGS method and the proposed HFGF method. $f(x)_{final}$ is the final function value obtained after optimisation. n_{iter} is the total number of iterations in the major loop. n_{fev} is the number of function evaluations. n_gev is the number of gradient evaluations. $Success[\%]$ is the rate of success in optimising each function.

Function	Optimisation Method	$f(x)_{final}$	n_{iter}	n_{fev}	n_{gev}	$Convergence[\%]$
Dropwave	TN	-2.92×10^{-1}	4	17	8	100
	NCG	-2.08×10^{-1}	3	41	9	67
	L-BFGS	-3.23×10^{-1}	3	28	6	67
	HFGF	-1.00×10^{0}	2	32	34	100
Shubert	TN	5.80×10^{-17}	4	13	7	100
	NCG	2.13×10^{-2}	3	39	8	50
	L-BFGS	3.33×10^{-15}	5	25	9	50
	HFGF	2.20×10^{-11}	8	24	32	100
Booth	TN	2.34×10^{-1}	7	21	15	100
	NCG	1.04×10^{-16}	2	20	5	100
	L-BFGS	1.23×10^{-12}	5	20	10	100
	HFGF	1.18×10^{-11}	15	45	60	100

of CPU times in most cases. This clearly illustrates the superiority of the proposed HFGF algorithm in very large-scale applications compared to NCG, L-BFGS, and TN methods.

Table 3: HFGF Results for 100,000-Dimension Test Functions.

$Function$	$f(x)_{final}$	n_{iter}	n_{fev}	n_{gev}	$CPUtime(s)$	$Convergence[\%]$
Rosenbrock	1.40×10^{-9}	409	105	514	743.47	100
Dejong	6.03×10^{-6}	75	25	100	38.16	100
Chung Reynolds	2.49×10^{-2}	58	19	77	50.77	100
Rastrigin	1.16×10^{6}	22	7	29	25.84	100
Ackley	2.272×10^{1}	22	72	80	72.07	100

4.2. Time Analysis

The HFGF method involves both function evaluations and gradient evaluations. We perform CPU time analysis to identify the most time-consuming steps in the optimizer. We calculate the CPU time spent on function evaluations, gradient evaluations and other steps respectively based on 10 different independent runs. The optimizer is run to solve for the optima of a 100-dimensional Rosenbrock function.

Table 4: HFGF Results for 1,000,000-Dimension Test Functions.

Function	$f(x)_{final}$	n_{iter}	n_{fev}	n_{gev}	$CPUtime(s)$	$Convergence[\%]$
Rosenbrock	2.86×10^{-9}	399	110	509	6112.24	100
Dejong	6.03×10^{-5}	75	25	100	418.40	100
Chung Reynolds	2.98×10^{-1}	69	23	92	543.41	100
Rastrigin	1.16×10^{7}	32	11	43	368.37	100
Ackley	6.560×10^{0}	27	74	81	702.19	100

Based on 10 experiments, the average number of function evaluations in one run is 60, costing a total CPU time of 0.00237 seconds (7.77% of total time). The average number of gradient evaluations in one run is 45, costing a total CPU time of 0.0186 seconds (60.99% of total time). The other steps cost a total CPU time of 0.0095 seconds (31.25% of total time). The average CPU time for one step of function evaluation is 3.97×10^{-5} seconds and the average CPU time for one step of gradient evaluation is 4.16×10^{-4} seconds.

From the results of this experiment, it is evident that the most of the CPU time is spent on gradient evaluations. This has demonstrated the importance of selecting an automatic differentiation package that has a shorter time-frame to evaluate derivatives. Moreover, the optimization of the algorithm can center on the reduction of the number of gradient evaluations.

