Neural Networks for Solving Differential Equations

Zhicheng Wu
Department of Electrical Engineering
Columbia University
zw2497@columbia.edu

Lingyi Cai
Department of Electrical Engineering
Columbia University
lc3352@columbia.edu

Xinlang Yue
Department of Applied Physics and Applied Mathematics
Columbia University
xy2383@columbia.edu

Abstract

Different methods are developed to solve partial differential equations (PDE). Classical numerical methods may omit information in approximation with unstable convergence and path-dependent problems. In this paper, we proposed a scheme of stable neural network (NN) framework that provides closely analytical solutions compared to traditional numerical results. Firstly, we presented mathematical deduction of NN model; Secondly we processed complex boundary, high dimensional PDE problems. We also showed training performance of activation functions and proposed implementation of "frequency principle" as an alternative in algorithm improvement.

1 Introduction

Successful applications of neural networks (NN) in different scientific fields prompt attempts in solving general types of differential equations since the latter are widely used in financial area and classical computational physics[1], for example, pricing model using Black-Scholes equations, Hamilton-Jacobi-Bellman Equation in optimal control, etc. Classical methods are generally challenged by numerical precision, convergence, and computational cost.

One of the traditional ways to solve these equations are based on Monte Carlo simulation[2], which randomizes the problem by presenting the average. It will lose information either in iteration or simulation because of randomness. Another category of approaches is numerical approximation to solve PDE. It relies heavily on the equation problem model with given condition to satisfy. If the model system is not stable enough, subtle changes will lead to completely different numerical estimates.

Therefore, generalized and stable model for solving PDE is required: NN. The intuition [3] built on NN solver comes from the connection between functions and their derivatives. Generally, the derivative of a function, if it exists, represents the strength of connections between different hidden neurons if expressed by NN. Therefore, if the function can be decomposed by NN, in terms of its gradient, the discretization will approximate the function based on information given in differential equations.

In this paper, we proposed a generalized NN model with the help of TensorFlow to process PDE problems with complex boundary information and presented visualized result of "frequency principles" based on our framework.

2 Related Works

Higher dimensional PDE have been widely applied in the field of engineering, finance[1] [4]. Their numerical simulation and results have constantly been a challenge. As a comparison, the existence of result in mathematical view in many cases is ensured by different theorems, but very few numerical methods for deriving solutions in explicit and closed form are not unknown. Some of the current approaches are presented as follows: Shooting Method, Finite Difference Method, Finite Element Method, Spline Based Method and NN Method, etc. In short, these are signs in the evolution of numerical methods due to the limit of analytical solutions.

Therefore, existing works on solving PDE are generally divided into numerical simulation and deep learning algorithms that reimplement or combine with classical methods.

2.1 Shooting Method

A representative of numerical methods is shooting method[5]. It basically modifies the original form of the target equation with boundary conditions by transforming the system into two initial value problems and adding sufficient amount of conditions to tuning the given conditions until they are well satisfied with the given setup.

For example, consider a problem with boundary condition value like this:

$$y" = u(t, y, y') \quad \text{with} y(a) = \alpha, y(b) = \beta$$

Follow the steps above, we make an appropriate guess of $y'(a)$, then integrate the differential equation and measure the difference between the result and $y(b) = \beta$ until the difference is under a given tolerance.

As we see through the method's procedure, the drawbacks[6] are that it highly depends on the integrability of the target equation and the initial guess since the equation may "blow up" before the problem can be integrated and initial guess on derivative can lead to completely different iteration within two boundaries. In sum, the method gives an excess of limits on the problem.

2.2 Finite Element Method

Finite element method is powerful compared to other numerical difference strategy. It represents target functions in terms of selected basis functions and solves the equation in weak integral form to avoid drawbacks we mentioned above.

For example, consider the following boundary value problem:

$$u" = u + f(x) \quad 0 \leq x \leq 1$$

with

$$u(0) = 0, \quad u(1) = 1$$

The goal is to choose piecewise smooth polynomial basis to represent the exact solution $u(x)$:

$$u(x) = \sum_{i}^{N} \alpha_i \phi_i(x)$$

where ϕ_i are specified basis functions and α_i are (unknown) corresponding constants. And the equation will then be written in weak form:

$$\left(\sum_{i=1}^{N} \alpha_i \phi_i', \phi_j' \right) + \left(\sum_{i=1}^{N} \alpha_i \phi_i, \phi_j \right) + (f, \phi_j) = 0$$

to obtain coefficients. And this approach has weakness[7] in that it depends on the choice of basis functions for reference elements and loses information in approximation since its simulation may highly fluctuate around the solution.

3 System Model and Problem Formulation

3.1 Taylor Series Approximation

According to the universal approximation theorem [8], feed-forward NN can approximate any (at least) continuous function with a single hidden layer. Then it reminds us to combine NN with Taylor expansion to express the value of a function at a given point x_0:

Assume that $u(\mathbf{x}) \in C^1(\Omega)$, where $\Omega \in R^n, n \geq 1$ is the bounded domain for $\mathbf{x}$.

Then we can write:

$$\begin{aligned} u(\mathbf{x}) &= u(\mathbf{x}_0) + Du(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0) + \epsilon \\ &= u(\mathbf{x}_0) + \left(\frac{\partial u}{\partial \mathbf{x}} |_{\mathbf{x}=\mathrm{x}_0} \right)^T \cdot (\mathbf{x} - \mathbf{x}_0) + \epsilon \\ &= u(\mathbf{x}_0) + \sum_{i=1}^{n} \frac{\partial u(\mathbf{x}_0)}{\partial x_i} \cdot (x_i - x_{0_i}) + \epsilon \end{aligned} \tag{1}$$

where ϵ is the truncated taylor expansion residue of $u(\mathbf{x})$.

