Independent Study

Matthieu Meeus

March 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 goes beyond the initial exploration from the last couple of weeks.

First, recall the analytical of the two-dimensional, homogeneous Laplace equation in a circular domain. The same method, being separation of variables will now be used to solve the same equation but in three dimensions while maintaining the same circular form, therefore in cylindrical coordinates. The hope is obtain Bessel functions in the solution, rather than the infinite sum of polynomials and trigonometric functions of last time. 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. Note that this has been incorporated in the document since the beginning.

Next, recall the Pytorch implementation of the neural network to solve a simple ODE. After a successful individual implementation, for now only the publicly available code developed by Feiyu will be used and expanded. First, the theory about Chebyshev interpolation was tested and appeared to be not very useful. Second, the performance of the proposed masternode to guarantee periodicity in the solution will be evaluated. Last week, some preliminary results were very positive. Two questions will be asked to verify its performance in this documents:

- Is the model able to predict the frequency through the weight inside the masternode?
- Is the resulting NN solution truly periodic?
- Is this good performance specific to the sine-problem? This will be verified by checking the performance on an ODE with periodic, non-goniometric results.

Lastly, the 2D/3D part of the code will be explored with boundary conditions rather than initial conditions.

2 Problem formulation and analytical solution

2.1 Problem formulation

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

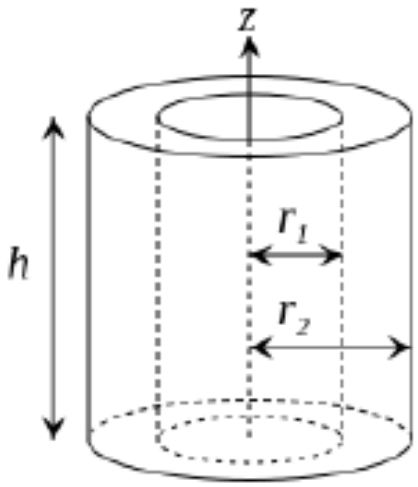

Figure 1: The considered cylinderical domain for the PDE, (ref)

The general, homogeneous Laplace equation is equal to:

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

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

$$\nabla^2 u = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} + \frac{\partial^2 u}{\partial z^2} = 0. \tag{2}$$

We now have to specify the boundary conditions applicable to the problem. For the radial dimensions, we will consider be a homogeneous Dirichlet boundary condition for $r = r_1$ and a non-homogeneous Neumann boundary condition for $r = r_2$, or:

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

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

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

For the z-dimension, we will also consider a homogeneous Dirichlet boundary condition for $z = 0$ and a non-homogeneous one for $z = z_1$.

$$u(r, \theta, z = 0) = 0 \tag{5}$$

$$u(r, \theta, z = z_1) = h(r, \theta) \tag{6}$$

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), (4), (5) and (6).

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 $u(r, \theta, z)$ as a product of decoupled functions in its three variables r, θ and z. Or:

$$u(r, \theta, z) = R(r)T(\theta)Z(z) \tag{7}$$

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

$$\nabla^2 u(r, \theta, z) = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} + \frac{\partial^2 u}{\partial z^2} = \frac{TZ}{r}\frac{d}{dr}\left(r\frac{dR}{dr}\right) + \frac{RZ}{r^2}\frac{d^2T}{d\theta^2} + RT\frac{d^2Z}{\partial z^2} = 0 \tag{8}$$

Which leads to:

$$\frac{TZ}{r}\frac{dR}{dr} + TZ\frac{d^2R}{dr^2} + \frac{RZ}{r^2}\frac{d^2T}{d\theta^2} + RT\frac{d^2Z}{\partial z^2} = 0 \tag{9}$$

$$\frac{1}{r}\frac{R' + rR''}{R} + \frac{1}{r^2}\frac{T''}{T} + \frac{Z''}{Z} = 0 \tag{10}$$

Note that the first two terms can only depend on r and θ, while the third can only depend on z and their sum should be equal to zero. This can only be true if -with λ being a real number:

$$\begin{aligned} -\frac{Z''}{Z} &= \lambda \\ \frac{1}{r}\frac{R' + rR''}{R} + \frac{1}{r^2}\frac{T''}{T} &= \lambda. \end{aligned} \tag{11}$$

Note that through the separation of variables, we are able to decouple the PDE into multiple equations.

Let's start solving for $Z(z)$, or:

$$\frac{Z''}{Z} = -\lambda \tag{12}$$

We now suppose that λ is a negative number, or $\lambda = -\gamma^2$. With this, we know that solution for $Z(z)$:

$$Z_0(z) = A_0 + B_0 z \tag{13}$$

$$Z_\gamma(\theta) = A_\gamma cosh(\gamma z) + B_\gamma sinh(\gamma_j z) \tag{14}$$

Considering the boundary condition (5), Z should be equal to zero when $z = 0$, which leads to $A_0 = A = 0$.

Next, we proceed with the second part of the (11) and multiply both sides by r^2:

$$r\frac{R' + rR''}{R} + \frac{T''}{T} = \lambda r^2 \tag{15}$$

Again, we know that $\frac{T''}{T}$ should be equal to a real number, so:

$$-\kappa = \frac{T''}{T} = \lambda r^2 - r\frac{R' + rR''}{R} \tag{16}$$

Let's say $\kappa = \nu^2$ and thus a positive number. 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:

$$T_0(\theta) = C_0 \tag{17}$$

$$T_\nu(\theta) = C_\nu cos(\nu\theta) + D_\nu sin(\nu\theta) \tag{18}$$

Where ν should be a strictly positive integer, so $\kappa_j = \nu^2$ for $\nu = 1, 2, ...$

We can know return to equation (16) and solve for $R(r)$, which will be more complicated.

$$\begin{aligned} -\kappa &= \lambda r^2 - r\frac{R' + rR''}{R} \\ -\kappa R &= \lambda r^2 R - rR' - r^2 R'' \\ r^2 R'' + rR' + (-\kappa - \lambda r^2)R &= 0 \\ r^2\frac{R''}{R} + r\frac{R'}{R} + (-\nu^2 + \gamma^2 r^2) &= 0 \end{aligned} \tag{19}$$

