Independent Study: Initial Exploration

Matthieu Meeus

February 2020

1 Introduction

This document marks the beginning of the independent research study in Spring 2020, under supervision of Dr. Sondak, Dr. Protopapas and Dr. Mattheakis.

The goal of the overall project is to experiment with different neural network architectures (with an initial focus on activation functions) to solve partial differential equations (PDEs) in an unsupervised, data-free way.

This document focuses on getting familiar with the problem statement. First, a specific two-dimensional PDE is formulated on a circular domain. Through separation of variables, the analytical solution is computed. Note that last week, we attempted to solve the non-homogeneous Poisson equation, which appeared to be to hard for the application. Therefore, we now focus on the homogeneous Laplace equation. The general form of this solution inspires us to implement a similar mathematical procedure in our neural network.

Next, a simple neural network architecture is being examined mathematically. The mathematical formulation of a regular neural network with two inputs, an arbitrary number of nodes in one layer and a sinusoidal activation function is derived. This approach is then extended for two layers.

Lastly, and this is entirely new for this week, a simple ordinary differential equation is solved with a neural network in Pytorch.

2 Problem formulation and analytical solution

2.1 Problem formulation

As an example PDE, we will solve the two-dimensional, homogeneous Laplace equation on a circular plate. Note that because of the circular dimension, polar coordinates will be used in the analysis. Figure 1 below illustrates the domain that will be considered.

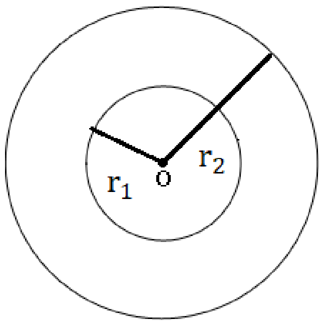

Figure 1: The considered circular domain for the PDE

The general Poisson equation is equal to:

$$\nabla^2 u(r,\theta) = 0 \tag{1}$$

For polar coordinates in particular, the Poisson operator has the following mathematical form:

$$\nabla^2 u(r,\theta) = \frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial u}{\partial r}) + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} \tag{2}$$

For this problem there will be a homogeneous Dirichlet boundary condition for $r = r_1$ and a non-homogeneous Neumann boundary condition for $r = r_2$, or:

$$u(r = r_1, \theta) = 0 \tag{3}$$

$$\frac{\partial u}{\partial r}\Big|_{r=r_2} = g(\theta) \tag{4}$$

Note that the circular geometry requires the solution to be periodic in θ, which will also play a crucial part in the analytical solution.

The section below will attempt to find the analytical solution of equation (1) with the specified definition of the Laplace operator (2) with boundary conditions (3) and (4).

2.2 Analytical solution through separation of variables

The general method that we will use is the separation of variables, meaning that we could write the solution of the PDE $u(r,\theta)$ as a product of decoupled functions in its two variables r and θ. Or:

$$u(r,\theta) = R(r)T(\theta) \tag{5}$$

Let's plug this formulation into the polar formulation of equation (1):

$$\nabla^2 u(r,\theta) = \frac{1}{r}\frac{\partial u}{\partial r}(r\frac{\partial u}{\partial r}) + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} = \frac{T}{r}\frac{d}{dr}(r\frac{dR}{dr}) + \frac{R}{r^2}\frac{d^2T}{d\theta^2} = 0 \tag{6}$$

Which leads to:

$$T\frac{d^2R}{dr^2} + \frac{1}{r}\frac{dR}{dr}T + \frac{1}{r^2}\frac{d^2T}{d\theta^2} = 0 \tag{7}$$

$$\frac{r^2R'' + rR'}{R} + \frac{T''}{T} = 0 \tag{8}$$

As the first term is a function of only r, the second of only θ and they both sum up to zero, they have to be equal to a real number. Or:

$$\frac{r^2R'' + rR'}{R} = \lambda = -\frac{T''}{T} \tag{9}$$

Note that through the separation of variables, we are able to decouple the PDE into two ODE's in the respective variables r and θ.