Now, we consider to construct the model in terms of the given general form of differential equation listed below:

$$G\left(\vec{\mathbf{x}}, \nabla u(\vec{\mathbf{x}}), \nabla^2 u(\vec{\mathbf{x}}), \cdots, \nabla^m u(\vec{\mathbf{x}})\right) = 0 \tag{2}$$

where $\mathbf{x} \in \Omega, \Omega$ bounded, $\Omega \subseteq R^n, m, n \geq 1$

3.2 Setup and Function Fitting

For n dimensional scalar-valued functions, we can build a single layer NN based on matrix operation form to absorb expanded items in the above equations and measure the error ϵ.

Then we write it as :

$$N(\mathbf{x}; w) = W_2 \sigma(W_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \tag{3}$$

where W_1, W_2 are weight matrices, $\mathbf{b}_1, \mathbf{b}_2$ are bias vectors, and σ is a given activator.

Let $w = (W_1, W_2, \mathbf{b}_1, \mathbf{b}_2)$ for $N(\mathbf{x}, w)$. And it is clear that for the above scalar $u(\mathbf{x})$, will take all n scalar inputs in $R^n, n \geq 1$ and transform into an scalar for $N(\mathbf{x}; w)$.

Intuitively, parameter pairs in w can be optimized by minimizing the loss in L^2 norm:

$$L(w) = \|u(\mathbf{x}) - N(\mathbf{x}; w)\|_2 \quad \mathbf{x} \in \Omega \tag{4}$$

Without loss of generality, we assume $\Omega \subseteq R^1$:

$$\begin{aligned} L(w) &= \|u(\mathbf{x}) - N(\mathbf{x}; w)\|_2, \quad \mathbf{x} = x \in [a, b] = \Omega \\ &= \left(\int_a^b [u(x) - N(x; w)]^2 dx \right)^{\frac{1}{2}} \\ &= \left(\sum_i [u(x_i) - N(x_i; w)]^2 \right)^{\frac{1}{2}} \end{aligned} \tag{5}$$

where $\{x_i\}$ is the set of discretized training points over the interval $[a, b]$.

Naturally, if the loss $L(w)$ above is under given tolerance, then $N(x; w)$ well approximates the target function:

$$N(x; w) \approx u(x) \tag{6}$$

3.3 Fundamentals of Approximation Manner

Question How do Taylor series work in approximating functions combined with (3)?

Recall: Let $u(x) \in C^{N+1}[a,b]$ and $x_0 \in [a,b] \subseteq R^1$, then for all $x \in (a,b)$ there exists a number $c = c(x)$ that lies between x_0 and x such that

$$u(x) = T_N(x) + R_N(x) \tag{7}$$

where $T_N(x)$ is the Taylor polynomial approximation

$$T_N(x) = \sum_{n=0}^{N} \frac{f^{(n)}(x_0) \cdot (x - x_0)^n}{n!} \tag{8}$$

and $R_N(x)$ is the residual (the part of the series we left off)

$$R_N(x) = \frac{f^{(n+1)}(c) \cdot (x - x_0)^{n+1}}{(n+1)!} = \epsilon \tag{9}$$

This in some degree shows a fixed relation between the target function and taylor expansion coefficients. But, for the same variable x, we can modify the relations between coefficients of powers of x.

For example, $u(x) = e^x$ with $x_0 = 0$:

$$\begin{aligned} e^x &= \sum_{n=0}^{N} e^0 \frac{x^n}{n!} = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots \\ e^{2x} &= \sum_{n=0}^{N} e^0 \frac{(2x)^n}{n!} = 1 + 2x + \frac{2^2}{2!}x^2 + \frac{2^3}{3!}x^3 + \cdots \end{aligned} \tag{10}$$

The expansion relations are fixed for each power of x, but

$$e^x + e^{2x} = 2 + 3x + \frac{1+2^2}{2!}x^2 + \frac{1+2^3}{3!}x^3 + \cdots \tag{11}$$

The relations for powers are modified as corresponding coefficients are changed.

And what are the coefficients here for (3)? The weights pairs: w.

3.4 Single ODE illustration

We first consider the NN model for lower (one) dimensional ODE and rewrite (2) as:

$$\begin{aligned} u'(x) &= F(x, u(x)) \\ u(x_0) &= u_0 \qquad\qquad\qquad \text{in } \Omega \subset R^1 \end{aligned} \tag{12}$$

And we find that if we plug in $(*)$ directly, the given initial condition cannot be satisfied, since the y-axis value will fluctuate:

$$N(x_0; w) \neq u_0 \tag{13}$$

So we force the initial condition in (12) to get:

$$\overline{u}(x; w) = u_0 + (x - x_0) \cdot N(x; w) = \tilde{N}(x; w) \tag{14}$$

and it is clear that $\overline{u} = u_0$ always holds for any w.

Now we and plug in (14) and rewrite (12) as:

$$\overline{u}'(x; w) \approx F(\overline{u}(x; w), x) = \tilde{N}'(x; w) \tag{15}$$

$\Rightarrow$

$$\begin{aligned} \overline{u}' &= \frac{d[u_0 + (x - x_0) \cdot N(x; w)]}{dt} \\ &= 0 + \frac{d(x - x_0)}{dx} \cdot N(x; w) + (x - x_0) \cdot \frac{d(N(x; w))}{dt} \\ &= N(x; w) + (x - x_0) \cdot N'(x; w) \end{aligned} \tag{16}$$

Similarly, we can obtain optimal parameter pairs in w by minimizing the loss in L^2 norm:

$$\begin{aligned} L(w) &= \|\overline{u}'(x;w) - \tilde{N}'(x;w)\|_2 \quad x \in \Omega \\ &\approx \sum_i [\overline{u}'(x_i;w) - F(\overline{u}(x_i;w), x_i)]^2 \end{aligned} \tag{17}$$

where x_i is the set of discretized training points over the interval $[x_0, x_n] \in \Omega$.

