Application of Lie Group-based Neural Network Method to Nonlinear Dynamical Systems

Ying Wen1,2, Temuer Chaolu∗1,2,

1 College of Information Engineering, Shanghai Maritime University, 201306, Shanghai, China
2 College of Arts and Sciences, Shanghai Maritime University, 201306, Shanghai, China

∗These authors contributed equally to this work.

Abstract

In this paper, a Lie group-based neural network method is proposed for solving initial value problems of nonlinear dynamics. Due to its single-layer structure (MLP), the approach is substantially cheaper than the multilayer perceptron method used in literature. The higher performance ability of the method is demonstrated by several examples.

1 Introduction

In recent years, a number of researchers have been devoted to the study of neural network (NN) approaches for solving differential equation (DE) problems. The study has become a new developing trend of solving initial or boundary value (IVP or BVP) problem of an ordinary and partial differential equations (ODEs and PDEs). The main encourage of this study not only from the fact that NNs have general approximation ability [1] but also due to their many advantages to study DEs, such as differ from many numerical methods to solve DEs, the methods do not rely on numeric difference scheme and can provide closed differentiable solutions to the problems of the DEs, etc.

Lee et al in [2] solved the first order ODEs through using Hopfiled NN minimization methodology and compared the results to the traditional algorithm. Lagaris et al in [3] proposed a NN trial solution method for the IVP of an ODE and PDEs; subsequently, irregular boundary problems were investigated by the method in [4]; Maria et al in [5] gave an extension on this method by training the network output directly as the equations solution, rather than relying on the trial solution, and successfully applied it to the calculation of the cosmological phase transition tunneling profile. Baymani et al in [6] obtained the solutions of the Navier-Stokes equations in the form of analytical functions based on a NN method. Recently, researchers are moving further away from shallow NNs and toward deep learning. Raissi et al in [7] involves the development of data-driven solutions to PDEs and applies to quantum mechanics, reaction-diffusion systems, and so on. It is worthy to note that the network structure evolved as the research progressed, and new mathematical principles were incorporated. Chaouki et al in [8] studied recurrent NNs with mixed time lag and time varying coefficients. Mall
et al in [9] employed Chebyshev networks to solve the singular IVP of Lane-Emden type equations, in which the computational difficulty is reduced through using Chebyshev polynomials with eliminate hidden layers in feedforward NNs. Marieme et al in [10] proposed a Fourier NN with the period function as the activation function, which simulated Fourier decomposition and was effectively applied to the Poisson equation and Heat equation. In a continuous-depth Bayesian NN family, Xu et al in [11] conduct scalable approximate inference. In there, the hidden units follow a stochastic DE due to uncertainty about independent weights in each layer. To acquire continuous-time PDEs from sparse data, Iakovlev et al. in [12] used graphical NNs.

In most recent, as the extension and deepening of NN methods, people put forward various machine learning methods. However, the basis of these methods is still the forward NN algorithm (as the basic structural unit of these methods), which determines the ultimate efficiency of these algorithms. At the same time, capturing the mathematical essence of the equation solution is the key to improve the efficiency of the algorithm. In real-world applications, these methods are also becoming more widely used. Eduardo et al in [13] applied deep learning methods to solve PDEs for transport models. Zichao et al in [14] proposed a new feed-forward deep network, called PDE-Net for estimating the dynamics of complex systems and their underlying implied PDE models. Most of the systems of DEs encountered in control applications are ODEs, and CK Ye et al in [15] used artificial neural networks (ANNs) to solve the ODE in the state observers. Bingyu et al in [16] applied deep learning techniques were applied to solve option prices using the backward stochastic differential equation (BSDE) method.

Although there are many researches, in considering convergence of the approximate solution, the most of literatures only tend to consider the equation itself with associated conditions, and do not sufficiently considered more implied information about the solution. This is one of the main reasons for some unsatisfactory performance of NN algorithm. For example, Lie algebraic structures, conservation laws, and possible expressions of solutions have not been applied in various machine learning methods. Raisia M et al in [7] proposed a physical information NN algorithm to solve DEs, in which some extra physical properties were tried to be used and got more accuracy solution to the problem. This enlightens more further study on this topics.

In this article, we try to use an algebra structure, admitted Lie groups, of underlying ODEs to get an alternative NN algorithm for solving the IVP of the nonlinear dynamic system.

A Lie group (symmetry) of a DEs is a one parameter transformations mapping a solution to another and providing formal exponential expression to the solution of an IVP of the DEs. Lie group is vital to understanding the solutions characteristics of a nonlinear DEs since it provides a reasonable framework for analysing the solutions to the DEs. For further information on Lie groups of a DEs, one refers to see [17–20]. Best in our knowledge, the essential combination of Lie group method, a powerful theoretical and computational tool, and NN method has not yet explored so far.