Let's start solving for $T(\theta)$, or:

$$\frac{1}{T}\frac{d^2T}{d\theta^2} = -\lambda \tag{10}$$

For our solution to make physical sense, we need $T(\theta)$ to be periodic, or that $T(\theta) = T(\theta + 2\pi)$ and $T(\theta) = T(\theta + 2\pi)$ for all θ. From online resources (link), we know that the general solution for this is equal to the following:

$$T_0(\theta) = A_0 j = 0 \tag{11}$$

$$T_j(\theta) = A_j cos(\sqrt{\lambda_j}\theta) + B_j sin(\sqrt{\lambda_j}\theta) \qquad j = 1, 2... \tag{12}$$

And the valid eigenvalues are $\lambda_j = j^2$.

With the ODE in terms of θ being solved, we can move on to the one in r:

$$\frac{r^2R'' + rR'}{R} = \lambda_j \tag{13}$$

$$r^2R'' + rR' - \lambda_j R = 0 \tag{14}$$

The equation above is a simple Cauchy-Euler equation with the following general solution:

$$R_0(r) = C_0 + D_0 \ln r \tag{15}$$

$$R_j(r) = C_j r^{\sqrt{\lambda_j}} + D_j r^{-\sqrt{\lambda_j}} \qquad j = 1, 2... \tag{16}$$

Recall that we have a homogeneous Dirichlet boundary condition at $r = r_1$. For this to be true for all θ, the coefficients C_j and D_j should be as such that $R_j(r_1) = 0$ for all j. Or:

$$\frac{D_0}{C_0} = -\frac{1}{\ln r_1} \tag{17}$$

$$\begin{aligned} C_j r_1^{\sqrt{\lambda_j}} + D_j r_1^{-\sqrt{\lambda_j}} &= 0 \\ \frac{D_j}{C_j} = -\frac{r_1^{\sqrt{\lambda_j}}}{r_1^{-\sqrt{\lambda_j}}} &= -r_1^{2\sqrt{\lambda_j}} \end{aligned} \tag{18}$$

Using the results from above and the separability of variables, we can write the overall, general solution for $u(r, \theta)$:

$$u(r, \theta) = \sum_{j=0}^{\infty} R_j(r) * T_j(\theta) \tag{19}$$

Note that the solutions for the decoupled functions are determined up until some proportionality constants that depend on j. When multiplying, these can be combined into new, unknown variables A_j:

$$\begin{aligned} u(r, \theta) = (A_0) * (1 - \frac{\ln r}{\ln r_1}) &+ \sum_{j=1}^{\infty} A_j (r^{\sqrt{\lambda_j}} - r_1^{2\sqrt{\lambda_j}} r^{-\sqrt{\lambda_j}}) \cos \sqrt{\lambda_j}\theta \\ &+ \sum_{j=1}^{\infty} B_j (r^{\sqrt{\lambda_j}} - r_1^{2\sqrt{\lambda_j}} r^{-\sqrt{\lambda_j}}) \sin \sqrt{\lambda_j}\theta \end{aligned} \tag{20}$$

We now only have to compute the coefficients A_0, A_j and B_j for $j = 1, 2, ...$ This can be done using the non-homogeneous Neumann boundary condition:

$$\frac{\partial u}{\partial r}\bigg|_{r=r_2} = g(\theta) \tag{21}$$

The left hand side of this equation becomes:

$$\begin{aligned} \frac{\partial u}{\partial r}\bigg|_{r=r_2} = A_0 \frac{-1}{r_2 ln(r_1)} &+ \sum_{j=1}^{\infty} A_j (\sqrt{\lambda_j} r_2^{\sqrt{\lambda_j}-1} + r_1^{2\sqrt{\lambda_j}} \sqrt{\lambda_j} r_2^{-\sqrt{\lambda_j}-1}) \cos \sqrt{\lambda_j}\theta \\ &+ \sum_{j=1}^{\infty} B_j (\sqrt{\lambda_j} r_2^{\sqrt{\lambda_j}-1} + r_1^{2\sqrt{\lambda_j}} \sqrt{\lambda_j} r_2^{-\sqrt{\lambda_j}-1}) \sin \sqrt{\lambda_j}\theta \end{aligned} \tag{22}$$