Also if the loss $L(w)$ above is under given tolerance, then $\tilde{N}(x;w)$ may well approximate the analytical solution in (12):

$$\overline{u}(x;w) \approx u(x)_{true} \tag{18}$$

3.5 Trial Solution Form

The above implementation reminds us that through the construction in (14), the NN should take into account of all boundary conditions (BC) to express information given in differential equations.

Basically, this can be achieved by truncating the Taylor expansion of the target function at fixed boundaries to satisfy all BC and absorb all unknown parameters independent of BC in the NN. Then the solution is inspired by classical method of trial solution for solving differential equations.

So for$\vec{\mathbf{x}} \in R^n$, we write our trial solution $u_t(\vec{\mathbf{x}})$ as:

$$u_t(\vec{\mathbf{x}}) = A(\vec{\mathbf{x}}) + F(\vec{\mathbf{x}}, N(\vec{\mathbf{x}}, w)) \tag{19}$$

where $N(\vec{\mathbf{x}}, w)$ is the single-output NN defined in (3) above, $A(\vec{\mathbf{x}})$ contains all BC, and $F(\vec{\mathbf{x}}, N)$ contains information independent of BC and deals with minimization problems for optimal w.

(Remark: we only discuss Dirichlet BC in our model)

In arbitrarily complex cases of BC and higher dimensions, the definition for (19) becomes difficult compared to (16). So we have to force some requirements for F and introduce a distance measurement indicator to construct the term F:

$$F(\vec{\mathbf{x}}, N) = N \cdot M_D(\vec{\mathbf{x}}) \tag{20}$$

Definition: the function M_D is an indicator of distance from the boundary in any form of (2) such that:

$$F(\vec{\mathbf{x}}, N) = \begin{cases} N \cdot M_D(\vec{\mathbf{x}}) & \mathbf{x} \in \Omega \setminus \partial\Omega \\ 0 & \mathbf{x} \in \partial\Omega \end{cases} \tag{21}$$

Then the model of NN can be trained in the domain through $N(\vec{\mathbf{x}}, w)$. As we can see, the the indicator M_D doesn't critically have to be the same, it may be distance functions in any form depending on expression, for example, perpendicular distance from $\mathbf{x}$ to positions on the boundary will work for most convex and non-convex cases.

3.6 Implementation with Examples

Apart from the examples given in (12), we show several other NN form in differential equations.

For scalar functions, the NN defines a mapping $R^n \to R^1$.

3.6.1 Single second order ODE with initial value problem (IVP)

$$\begin{aligned} u''(x) &= F\left(x, u'(x), u(x)\right) \\ \text{with} \quad u(x_0) &= u_0, \quad u'(x_0) = u_0' \quad \text{in } \Omega \subset R^1 \end{aligned} \tag{22}$$

The trial solution $u_t(x)$ then becomes[3]:

$$\begin{aligned} u_t(x) &= A(x) + F(x, N(x,w)) \\ &= \underbrace{u_0 + \frac{u_0'}{1!} \cdot (x - x_0)}_{A(x)} + \underbrace{\sum_{n=2}^{N} \frac{f^{(n)}(x_0) \cdot (x - x_0)^n}{n!} + \epsilon}_{F(x, N(x,w))} \\ &= u_0 + u_0' \cdot (x - x_0) + (x - x_0)^2 \cdot N(x;w) \end{aligned} \tag{23}$$

And optimal parameter pairs in w is presented in the following equation:

$$\begin{aligned}\min L(w) &= \|u''(x) - \tilde{N}(x;w)\|_2, \quad x \in \Omega \\ &= \|u''(x) - F\left(x, u'(x), u(x)\right)\|_2 \\ &= \left\{\sum_i [\frac{d^2u(x_i)}{dx^2} - F\left(x_i, u'(x_i), u(x_i)\right)]^2\right\}^{\frac{1}{2}}\end{aligned} \tag{24}$$

and

$$w = \arg\min L(w) \tag{25}$$

3.6.2 Single second order ODE with boundary value problem (BVP)

$$\begin{aligned}u''(x) &= F\left(x, u'(x), u(x)\right) \\ \text{with} \quad u(x_0) = u_0, &\quad u(x_N) = u_1 \quad \text{in } [x_0, x_N] = \Omega \subset R^1\end{aligned} \tag{26}$$

The trial solution $u_t(x)$ then becomes:

$$\begin{aligned}u_t(x) &= A(x) + F(x, N(x,w)) \\ &= \begin{Bmatrix} u_0, & x = x_0 \\ u_N, & x = x_N \end{Bmatrix} + N(x;w) \cdot M_D(x_0, x_N, x) \\ &= u_0(x_N - x) + u_N(x - x_0) + (x - x_0)(x_N - x) \cdot N(x;w)\end{aligned} \tag{27}$$

And optimal parameter pairs in w is presented in the following equation:

$$\begin{aligned}\min L(w) &= \|u''(x) - \tilde{N}(x;w)\|_2, \quad x \in \Omega \\ &= \left\{\sum_i [\frac{d^2u(x_i)}{dx^2} - F\left(x_i, u'(x_i), u(x_i)\right)]^2\right\}^{\frac{1}{2}}\end{aligned} \tag{28}$$

and

$$w = \arg\min L(w) \tag{29}$$

which is the same as in 3.6.1.