We can now substitute $x = \gamma r$ and $r = \frac{x}{\gamma}$, which leads to -after some algebra:

$$\frac{d^2R}{dx^2} + \frac{1}{x}\frac{dR}{dx} + \left(1 - \frac{\nu^2}{x^2}\right)R = 0. \tag{20}$$

The solution of this ODE is a linear combination of the Bessel function of the first and second kind, or $J_{\nu x}$ and $Y_{\nu x}$ respectively. This leads to:

$$R_\nu(r) = E_\nu J_\nu(\gamma r) + F_\nu Y_\nu(\gamma r) + G_\nu \tag{21}$$

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

$$u(r, \theta) = R(r) * T(\theta) * Z(z) = \sum_\nu \sum_\gamma \begin{Bmatrix} sinh(\gamma z) \\ cosh(\gamma z) \end{Bmatrix} \begin{Bmatrix} \cos(\nu\theta) \\ \sin(\nu\theta) \end{Bmatrix} \begin{Bmatrix} J_\nu(\gamma r) \\ Y_\nu(\gamma r) \end{Bmatrix} \tag{22}$$

With the coefficients of $cosh(\nu z)$ being zero and the incorporating the constants, we get:

$$u(r,\theta,z) = A_0 + \sum_{\nu}\sum_{\gamma}\left\{\begin{array}{c} A_{\nu\gamma}sinh(\gamma z)\cos(\nu\theta)J_\nu(\gamma r) \\ +B_{\nu\gamma}sinh(\gamma z)\cos(\nu\theta)Y_\nu(\gamma r) \\ +C_{\nu\gamma}sinh(\gamma z)\sin(\nu\theta)J_\nu(\gamma r) \\ +D_{\nu\gamma}sinh(\gamma z)\sin(\nu\theta)Y_\nu(\gamma r)\end{array}\right\}. \tag{23}$$

The only task that remains is determining all coefficients associated with each γ and ν. This can be done using non-homogeneous boundary conditions (4) and (6).

I do have some difficulties moving forward into the details here. My question is if we could simplify this by saying that $r_1 = 0$, which leads to the exclusion of the Bessel function of the second kind $Y_\nu(\gamma r)$ as this is not finite for $r = 0$.

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:

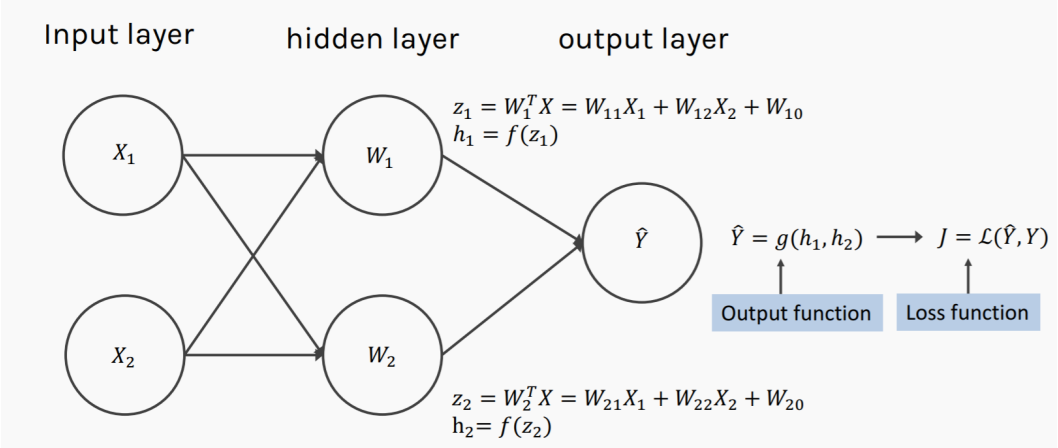

Figure 2: Illustration of a simple Artificial Neural Network (ANN), ref. to CS209a

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:

$$z_1 = W_1^{(1)} \cdot X = \begin{bmatrix} w_{11}^{(1)} \\ w_{12}^{(1)} \\ w_{10}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ 1 \end{bmatrix} = w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)} \tag{24}$$

$$z_2 = W_2^{(1)} \cdot X = \begin{bmatrix} w_{21}^{(1)} \\ w_{22}^{(1)} \\ w_{20}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ 1 \end{bmatrix} = w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)} \tag{25}$$

These nodes will now be 'activated' as follows:

$$h_1^{(1)} = sin(z_1) = sin(W_1^{(1)} \cdot X) = sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \tag{26}$$

$$h_2^{(1)} = sin(z_2) = sin(W_2^{(1)} \cdot X) = sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) \tag{27}$$

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

$$\tilde{u}_{NN} = W^{(o)} \cdot H^{(1)} = \begin{bmatrix} w_1^{(o)} \\ w_2^{(o)} \\ w_0^{(o)} \end{bmatrix} \cdot \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ 1 \end{bmatrix} = w_1^{(o)} h_1^{(1)} + w_2^{(o)} h_2^{(1)} + w_0^{(o)} \tag{28}$$

Or in terms of the input variables, this becomes:

$$\tilde{u}_{NN} = w_1^{(o)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) + w_2^{(o)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_0^{(o)} \tag{29}$$

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

$$\tilde{u}_{NN} = \sum_{i=1}^{N} w_i^{(o)} sin(w_{i1}^{(1)} X_1 + w_{i2}^{(1)} X_2 + w_{i0}^{(1)}) + w_0^{(o)} = \sum_{i=1}^{N} w_i^{(o)} sin(W_i^{(1)} \cdot X) + w_0^{(o)} \tag{30}$$