With renaming the coefficients, the boundary condition becomes:

$$A_{0,r2} + \sum_{j=1}^{\infty} A_{j,r2} \cos \sqrt{\lambda_j}\theta + \sum_{j=1}^{\infty} B_{j,r2} \sin \sqrt{\lambda_j}\theta = g(\theta) \tag{23}$$

Note that this is equal to a Fourier series expansion of $g(\theta)$. Given that the frequencies are equal to $\sqrt{\lambda_j} = j$, the coefficients can be easily computed as follows:

$$A_{0,r2} = \frac{1}{2\pi} \int_{-\pi}^{\pi} g(\theta) d\theta \tag{24}$$

$$A_{j,r2} = \frac{1}{\pi}\int_{-\pi}^{\pi} g(\theta)\cos\sqrt{\lambda_j}\theta d\theta \tag{25}$$

$$B_{j,r2} = \frac{1}{\pi}\int_{-\pi}^{\pi} g(\theta)\sin\sqrt{\lambda_j}\theta d\theta \tag{26}$$

Note that now $A_{0,r2}$ and all $A_{j,r2}$ and $B_{0,r2}$ can be computed. The only thing missing is the connection to the general solution in equation (20), which is solved through:

$$A_0 = -A_{0,r2} * r_2 \ln(r_1) \tag{27}$$

$$A_j = \frac{A_{j,r2}}{\sqrt{\lambda_j} r_2^{\sqrt{\lambda_j}-1} + r_1^{2\sqrt{\lambda_j}} \sqrt{\lambda_j} r_2^{-\sqrt{\lambda_j}-1}} \tag{28}$$

$$B_j = \frac{B_{j,r2}}{\sqrt{\lambda_j} r_2^{\sqrt{\lambda_j}-1} + r_1^{2\sqrt{\lambda_j}} \sqrt{\lambda_j} r_2^{-\sqrt{\lambda_j}-1}} \tag{29}$$

Combining these expressions for the coefficients with equation (20), the entire analytical solution $u(r, \theta)$ is known.

3 Exploring the neural network architecture

3.1 One hidden layer

This section explores the mathematical formulation of a neural network architecture with two input, one hidden layer and a continuous, single output. Note that the two inputs correspond to the two dimensions of the PDE in the section above, and the output function is a neural network prediction $\tilde{u}_{NN}$ of the exact solution u. The following figure illustrates a basic architecture with two nodes:

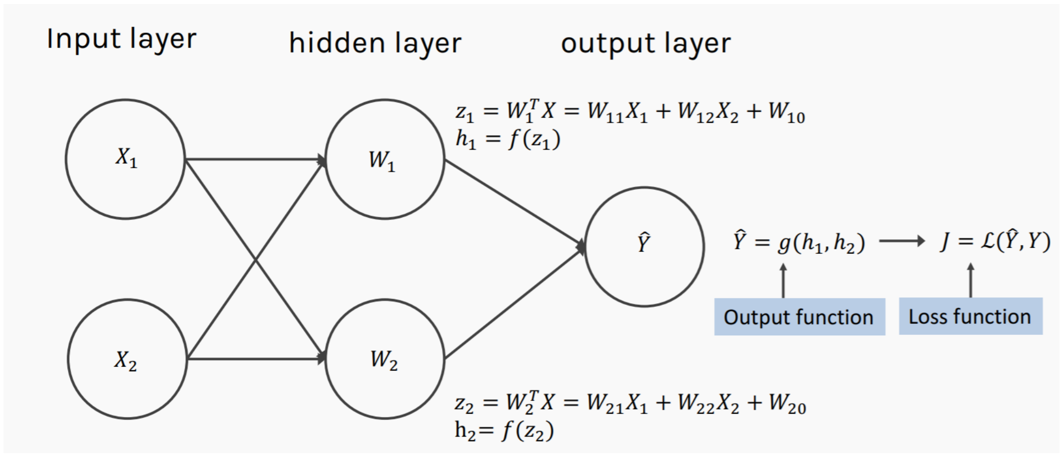
Input layer
hidden layer
output layer
X_1
X_2
W_1
W_2
$\hat{Y}$
$z_1 = W_1^T X = W_{11}X_1 + W_{12}X_2 + W_{10}$
$h_1 = f(z_1)$
$z_2 = W_2^T X = W_{21}X_1 + W_{22}X_2 + W_{20}$
$h_2 = f(z_2)$
$\hat{Y} = g(h_1, h_2) \longrightarrow J = \mathcal{L}(\hat{Y}, Y)$
Output function
Loss function