3.6.3 Single second order PDE with BVP in R^2

For illustration, we show for Poisson equation in R^2:

$$-\Delta u(x,y) = f(x,y) \tag{30}$$

with BC

$$\begin{cases} u(x_0, y) &= f_0(y) \\ u(x_N, y) &= f_1(y) \\ u(x, y_0) &= g_0(x) \\ u(x, y_N) &= g_1(x) \end{cases} \tag{31}$$

in $x, y \in [x_0, x_N] \times [y_0, y_N]$

The trial solution $u_t(x)$ then becomes:

$$\begin{aligned}u_t(x,y) &= A(x,y) + F(\mathbf{x}, N(\mathbf{x}, w)) \\ &= A(x,y) + N(\mathbf{x};w) \cdot M_D(x_0, x_N, y_0, y_N, x, y) \\ &= A(x,y) + (x - x_0)(x_N - x)(y - y_0)(y_N - y) \cdot N(\mathbf{x};w)\end{aligned} \tag{32}$$

where $A(x,y)$ is by definition,

$$\begin{aligned}A(x,y) &= (x_N - x)f_0(y) + (x - x_0)f_1(y) \\ &+ (y_N - y)\{g_0(x) - [(x_N - x)g_0(x_0) + (x - x_0)g_0(x_1)]\} \\ &+ (y - y_0)\{g_1(x) - [(x_N - x)g_1(0) + (x - x_0)g_1(x_N)]\}\end{aligned} \tag{33}$$

And optimal parameter pairs in w is presented in the following equation:

$$\begin{aligned} \min L(w) &= \|-\Delta u(\mathbf{x}) - \tilde{N}(\mathbf{x}; w)\|_2, \quad x \in \Omega \\ &= \left\{ \sum_i [-\Delta u(x_i, y_i) - f(x_i, y_i)]^2 \right\}^{\frac{1}{2}} \end{aligned} \tag{34}$$

for $x_i, y_i \in [x_0, x_N] \times [y_0, y_N]$ and

$$w = \arg\min L(w) \tag{35}$$

4 Neural Networks for Solving Differential Equations Experiment

There are various applications to images, sound, video, sequential actions processing. Neural networks are used for solving many specific problems such as image processing, character recognition and forecasting. In this section, neural networks will be applied to three kinds of differential equations: ODEs, second order ODEs and PDEs. Our code are based on TensorFlow 2.0 in our experiments.

4.1 First Order Ordinary Differential Equation

In the first experiment, we started with simple ODE in the form of:

$$\frac{d\Psi(x)}{dx} = f(x, \Psi) \tag{36}$$

According to I.E. Lagaris' team's work [3], we can get the minimization problem:

$$min \sum_{\vec{x_i} \in \hat{D}} G(\vec{x_i}, \Psi_t(\vec{x_i}, \vec{p}), \bigtriangledown \Psi(\vec{x_i}, \vec{p}), \bigtriangledown^2 \Psi(\vec{x_i}, \vec{p}))^2 \tag{37}$$

In the proposed approach, we can have the trial solution of this problem:

$$\Psi_t(\vec{x}) = A(\vec{x}) + F(\vec{x}, N(\vec{x}, \vec{p})) \tag{38}$$

We can get the optimization objective:

$$E[\vec{p}] = \sum_i \left\{ \frac{d\Psi_t(x_i)}{dx} - f(x_i, \Psi_t(x_i)) \right\}^2 \tag{39}$$

Then we try to solve the following problem:

$$\frac{d}{dx}\Psi + (x + \frac{1+3x^2}{1+x+x^3})\Psi = x^3 + 2x + x^2 \frac{1+3x^2}{1+x+x^3} \quad \Psi(0) = 1 \tag{40}$$

Trail solution:

$$\Psi_t(x) = 1 + xN(W, x) \tag{41}$$

We built a two-layer neural network and used ReLU activation function in the first layer. By comparing the analytical solution, iteration solution and Neural network solution in Figure 1, we found that the neural network solution is closer to the analytical solution than the iteration solution. Then we can get the conclusion: neural networks perform better than iteration solution when solving ODE problem.

4.2 Second Order Ordinary Differential Equation

In the second experiment, we go further and extend our solution to second order ODE in the form of:

$$\frac{d^2\Psi(x)}{dx^2} = f(x, \Psi, \frac{d\Psi}{dx}) \tag{42}$$

We can get the second order ODE as Dirichlet problem trial solution:

$$\Psi_t(x) = A(1-x) + Bx + x(1-x)N(x, \vec{p}) \tag{43}$$

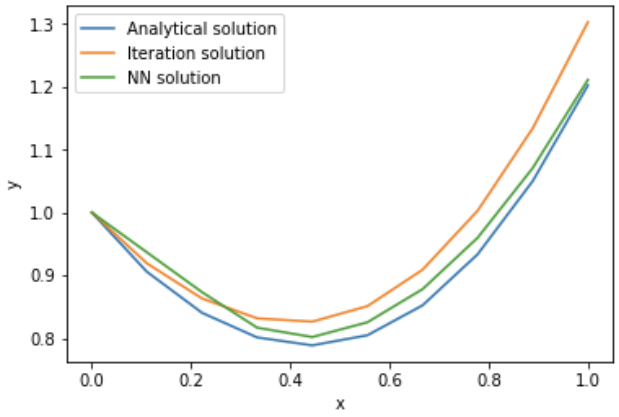

Figure 1: ODE solutions

Then we try to solve the second order ODE as the following equation:

$$\frac{d^2}{dx^2}\Psi + \frac{1}{5}\frac{d}{dx}\Psi + \Psi = -\frac{1}{5}e^{-\frac{x}{5}}cos(x) \tag{44}$$

$$\Psi(0) = 0 \quad and \quad \frac{d}{dx}\Psi(0) = 1 \, with \, x \in [0, 2] \tag{45}$$

The trial solution can be defined as the following form:

$$\Psi_t(x) = x + x^2 N(W, x) \tag{46}$$

We built a three-layer neural network, at the beginning, we use ReLU activation function in the first two layers and tanh activation function was only used in the last layer in model 1. With the comparison of different kinds of activation functions (activation functions comparison will be discussed in section), we changed the ReLU activation function to tanh activation function in model 2. Both of models were trained with 1000 epochs and the results are shown in figure 2.