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

$$\sin(\alpha + \beta) = \sin\alpha \cos\beta + \cos\alpha \sin\beta \tag{31}$$

$$\cos(\alpha + \beta) = \cos\alpha \cos\beta - \sin\alpha \sin\beta \tag{32}$$

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

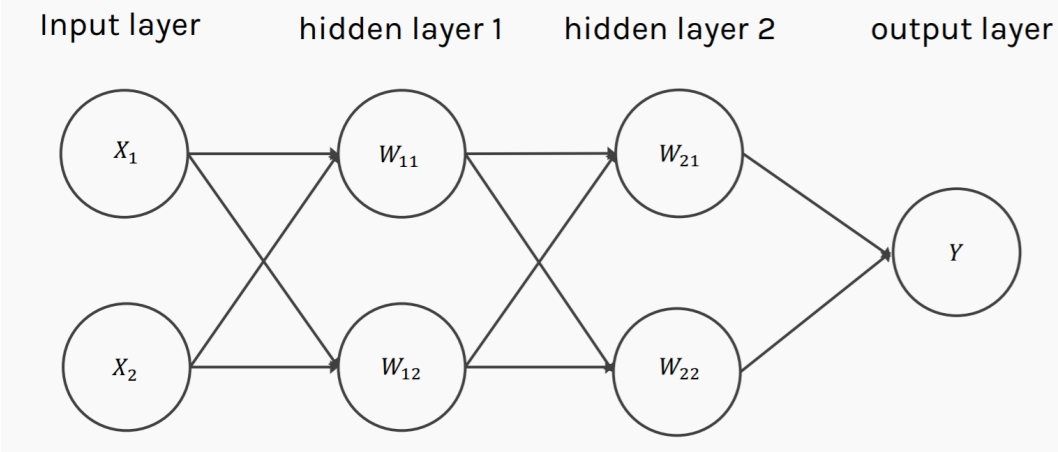

Figure 3: Illustration of a simple Artificial Neural Network (ANN) with 2 layers, ref. to CS209a

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:

$$H^{(1)} = \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ 1 \end{bmatrix} = \begin{bmatrix} sin(W_1^{(1)} \cdot X) \\ sin(W_2^{(1)} \cdot X) \\ 1 \end{bmatrix} = \begin{bmatrix} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \\ sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) \\ 1 \end{bmatrix} \tag{33}$$

For the second layer, this now becomes:

$$H^{(2)} = \begin{bmatrix} h_1^{(2)} \\ h_2^{(2)} \\ 1 \end{bmatrix} = \begin{bmatrix} sin(W_1^{(2)} \cdot H^{(1)}) \\ sin(W_2^{(2)} \cdot H^{(1)}) \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} sin(w_{11}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) + w_{12}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{10}^{(2)}) \\ sin(w_{21}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) + w_{22}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{20}^{(2)}) \\ 1 \end{bmatrix} \tag{34}$$

The output can then be written as:

$$\begin{aligned}\tilde{u}_{NN} &= W^{(o)} \cdot H^{(2)} \\ &= w_1^{(o)} sin(w_{11}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \\ &+ w_{12}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{10}^{(2)}) \\ &+ w_2^{(o)} sin(w_{21}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \\ &+ w_{22}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{20}^{(2)}) \\ &+ w_0^{(o)}\end{aligned} \tag{35}$$

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:

$$\frac{d^2x}{dt^2} + \omega^2 x = 0 \tag{36}$$

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

$$x(t) = \frac{v_0}{\omega} \sin(\omega t) + x_0 \tag{37}$$

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

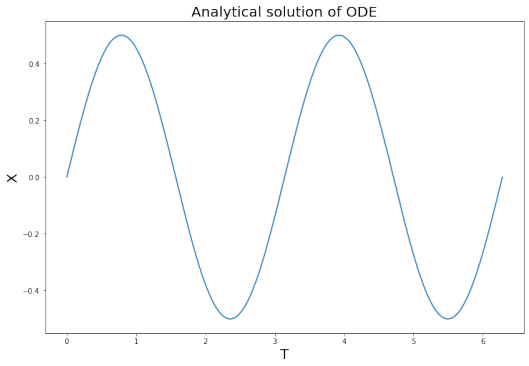

Figure 4: Solution of ODE

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:

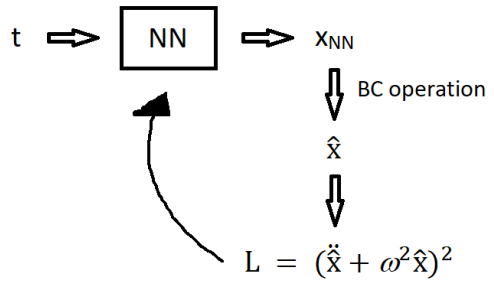

Figure 5: NN representation

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:

$$\hat{x} = x_0 + (1 - e^{-t})v_0 + (1 - e^{-t})^2 x_{NN} \tag{38}$$

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.

$$L = (\ddot{\hat{x}} + \omega^2 \hat{x})^2 \tag{39}$$

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 32 nodes. The following illustrates the results.

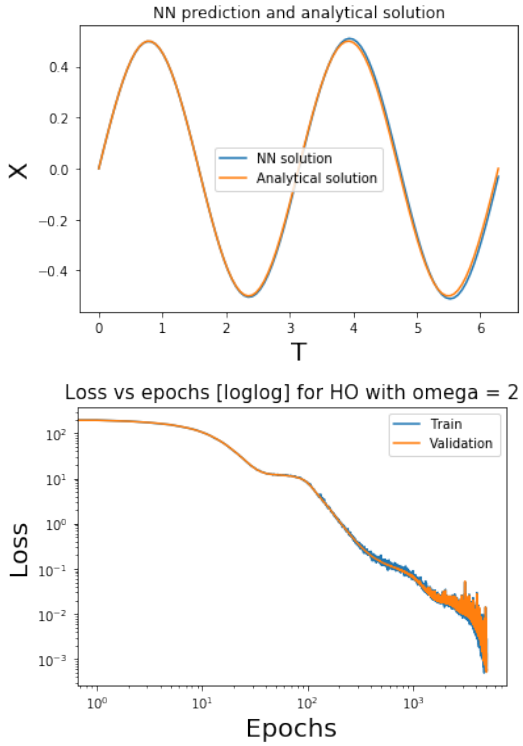

Figure 6: NN prediction of simple ODE

It is clear that the neural network does a pretty good job in predicting the solution to the ODE. Two things made me wonder about potentially improving the current design.

First, the interpolation method to generate the training data that is currently used is either uniformly distributed (default), equally spaced, or a noisy version of either one. This raises the question is there might be a modified sampling approach that leads to better results. In the section below, the Chebyshev interpolation is tested.

Secondly, it is interesting to analyse the performance of the neural network outside the training domain. The figure below shows what happens when the same NN as trained above is used to predict the solution of the ODE.