4.3. Memory Storage

The proposed method requires the storage of 16 vectors of length n (the vectors x_k, $g(x_k)$, and 14 working vectors). This is similar to the requirement of L-BFGS method, a popular quasi-Newton method, which has a memory storage of $14n$, based on the scheme that has 7 vectors of past information storage. In either case, we do not need to store second-order derivatives information. There is also no storage of matrices; only storage of vectors suffices. It should be noted that although our proposed method and L-BFGS use different principles to compute search direction, the routines used for the linesearch are similar: both use cubic interpolation to obtain the strong Wolfe conditions [41].

5. Applications

The analysis above provides a theoretical understanding of the algorithmic performance. In this section, the optimiser is applied to several test cases to determine its real-world performance. The derived algorithm is applied to large-scale optimisation of DNNs to test its speed, robustness and accuracy. In particular, DNNs are optimised to confirm the efficacy of the optimiser to large-scale problems.

5.1. Datasets

In order to test the performance of our proposed optimiser, we perform one classification test and one regression task on the proposed optimiser adopting real-life dataset. First, we perform classification on the MNIST dataset. The MNIST dataset has a training set of 60,000 examples and a test set of 10,000 examples, consisting of hand-written digits representing the numbers 0-9. It is a commonly used database in the machine learning community to test pattern recognition capability of a designed network with minimal efforts on pre-processing and formatting the images. We adopt LeNet-5, consisting of Convolutional Neural Network (CNN) architecture, to analyse MNIST data base. The network consists of two convolutional layers separated by max-pooling layers, followed by three fully-connected layers. Here we adopt mini-batches to perform optimization. The activation function used is ReLU.

Second, we perform a regression task on the OILDROPLET dataset. The dataset contains observations generated from dropping an oil droplet to a surface of water. The oil is composed of several organic chemicals including diethyl phthalate, octanol, octanoic acid and pentanol. The input features of the dataset include composition of the droplet, environment temperature, oil viscosity, oil surface tension and oil density. The output features are generated by observing the movement and merging of the oil droplet on the water surface, including average movement speed, maximum speed of a single droplet, average number of droplets in the last second, average number of droplets throughout the experiment. We construct an optimised neural network consisting of three hidden layers with 50, 20 and 20 neurons respectively. Between each layer we apply the ReLU activation function. Since the key is to compare performance of the optimiser, the architectural parameters are held constant.

We have performed data pre-processing on both the MNIST dataset and the OILDROPLET dataset. In the MNIST dataset, the black-and-white images are converted to a series of gray-scale values for each pixel. In the OILDROPLET dataset, we perform normalisation of the x values as a pre-processing step. In either case, the pre-processing of data remains the same for the application of different optimisation methods.

5.2. CNN on MNIST Data

The network is run on several optimizers and the comparison of the performance is shown in Table 5. We have selected SGD, Adam and L-BFGS as the target of comparison. SGD and Adam are selected because these are the first-order stochastic methods. SGD is the most basic form and Adam is the commercially-available best-performing counterpart. L-BFGS is adopted because it is a quasi-Newton method. It acts as a benchmark to the current quasi-second order methods and is known to be very fast. Before comparison, we optimise the performance of the optimisers through hyperparameter tuning. The learning rate set for SGD is set at 0.01, the optimal hyperparameter obtained after tuning. The learning rate for Adam is 0.001. The hyper-parameter values for HFGF are optimised using a grid search. We assume the results from a grid search is relatively

optimal in this analysis. The performance of the three optimisers are compared based on accuracy of prediction on the test set.