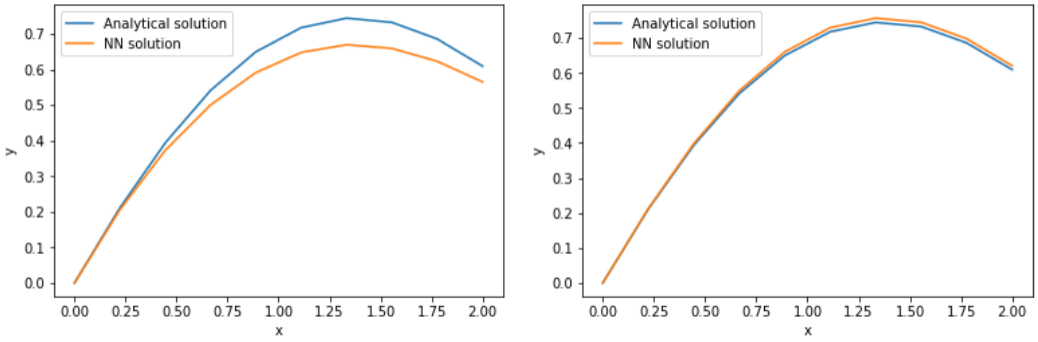

Figure 2: Second order ODE solutions, model 1(left) and model 2(right)

Furthermore, another second order ODE was also computed by using neural network:

$$-\Delta u(x) = g(x), \quad x \in \Omega = (-1, 1) \tag{47}$$

$$u(-1) = u(1) = 0 \tag{48}$$

$$g(x) = sin(x) + 4sin(4x) - 8sin(8x) + 16sin(24x) \tag{49}$$

Firstly, shooting method, which is a method for solving BVP mentioned earlier in 2.1 by reducing equation to a system of IVP, provided a solution to this equation. Secondly, neural network was applied.

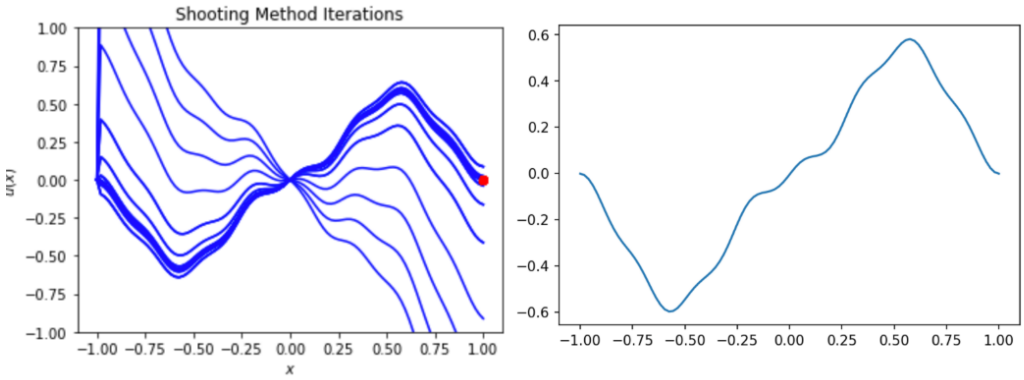

Figure 3: Second order ODE solutions 2

4.3 Partial Differential Equation

In the third experiment, we applied neural networks to PDE, which has the following form:

$$\frac{\partial^2}{\partial x^2}\Psi + \frac{\partial^2}{\partial y^2}\Psi(x,y) = f(x,y) \tag{50}$$

In this PDE problem, according to I.E. Lagaris' team's work [3], trail solution has the following form:

$$\Psi_t(x,y) = A(x,y) + x(1-x)y(1-y)N(x,y,\vec{p}) \tag{51}$$

Optimization objective has the following form:

$$E[\vec{p}] = \sum_i \left\{ \frac{\partial^2}{\partial x^2}\Psi(x_i,y_i) + \frac{\partial^2}{\partial y^2}\Psi(x_i,y_i) - f(x_i,y_i) \right\}^2 \tag{52}$$

Then we solved the Laplace equation from previous work [9]:

$$\nabla^2\Psi(x) = 0, \forall x \in D \tag{53}$$

With the following BCs:

$$\Psi(x) = 0, \forall x \in \{(x_1,x_2) \in \partial D \,|\, x_1 = 0, x_1 = 1, or\, x_2 = 0\} \tag{54}$$

$$\Psi(x) = sin\pi x_1,\ \forall x \in \{(x_1,x_2) \in \partial D \,|\, x_2 = 1\} \tag{55}$$

And then, trial solution can be computed:

$$\Psi_t(x,v,\vec{W}) = x_2 sin\pi x_1 + x_1(x_1-1)x_2(x_2-1)N(x,v,\vec{W}) \tag{56}$$

Five-layer neural network was trained for 1000 epochs and tanh activation was used in this model as shown in Figure 4. As shown in Figure 5, neural networks perform well in partial differential equation problem.

From the experiments above, there are many advantages of using neural networks in solving differential equations. Firstly, the solution via NN's is a differentiable, closed analytic form easily used in any subsequent calculation compared to discretization, limited differentiability in other methods. Secondly, neural networks provides a solution with generalized property. Thirdly, number of sampling points does not affect computational complexity drastically, amount of model parameters are far less than those of other solution technique and therefore, with quite low demand on memory space. Fourthly, the method can be generalized to ODEs, systems of ODEs and to PDEs as well even with irregular BC. Last but not least, the NN model can also be efficiently operated on parallel architectures.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, None, 512)         1536
_________________________________________________________________
dense_15 (Dense)             (None, None, 128)         65664
_________________________________________________________________
dense_16 (Dense)             (None, None, 64)          8256
_________________________________________________________________
dense_17 (Dense)             (None, None, 32)          2080
_________________________________________________________________
dense_18 (Dense)             (None, None, 16)          528
_________________________________________________________________
dense_19 (Dense)             (None, None, 1)           17
=================================================================
Total params: 78,081
Trainable params: 78,081
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Model summary