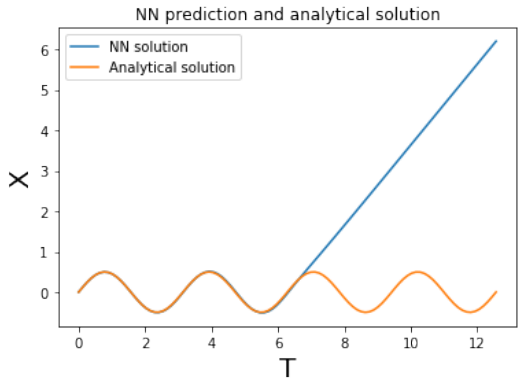

Figure 7: NN prediction outside training domain

We can also plot the difference between the predicted solution and the analytical solution over the t-domain. Here the t-domain is chosen to be slightly smaller than the training domain (up until 2.2π instead of 2π:

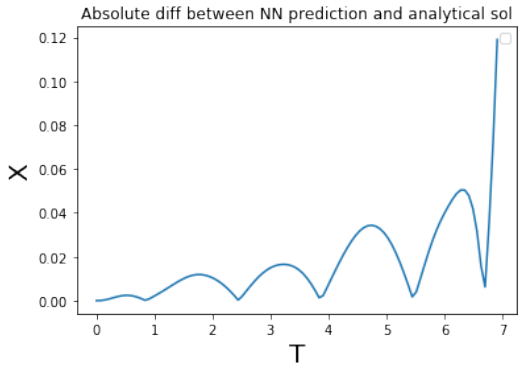

Figure 8: Difference between analytical solution and prediction

From this, we conclude that the network is not able to predict outside the training domain and predicts increasingly worse further away from the initial condition. Next, we will dive deeper into potentially incorporating periodicity into the network, in the hope to expand our performance outside the training domain.

4.3 Chebyshev Interpolation

I wondered whether it matters how you are interpolating the points in the t-domain. In the analysis above, a uniformly distrubuted sampling was used. Given that Chebyshev interpolation has the characteristic of minimizing the 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 $[a, b]$:

$$x_k = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \ldots, n. \tag{40}$$

After training the network, the NN predictions with the Chebyshev interpolation points are illustrated in the figure below. The points tend to be more concentrated towards the ends of the interval.

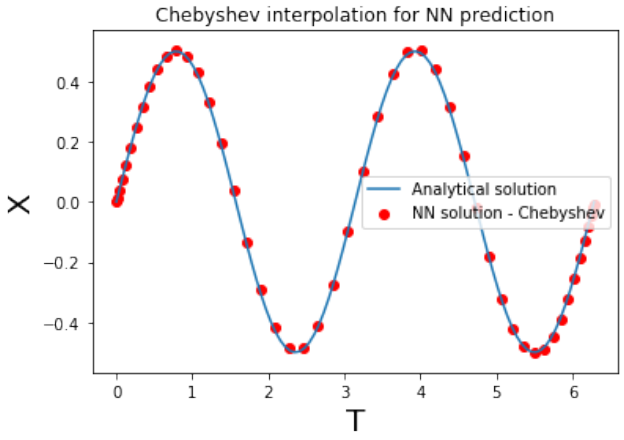

Figure 9: NN prediction with Chebyshev points

Different interpolation methods can only be compared when looking at the losses over amount of epochs. In the following three graphs, I plotted this for three different ODE's: one harmonic oscillator with $\omega = 2$, one with $\omega = 4$ and one exponential decaying function. From all three cases, the Chebyshev interpolation does not seem to have any particular advantage, which is unfortunate. Looking back at Figure 9, this might make sense, as we are actually sampling more close to the initial condition, which is already fit closely through the initial condition operator. In this regard, it might be smart to sample more the further away from the initial condition, but this should be tested (if relevant). In any case, the Chebyshev remains a nice interpolation option to include for the user, as it still might have better performance in particular application.

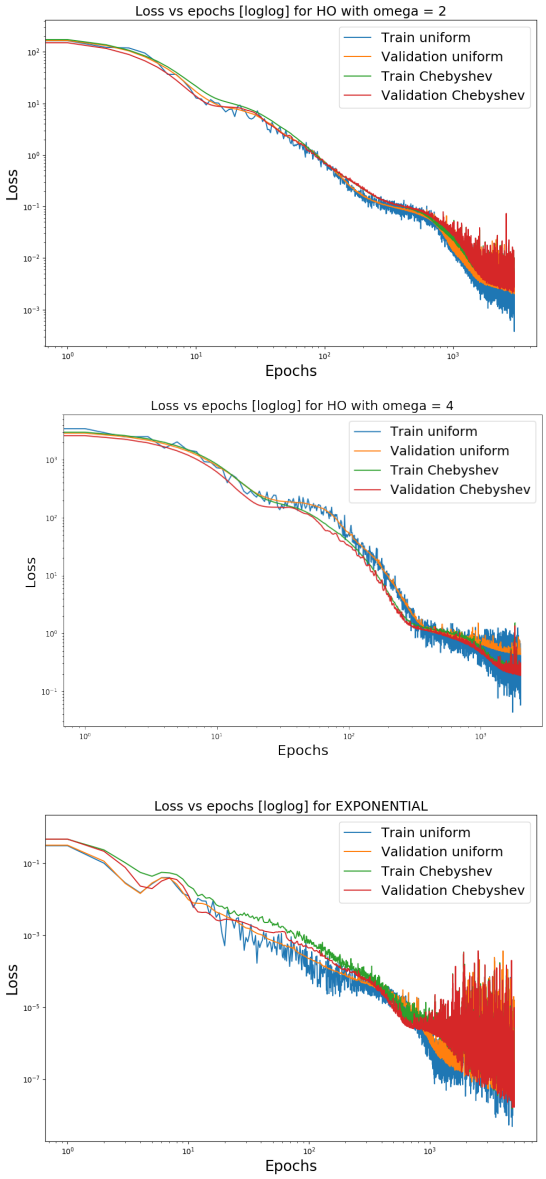

Figure 10: Loss vs epochs for different ODE's, Chebyshev vs uniform

5 Experimenting with Pavlos' Masternode

5.1 Last week analysis

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

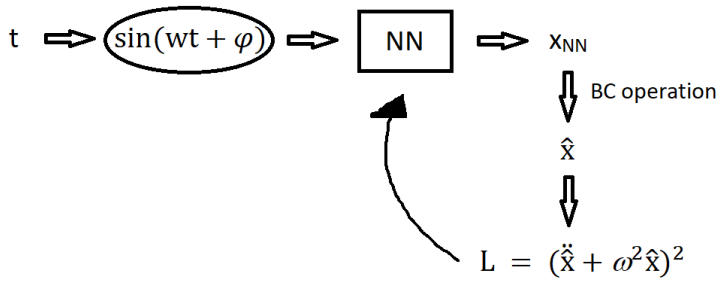

Figure 11: NN Architecture with masternode

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:

$$sin(wt + \phi) = sin(wt + \phi + 2 * \pi) \tag{41}$$

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.