Figure 2: Illustration of a simple Artificial Neural Network (ANN), ref. to CS209a

The input layer has two continuous values, X_1 and X_2. Two nodes z_1 and z_2 result from a linear combination of these two values with an added bias. Each node therefore has two weights and a bias term, which will be tuned during training. Next, the resulting linear combinations z_1 and z_2 are 'activated' with a non-linear activation function $f(z)$. In what follows, this will be assumed to be a sinusoidal function. As the output should be a single, continuous value $\tilde{u}_{NN}$, the output function will again be a simple linear combination of the activated outputs of all nodes, with according weights and bias term.

Let's now try to find the mathematical expression of $\tilde{u}_{NN}$ in terms of the two input variables X_1 and X_2 and all the weights and biases. First, it is important to clearly define notation. W_j^i corresponds to the vector containing the weights and bias term for layer i and node j. $W^{(o)}$ contains the weights of the output layer. Capital letters correspond to the vectors, and small letters to the real numbers.

The two nodes are being computed as follows:

$$z_1 = W_1^{(1)} \cdot X = \begin{bmatrix} w_{11}^{(1)} \\ w_{12}^{(1)} \\ w_{10}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ 1 \end{bmatrix} = w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)} \tag{30}$$

$$z_2 = W_2^{(1)} \cdot X = \begin{bmatrix} w_{21}^{(1)} \\ w_{22}^{(1)} \\ w_{20}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ 1 \end{bmatrix} = w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)} \tag{31}$$

These nodes will now be 'activated' as follows:

$$h_1^{(1)} = sin(z_1) = sin(W_1^{(1)} \cdot X) = sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \tag{32}$$

$$h_2^{(1)} = sin(z_2) = sin(W_2^{(1)} \cdot X) = sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) \tag{33}$$

With the output weights contained in vector $W^{(o)}$, we can now write the output:

$$\tilde{u}_{NN} = W^{(o)} \cdot H^{(1)} = \begin{bmatrix} w_1^{(o)} \\ w_2^{(o)} \\ w_0^{(o)} \end{bmatrix} \cdot \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ 1 \end{bmatrix} = w_1^{(o)} h_1^{(1)} + w_2^{(o)} h_2^{(1)} + w_0^{(o)} \tag{34}$$

Or in terms of the input variables, this becomes:

$$\tilde{u}_{NN} = w_1^{(o)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) + w_2^{(o)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_0^{(o)} \tag{35}$$

For an total number of N nodes in one hidden layer, the general expression becomes:

$$\tilde{u}_{NN} = \sum_{i=1}^{N} w_i^{(o)} sin(w_{i1}^{(1)} X_1 + w_{i2}^{(1)} X_2 + w_{i0}^{(1)}) + w_0^{(o)} = \sum_{i=1}^{N} w_i^{(o)} sin(W_i^{(1)} \cdot X) + w_0^{(o)} \tag{36}$$

As a sidenote, we keep in mind the classic mathematical expressions for the sine and cosine of sums:

$$\sin(\alpha + \beta) = \sin\alpha \cos\beta + \cos\alpha \sin\beta \tag{37}$$

$$\cos(\alpha + \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta \tag{38}$$

This means that sines and cosines of sums can be written as sums of products of sines and cosines. Hence, one could possibly rewrite equation (22) such that only sines and cosines of every input value individually appear.