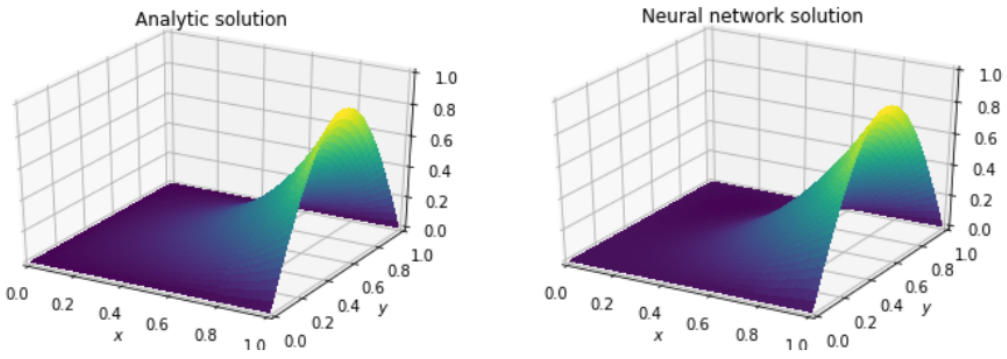

Figure 5: PDE solutions

5 Comparisons of Different Activation Functions

In this section, we will compare four different activation functions,ReLU, softplus, sigmoid and tanh. In neural networks, the activation function as shown in Figure 6 is responsible for mapping the input of neurons to the output. Without the activation function, each input of next layer is a linear combination of the previous output. No matter how many layers in the neural network, the output is linearly related to the inputs, this is the most primitive perception. If we use specific activation like max, they brings in nonlinear factors to the network. Actually, most of the problem we need to solve is not linear. These nonlinear activations allows the network to fit nonlinear function, so that these complicate problem like image and speech can be solved.

5.1 ReLU Activation Function

ReLU activation function is widely used in neural networks and it is much easier and efficient to compute ReLU. First, when we use s-shaped function to calculate the activation function and the back propagation is used to find the error gradient, there exists problem of big calculation. ReLU can help us reduce the computation cost. Second, ReLU will make a part of outputs equal zeros, which will

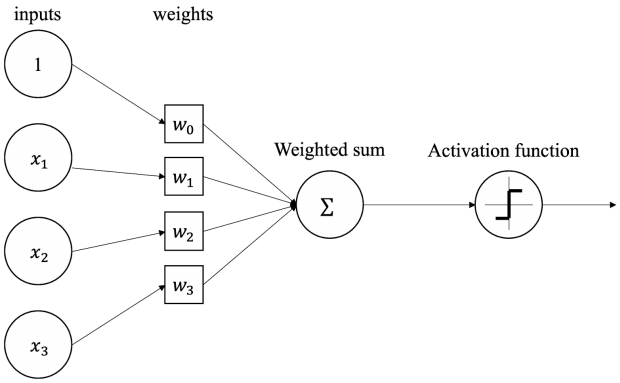

Figure 6: Activation function in neural network

cause the sparseness of the network, and reduce the interdependence of parameters. ReLU alleviates the occurrence of overfitting problem. Neural network solution with ReLU activation function drawn in the following Figure 7 and shown by the equation:

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{57}$$

The derivative equation of the ReLU activation function:

$$f'(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{58}$$

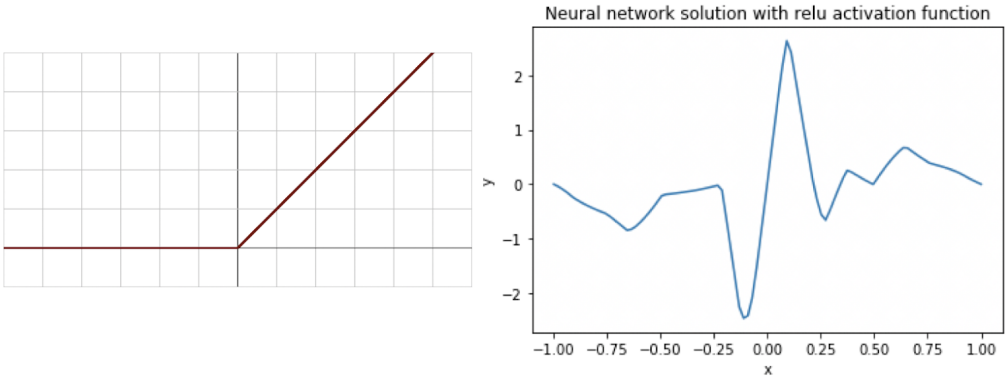

Figure 7: NN solution with ReLU activation function

5.2 Softplus Activation Function

Both ReLU and softplus are largely similar, except near zero where the softplus is smooth and differentiable which is defined by the formula:

$$f(x) = ln(1 + e^x) \tag{59}$$

The derivative of softplus function:

$$f'(x) = \frac{1}{1 + e^{-x}} \tag{60}$$

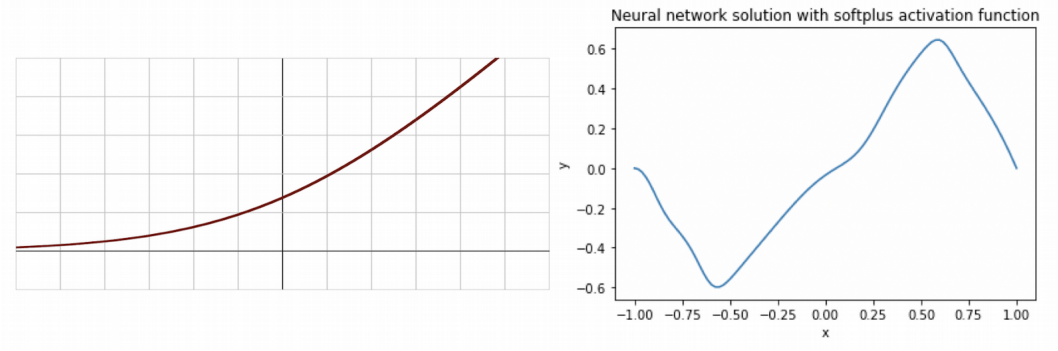

Figure 8: NN solution with softplus activation function

5.3 Sigmoid Activation Function

Sigmoid function has a well S shape. It refers to special case of the logistic function shown in the Figure 9 below and shown by the formula:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{61}$$

The derivative of sigmoid activation:

$$f'(x) = f(x)(1 - f(x)) \tag{62}$$

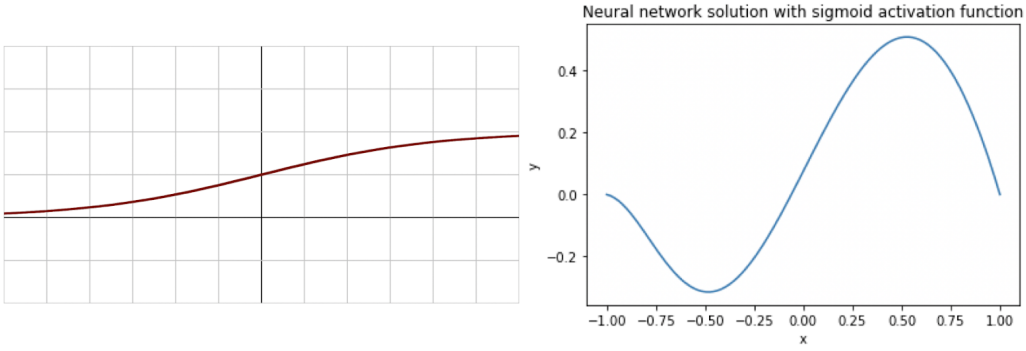

Figure 9: NN solution with sigmoid activation function

5.4 Tanh Activation Function

Tanh function acts like sigmoid but it is better than sigmoid. Tanh function has a s shape. The range of it is from -1 to 1. The tanh function squeezes the input value to a range of -1 to 1, so it's mean is equal to zero, which solves the non zero-centered problem of the sigmoid function. In the tanh graph, the negative value will be mapped to negative far away from zero and the zero inputs will be mapped close to zero in the tanh graph. In addition, tanh function is differentiable. From the Figure 10 below, we can find that tanh function performs well and the function is and defined by the formula:

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{63}$$

The derivative of tanh activation:

$$f'(x) = 1 - f(x)^2 \tag{64}$$

From the figures above, we find that ReLU activation function cannot be used in second order ODE, the second derivative of ReLU function will always be zero. Softplus, sigmoid activation functions

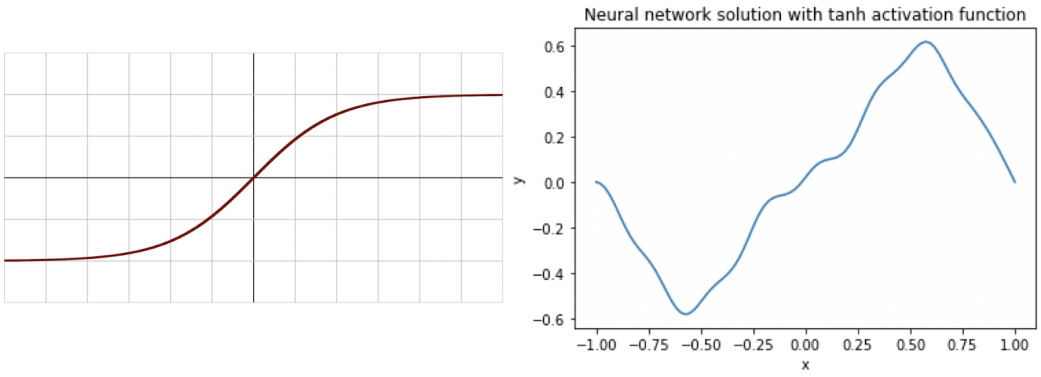

Figure 10: NN solution with tanh activation function

provide us similar shapes with analytical solution because both of them are differentiable and the second derivative of them will not always be zero. Beyond that, tanh activation function performs best in second order ODE problem which is almost the same as analytical solution.

6 F-principle Verification

Frequency principle (F-principle) was proposed by Zhi-Qin John Xu in 2018 [10], they indicate DNNs initialized with small parameters usually fit target functions from low to high frequencies.

As an example, we trained a deep learning neural networks (DNN) to recognize a picture. The input to the DNN is a position coordinate (x, y), and we want it to output the gray value corresponding to that position. In Figure 11, the series of diagrams below shows the different training steps, the images learned by DNN from the rough outline to the details in the learning process which is different from computational mathematics. In computational mathematics, many iterative formats first converge high frequencies, such as the Jacobi method and the Gauss-Seidel method.

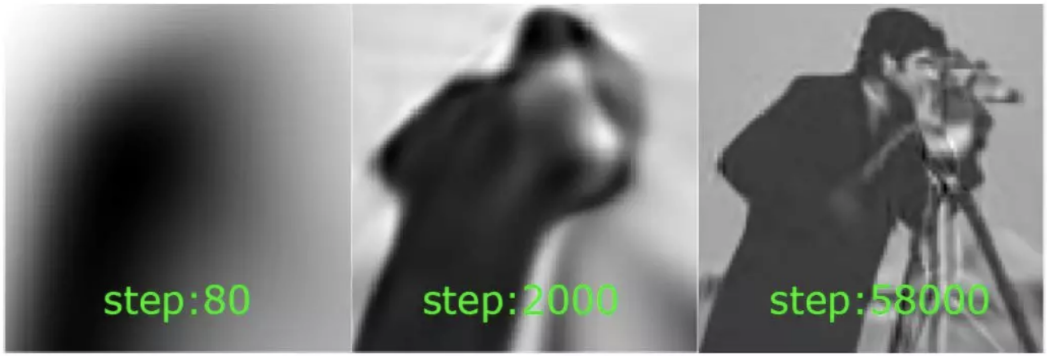

Figure 11: Images learned by DNN in different steps

In this section, as shown in Figure 12, we applied frequency domain analysis on the second order ODE. The input to the DNN is x. We used the output of the DNN to approximate the true solution $u(x)$. The loss function was the energy functional of the Poisson equation and the process of the specific solution can be found in the paper. As shown in the Figure 12 above, neural networks work on low frequencies first from 0 to 300 epochs and then work on the high frequencies from 400 to 900 epochs. In Figure 13, we can observe low frequencies converge first from zero to 60 epochs. The F-Principle in neural networks is very obvious, the low frequency converges much faster than the high frequency.

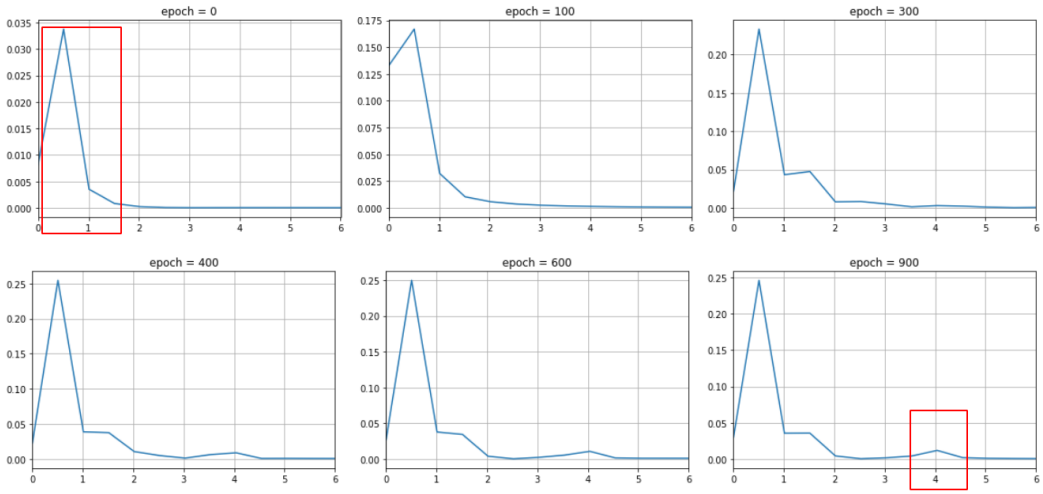

Figure 12: Frequency domain analysis (X axis: Fourier transform index, Y axis: Amplitude)

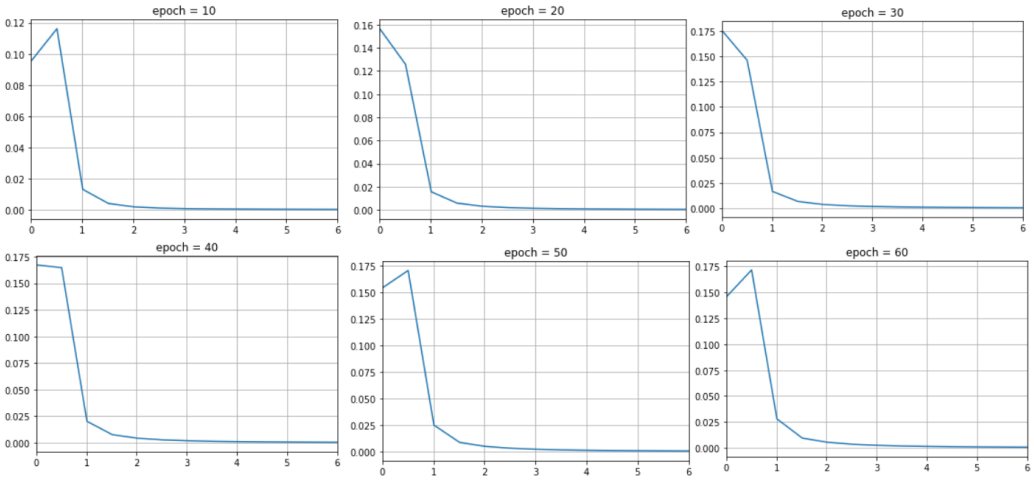

Figure 13: Frequency domain analysis, from 0 epoch to 60 epochs

7 Conclusion

In this project, firstly, we have solved ODE, second order ODE, PDE by applying neural network framework which provides closely analytical solutions. Secondly, we compared different activation functions and found that tahn activation function has good performance in second order ODE. We also verified F-principle in second order ODE. In the future, we will generalize our approach in solving high-dimensional PDE of higher orders (>2) and analyze problems with arbitrary boundary conditions, including mixed Neumann BC. In addition, we will also design more efficient algorithms to improve stability when cases of ill-conditioned analytical solution occur and extend the method on issues with similar nature using differential operators, such as, variational problems, flow problems in fluid mechanics.

References

[1] Jiequn Han, Arnulf Jentzen, and E Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[2] William Edmund Milne and WE Milne. *Numerical solution of differential equations*, volume 19. Wiley New York, 1953.

[3] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[4] Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe Von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv preprint arXiv:1809.02362*, 2018.

[5] Andrzej Granas, Ronald Guenther, and John Lee. Nonlinear boundary value problems for ordinary differential equations. 1985.

[6] David D Morrison, James D Riley, and John F Zancanaro. Multiple shooting method for two-point boundary value problems. *Communications of the ACM*, 5(12):613–614, 1962.

[7] Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

[8] Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48, 2001.

[9] MM Chiaramonte and M Kiener. Solving differential equations using neural networks. *Machine Learning Project*, 2013.

[10] Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *arXiv preprint arXiv:1901.06523*, 2019.