However, recall the initial value operator:

$$\hat{x} = x_0 + (1 - e^{-t})v_0 + (1 - e^{-t})^2 x_{NN} \tag{42}$$

Despite the intrinsic periodicity introduced by the masternode, the initial value operator has non-periodic terms in t, therefore making it impossible to recover a periodic prediction from the NN. This could be solved by looking into an alternative initial value operator, that is periodic in t, and also enforces the initial values of the ODE. The general form is:

$$\hat{x} = x_0 + f_1(t)v_0 + f_2(t)x_{NN} \tag{43}$$

Where f_1 and f_2 should both be periodic in t, and $f_1(0) = 0$, $f_2(0) = 0$, $f_1'(0) = 1$ and $f_2'(0) = 0$. Before going any further, it is important to note that the periodicity in t only makes sense when it's consistent in the masternode and the initial value operator. If we call the first weight used as frequency in the masternode w_0, we should have the same periodicity in t for both functions f_1 and f_2. Keeping this in mind and the specified conditions, the following operator is proposed:

$$\hat{x} = x_0 + \frac{1}{w_0}\sin(w_0 t)v_0 + \sin^2(w_0 t)x_{NN} \tag{44}$$

It is important to notice that the weight w_0 from the masternode now returns in the initial condition operator, and thus in the loss function.

It took a while to incorporate this into Feiyu's code but I believe it worked. The following graphs illustrates the loss in the training domain over the epochs for three methods: the original code with no masternode and the non-periodic initial value operator, the code with the masternode and the non-perodic operator and the code with the masternode and the periodic operator.

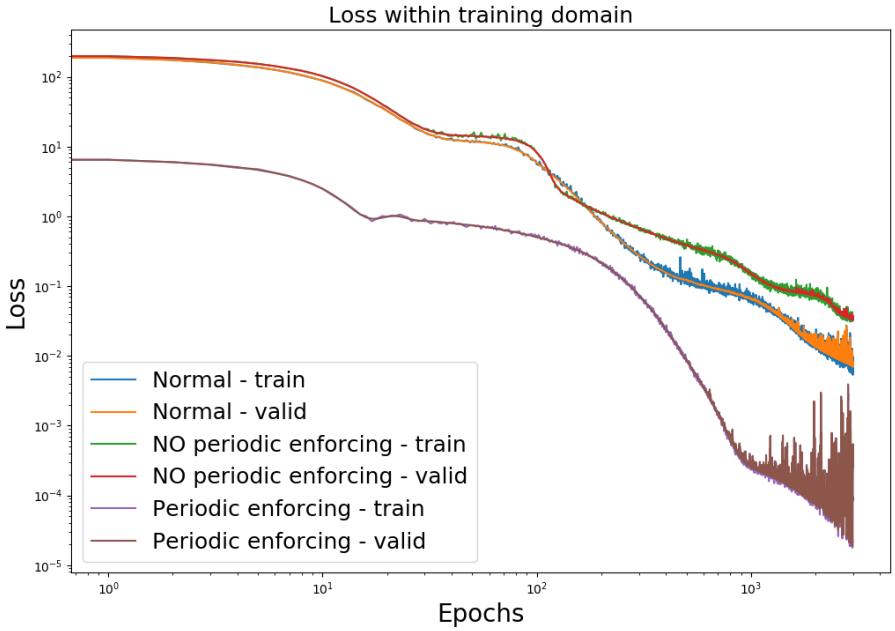

Figure 12: Losses for masternode design

This graph is very interesting! First, it is clear that including the masternode, without adapting the initial value operator, leads to worse performance. This makese sense in my opinion, as you're only partially forcing periodicity. Interestingly, the loss of the network with the masternode and the periodic initial value operator starts with a lower value and decreases faster than the other networks. The first makes sense, as you're using a different initial value operator and thus a different loss function - which appears to be lower. For more iterations, the network seems to benefit from the enforced periodicity, leading to significantly lower values.

Note that the loss computed above is the loss in the training domain. It is now interesting to verify the performance outside the training interval, which is illustrated on the graph below. As expected, the network with masternode and periodic initial value operator is entirely periodic in t, therefore predicting equally well outside the training domain, while the other networks do not have this periodicity and perform very poorly as discussed previously.

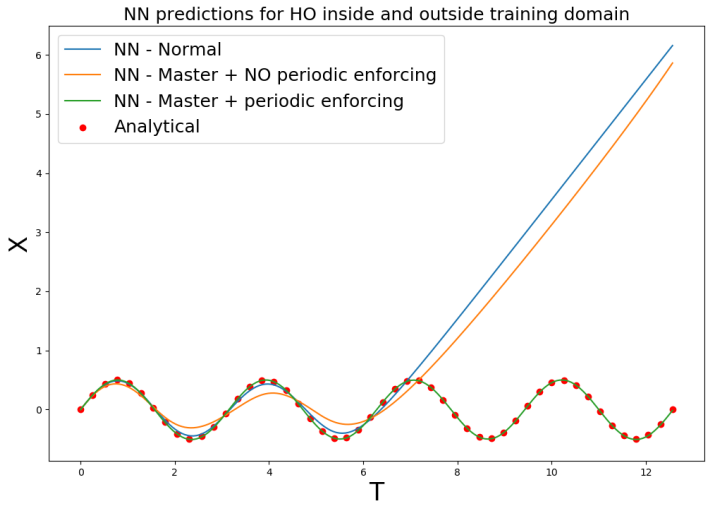

Figure 13: Outside training domain performance for masternode design

We can also plot the error between the predicted values and the corresponding analytical solution. This leads to the following graph:

Figure 14: Errors inside and outside training domain

Knowing that the training domain for the network above was between 0 and 2π, the errors are relatively small even far from the initial condition and outside the training domain. This accuracy was not possible to achieve with other non-periodic implementations.