In current work, we propose a NN method with combining Lie group of a DEs to solve the IVP of the DEs. Unlike most cases, our proposed neural network algorithm with only one hidden layer can achieve high accuracy to approximate the solution. The simple structure of using a single hidden layer is beneficial not only in data preprocessing but subsequent stages of integration with any other more complex network. Our primary objective is to provide a general form of a NN solution for nonlinear dynamical systems by dividing the solution into two parts: the first term is a solution of an IVP of an ODEs, derived from the original IVP, can be solved by Lie group method without training; the second part is a NN with adjustable parameters, which will be learnt from an NN algorithm to make the final network solution
approximates the solution of the DE. The NN is trained using an unsupervised approach throughout the procedure. The main advantages of the current method are: the method can provide more accurate solution expression, which will provide more accurate nonlinear information admitted by solution to make the subsequent network adjustment more effective; the effectiveness not only in the training interval, but also extrapolation outside the training interval. The efficiency and interpretability of the proposed algorithm are demonstrated by comparing the applications of the method with that of existing numerical method to some specific dynamics systems. Note that the idea used in this paper is a specific form of the Lie group method for the solution expression of the first-order ODEs rather that its reduction ability for a DEs (It can not reduce an IVP of a DEs). This is an innovative thinking and may provide some inspiration for further research on this issue.

The rest of present article is arranged as following. In Section II, we present the basic architecture of the Lie group-based NN and the learning algorithm. In Section III, we provide details of modeling examples and results as applications of our algorithm. In Section IV, we give some conclusion remarks.

2 Preliminaries

2.1 System of initial value problems for first-order ordinary differential equations

For a system of ODEs, the general form is given by

\[
\begin{align*}
  y'_i &= f_i(t, y_1, y_2, \ldots, y_n), \\
  y_i(0) &= \alpha_i,
\end{align*}
\]

where, \( y_i(t) \) is the dependent variable, \( t \in \mathbb{R} \) is the independent variable, and \( f_i \) on the right side of the equation is a linear or nonlinear differentiable function about the independent and dependent variables.

We also know that first-order ODEs are just a special case of (1). Similarly, the IVPs for higher ODEs can be transformed into the forms (1).

2.2 One-parameter continuous transformation group

Suppose there are transformations \( T_\epsilon \) in the \((x, y)\) plane:

\[
T_\epsilon : \hat{x} = \phi(x, y, \epsilon), \quad \hat{y} = \psi(x, y, \epsilon)
\]

Definition 1 \( G \) is a family of transformations \( T_\epsilon \) which depends on the real parameters \( \epsilon \) varying in an interval \( \Delta \). If the following properties are satisfied, \( G = \{T_\epsilon\} \) is called a one-parameter transformation group.

1) There are the following identity transformation when \( \epsilon = 0 \).

\[
T_0 : \hat{x} = \phi(x, y, 0), \quad \hat{y} = \psi(x, y, 0)
\]

2) When \( \epsilon \) changes to \(-\epsilon\), there is an inverse transformation.

\[
T_\epsilon : x = \phi(\hat{x}, \hat{y}, -\epsilon), \quad y = \psi(\hat{x}, \hat{y}, -\epsilon)
\]

3) Suppose

\[
T_\delta : x_2 = \phi(\hat{x}, \hat{y}, \delta), \quad y_2 = \psi(\hat{x}, \hat{y}, \delta)
\]
Then the product of these two transformations also belongs to (2). And the parameter becomes \( \epsilon + \delta \), that is
\[
T_{\epsilon + \delta} : x_2 = \phi(\hat{x}, \hat{y}, \delta) = \phi(x, y, \epsilon + \delta), \quad y_2 = \psi(\hat{x}, \hat{y}, \delta) = \psi(x, y, \epsilon + \delta)
\] (6)

The \( \phi(x, y, \epsilon) \), \( \psi(x, y, \epsilon) \) in (2) is generally referred to as the global form of the continuous group. For a small parameter \( \epsilon \), it can be expanded around \( \epsilon = 0 \) as
\[
\hat{x} = x + \frac{d\hat{x}}{d\epsilon}|_{\epsilon=0} + O(\epsilon^2), \quad \hat{y} = y + \frac{d\hat{y}}{d\epsilon}|_{\epsilon=0} + O(\epsilon^2)
\] (7)

where, \( O(\epsilon^2) \) represents the sum of all higher-order expansions including the second-order minima. Where the auxiliary functions \( \xi(x, y) \) and \( \eta(x, y) \) are introduced
\[
\xi = \frac{d\hat{x}}{d\epsilon}|_{\epsilon=0}, \quad \eta = \frac{d\hat{y}}{d\epsilon}|_{\epsilon=0}
\] (8)

and relate it to the group \( G \) given by (2), then we’ll have
\[
\hat{x} = x + \epsilon \xi + O(\epsilon^2), \quad \hat{y} = y + \epsilon \eta + O(\epsilon^2)
\] (9)

This expansion of (7) around \( \epsilon = 0 \) is called the infinitesimal of the transformation group (2).

**Theorem**  The functions \( \phi(x, y, \epsilon) \), \( \psi(x, y, \epsilon) \) defining a group of transformations satisfy the system of differential equations
\[
\frac{d\hat{x}}{d\epsilon} = \xi(\hat{x}, \hat{y}), \quad \frac{d\hat{y}}{d\epsilon} = \eta(\hat{x}, \hat{y})
\] (10)

with initial value \( \hat{x}|_{\epsilon=0} = x, \quad \hat{y}|_{\epsilon=0} = y \). Conversely, when \( \epsilon = 0 \) and the infinitesimal form of the continuous group is known, integration can be used to find the global form of the continuous group \( G \).