Table 5: Comparison between the performance of different optimizers on the classification task of MNIST data

Optimizer	Accuracy	Final training loss	Time of convergence (s)
SGD	92.5%	0.0801	55.98
Adam	91.6%	0.2448	65.13
L-BFGS	90.9%	0.3565	2225.21
HFGF	96.4%	0.0643	142.11

From Table 5, it can be seen that our proposed method obtained comparable result in terms of accuracy of classification. While the conventional quasi-Newton method of L-BFGS sometimes breaks down with a high time of convergence, our proposed quasi-Netwon method is robust to the problem. Although the final loss achieved is higher than the first-order methods, the classification accuracy is comparable to the other two optimisers. Moreover, although the time of convergence is longer than stochastic first-order methods, it is lower than the benchmark of L-BFGS. For problems that rely on speed of computation, an improvement from 92.5% to 96.4% in accuracy at the expense of computation time is not desired. This can be viewed as a limitation of the algorithm. In cases where accuracy is the key, the results show promises in improving performance. Moreover, compared to other quasi-Newton methods such as L-BFGS, the speed becomes an edge as there is significant improvement in terms of computation time.

5.3. Traditional DNN on OILDROPLET

We apply the same set of optimisers, SGD, Adam, L-BFGS and the proposed optimisation algorithm, on the OILDROPLET dataset. The result of application is shown in Table 6. The learning rate used in SGD is 0.01, and in Adam is 0.05. In HFGF method, we have mostly used the default settings or settings of a similar scale. The settings used include the following: the increase in the value of h after each step is 8×, the decrease is 0.125×. The hyperparameters defined to have infinitesimal values are all close to zero. The value of ϵ is 10^{-7} and of tolerance is 10^{-5}. The value of μ is 10^{-4} and of initial h is 10^{-3}. The maximum number of inner iterations is originally 15 but is now tuned to 2. This tuning has greatly increased the processing speed of the algorithm. To evaluate the effectiveness of optimisation, we use the mean squared error (MSE) loss on the test dataset as the metric for performance. Since there are four output feature, we tabulate an MSE for each dataset.

On the OILDROPLET dataset, the HFGF method achieved better result than SGD in terms of final loss and regression error. It is under-performing compared to Adam in terms of final loss, test error and speed. However, it is faster than the quasi-Newton method of L-BFGS. Considering that L-BFGS sometimes fails for

Table 6: Comparison between the performance of different optimisers on the classification task of OILDROPLET data. S_{Ave} represents the average speed of droplets. $D_{FinalAve}$ represents the average number of droplets in the last second. S_{Max} represents the maximum average single droplet speed. D_{Ave} represents the average number of droplets.

Optimizer	Training loss	Test error				Time of convergence (s)
		S_{Ave}	$D_{FinalAve}$	S_{Max}	D_{Ave}	
SGD	0.389	0.3782	0.3785	0.4616	0.3365	162.89
Adam	0.224	0.1960	0.2297	0.2589	0.2104	160.76
L-BFGS	0.188	0.1583	0.1938	0.2223	0.1789	4249.79
HFGF	0.290	0.2800	0.2707	0.3478	0.2577	535.89

complicated problems, our optimiser is able to perform more complicated optimisation tasks and is more robust to datasets of high dimensionality.

6. Conclusion and Future Work

We have demonstrated the derivation of a novel quasi-Newton optimisation method with a proof of convergence. The method is named Hessian-free Gradient Flow (HFGF) and has been designed for the optimization of DNNs. We first tested the HFGF method on standard testing functions and was then compared with the other common optimization algorithms to test for its convergence. Then we performed time analysis to identify the most time-consuming steps in the proposed algorithm. We also briefly discussed the storage requirement. The HFGF method was then applied to case studies where open-source databases and real-world industrial databases are used to test the effectiveness of the optimization algorithm.

The aim of the algorithm is not to beat cutting-edge first stochastic algorithms as the second-order method itself carries with it a computational cost. However, our algorithm has demonstrated comparable performance in tasks of classification and regression with regard to popular first-order algorithms. It is also more robust than L-BFGS algorithm if we set L-BFGS as a benchmark for quasi-Newton algorithms.

In summary, although the method was not the fastest optimization algorithm compared to L-BFGS algorithms in simple cases, and not the simplest compared to SGD or Adam, it introduces a trade-off between speed and robustness. Moreover, the method has demonstrated advantages in classification accuracy in large datasets compared to most first-order algorithms.