5.2 This week's analysis

It is clear that the masternode potentially has some positive impact on performance. Before making any statements, we need to dive deeper, which will be done in the following paragraphs.

First, it is crucial to know whether the enforced periodicity that is used as a trainable parameters is predicted correctly by the neural network. Therefore, the analysis is repeated for the same ODE as before but with different values of ω. The following table lists both the trained value of the masternode weight w_0 as the loss in the training domain associated with the network prediction.

Omega	Masternode weight	Loss after 7k epochs
0.75	-0.25000116	1.8429933e-08
1	-1.0000070333	2.69562495e-07
1.25	-0.416494817	4.952709787e-05
1.5	0.49993616	1.56429246e-06
1.75	-0.350078344	0.0073964195
2	0.40003186464	6.240822131e-06
2.25	-0.322204768	0.01389456
2.5	0.833506167	0.00035869

Table 1: Caption

In none of the cases above, the model is able to exactly predict the value of omega. Interestingly, the model is still able to achieve very low losses with these 'wrong' frequencies. Let's have a look at the evolution of the loss vs epochs for the model with ω equal to 0.75 and 2.5.

It is great to see that the loss function decreases faster for the network with the masternode, and that the latter's prediction shows some periodicity which leads to good predictions also outside the training domain. Note that this periodicity arises despite the frequencies being entirely wrong. My thoughts are that this might be because of the goniometric relationships between sines and cosines and multiples of their angles. For instance, the predicted frequencies for $\omega = 0.75$ and $\omega = 1.5$ correspond to approximately $\frac{1}{3}$ of the real value, while this seems to be a fraction of $\frac{1}{5}$ for $\omega = 1.75$ and $\omega = 2$. This would mean that the masternode could predict a frequency or a fraction of a frequency for a periodic function.

Next, it is necessary to verify whether the solutions show true periodicity. The definition of a periodic function $x(t)$ is that there exists some non-zero constant P for which:

$$x(t + P) = x(t) \tag{45}$$

which should count for all t. If we look at the figures above, this might be the case. However, during the iteration over values for ω, the following plots arose as well:

Looking at the predictions of the masternode outside the training domain, it seems that the solution is not automatically periodic in time. In order to be entirely sure, we should maybe look at the period associated with the predicted frequency. We know that the following relationship holds:

$$T = \frac{2\pi}{\omega} \tag{46}$$

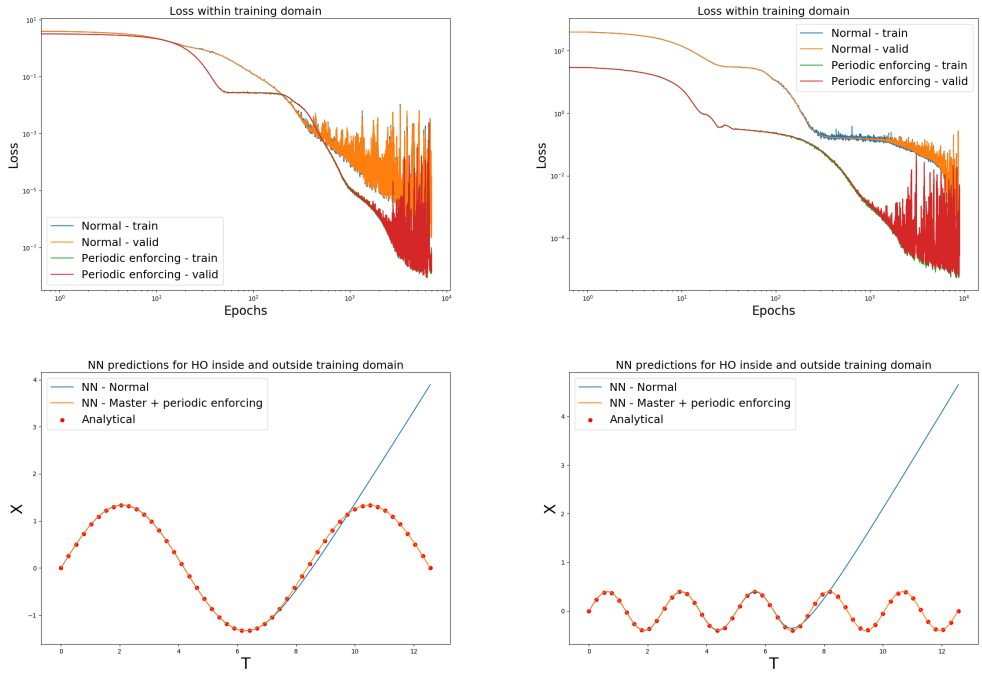

Figure 15: Loss (above) and solution (below) for omega equal to 0.75 and 2.5

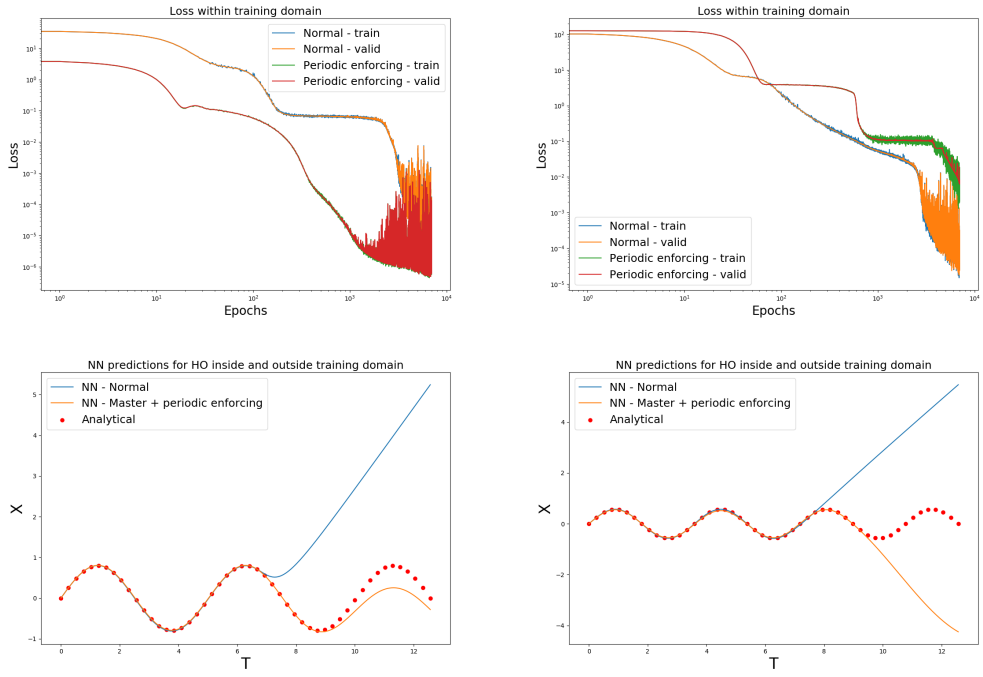

Figure 16: Loss (above) and solution (below) for omega equal to 1.25 and 1.75

In both cases above, the values of the predicted frequencies (the masternode weights) lead to $T \approx 15$ and $T \approx 17.5$. That means that the periodicity cannot be judged based on the graphs above and that the domain in t should be expanded. Let's have a look now:

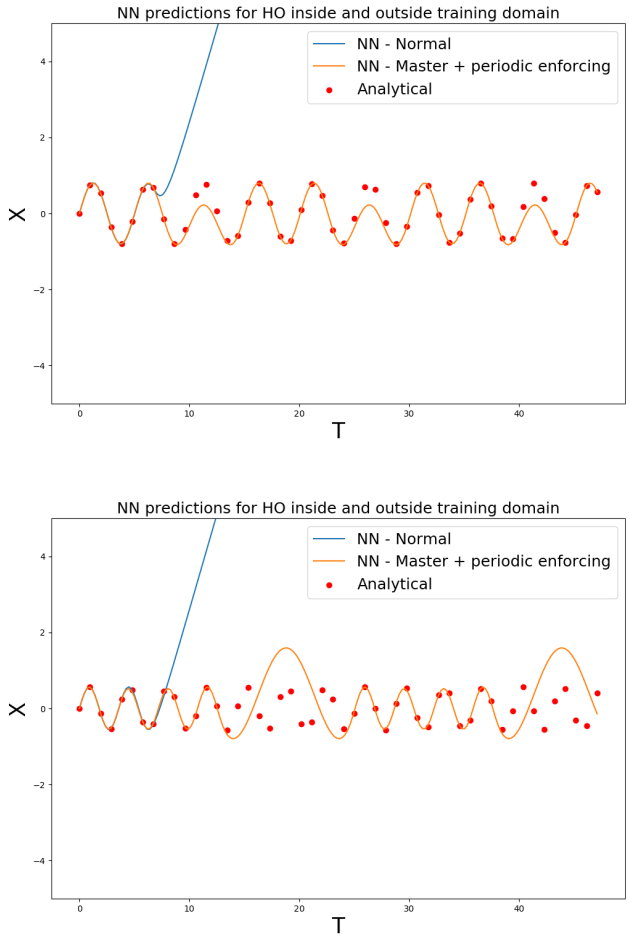

Figure 17: NN solutions far outside the training domain for omega equal to 1.25 and 1.75

These graphs show that, how bad the prediction of the frequency may be, there will always be some periodicity in time for the masternode network solution.

From this analysis, we could make some conclusions:

- The mathematical formulation of the network is intrinsically periodic with a frequency as trainable parameter.
- In the cases tested thus far, this seems to have a positive impact on the loss vs epochs evolution.
- Despite the two points above, the masternode network becomes incredibly sensitive with respect to the value of the frequency. This raises the question whether there can

be put more 'weight' on the training on this masternode weight/frequency. Should we/I think about a clever way to implement this in the loss function? Or is this not useful direction to follow?

Lastly, it is interesting to apply the same masternode and periodic initial condition enforcing to an initial value problem ODE, which shows periodicity but not in the explicit form of sines or cosines. For this, the Van der Pol equation has been chosen:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0, \tag{47}$$

In dynamics, the Van der Pol oscillator is a non-conservative oscillator with non-linear damping. The scalar parameter μ determines the strength of the non-linear damping. For μ non-zero, the equation does not have an analytical solution. The figure below illustrates the numerically computed solution for $\mu = 5$ (source).

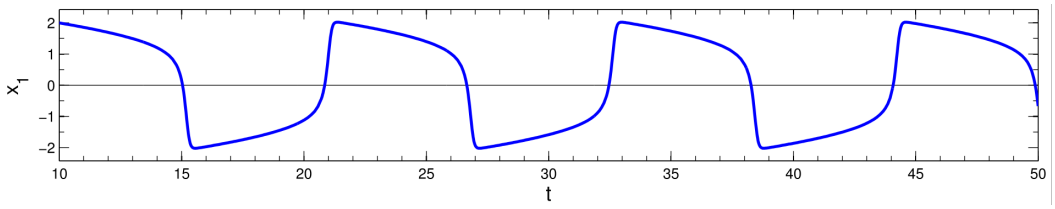

Figure 18: Solution to the Van der Pol equation for $\mu = 5$

Clearly, the solution shows periodicity in the time domain. Hence, together with the fact that the solution does not contain any goniometric functions, this is the perfect function to test the functionality of the neural network with the masternode and the periodic initial value enforcing.