3.2 Two hidden layers

It is now interesting to check what happens to this mathematical expression if there are two hidden layers. The following figure illustrates what such an architecture would look like

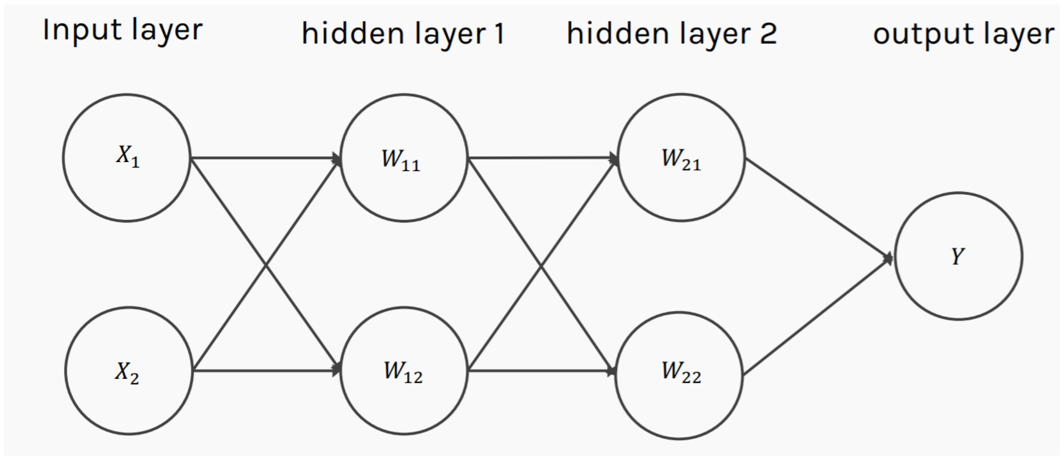

Figure 3: Illustration of a simple Artificial Neural Network (ANN) with 2 layers, ref. to CS209a

For this, let's define vector $H^{(1)}$ containing all activated nodes from the first layer and a 1 for the bias term. Using derivations from before we get:

$$H^{(1)} = \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ 1 \end{bmatrix} = \begin{bmatrix} sin(W_1^{(1)} \cdot X) \\ sin(W_2^{(1)} \cdot X) \\ 1 \end{bmatrix} = \begin{bmatrix} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \\ sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) \\ 1 \end{bmatrix} \tag{39}$$

For the second layer, this now becomes:

$$H^{(2)} = \begin{bmatrix} h_1^{(2)} \\ h_2^{(2)} \\ 1 \end{bmatrix} = \begin{bmatrix} sin(W_1^{(2)} \cdot H^{(1)}) \\ sin(W_2^{(2)} \cdot H^{(1)}) \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} sin(w_{11}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) + w_{12}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{10}^{(2)}) \\ sin(w_{21}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) + w_{22}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{20}^{(2)}) \\ 1 \end{bmatrix} \tag{40}$$

The output can then be written as:

$$\begin{aligned} \tilde{u}_{NN} &= W^{(o)} \cdot H^{(2)} \\ &= w_1^{(o)} sin(w_{11}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \\ &\quad + w_{12}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{10}^{(2)}) \\ &\quad + w_2^{(o)} sin(w_{21}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \\ &\quad + w_{22}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{20}^{(2)}) \\ &\quad + w_0^{(o)} \end{aligned} \tag{41}$$

Interestingly, implementing more layers leads to a serial application of the activation function (so a sine of a sum of sines), while more nodes increases the length of the linear combinations within one sine. Both options lead to an equal increase in weights.

4 Solving an ODE with a Neural Network

In this section, we will attempt to solve a simple ordinary differential equation (ODE) with a neural network (NN). We first discuss the equation to be solved and its exact solution and subsequently expand on the neural network architecture needed to solve this.