Currently ongoing future research is focused on improving the speed of the new method, particularly by reducing the number of gradient evaluations and a more optimised algorithmic implementation architecture. To prevent convergence to local minima, we can also introduce stochastic components to the optimiser. According to the principle of “no free lunch”, each optimization method has its own advantage in certain contexts. We believe our new algorithm may be more suited for the fitting of DNNs, compared to other standard optimization

Table 7: HFGF Results for 10,000-Dimension Test Functions.

Function	Optimisation Method	$f(x)_{final}$	n_{iter}	n_{fev}	n_{gev}	CPU Time	Success
Rosenbrock	TN	7.54×10^{3}	433	9219	-	59932.24	No
	NCG	3.99×10^{0}	183	2.83×10^{6}	283	2215.28	Yes
	L-BFGS	6.49×10^{4}	1	2.00×10^{4}	-	13.122	No
	HFGF	2.31×10^{-8}	62	308	278	17.811	Yes
Dejong	TN	1.01×10^{-7}	4	18	-	118.22	Yes
	NCG	2.90×10^{-6}	1	5.00×10^{4}	5	30.64	Yes
	L-BFGS	8.14×10^{2}	1	4.00×10^{4}	-	25.24	No
	HFGF	1.69×10^{-5}	14	28	42	0.84	Yes
Chung Reynolds	TN	2.02×10^{-6}	3	15	-	101.49	Yes
	NCG	6.87×10^{5}	1	1.60×10^{5}	16	101.49	No
	L-BFGS	4.10×10^{6}	1	3.00×10^{4}	-	18.62	No
	HFGF	3.62×10^{-3}	14	28	42	1.13	Yes
Rastrigin	TN	5.97×10^{-6}	4	17	-	121.04	Yes
	NCG	6.42×10^{-4}	1	8.00×10^{4}	8	53.79	Yes
	L-BFGS	2.29×10^{-4}	1	8.00×10^{4}	-	55.22	Yes
	HFGF	4.15×10^{-6}	16	66	43	2.29	Yes
Ackley	TN	6.02×10^{-9}	4	53	-	7387.56	Yes
	NCG	4.25×10^{0}	0	1.00×10^{4}	1	128.38	No
	L-BFGS	4.25×10^{0}	0	1.000×10^{4}	-	127.40	No
	HFGF	2.27×10^{-2}	22	72	80	3.98	Yes

methods.

7. Supplementary Material

We have performed a comparison of the CPU time between different quasi-Newton algorithms. The results are tabulated in Table 7 for the dimension of 10,000 and in Table 8 for the dimension of 100,000. The optimisers of Truncated Newton, Newton Conjugate Gradient and L-BFGS are coded in SciPy [42] and HFGF is coded using the package Casadi [43]. We have only included comparisons of CPU time for Dejong, Chung Reynolds and Rastrigin functions for the dimensionality of 100,000 because the optimisers other than HFGF fail to converge within a time frame of 100,000 seconds. The operating system used to run the code is Ubuntu 18.04.1 LTS and the CPU used is Intel Core i7-8700K with a memory of 3.70GHz. A few points we have noted from the data:

Table 8: HFGF Results for 100,000-Dimension Test Functions. Only HFGF results are included for Rosenbrock and Ackley functions because the other optimisers fail to converge within 100,000 seconds of CPU time.