The loss function now becomes:

$$L = (\ddot{\hat{x}} - \mu(1 - \hat{x}^2)\dot{\hat{x}} + \hat{x}) \tag{48}$$

Where $\hat{x}$ still comes from the same operation executed on the output of the neural network x_{NN}:

$$\hat{x} = x_0 + \frac{1}{w_0}\sin(w_0 t)v_0 + \sin^2(w_0 t)x_{NN} \tag{49}$$

I have been trying to implement this equation, but the results were surprisingly unsatisfying. Fist, the non-linearity makes the computational graphs very heavy in memory space. Second, even with 3 layers of 40 nodes, 200 points in the training domain and 2000 epochs, the results are only as good as on the figure below:

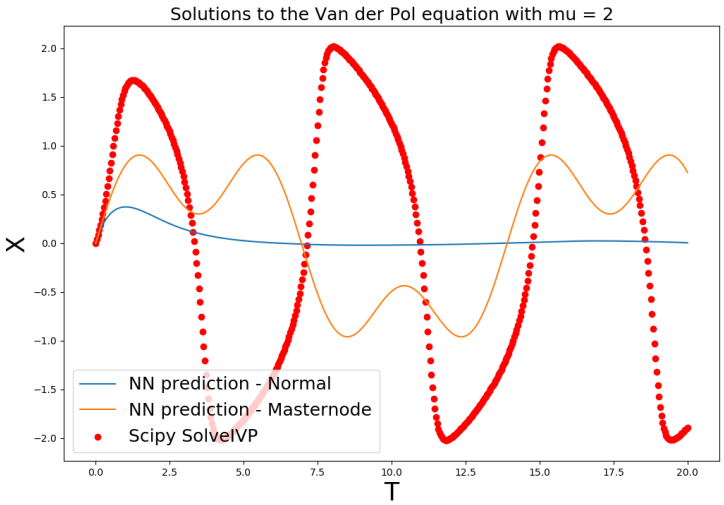

Figure 19: Prediction of the Van der Pol equation for $\mu = 2$

Hence, I'm a bit stuck moving forward. I think that, once we have figured out how to train the frequency correctly, the model performance will increase significantly.

6 Playing around with the 2D Boundary Value Problems

Lastly, I started to use Feiyu's code to solve two-dimensional boundary value problems. First, let's just solve the two dimensional Laplace equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{50}$$

The boundary conditions are the following:

$$\begin{aligned} u(x,y)\Big|_{x=0} &= \sin \pi y \\ u(x,y)\Big|_{x=1} &= 0 \\ u(x,y)\Big|_{y=0} &= 0 \\ u(x,y)\Big|_{y=1} &= 0 \end{aligned} \tag{51}$$

The analytical solution is the following:

$$u(x,y) = \frac{\sin(\pi y)\sinh(\pi(1-x))}{\sinh \pi} \tag{52}$$

Implementing this in Feiyu's code results in the following graphs:

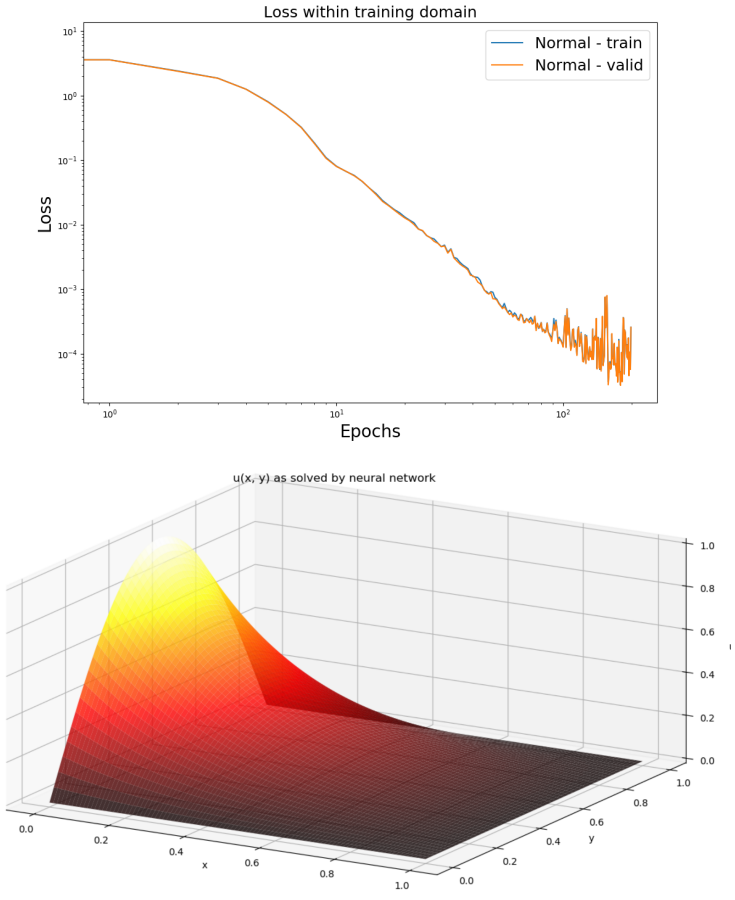

Figure 20: NN solutions of the two dimensional Laplace equation

Both the low loss for a relatively low amount of epochs and the smooth solution, with the boundary conditions seemingly met, this looks very good. This initial (brief) exploration will enable us to go further into the PDE's.

7 Conclusion

This documents has touched upon multiple topics.

First, it discusses the analytical solution of the Laplace equation in cylindrical coordinates with particular boundary conditions. The final solution consists of an infinite sum of multiplications of hyperbolics in z, sines and cosines of θ and Bessel functions of the first and second order in 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 error grows the further away from the initial condition and that the network performs very poorly outside the training domain. As deeper exploration, I evaluated the convergence of the NN with Chebyshev interpolation rather than uniform sampling. After thorough evaluation, it seems that there is no reason to believe that Chebyshev performs better, which does not mean that it is a nice additional feature to be incorporated in the code.

Next, the masternode of Pavlos has been implemented in the same NN in the hope periodic solutions would arise. With a modified initial condition operator, it turns out that periodic solutions can be derived. In some cases, when a reasonable value for the frequency is computed, this method can lead to very good performance. However, this changes drastically for slightly different values of the frequency, which was seen in the prediction of a simple linear oscillator but more strongly in the predictions of the solutions to Van der Pol equation. For the latter, it is however important to note that the non-linearity makes it hard to solve it without the masternode as well. In conclusion, I believe the masternode has some good characteristics but before we can say anything about its actual use cases, we should come up with a way to increase the importance of the frequency during training.

Lastly, a very simple PDE's, the 2D Laplace equation, has been solved using a the NN implementation of Feiyu.