4.1 Problem statement and exact solution

The ODE we wish to solve is the following:

$$\frac{d^2x}{dt^2} + \omega^2 x = 0 \tag{42}$$

With initial conditions $x(0) = x_0$ and $\frac{dx}{dt}(0) = v_0$. The analytical solution is equal to:

$$x(t) = \frac{v_0}{\omega} \sin(\omega t) + x_0 \tag{43}$$

For $x_0 = 0$, $v_0 = 1$ and $\omega = 2$, this leads to the following graphical result:

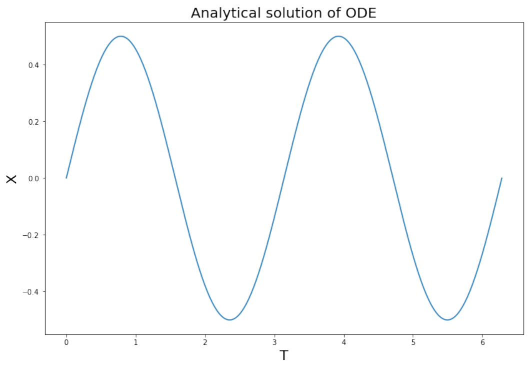

Figure 4: Solution of ODE

4.2 Neural Network solution

We now wish to solve equation (42) with a NN. The following figure illustrates how such a network can be designed:

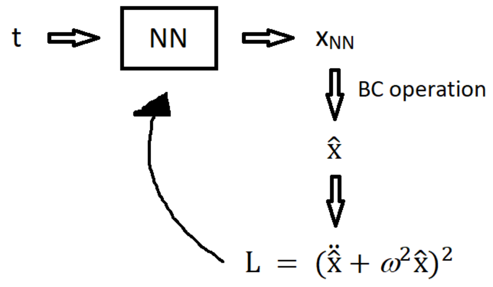

Figure 5: NN representation

The input variable is one scalar t, which leads through a series of nodes and layers of linear combinations and activation functions to a value x_{NN}. In order to force the solution to satisfy the specified boundary conditions, the following operation is computed:

$$\hat{x} = x_0 + (1 - e^{-t})v_0 + (1 - e^{-t})^2 x_{NN} \tag{44}$$

The loss function L is then computed based on the structure of the ODE. For a correct solution, the loss must be equal to zero.

$$L = (\ddot{\hat{x}} + \omega^2 \hat{x})^2 \tag{45}$$

Note that the loss function is a complication function of t and the weights of the neural network. The derivatives that explicitly appear in the loss function are with respect to the input variable t. For minimization, the algorithm will have to compute the gradient of the loss function with respect to the weights.

This has been implemented in Pytorch, with a tanh as activation function and two layers with each 20 nodes. The following illustrates the results, for a different number of epochs. The loss function of the prediction after 2000 epochs was approximately equal to 0, while the one with 1000 epochs had not converged yet. Interestingly, it seems that the boundary condition operation forces the solution to first fit the beginning of the t-interval.

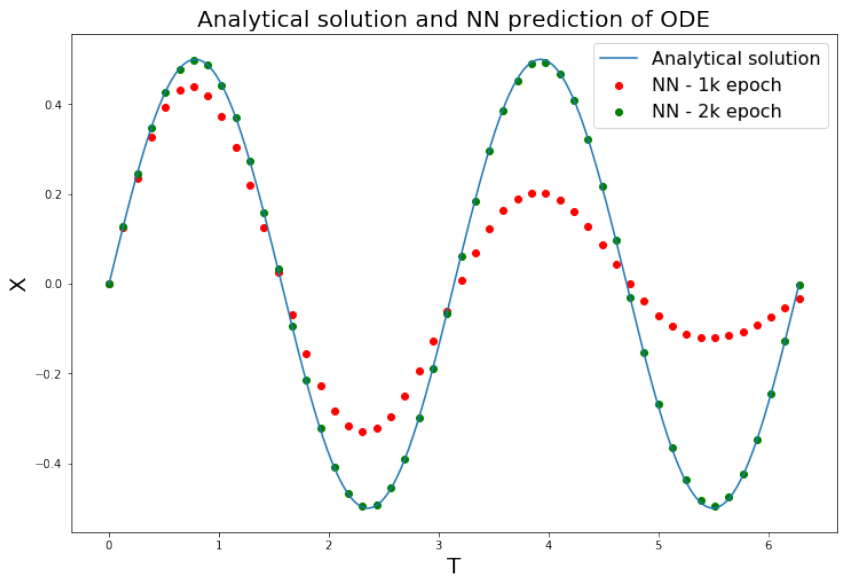

Figure 6: NN prediction

4.3 Chebyshev Interpolation

Next, I wondered whether it matters how you are interpolating the points in the t-domain. In the analysis above, a simple linear interpolation was used. Given that Chebyshev interpolation has the characteristic of minimizing interpolation error when approximating functions by polynomials, this might be result in a better performance of the network. Let's find out.

Recall the Chebyshev interpolation on an interval $[a, b]$:

$$x_k = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \dots, n. \tag{46}$$

The following graph illustrates how the loss function decreases over the number of iterations, using the same number of interpolation points and learning rate. Next, the solution is plotted for both trained networks. In both cases, it seems that there might be a small advantage for the Chebyshev interpolation, but it is hard to call it very significant.

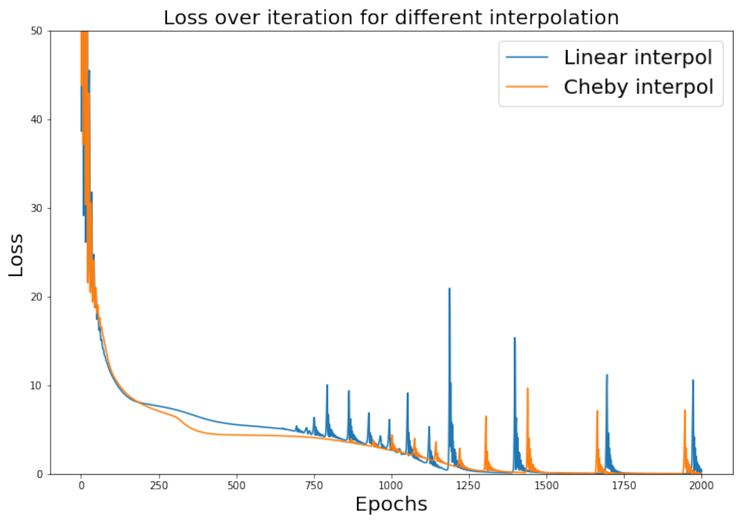

Figure 7: Chebyshev vs linear interpolation - loss

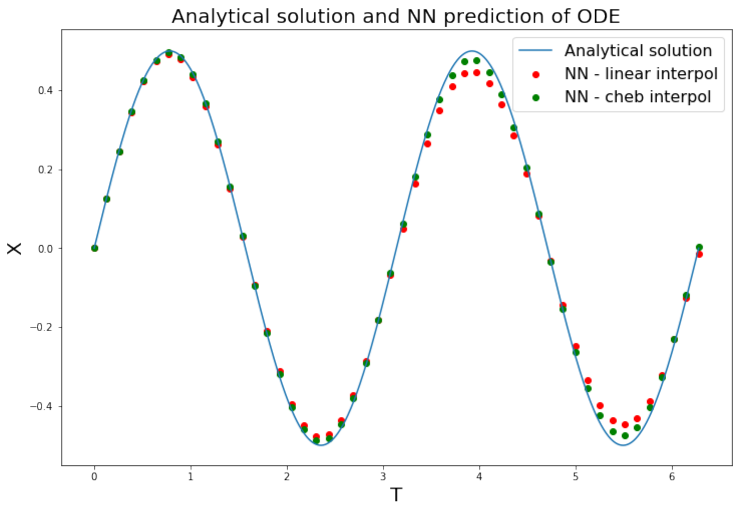

Figure 8: Chebyshev vs linear interpolation - results

4.4 Experimenting with Pavlos' Masternode

Last week, Pavlos proposed to implement one masternode in front of the neural network in order to incorporate periodicity of a solution.

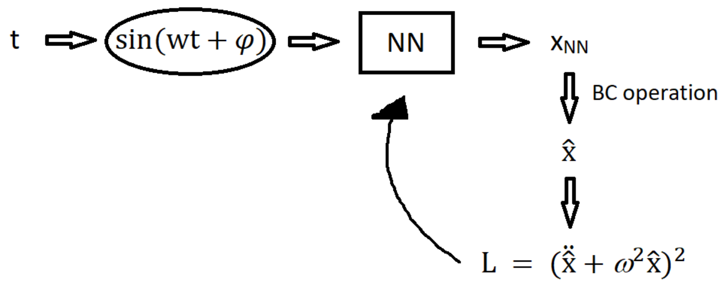

Figure 9: NN Architecture with masternode

From the figure above, it is clear that the input parameter is first 'activated' by a sine function with a certain frequency and phase that are incorporated as weights to be trained. As such, the hope is to have periodic output functions in t. This would make sense, as:

$$sin(wt + \phi) = sin(wt + \phi + 2 * \pi) \tag{47}$$

As such, every $\frac{2\pi}{w}$, the output of the masternode and thus the input of the remaining neural network will be same, resulting in a periodic output.

After implementing this, it turns out that it takes more iterations to train the network with the masternodes to achieve the same accuracy on the training domain as before. Despite of that, it is interesting to see how the network performs outside the training domain. The figure below shows the difference in performance with and without a masternode in a domain that is 'unknown' for the network.

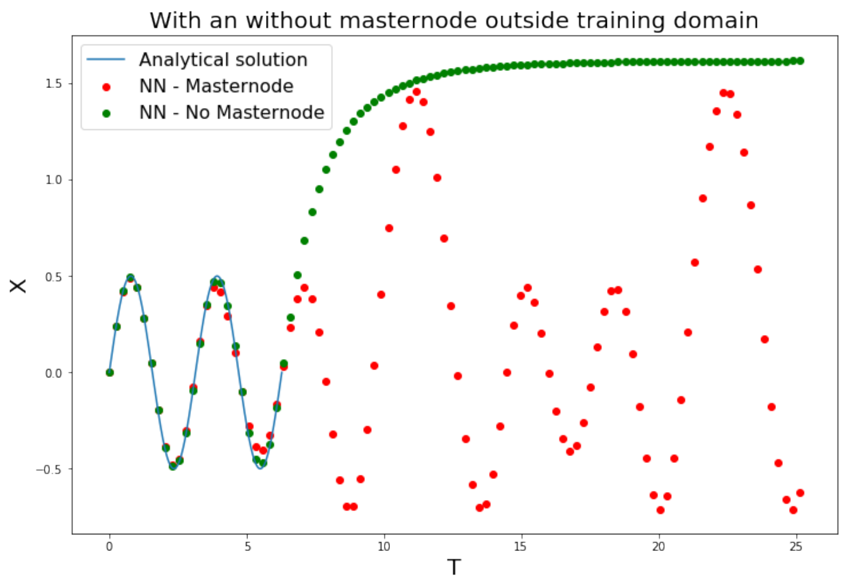

Figure 10: Results with masternode

Interestingly, the network without masternode performs very poorly, not capturing the trends at all. The network with masternode does seem to continue somehow periodically, but the results are not correct at all. I think this is because we eliminate the presumed periodicity from equation (47) by applying the boundary condition operation from equation (44). I have tried to implement the same periodicity into the boundary condition operation, but I could not find it. I think I have to understand more on why this particular boundary condition operation has been chosen.

5 Conclusion

This documents has touched upon multiple topics.

First, it discusses the analytical solution of the Laplace equation in polar coordinates with particular boundary conditions. The final solution consists of an infinite sum of sines and cosines of θ, multiplied by powers of r.

Second, the mathematics behind a simple neural network with sine as activation function has been developed, both for one layer with multiple nodes as for two layers. It appears that the first leads to a linear combination of sines of the input variables, while the latter leads to weighted sums of sines of sines.

Third, a simple ODE has been solved with a neural network through a Pytorch implementation of the cleverly designed, unsupervised NN architecture. It was interesting to see how the boundary condition operation forces the network to first (in the first iterations) fit around the boundaries. Also, I evaluated the convergence of the NN with Chebyshev interpolation rather than linear interpolation. It seems that there is a possibility that Chebyshev can do better. Lastly, the masternode of Pavlos has been implemented in the same NN in the hope periodic solutions would arise. The model seems to grasp some periodicity outside the training domain, but we should think of a way to mathematically ensure the periodicity in the boundary condition operation as well.