# Artificial Neural Networks for Solving ODEs and PDEs

Jun Hao (Simon) Hu [*]

August 9, 2018

**Abstract**

Neural Networks (NNs) have become the holy grail of modern computational theory. They have seen their fair share of applications across numerous scientific disciplines. In this paper, I propose a method for solving initial and boundary-value problems using artificial neural networks (ANNs). I will study its applications to solving differential equations, up to the second order. Several examples and comparisons against other popular numerical methods will be presented.

## 1 Introduction.

For many problems in physics and engineering, numerical methods for solving differential equations have proved to be critical. The task of numerically solving differential equations, however, can be a computationally difficult challenge. This is in part due to the curse of dimensionality, where the time required for the computation increases exponentially. Traditionally, finite differences, element, volume, and its variants, are adequate for most applications. The problem however is that these numerical methods depend on a particular collocation of the domain, and as such only output the solution to the PDE at the collocation points. This is acceptable for most real-world applications, but not in theoretical applications where we want to understand, or confirm, properties of solutions to the PDE.

Since a multilayer perceptron, with one hidden layer, has excellent function approximation properties, it is not surprising that we would want to employ NNs for solving differential equations. To employ NNs, we need to first cast the problem of solving a PDE into an approximation problem. By employing a feed-forward neural network architecture to solve differential equations, we obtain an analytic approximation to the solution which has very nice topological properties such as continuity and differentiability, and has a closed form expression. This is what differentiates the proposed method from any popular Galerkin numerical method; with this method, we can obtain an analytic, closed form solution.

The main idea of this method is to first assume a form of the solution, called the trial function, and then train a NN to learn the network parameters, which in this case are the weights and biases. The trial function contains a function that contains tunable network parameters. Once the NN has found the best network parameters, obtained by minimizing an appropriately defined cost function, we hope that the trial solution is close to the actual solution. In fact, when we study the error properties of this method, we will see that this method has error properties that are very nice.

### 1.1 Preliminary Remarks.

Existence and uniqueness theorems for ODE and PDE is a rich and interesting topic of study, but we will not concern ourselves with these issues. These considerations should be taken into account before employing any numerical method, as it provides insight as to what the solution is. It also allows us to confirm that our numerical solution is indeed, a correct solution. However, to dedicate a section to uniqueness and existence theorems for differential equations would be ridiculous. As a caveat, I will assume that the differential equation problem has a solution, though this solution does not need to be unique. Furthermore, I will only

[*]Department of Electrical and Computer Engineering, University of California San Diego

consider differential equations up to and including the second order. This method works for higher-order differential equations, but due to computational resource restraints, we will not touch the topic.

With respect to notation, throughout the paper we will assume that $\Omega \subset \mathbb{R}^m$ is an open set, and for any function $f$, we use the shorthand $f_i = f(x_i)$. Additionally, $D$ and $D^2$ will denote the gradient and Hessian operators, respectively.

### 1.2 Structure of Paper.

In section 2, I present the framework for the proposed numerical method. We will consider the general problem at hand, and provide an outline of the method. In sections 3, 4, and 5, I focus on classes of linear ODEs, systems of ODEs, and PDEs respectively, that can be tackled using the proposed method. Examples and comparisons will be presented. In section 6, I discuss the possibility of employing this method for non-linear PDE. In section 7, I discuss further improvements that can be made, for this method.

## 2 Outline of the Method.

Consider the problem of finding a function $\psi = \psi(x)$, $\psi : \overline{\Omega} \to \mathbb{R}$ that, for some specified function $f = f(x)$, $f : \Omega \to \mathbb{R}$, satisfies the equation

$$G(x, \psi, D\psi, D^2\psi) = f(x) \tag{2.1}$$

for all $x \in \Omega$, subject to appropriately defined initial and boundary conditions. As simple as this equation looks, this is not a particularly easy problem to solve especially if the left-hand side is complicated or non-linear. The condition that (2.1) hold for all $x \in \Omega$ creates a computational burden. To relieve this burden, we perform a discretization of the domain $\Omega$ and its boundary $\partial\Omega$. Let us define by $U \subset \Omega$, the discretized domain and let $\partial U = U \cap \partial\Omega$. One particularly simple discretization is a uniform grid discretization, where each grid point is spaced uniformly across the domain. The reader should note that this is a *very* simple discretization and is not optimal for domains that are irregular or not square. Once the domain is discretized, the problem reduces to finding a function $\psi$ that satisfies the equation

$$G(x_i, \psi_i, D\psi_i, D^2\psi_i) = f(x_i) \tag{2.2}$$

for all $x_i \in U$, $(i = 1, \ldots, n)$, subject to an appropriately defined, discretized version of the initial and boundary conditions. It would be at this point that a Galerkin method would be implemented to solve the PDE, thus obtaining the solution at the $x_i$s. But this can still be too much to ask. For high-dimensional PDEs, taking a large collection of points presents an issue due to computational restraints. As such, we relax the condition even further, by saying that we want to find a solution $\psi$ that almost satisfies (2.2) by minimizing an appropriately defined error index. To do so, we must define the *trial function.*

Our goal is to approximate the solution $\psi$ in (2.1) via function approximation. The trial function is a function $\psi_{\text{trial}} = \psi_{\text{trial}}(x, p)$, $\psi_{\text{trial}} : \overline{\Omega} \to \mathbb{R}$, where $p$ are the parameters of the network, that does the job. For the proposed neural network structure, the parameters of the network are the weights and biases which will be learned by the network using an optimization algorithm. The idea is to use the trial function as an initial guess, and fine-tune it using the network so that the approximation differs to the actual solution by a small absolute error.

We assume that the trial function takes the form

$$\psi_{\text{trial}}(x; \theta) = \xi(x) + \gamma(x)N(x; \theta) \tag{2.3}$$

where $N(x, p)$ is the output of the feed-forward neural network, $\xi = \xi(x)$, $\xi : \overline{\Omega} \to \mathbb{R}$ and $\gamma = \gamma(x)$, $\gamma : \overline{\Omega} \to \mathbb{R}$ are functions chosen with the initial and boundary conditions in mind. The function $\xi(x)$ is chosen so that the initial and boundary conditions are satisfied, and $\gamma(x)$ is chosen so that it does not contribute to the boundary and initial conditions. There are many ways to construct the functions $\xi$ and $\gamma$ according to our requirements, but it is usually obvious which construction will fare better. Our hope is that, with the learned parameters in hand, the function $\psi_{\text{trial}}$ smoothly approximates $\psi$ to some degree of accuracy. To measure the effectiveness of our approximation, we turn to the cost function.

Defining an appropriate cost function is an art in and of itself but it is usually clear, from the problem at hand, what the cost function should be since it has a physical interpretation. A good choice of the cost function can make the difference between an effective and ineffective model. A very popular choice of a cost function is the standard squared-error cost function, given by

$$J(\theta) = \sum_{x_i \in U} [G(x_i, \psi_{\text{trial}}(x_i; \theta), D\psi_{\text{trial}}(x_i; \theta), D^2\psi_{\text{trial}}(x_i; \theta)) - f(x_i)]^2. \tag{2.4}$$

Regardless, for the sake of generality, let us denote by $J(p)$, the cost function associated with (2.2). Training the network is therefore equivalent to finding $\theta^*$ such that

$$\theta^* := \arg\min_{\theta} [J(\theta)].$$

To solve this minimization problem, we employ an appropriately chosen optimization algorithm.[1] We found that the stochastic gradient descent algorithm works very well for this problem. With $\theta^*$ in hand, the optimal solution is therefore given by $\psi_{\text{trial}}(x; \theta^*)$, which is the best approximate solution to (2.2). We now move to discuss the structure of the neural network.

The structure of the neural network is shown in the figure below.

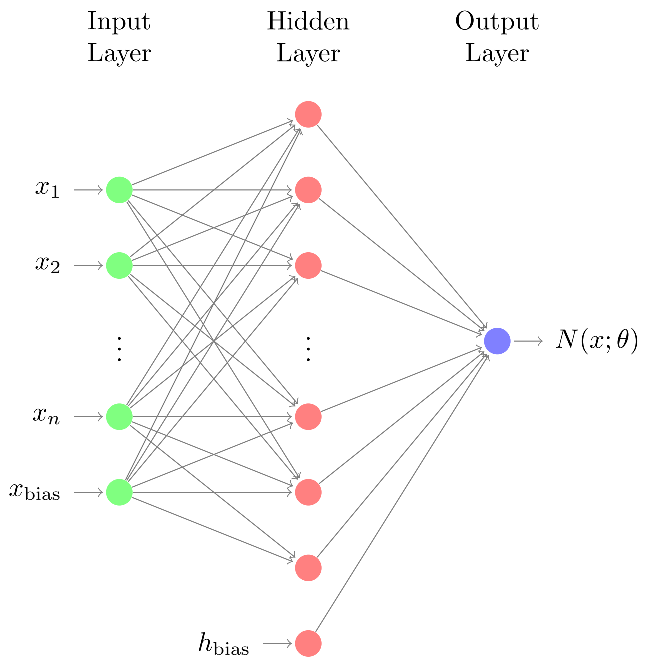

Figure 1: Schematic of a feed-forward neural network with one hidden layer.

The neural network has an input layer with $n + 1$ nodes, a hidden layer with $H + 1$ nodes, and an output layer with 1 node. The output of this neural network is the function $N(x; \theta)$, which appears in the definition of the trial function. To complete the construction of our network, we need to define an appropriate underlying activation function.

A good choice for the activation function is essential to the operation. For simplicity, we will not construct our own activation function and use ones that are commonly used in real-world applications. To this end, the underlying activation function will be the ReLU function. Whenever appropriate, we will use the leaky ReLU activation function to rectify issues arising from dead neurons. The ReLU function is also chosen for its effectiveness at deterring the vanishing gradient effect. For a discussion on the comparison between popular activation functions, see (enter reference here).

[1]The choice of the minimization algorithm should take into account, the structure of the cost function, in particular the number of local minima. Gradient descent, for example, does not guarantee that we converge to the global minima, only a local minima.

## 3 Ordinary Differential Equations.

*Remark* 3.1. For this section, let $\Omega = [a, b] \subset \mathbb{R}$ for $a < b$.

*Remark* 3.2. In the following, we will consider the general case where there are both initial and boundary conditions. If one or the other does not exist, then the reader should be careful not to use the formulas verbatim. Instead, the reader should take the general idea, and modify the method for the problem at hand.

### 3.1 First-Order ODEs.

Consider the first-order IBVP

$$\begin{cases} p(x)\psi'(x) = f(x, \psi), & x \in \Omega, \\ \psi(x) = h(x), & x \in \partial\Omega, \\ \psi(y) = g(y), & y \in \Omega \end{cases} \tag{3.1}$$

where $p(x), f(x, \psi), h(x), g(x)$ are $C^\infty(\Omega)$ functions. For the purpose of existence and uniqueness, we require that the function $f_\psi(x, \psi)$ must be continuous and satisfy the Lipschitz condition. That is, for some $\psi_1, \psi_2 \in \Omega$, there exists a constant $L > 0$ such that

$$\left| f(x, \psi_1) - f(x, \psi_2) \right| \leq L |\psi_1 - \psi_2| \,.$$

As mentioned in the previous section, we discretize $\Omega$ and $\partial\Omega$, which will be denoted by $U$ and $\partial U = U \cap \partial\Omega$ respectively.

Next, we construct the trial function. To construct the trial function, we break the problem into two parts: first, we construct the function $\xi$ and then we construct $\gamma$. Recall, that the goal is to construct $\xi$ so that the boundary/initial conditions are satisfied. Put another way, we want to construct $\xi(x)$ so that it takes on certain values at specific points. To this end, we will use the theory of Lagrange interpolation. Using the Newton form of the Lagrange interpolating polynomial, we find that $\xi$ takes the form

$$\xi(x) = \xi[a] + \xi[y, a](x - a) + \xi[b, y, a](x - a)(x - y) \tag{3.2}$$

where $\xi[\,\cdot\,]$ is a divided difference. This completes the construction of $\xi$. To construct $\gamma$, we require that it does not contribute to the boundary/initial conditions. While we are free to again use the theory of Lagrange interpolation, it is not necessary. Instead, we construct $\gamma$ as a polynomial that has roots at the boundary/initial point(s). Thus, our construction of $\gamma$ takes the form

$$\gamma(x) = (x - a)(x - y)(x - b). \tag{3.3}$$

This completes the construction of $\gamma$.

With the construction of the trial function complete, we can now define the cost function $J(\theta)$, which will be a function of the trial function. The choice of the cost function varies with each problem, but as an example, let us consider the squared-error function given by

$$J(\theta) = \sum_{x_i \in U} [p(x_i)\psi'_{\text{trial}}(x_i; \theta) - f(x_i, \psi_{\text{trial}}(x_i; \theta))]^2. \tag{3.4}$$

### 3.2 Second-Order ODEs.

Let us now consider the general second-order IBVP

$$\begin{cases} p(x)\psi''(x) + q(x)\psi'(x) + r(x)\psi(x) = f(x), & x \in \Omega, \\ \psi(x) = h(x), & x \in \partial\Omega, \\ \psi'(y) = g(y), \ \psi(y) = v(y) & y \in \Omega. \end{cases} \tag{3.5}$$

where $p(x), q(x), r(x), h(x), g(y)$ and $f(x)$ are $C^\infty(\Omega)$ functions.

To construct the function $\xi$ for general second-order IBVP, we may have to use the theory of osculating polynomials, as not only do we require $\xi$ take on certain values at specific locations, but perhaps its derivative as well. For the case where only boundary conditions are present, it is not required to use osculating interpolation theory as the Lagrange interpolating polynomial will suffice. For the Newton basis for the osculating polynomial, the function $\xi$ takes the form

$$\xi(x) = \xi[a] + \xi[y,a](x-a) + \xi[y,y,a](x-a)(x-y) + \xi[b,y,y,a](x-a)(x-y)^2. \tag{3.6}$$

This completes the construction of $\xi$. To construct $\gamma$, we are free to use the theory of osculating polynomials and Lagrange interpolation, we can borrow the idea in the previous section and assume $\gamma$ is a polynomial with roots at the boundary/initial point(s). However, as with $\xi$, we have to consider the first derivative of $\gamma$ as well. As such, we require that the root $y$ is a second-order root so that if $y$ is a root of $\gamma$, it is also a root of $\gamma'$. Thus, our construction of $\gamma$ takes the form

$$\gamma(x) = (x-a)(x-y)^2(x-b). \tag{3.7}$$

This completes the construction of $\gamma$. Now that the trial function is constructed, we can now define an appropriate cost function.

## 3.3 Higher-Order ODEs.

The above can be extended to higher-order ODEs. Let $p_0(x), \dots, p_m(x), f(x), g_0(x), \dots, g_{m-1}(x), h(x)$ be given $C^\infty(\Omega)$ functions. Consider the general $m$-th order IBVP

$$\begin{cases} p_m(x)\psi^{(m)}(x) + \cdots p_1(x)\psi(x) + p_0(x) = f(x), & x \in \Omega, \\ \psi(x) = h(x), & x \in \partial\Omega, \\ \psi^{(m-1)}(y) = g_{m-1}(x), \dots, \psi'(y) = g_1(y), \psi(y) = g_0(y), & y \in \Omega. \end{cases} \tag{3.8}$$

To construct the function $\xi$ for the general $m$-th order IBVP, we have to use the theory of osculating polynomials, as we require that $\xi$ and its $m-1$-th derivatives take on certain values at specific points. For the Newton basis for the osculating polynomial, the function $\xi$ takes the form

$$\xi(x) = \xi[a] + \xi[y,a](x-a) + \xi[y,y,a](x-a)(x-y) + \cdots + \xi[b,y,\dots,y,a](x-a)(x-y)^m. \tag{3.9}$$

This completes the construction of $\xi$. To construct $\gamma$, we use the same idea as the previous section. The function $\gamma$ takes the form

$$\gamma(x) = (x-a)(x-y)^{m-1}(x-b). \tag{3.10}$$

The root $y$ is a $m-1$-th order root so that the $m-1$-th derivative of $\gamma$ still has a root at $y$. With the trial function completed, we can appropriately define a cost function.

## 3.4 Example 1.

Consider the IVP

$$\begin{cases} \psi' = 1 + \frac{\psi}{x} + \left(\frac{\psi}{x}\right)^2, & 1 \le t \le 3, \\ \psi(1) = 0 \end{cases} \tag{3.11}$$

whose closed form solution is given by

$$\psi(x) = x\tan(\ln x). \tag{3.12}$$

We discretize the domain using a uniform spacing, of 0.1 units apart.

### 3.5 Example 2.

Consider the IVP

$$\begin{cases} \psi''(x) - 2\psi'(x) + \psi(x) = xe^x - x, \quad 0 \leq x \leq 1, \\ \psi(0) = \psi'(0) = 0. \end{cases} \tag{3.13}$$

The analytic solution is

$$\psi(x) = \tfrac{1}{6}x^3e^x - xe^x + 2e^x - x - 2, \;\; 0 \leq x \leq 1. \tag{3.14}$$

We discretize the domain using a standard grid discretization, where the grid points are spaced $h = 0.1$ apart.

## 4 Systems of Ordinary Differential Equations.

### 4.1 Example 1.

## 5 Linear Partial Differential Equations.

### 5.1 Example 1.

## 6 Non-Linear Partial Differential Equations.

### 6.1 Example 1.

## 7 Further Remarks.

## 8 References.