Function	Optimisation Method	$f(x)_{final}$	n_{iter}	n_{fev}	n_{gev}	CPU Time	Success
Rosenbrock	HFGF	1.40×10^{-9}	409	105	514	743.47	Yes
Dejong	TN	3.35×10^{-12}	13	83	-	53195.35	Yes
	NCG	9.33×10^{-3}	1	6.00×10^{5}	6	9379.80	Yes
	L-BFGS	1.88×10^{4}	1	4.00×10^{5}	-	2678.09	No
	HFGF	6.03×10^{-4}	75	25	100	38.16	Yes
Chung Reynolds	TN	1.95×10^{-4}	12	101	-	66195.31	Yes
	NCG	2.72×10^{7}	1	1.70×10^{6}	17	11539.14	No
	L-BFGS	3.53×10^{8}	1	4.00×10^{5}	-	2688.95	No
	HFGF	2.49×10^{-2}	58	19	77	50.77	Yes
Rastrigin	TN	1.90×10^{-3}	5	28	-	24269.53	Yes
	NCG	1.98×10^{-4}	1	1.10×10^{6}	11	9795.51	Yes
	L-BFGS	3.96×10^{1}	1	1.00×10^{6}	-	8185.25	No
	HFGF	1.16×10^{6}	22	7	29	25.84	Yes
Ackley	HFGF	2.27×10^{1}	22	72	80	72.07	Yes

- The time of convergence for HFGF is much faster compared to other quasi-Newton method.
- HFGF converges successfully in all cases.
- Optimisers other than HFGF succeed and fail in different cases, well demonstrating the "No Free Lunch" Theorem.
- The optimiser L-BFGS fails in most cases, indicating its lack of robustness.

8. Acknowledgement

The first author would like to thank the Cambridge Overseas Trust and the China Scholarship Council for their funding of this research.

References

[1] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[2] A. Agrawal, A. Jain, and B. S. Kumar, "Deep learning based classification for assessment of emotion recognition in speech," *Available at SSRN 3356238*, 2019.

[3] G. Krishna, C. Tran, J. Yu, and A. H. Tewfik, "Speech recognition with no speech or with noisy speech," *arXiv preprint arXiv:1903.00739*, 2019.

[4] Y. Bao, Z. Tang, H. Li, and Y. Zhang, "Computer vision and deep learning–based data anomaly detection method for structural health monitoring," *Structural Health Monitoring*, vol. 18, no. 2, pp. 401–421, 2019.

[5] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational intelligence magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[6] S. Min, B. Lee, and S. Yoon, "Deep learning in bioinformatics," *Briefings in bioinformatics*, vol. 18, no. 5, pp. 851–869, 2017.

[7] H. Yan, P. Yu, and D. Long, "Study on deep unsupervised learning optimization algorithm based on cloud computing," in *2019 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, pp. 679–681, IEEE, 2019.

[8] D. Xu, T. Xiong, D. Liu, S. K. Zhou, M. Chen, and D. Comaniciu, "Multiple landmark detection in medical images based on hierarchical feature learning and end-to-end training," Feb. 19 2019. US Patent App. 10/210,613.

[9] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pp. 265–272, Omnipress, 2011.

[10] D. Fouskakis and D. Draper, "Stochastic optimization: a review," *International Statistical Review*, vol. 70, no. 3, pp. 315–349, 2002.

[11] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[12] C. Jin, P. Netrapalli, R. Ge, S. M. Kakade, and M. I. Jordan, "Stochastic gradient descent escapes saddle points efficiently," *arXiv preprint arXiv:1902.04811*, 2019.

[13] Y. Wen, K. Luk, M. Gazeau, G. Zhang, H. Chan, and J. Ba, "Interplay between optimization and generalization of stochastic gradient descent with covariance noise," *arXiv preprint arXiv:1902.08234*, 2019.

[14] G. Denevi, C. Ciliberto, R. Grazzi, and M. Pontil, "Learning-to-learn stochastic gradient descent with biased regularization," *arXiv preprint arXiv:1903.10399*, 2019.

[15] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[16] S. Klein, J. P. Pluim, M. Staring, and M. A. Viergever, "Adaptive stochastic gradient descent optimisation for image registration," *International journal of computer vision*, vol. 81, no. 3, p. 227, 2009.

[17] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[18] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.