### 2.2.1 Infinitesimal operator of the group

**Definition 2** An infinitesimal operator of the group \( G \) is the linear differential operator
\[
X = \xi(x)^i \frac{\partial}{\partial x^i}
\] (11)

where \( \xi(x)^i \) are determined in (8). Functions \( \xi(x)^i \) are coordinates of the operator \( x \).

Let a function \( f(x, y) \) be defined in the plane \( (x, y) \), and under the transformation of (7), it becomes its overall similar \( f(\hat{x}, \hat{y}) \). Expanding around \( \epsilon = 0 \), as follows
\[
f(\hat{x}, \hat{y}) = f(x, y) + \epsilon X f + \frac{1}{2!}\epsilon^2 X^2 f + \frac{1}{3!}\epsilon^3 X^3 f + \cdots + \frac{1}{n!}\epsilon^n X^n f + \cdots
\] (12)

This series is called the Lie series. \( X \) stands for Infinitesimal operator. From (11), it is defined as
\[
X = \xi(x, y) \frac{\partial}{\partial x} + \eta(x, y) \frac{\partial}{\partial y}
\] (13)
From \[ \text{(12)} \] it can be seen that Lie series can be expressed as
\[
 f(\hat{x}, \hat{y}) = e^{\epsilon X} f(x, y)
\]
where \( e^{\epsilon X} = \sum_{n=0}^{\infty} (1/n!) \epsilon^n X^n \) is an operator. If we take \( f(\hat{x}, \hat{y}) \) as \( \hat{x} \) and \( \hat{y} \) respectively in \[ \text{(14)} \], we get
\[
 \hat{x} = e^{\epsilon X} x, \quad \hat{y} = e^{\epsilon X} y
\]

2.3 Initial value problems for systems of first-order ordinary differential equations admitted by Lie group

The IVPs of system of first-order ODEs is satisfied by the transformation
\[ T_\epsilon : \dot{x} = \phi(x, y, \epsilon), \dot{y} = \psi(x, y, \epsilon) \] in the \( G = \{ T_\epsilon \} \) of \[ \text{(2)} \]. From \[ \text{(10)} \], there are the following
\[
 \begin{align*}
 \frac{d\dot{x}}{d\epsilon} &= X\dot{x}, \quad \dot{x}|_{\epsilon=0} = x \\
 \frac{d\dot{y}}{d\epsilon} &= X\dot{y}, \quad \dot{y}|_{\epsilon=0} = y \\
 \end{align*}
\]

Under the transformation \( \dot{x} = \phi(x, y, \epsilon), \dot{y} = \psi(x, y, \epsilon) \), the solution of \[ \text{(16)} \] are \( \dot{x} = e^{\epsilon X} x \) and \( \dot{y} = e^{\epsilon X} y \).

In \[ \text{(21)} \] (The operator is denoted by \( D \)), the operator \( X \) is divided into two main parts for convenience of calculation, \( X = X_1 + X_2 \), and the solution of problem \[ \text{(16)} \] are rewritten as \( \dot{x} = e^{\epsilon (X_1+X_2)} x \) and \( \dot{y} = e^{\epsilon (X_1+X_2)} y \). We extend them to the following expressions
\[
 \begin{align*}
 e^{\epsilon X_1} x &= e^{\epsilon X_1} x + \sum_{\alpha=1}^{\infty} \sum_{k=\alpha}^{\infty} \frac{\epsilon^k}{k!} X_1^{k-\alpha} X_2^\alpha x \\
 e^{\epsilon X_1} y &= e^{\epsilon X_1} y + \sum_{\alpha=1}^{\infty} \sum_{k=\alpha}^{\infty} \frac{\epsilon^k}{k!} X_1^{k-\alpha} X_2^\alpha y \\
 \end{align*}
\]

A proof of the Gröbner’s formula for the Lie series and its convergence on \[ \text{(17)} \] is given in \[ \text{(21)} \].

To redescribe, the succinct equation has been devised. The Eq. \[ \text{(17)} \] are
\[
 \begin{align*}
 \dot{x}^* &= e^{\epsilon X_1} x + N_1(x, \epsilon) \\
 \dot{y}^* &= e^{\epsilon X_1} y + N_2(y, \epsilon) \\
 \end{align*}
\]

From \[ \text{(10)} \], we know that \( \dot{x} = e^{\epsilon X_1} x \) and \( \dot{y} = e^{\epsilon X_1} y \) corresponding to the IVPs of a new system of ODEs \( d\tilde{x}/d\epsilon = X_1 \tilde{x}, d\tilde{y}/d\epsilon = X_1 \tilde{y} \) with invariant initial value \( \tilde{x}|_{\epsilon=0} = x \) and \( \tilde{y}|_{\epsilon=0} = y \). The problem is simplified by this step. Solving a complex system of ODEs is extremely difficult, but choosing a simple component of the original equation makes the problem simple. In particular, it is also a partial solution of the original equation.

Some methods, such as Lie series, can be solved second portion \( N_i \) in \[ \text{(18)} \]. When the equation is complex, it is difficult to solve. We propose that the second part be replaced with a black-box (a NN). It’s worthy to note that \( N_i \) in \[ \text{(18)} \] are replaced by \( \epsilon N_i \) in order to satisfy the criterion that \( \dot{x}^*(0) = x \) and \( \dot{y}^*(0) = y \) when \( \epsilon = 0 \).

Based on the expression of the Lie group-based solution in \[ \text{(10)} \], Eq. \[ \text{(18)} \] are rewritten as
\[
 \begin{align*}
 \dot{x}^* &= e^{\epsilon X_1} x + \epsilon N_1(x, \epsilon) \\
 \dot{y}^* &= e^{\epsilon X_1} y + \epsilon N_2(y, \epsilon) \\
 \end{align*}
\]
The first part of the problem can be solved by selecting a simpler $X_1$, and the second part $N_i$, should always be solved. The network solution $\hat{x}^*$ and $\hat{y}^*$ are being used to approximate the equation’s solution $x$ and $y$. It is feasible to use the following expression

$$\hat{x} \approx \bar{x} + \epsilon N_1(\epsilon, p), \quad \hat{y} \approx \bar{y} + \epsilon N_2(\epsilon, p)$$

(20)

The network output $N_i$ is a value that incorporates just the input value $\epsilon$ and the network parameter $p$, with no equation boundary value.

The above is for the IVPs of a system of first-order ODEs, and is also applicable to the IVPs of higher-order ODEs and first-order ODE. The mechanism is detailed below.

### 3 Approximation of a system of ordinary
differential equations by a neural network based
on Lie groups

In this section, we investigate the common types of nonlinear dynamical systems for which the Lie group-based NN algorithm solves the procedure as follows.

Consider the following coupled first-order ODEs,

$$\begin{align*}
x' &= f_1(t, x, y), \\
y' &= f_2(t, x, y).
\end{align*}$$

(21)

With the initial value $x(0) = \alpha_0$, $y(0) = \alpha_1$, convert it to the form \[10\],

$$\begin{align*}
\frac{dx}{dt} &= X x, \quad x(0) = \alpha_0 \\
\frac{dy}{dt} &= X y, \quad y(0) = \alpha_1
\end{align*}$$

(22)

The differential operator $X = f_1(t, x, y) \partial_x + f_2(t, x, y) \partial_y$. Written in the form of \[19\] as

$$\begin{align*}
x &= e^{tX} \alpha_0, \\
y &= e^{tX} \alpha_1.
\end{align*}$$

(23)

By the decomposability of the operator in \[17\], Write the above equation in the form containing the network output

$$\begin{align*}
\hat{x} &= e^{tX_1} \alpha_0 + tN_1(t, p), \\
\hat{y} &= e^{tX_1} \alpha_1 + tN_2(t, p),
\end{align*}$$

(24)

Where $e^{tX_1} \alpha_0$ is the solution of equation $\bar{x}' = X_1 \bar{x} = \bar{f}_1(t, \bar{x}, \bar{y})$ with initial value $\bar{x}(0) = x(0) = \alpha_0$, and $e^{tX_1} \alpha_1$ is the solution of equation $\bar{y}' = X_1 \bar{y} = \bar{f}_2(t, \bar{x}, \bar{y})$ with initial value $\bar{y}(0) = y(0) = \alpha_1$. We clearly know that $e^{tX_1} \alpha_0$ is equivalent to $e^{tX_1} \alpha_0 = e^{tX_1} \bar{x} |_{x \rightarrow \alpha_0}$, $e^{tX_1} \alpha_1$ is the same, and $t$ corresponds to the parameter $\epsilon$ of \[16\]. The number of equations determines $i$ in $N_i$.

Any continuous function can be approximated by a feed-forward NN with a single hidden layer, according to the universal approximation theorem \[1\]. This ANN (N) can also be expressed as a matrix multiplication.

$$N(t, p) = W_2 \sigma (W_1 t + b_1) + b_2$$

(25)
$W_1 \in R^{m \times n}$, which is the hidden layer weight from the input layer $t_i \in R^n (i = 1, 2, \ldots, n)$, $b_1 \in R^m$, which is the hidden layer bias term, $W_2 \in R^m$, which is the weight matrix of the weights from the hidden layer to the output layer, and $b_2 \in R$, which is the output layer bias term. The number of neurons in the hidden layer is given by $m$, $\sigma = \tanh$ is a nonlinear activation function. The summation formula is expanded and written in vector form as $N(t, p) = \sum_{j=1}^m (w_{2j} \cdot e^{x_j - i \cdot \pi}) + b_2$, $z_j = t \cdot w_{1j} + b_{1j}$, $j = 1, 2, \ldots, m$. The network structure is shown in Fig 1. The number of networks grows in lockstep with the number of equations. $p$ represents the set of $W_1, W_2, b_1, b_2$. The NN takes a scalar input $t(t = \{t_1, t_2, \ldots, t_n\} in [0, a])$ and returns a scalar output $N(t)$, which fits the solution in (21) of scalar functions $x(t)$ and $y(t)$ by $\hat{x} = \bar{x} + tN_1(t, p)$ and $\hat{y} = \bar{y} + tN_2(t, p)$. We could even use a more explicit expression afterwards.

Fig 1. Neural network models for systems of ODEs based on Lie groups

\[
\dot{x}'(t, p) \approx f_1(t, \hat{x}(t, p), \hat{y}(t, p)), \quad \dot{y}'(t, p) \approx f_2(t, \hat{x}(t, p), \hat{y}(t, p))
\]  

(26)

This can be easily calculated analytically $x'(t, p)$ and $y'(t, p)$ without the need of techniques like differencing.

\[
\dot{x}'(t, p) = \frac{\partial (\bar{x} + tN_1(t, p))}{\partial t} = \frac{\partial \bar{x}}{\partial t} + N_1(t, p) + t \frac{\partial N_1(t, p)}{\partial t}
\]

\[
\dot{y}'(t, p) = \frac{\partial (\bar{y} + tN_2(t, p))}{\partial t} = \frac{\partial \bar{y}}{\partial t} + N_2(t, p) + t \frac{\partial N_2(t, p)}{\partial t}
\]

(27)

\[
\frac{\partial \bar{x}}{\partial t} \text{ and } \frac{\partial \bar{y}}{\partial t}
\]
can be obtained by solving the differential equation by the properties of the Lie group.

The previously studied solution problem has been turned into an optimization problem in which the loss function $L(p)$ is minimized by optimizing the parameter $p$

\[
L(p) = \int_0^a [\dot{x}'(t, p) - f_1(t, \hat{x}(t, p), \hat{y}(t, p))]^2 \, dt + \int_0^a [\dot{y}'(t, p) - f_2(t, \hat{x}(t, p), \hat{y}(t, p))]^2 \, dt
\]  

(28)

(29)

To accomplish the calculation, we’ll utilize the summation formula below.

\[
L(p) = L_1(p) + L_2(p)
\]

\[
= \frac{1}{n} \left( \sum_{i=1}^n [\dot{x}'(t_i, p) - f_1(t_i, \hat{x}(t_i, p), \hat{y}(t_i, p))]^2 \\
+ \sum_{i=1}^n [\dot{y}'(t_i, p) - f_2(t_i, \hat{x}(t_i, p), \hat{y}(t_i, p))]^2 \right)
\]

(30)

where $\{t_i\}$ is a set of training points that spans the $[0, a]$ domain. The number of $t$ values in the range $[0, a]$ is $n$. The error function $L_1$ relates to $\hat{x}$, while the error function $L_2$ relates to $\hat{y}$.

When the loss function $L(p)$ is small enough, the approach is capable of approximating the true solution $x(t)$ and $y(t)$ in the $[0, a]$. We not only use the above
equation’s mean square error to generate the loss function, but we also utilize the average root mean square error to assess the algorithm’s superiority.

\[ L_{RMSE} = \frac{1}{2} (L_1(p) + L_2(p)) \]  

where \( L_1(p) \) is the mean square error between the network solution \( \hat{x}(t, p) \) and the exact solution \( x(t) \) (if the exact solution is unavailable, the numerical solution \( x(t) \) is used), \( L_1(p) = \sqrt{\frac{1}{n} \left( \sum_{i=1}^{n} [\hat{x}(t_i, p) - x(t_i)]^2 \right)} \). Similarly, \( L_2(p) = \sqrt{\frac{1}{n} \left( \sum_{i=1}^{n} [\hat{y}(t_i, p) - y(t_i)]^2 \right)} \).

**Extension** This algorithm can also be used for IVPs for systems of higher-order ODEs, for the sake of brevity, will not be discussed here.

All the cases discussed in this paper are based on that scenario. The number of equations dictates the number of networks, and for two or more equations, the call between the solutions of the equations and the nonlinear characteristics of the equations causes the call between networks, and the final result is not as good as that of a single equation. To optimize network parameters and reduce the loss function \( L(p) \), we used the more stable BFGS approach.

The proposed process approach is summarized in Algorithm 1.

**Algorithm 1** The following are the steps in solving an ODEs using a feed forward NN model based on the Lie Group method.

1. Consider the discrete points \( t_i \) in the range \([0, a]\) \((i = 1, 2, ..., n)\)
2. Configure the ANN architecture (number of layers, dimensionality of each layer and activation function, in this paper, we choose a hidden layer, the number of neurons in the hidden layer is \( m \), and the activation function uses tanh)
3. Initialize the network parameters \( p_0 \), i.e., \( W_1, b_1, W_2, b_2 \)
4. Select the appropriate operator \( X_1 \) for (21)
5. Calculate the objective function (trial solution) \( \hat{x} = \bar{x} + tN_1(t, p) \) and \( \hat{y} = \bar{y} + tN_2(t, p) \) by solving the new systems of equations corresponding to \( X_1 \)
6. Form the loss function \( L(p) \) according to (30)
7. To update the parameter \( p \), adopt the BFGS optimization approach
8. When the conditions are met, the optimal parameter \( p^* \) is determined. The maximum number of iterations or the necessary accuracy of \( L(p) \) can be selected
9. for all \((W_1, b_1, W_2, b_2) \in p\) do
10. Update \( W_1 \) based on gradient \( dL(p)/dW_1 \) using BFGS
11. Update \( b_1 \) based on gradient \( dL(p)/db_1 \) using BFGS
12. Update \( W_2 \) based on gradient \( dL(p)/dW_2 \) using BFGS
13. Update \( b_2 \) based on gradient \( dL(p)/db_2 \) using BFGS
14. end for
15. return \( W_1, b_1, W_2, b_2 \)
16. To determine the algorithm effectiveness, use the average RMSE (31)

### 4 Numerical Experiments and Applications

In this section, we perform experiments for some examples of nonlinear dynamical systems to demonstrate the behavior and properties of this new approach, comparing the accuracy and convergence of traditional numerical methods with the proposed method when the analytical solution is unknown, using the average mean square error
to measure the accuracy of the numerical solution, and testing the algorithm’s
generalization capability. We select different types of practical problems and, where
appropriate, point out the difficulties encountered. Although our NN just has one
hidden layer, different models may use a different number of nodes for the length of
the interval they examine. The following numerical experiments use the BFGS
optimization algorithm (which is the most stable when compared to other
optimization algorithms) to minimize the loss function in (30), instead of directly
calculating the derivatives of the NN, we use an automatic differentiation technique
that has been proven to work well. Use the symbolic calculator mathematica to
calculate the first part of the decomposition corresponding to the operator $X_1$. For
the traditional numerical method using Scipy in Python.

Example 1. Ecosystems contain classic nonlinear dynamics equations for
three-species food chains [22], in which the bottom prey $x$ is preyed upon by the
middle species $y$, which is then preyed upon by the top predator $z$. In the real world,
the mouse-snake-owl is an example of a three-species food chain.

$$\frac{dx}{dt} = ax - bxy$$ (32)

$$\frac{dy}{dt} = -cy + dxy - eyz$$ (33)

$$\frac{dz}{dt} = -fz + gyz$$ (34)

where $x$ is the dependent variable representing the density of the bottom prey
population, $y$ is the dependent variable representing the density of the middle
predator (prey) population, $z$ is the dependent variable representing the density of the
higher predator population, and $t$ is the independent variable representing time, which
is $\epsilon$ in [20], the initial conditions are $x(0) = 0.5, y(0) = 1, z(0) = 2$, and we consider
the case where $t$ is in the interval $[0, 3]$. In the absence of a predator ($y = 0$), Eq. (32)
shows that the prey $x$ would grow at a constant rate $a$, assuming that the prey $x$ have
an unlimited food supply. Predation on the prey is considered to be proportional to
the rate at which predators $y$ and prey $x$ are present at the same time, denoted by $bxy$.
Predation is impossible if either $x$ or $y$ is zero. Similarly, in the absence of prey
($x = 0$), Eq. (33) demonstrates that the density of predators $y$ would decline at a
constant rate $c$ due to natural death or emigration, $e$ represents the effect of predation
on species $y$ by species $z$. This equation assumes that the predator population $y$ only
hunts the same prey species as in Eq. (32). In Eq. (34), $f$ represents the natural
death rate of species $z$ in the absence of prey, $g$ represents the reproduction rate of
species $z$ in the presence of prey $y$. $a, b, c, d, e, f, g > 0$.

According to the operator $X = (ax - bxy) \partial_x + (-cy + dxy - eyz) \partial_y + (-fz + g
yz) \partial_z$, from which the operator $X_1$ is chosen as $ax\partial_x - cy\partial_y - fz\partial_z$, initial
parameters $a = b = c = d = e = f = g = 1$, we can easily get the solution of this part
$x = \frac{1}{2} e^t, y = e^{-t}, z = 2e^{-t}$. Thus, the trial solution is
$x = \frac{1}{2} e^t + tN_1, y = e^{-t} + tN_2, z = 2e^{-t} + tN_3$.

We use a uniform distribution of 100 training points in the interval $[0, 3]$ and
$m = 100$ to train the NN, similarly, $N_2$ and $N_3$ is a network model of this type. Since
the model has no analytical solution, the fit to the numerical solution is plotted as
follows Fig[2]. The extrapolation capability of the test set at $[0, 3.5]$ is shown in Fig[3]
The optimization method is BFGS, as shown in Fig[4] the log loss is the minimum
compared to other methods at 1000 iterations. The loss function of Eq. (30),
$L(p) = 7.303 \times 10^{-5}, RMSE = 0.00851$. We discovered during the experiment that
when the training interval is more than 10, the training effect does not reach the ideal state. It’s also worth mentioning that the number of nodes necessary for training nonlinear differential equations in the hidden layer is more than before, which is another feature of nonlinear systems of equations.

**Fig 2.** Comparison between neural network solutions \( \hat{x}, \hat{y}, \hat{z} \) and numerical solutions \( x, y, z \) in Example 1

**Fig 3.** Network network solution \( \hat{x}, \hat{y}, \hat{z} \) and numerical solution \( x, y, z \) in the test set

**Fig 4.** Log loss function \( L(p) \) in Example 1

---

**Example 2.** We consider the van der Pol oscillator [23], is a non-conservative oscillator with a linear spring force and a non-linear damping force. The equation is given

\[
\frac{d^2x}{dt^2} - \mu (1 - x^2) \frac{dx}{dt} + x = 0 \tag{35}
\]

Applying the Liénard transformation [24] \( y = x - x^3 - \frac{1}{\mu} \frac{dx}{dt} \), the equation can be written as a system of ODEs,

\[
\begin{align*}
\frac{dx}{dt} &= \mu \left( x - \frac{1}{3} x^3 - y \right) \\
\frac{dy}{dt} &= \frac{x}{\mu}
\end{align*}
\tag{36}
\]

The operator form is

\[
X = \mu \left( x - \frac{1}{3} x^3 - y \right) \partial_x + \frac{x}{\mu} \partial_y.
\]

We select to determine the analytical solution of \( X_1 = -\mu y \partial_x + \frac{x}{\mu} \partial_y \) from Eq. (36), \( x(0) = 1, y(0) = 2 \). The fit of the Runge-Kutta methods in the Scipy solver with the NN solution in the interval \([0, 10]\), where \( \mu = 1 \), is shown in Fig 5. We use 40 training data in the interval for training and 50 nodes in the hidden layer. Fig 6 shows the extrapolation capability of the NN solution using trained parameters in the interval \([0, 11]\) compared to the numerical solution, again using BFGS optimization. Fig 7 displays the loss function iteration of the training, we can see that the error is about \( 10^{-1} \) after roughly 70 iterations. Throughout the training process, the loss function \( L(p) = 2.07 \times 10^{-4}, RMSE = 0.082 \) is the measuring function.

**Fig 5.** Comparison between neural network solutions \( \hat{x}, \hat{y} \) and numerical solutions \( x, y \)

**Fig 6.** Network network solution \( \hat{x}, \hat{y} \) and numerical solution \( x, y \) in the Example 2

**Fig 7.** Log loss function \( L(p) \) in Example 2

---

**Example 3.** A stimulation of chaos phenomena of Lorenz system. Consider the Lorenz equations [25]

\[
\begin{align*}
\dot{y}_1 &= \sigma(y_2 - y_1), \\
\dot{y}_2 &= \rho y_1 - y_2 - y_1 y_3, \\
\dot{y}_3 &= -\beta y_3 + y_1 y_2,
\end{align*}
\tag{37}
\]

\((y_1, y_2, y_3) \in \mathbb{R}^3, \sigma, \rho, \beta > 0\)
As an alternative example of application of our method, we consider the case \( \sigma = 10, \rho = 28, \beta = 8/3 \) and \( y_1(0) = 1, y_2(0) = 5, y_3(0) = 10 \) discussed in \cite{25}. The system has been already in the standard form. As previous examples, we take \( X_1 = 10y_2\partial y_1 - y_2\partial y_2 - \beta y_3\partial y_3 \), It yields the exact solutions to the associated Initial value problems of system \((37)\)

\[
\begin{aligned}
\bar{y}_1 &= e^{-10t} \\
\bar{y}_2 &= 5e^{-t} \\
\bar{y}_3 &= 10e^{-28t}
\end{aligned}
\]  

(38)

Basing on the first parts of trial solutions \( \hat{y}_i = \bar{y}_i + tN_i(t,p), i = 1, 2, 3 \). To derive the approximate solution \( \hat{y}_i \), We trained the \( N_i(t,p) \) network on the interval \([0, 0.5]\) using uniformly spaced 40 points with 30 of hidden nodes. Fig.8 shows a comparison between the network solution and the numerical solution to \((37)\). Fig. 9 shows that the ability to extrapolate in the interval \([0, 0.6]\) is not terrible. Fig. 10 depicts the loss function convergence diagram. It can be seen that the optimization algorithm BFGS is the most stable and the easiest to reach the logarithmic minimum of the loss function.

Fig 8. Comparison between neural network solutions \( \hat{y}_i \) and numerical solutions \( y_i \)

Fig 9. Network network solution \( \hat{y}_i \) and numerical solution \( y_i \) in the Example 3

Fig 10. Log loss function \( L(p) \) in Example 3

Example 4. We are thinking about another chaotic dynamical system consisting of three nonlinear ODEs, consider the following Rosseler system \cite{26}

\[
\begin{aligned}
\dot{x} &= -y - z \\
\dot{y} &= x + ay \\
\dot{z} &= b + z(x - c)
\end{aligned}
\]  

(39)

The operator \( X_1 = -z\partial x + x\partial y - cz\partial z \) is determined from the operator \( X = (-y - z)\partial x + (x + ay)\partial y + (b + z(x - c))\partial z \), and this step is skipped owing to familiarity, the linear portion can be intercepted directly from Eq.\((39)\) to produce the new systems of linear equations \( \bar{x}' = -\bar{z}, \bar{y}' = \bar{x}, \text{ and } \bar{z}' = cz \), with initial value \( \bar{x} = 1, \bar{y} = 5, \bar{z} = 10 \). The trial solution is \( \hat{x} = -0.754386 + 1.75439e^{-5.7t} + tN_1, \hat{y} = 5.30779 - 0.307787e^{-5.7t} + tN_2, \text{ and } \hat{z} = 10e^{-5.7t} + tN_3 \), as expected.

Our neural net solution achieves a close fit to the numerical method, as seen in Fig.11 the hidden layer was trained with a grid of 40 equidistant points in \( t = [0, 1] \) and 50 hidden units. 200 points in the interval \([0, 1.4]\) are used to verify the trained network extrapolation capability, as shown Fig.12 Fig.13 can be observed through a comparison of different optimization methods that the optimization method we employed is stable, loss function \( L(p) = 3.266 \times 10^{-6} \) was achieved approximately 300 times, which was used to estimate the algorithm \( \text{RMSE} = 4.747 \times 10^{-5} \).

Fig 11. Comparison between neural network solutions \( \hat{x}, \hat{y}, \hat{z} \) and numerical solutions \( x,y,z \)
Fig 12. Network network solution $\hat{x}, \hat{y}, \hat{z}$ and numerical solution $x, y, z$ in the Example 4

Fig 13. Log loss function $L(p)$ in Example 4

5 Discussion and conclusions

Our network solves some useful problems and provides use cases for deep learning, and our network is structurally simpler and easier to study than other use cases. Our proposed lie group based NN algorithm, verified the accuracy of the method by solving the nonlinear model initial value problem. The initial network parameters are considered as random. From this study, it can be seen that the proposed model is easy to implement and the method is efficient and straightforward to solve for some nth ODEs or n sets of ODEs. It also provides a means to study nonlinear dynamical systems. We will utilize this method in PDEs in the future, or we will extend the network structure to deep networks, thinking about memory conditions and temporality through algorithmic upgrades.

Conflict of Interest:

As far as we know, there are no conflicts of interest, financial or other conflicts between the designated author and the editors, reviewers and readers of this magazine.

Acknowledgements

The authors thanks the supporting of National Natural Science Foundation of China with grant number 11571008.

References

1. G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals and Systems 2 (4) (1989) 303–314.

2. H. Lee, I. S. Kang, Neural algorithm for solving differential equations, Journal of Computational Physics 91 (1) (1990) 110–131. [doi:https://doi.org/10.1016/0021-9991(90)90007-N]

3. I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Transactions on Neural Networks 9 (5) (1998) 987–1000. [doi:10.1109/72.712178]

4. I. E. Lagaris, A. C. Likas, D. G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Transactions on Neural Networks 11 (5) (2000) 1041–1049. [doi:10.1109/72.870037]

5. M. L. Piscopo, M. Spannowsky, P. Waite, Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions, Physical Review D 100 (1). [doi:ARTNO1600210.1103/PhysRevD.100.016002]

6. M. Baymani, S. Effati, H. Niazman, A. Kerayechian, Artificial neural network method for solving the navier-stokes equations, Neural Computing & Applications 26 (4) (2015) 765–773. [doi:10.1007/s00521-014-1762-2]
7. M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707. doi:10.1016/j.jcp.2018.10.045

8. C. Aouiti, B. Ghanmi, M. Miraoui, On the differential equations of recurrent neural networks, International Journal of Computer Mathematics 98 (7) (2021) 1385–1407. doi:10.1080/00207160.2020.1820493

9. S. Mall, S. Chakraverty, Chebyshev neural network based model for solving lane-emden type equations, Applied Mathematics and Computation 247 (2014) 100–114. doi:10.1016/j.amc.2014.08.085

10. M. Ngom, O. Marin, Fourier neural networks as function approximators and differential equation solvers, Statistical Analysis and Data Mining 14 (6) (2021) 647–661. doi:10.1002/sam.11531

11. W. Xu, R. Chen, X. Li, D. Duvenaud, Infinitely deep bayesian neural networks with stochastic differential equations.

12. V. Iakovlev, M. Heinonen, H. Lhdesmki, Learning continuous-time pdes from sparse data with graph neural networks.

13. E. Abreu, J. Florindo, A study on a feedforward neural network to solve partial differential equations in hyperbolic-transport problems.

14. Z. Long, Y. Lu, X. Ma, B. Dong, Pde-net: Learning pdes from data.

15. C. K. Ye, P. L. Ooi, A faster and cheaper method of implementing states observers using artificial neural networks, in: Instrumentation & Measurement Technology Conference, 2007.

16. B. Yu, X. Xing, A. Sudjianto, Deep-learning based numerical bsde method for barrier options, Papers.

17. F. Schwarz, Algorithmic lie theory for solving ordinary differential equations, Algorithmic Lie Theory for Solving Ordinary Differential Equations, 2007.

18. V. D. Sharma, R. Radha, Exact solutions of euler equations of ideal gasdynamics via lie group analysis, Zeitschrift Fur Angewandte Mathematik Und Physik 59 (6) (2008) 1029–1038. doi:10.1007/s00033-007-6140-9

19. X. Y. Tang, J. Lin, Conditional similarity reductions of jimbo-miwa equation via the classical lie group approach, Communications in Theoretical Physics 39 (1) (2003) 6–8.

20. P. Devi, K. Singh, Lie symmetry analysis of the nonlinear schrödinger equation with time dependent variable coefficients, International Journal of Applied and Computational Mathematics 7 (1) (2021) 1–18.

21. S. Filippi, A new lie series method for the numerical integration of ordinary differential equations, with an application to the restricted problem of three bodies.

22. C. V. Pao, Global asymptotic stability of lotka volterra 3-species reaction diffusion systems with time delays, Journal of Mathematical Analysis and Applications 281 (1) (2003) 186–204.
23. U. Parlitz, W. Lauterborn, Period-doubling cascades and devil’s staircases of the driven van der pol oscillator, Physical Review A 36 (3) (1987) 1428–1434.

24. W. A. Albarakati, N. G. Lloyd, J. M. Pearson, Transformation to lienard form, Electronic Journal of Differential Equations 2000 (76) (2000) 449–478.

25. J. Guckenheimer, P. Holmes, Nonlinear oscillations, dynamical systems, and bifurcations of vector fields, Physics Today 38 (11) (1993) 102–105.

26. O. E. Rössler, An equation for continuous chaos, Physics Letters A 57 (5) (1976) 397–398.