[19] S. Yatawatta, L. De Clercq, H. Spreeuw, and F. Diblen, "A stochastic lbfgs algorithm for radio interferometric calibration," *arXiv preprint arXiv:1904.05619*, 2019.

[20] F. Carrara, F. Falchi, R. Caldelli, G. Amato, and R. Becarelli, "Adversarial image detection in deep neural networks," *Multimedia Tools and Applications*, vol. 78, no. 3, pp. 2815–2835, 2019.

[21] C. A. Floudas and P. M. Pardalos, *State of the art in global optimization: computational methods and applications*, vol. 7. Springer Science & Business Media, 2013.

[22] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[23] R. Ward, X. Wu, and L. Bottou, "Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization," *arXiv preprint arXiv:1806.01811*, 2018.

[24] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, 2012.

[25] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

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

[27] F. Orabona and D. Pál, "Scale-free algorithms for online linear optimization," in *International Conference on Algorithmic Learning Theory*, pp. 287–301, Springer, 2015.

[28] C. Tan, S. Ma, Y.-H. Dai, and Y. Qian, "Barzilai-borwein step size for stochastic gradient descent," in *Advances in Neural Information Processing Systems*, pp. 685–693, 2016.

[29] A. Défossez and F. Bach, "Adabatch: Efficient gradient aggregation rules for sequential and parallel stochastic gradient methods," *arXiv preprint arXiv:1711.01761*, 2017.

[30] M. C. Mukkamala and M. Hein, "Variants of rmsprop and adagrad with logarithmic regret bounds," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2545–2553, JMLR. org, 2017.

[31] S. J. Reddi, S. Sra, B. Póczos, and A. Smola, "Fast incremental method for smooth nonconvex optimization," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 1971–1977, IEEE, 2016.

[32] J. Chen and Q. Gu, "Padam: Closing the generalization gap of adaptive gradient methods in training deep neural networks," 2018.

[33] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," *arXiv preprint arXiv:1811.03804*, 2018.

[34] B. Ghorbani, S. Krishnan, and Y. Xiao, "An investigation into neural net optimization via hessian eigenvalue density," *arXiv preprint arXiv:1901.10159*, 2019.

[35] A. Brown and M. C. Bartholomew-Biggs, "Some effective methods for unconstrained optimization based on the solution of systems of ordinary differential equations," *Journal of Optimization Theory and Applications*, vol. 62, no. 2, pp. 211–224, 1989.

[36] D. M. Young and K. C. Jea, "Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods," *Linear Algebra and its applications*, vol. 34, pp. 159–194, 1980.

[37] C. A. Botsaris, "Differential gradient methods," *Journal of Mathematical Analysis and Applications*, vol. 63, no. 1, pp. 177–198, 1978.

[38] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact newton methods," *SIAM Journal on Numerical analysis*, vol. 19, no. 2, pp. 400–408, 1982.

[39] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimization problems," *arXiv preprint arXiv:1308.4008*, 2013.

[40] M. Molga and C. Smutnicki, "Test functions for optimization needs," *Test functions for optimization needs*, vol. 101, 2005.

[41] A. Skajaa, "Limited memory bfgs for nonsmooth optimization," *Master's thesis*, 2010.

[42] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[43] J. Andersson, J. Åkesson, and M. Diehl, "Casadi: A symbolic package for automatic differentiation and optimal control," in *Recent advances in algorithmic differentiation*, pp. 297–307, Springer, 2012.

CrediT author statement

Sushen Zhang: Methodology, Software, Validation, Formal analysis, Data Curation, Writing - Original Draft, Writing- Review & Editing

Ruijuan Chen: Conceptualisation, Methodology, Validation, Writing - Original Draft, Writing - Review & Editing

Wenyu Du: Software, Resources

Ye Yuan: Conceptualisation, Methodology, Resources, Valiation, Supervision

Vassilious S. Vassiliadis: Conceptualisation, Methodology, Validation, Supervision, Project administration

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: