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. 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, a short literature review is listed, based on my readings of last weeks.

2 Problem formulation and analytical solution

2.1 Problem formulation

As an example PDE, we will solve the two-dimensional, non-homogeneous Poisson 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.

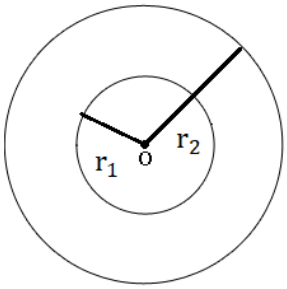

Figure 1: The considered circular domain for the PDE

The general Poisson equation is equal to:

$$\Delta u(r,\theta) = f(r,\theta) \tag{1}$$

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

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

The boundary conditions in this problem are:

$$u(r = r_1, \theta) = u(r = r_2, \theta) = 0 \tag{3}$$

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

2.2 Analytical solution of the homogeneous solution

The analytical solution of the problem formulated above is not straight-forward (for me). I therefore simplify it at this time to come up with the solution for the homogeneous equation, being equation (1) with $f = 0$.

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:

$$u(r,\theta) = R(r)T(\theta) \tag{4}$$

Let's plug this formulation into the polar formulation of the homogeneous Poisson equation:

$$\Delta v(r,\theta) = \frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial v}{\partial r}) + \frac{1}{r^2}\frac{\partial^2 v}{\partial \theta^2} = \frac{T}{r}\frac{d}{dr}(r\frac{dR}{dr}) + \frac{R}{r^2}\frac{d^2T}{d\theta^2} = 0 \tag{5}$$

Which leads to:

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

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:

$$\frac{r}{R}\frac{d}{dr}(r\frac{dR}{dr}) = \lambda = -\frac{1}{T}\frac{d^2T}{d\theta^2} \tag{7}$$

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:

$$\frac{1}{T}\frac{d^2T}{d\theta^2} = -\lambda \tag{8}$$

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:

$$T_0(\theta) = A_0 + B_0\theta \qquad j = 0 \tag{9}$$

$$T_j(\theta) = A_j cos(\sqrt{\lambda_j}\theta) + B_j sin(\sqrt{\lambda_j}\theta) \qquad j = 1, 2... \tag{10}$$

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:

$$\frac{r}{R}\frac{d}{dr}(r\frac{dR}{dr}) = \lambda_j \tag{11}$$

$$r^2R'' + rR' - \lambda_j R = 0 \tag{12}$$

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

$$R_0(r) = C_0 + D_0 \ln r \tag{13}$$

$$R_j(r) = C_j r^{\sqrt{\lambda_j}} + D_j r^{-\sqrt{\lambda_j}} \qquad j = 1, 2... \tag{14}$$

Using the results from (9), (10), (13) and (14) and the separability of variables, we can write the overall, general solution for $u(r, \theta)$:

$$u(r,\theta) = A_0 + B_0\theta + C_0 + D_0 \ln r + \sum_{j=1}^{\infty}(A_j r^{\sqrt{\lambda_j}} + C_j r^{-\sqrt{\lambda_j}}) \cos\sqrt{\lambda_j}\theta + \sum_{j=1}^{\infty}(B_j r^{\sqrt{\lambda_j}} + D_j r^{-\sqrt{\lambda_j}}) \sin\sqrt{\lambda_j}\theta \quad (15)$$
$$j = 1, 2, 3...$$

With respect to this general solution, I have some notes/difficulties moving forward:

- Last week, we proposed as boundary condition $u(r = r_1) = u(r = r_2) = 0$. As far as I understand it, I don't think this is possible in this mathematical formulation.
- Note that this is only the solution to the homogeneous Poisson equation. I have looked at different options to solve the non-homogeneous version.

  (i) Option 1 (link - page 5, 2.4) first looks for eigenfunctions of the Poisson operator, writes the solution in terms of the Fourier series and then computes the coefficients such that the non-homogeneous equality with function f is met. I have however not been able to formulate the problem in polar coordinates in such a way that I could compute the eigenfunctions.

  (ii) Option 2 (link - page 3, 1.6) offers a way to solve the transient, non-homogeneous heat equation by solving for the homogeneous solution and scaling this using the Duhamel principle (link). I have however not figured out if this also counts for non-time dependent/second order derivatives.
- Maybe the problem mentioned as first bullet point is not a problem when solving for the non-homogeneous solution.
- Maybe this is not worth the trouble and I should solve it numerically (?)

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:

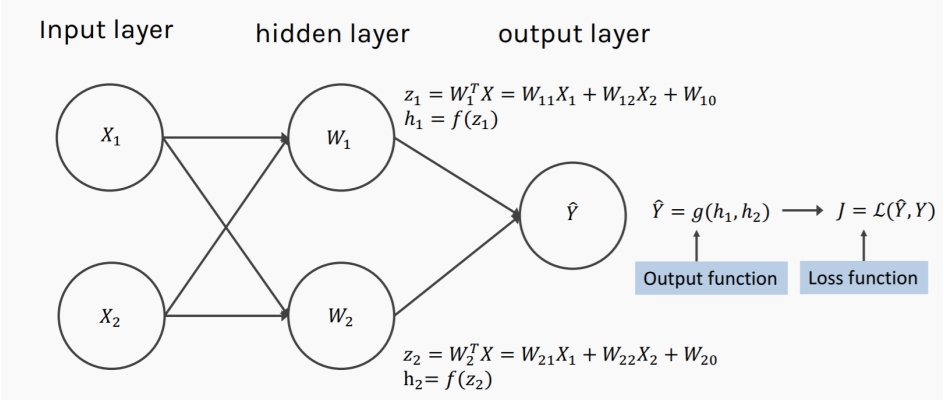

Figure 2: Illustration of a simple Artificial Neural Network (ANN), ref. to CS209a

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:

$$z_1 = W_1^{(1)} \cdot X = \begin{bmatrix} w_{11}^{(1)} \\ w_{12}^{(1)} \\ w_{10}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ 1 \end{bmatrix} = w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)} \tag{16}$$

$$z_2 = W_2^{(1)} \cdot X = \begin{bmatrix} w_{21}^{(1)} \\ w_{22}^{(1)} \\ w_{20}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ 1 \end{bmatrix} = w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)} \tag{17}$$

These nodes will now be 'activated' as follows:

$$h_1^{(1)} = sin(z_1) = sin(W_1^{(1)} \cdot X) = sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \tag{18}$$

$$h_2^{(1)} = sin(z_2) = sin(W_2^{(1)} \cdot X) = sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) \tag{19}$$

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

$$\tilde{u}_{NN} = W^{(o)} \cdot H^{(1)} = \begin{bmatrix} w_1^{(o)} \\ w_2^{(o)} \\ w_0^{(o)} \end{bmatrix} \cdot \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ 1 \end{bmatrix} = w_1^{(o)} h_1^{(1)} + w_2^{(o)} h_2^{(1)} + w_0^{(o)} \tag{20}$$

Or in terms of the input variables, this becomes:

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

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

$$\tilde{u}_{NN} = \sum_{i=1}^{N} w_i^{(o)} sin(w_{i1}^{(1)} X_1 + w_{i2}^{(1)} X_2 + w_{i0}^{(1)}) + w_0^{(o)} = \sum_{i=1}^{N} w_i^{(o)} sin(W_i^{(1)} \cdot X) + w_0^{(o)} \tag{22}$$

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

$$\sin(\alpha + \beta) = \sin\alpha \cos\beta + \cos\alpha \sin\beta \tag{23}$$

$$\cos(\alpha + \beta) = \cos\alpha \cos\beta - \sin\alpha \sin\beta \tag{24}$$

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

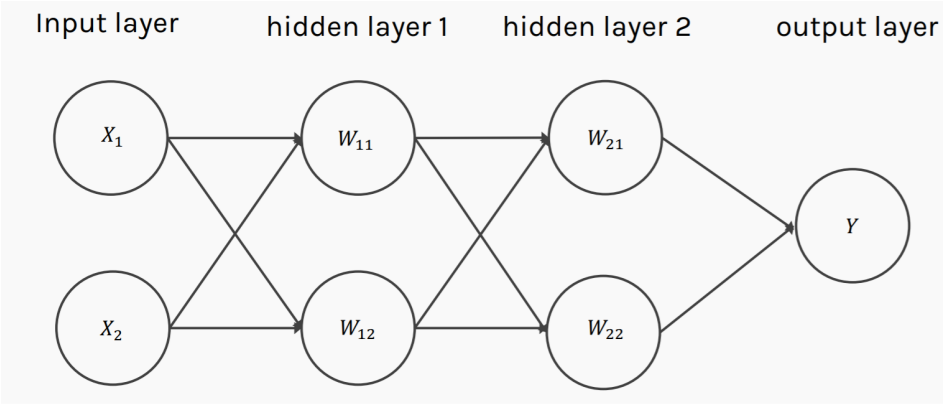

Figure 3: Illustration of a simple Artificial Neural Network (ANN) with 2 layers, ref. to CS209a

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:

$$H^{(1)} = \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ 1 \end{bmatrix} = \begin{bmatrix} sin(W_1^{(1)} \cdot X) \\ sin(W_2^{(1)} \cdot X) \\ 1 \end{bmatrix} = \begin{bmatrix} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \\ sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) \\ 1 \end{bmatrix} \tag{25}$$

For the second layer, this now becomes:

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

$$= \begin{bmatrix} sin(w_{11}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) + w_{12}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{10}^{(2)}) \\ sin(w_{21}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) + w_{22}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{20}^{(2)}) \\ 1 \end{bmatrix} \tag{26}$$

The output can then be written as:

$$\begin{aligned} \tilde{u}_{NN} &= W^{(o)} \cdot H^{(2)} \\ &= w_1^{(o)} sin(w_{11}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \\ &+ w_{12}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{10}^{(2)}) \\ &+ w_2^{(o)} sin(w_{21}^{(2)} sin(w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{10}^{(1)}) \\ &+ w_{22}^{(2)} sin(w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{20}^{(1)}) + w_{20}^{(2)}) \\ &+ w_0^{(o)} \end{aligned} \tag{27}$$

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 Conclusion

From the last paragraph, it is clear that the neural network actually computes a complicated linear combination of sinusoidal functions of the two inputs. From the first section, we realize that the general, analytical solution of the PDE, obtained through the separation of variables, is in fact a sum of sinusoidal terms. Bringing these two conclusions together - while acknowledging that no general proof has been provided here - it could therefore make mathematical sense that using a sinusoidal activation function leads to a faster convergence in predicting the solution of the PDE u. This initial exploration thus forms a solid foundation for next steps.

Independent study NNs + Symmetry: summary initial Literature review

1. *"Learning SO(3) Equivariant Representations with Spherical CNN's"*

- Addresses the 3D rotation equivariance in CNN's. This means that recognition still works when objects are rotated.
- SO(3) is the group of all rotations about the origin of 3D Euclidean space.
- Currently, CNN's are great with translations! Other nuisances are normally addressed with data augmentation. Equivariant CNN's allow rotations as well, the paper proposes the first NN with spherical convolutions.
- Pooling = a layer in CNN that reduces the spatial size, summarizing multiple features. Here a spectral pooling is used instead of spatial pooling, as this retains the equivariance. A special smoothing approach is used to select only a few 'anchor frequencies', which makes the number of weights independent of the input resolution.
- Uses 'Spherical harmonics', which comes down to the use of Spherical Fourier Transform (SFT). The convolution of signal f with a filter h is computed by first using SFT to f and h, computing their point-wise multiplication, and then using the inverse SFT. For this, they use equiangular samples.
- They parametrize the filters in the spectral domain and obtain localized filters by parametrizing the spectrum with anchor frequencies.
- They use spectral pooling, as this preserves equivariance.
- 64 citations in a year, seems like a very big paper.

2. *"Rotation equivariant vector field networks"*

- Tackles the same problem as previous paper, being developing a CNN that has rotational equivariance by design. Here, the built Rotation Equivariant Vector Field Networks (RotEqNet). Each convolutional filter is applied at multiple orientations and returns a vector field representing magnitude and angle of the highest scoring orientation at every spatial location.
- They make a distinction between rotation equivariance (rotating input, same rotation in output), invariance (same classification score rotated or not) and covariance (some relationship is present between rotated input and output).
- Traditional CNN's are translation equivariant thanks to the nature of their convolution operator.
- Trade-off. Doing R rotations of each filter leads to too large model size. Could also only retain max value, but then no idea about orientation. So, combo: keeping max value as 2D vector! This is called orientation pooling or OP.
- Two options for rotational equivariance: transform the representation or rotate the filters. Challenge of the first is loss of relative orientation of objects with respect to surroundings (not always relevant). Of the second, is dimensionality of the model. Traditional trade-off of the latter comes down to depth of network vs number of orientations included. RotEqNet bypasses this compromise.
- Note that data augmentation comes down to rotate the images.
- Segmentation problem in general comes down to identifying what part of the image belongs to what segment.

- In all examples provided by the paper, RotEqNet achieves better performance than previous work with significantly less parameters.

3. *"Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds"*

- A point cloud is a set of data points in space. Typically, this is produced by 3D scanners.
- Impressive that this works for high-dimensional tensors! Their filters are built from spherical harmonics. Each layer accepts as input scalars, vectors and higher-order tensors.
- Contains mathematically rigorous understanding and proof of equivariance.
- They use point convolution, which means that the convolution is executed on each point taking all other points and their relative location as input.
- Spherical harmonics = functions that are equivariant in SO(3).
- A cool application they show is the calculation of acceleration vectors of point masses (under Newtonian gravity and the moment of inertia tensor for every mass). Their network leads to perfect agreement with the exact solution.
- Another cool thing is their prediction of the location of a randomly removed point from a point cloud.
- Code is directly useable from github.
- 45 citations in a year, seems like a very big paper.

4. *"Harmonic Networks: Deep Translation and Rotation Equivariance"*

- From 2017, a year earlier than the other papers I have the feeling.
- They use Harmonic networks or H-nets to embed equivariance in CNN's, using circular harmonics.
- Filters are steerable if they can be constructed at any rotation as a finite, linear combination of base filters.
- Again, 'learning generalized transformations' from the input data is considered as alternative of playing with filters. But this leads to less interpretable and reliable results.
- H-nets hard-bake 360°-rotation invariance into their feature representation, by constraining the convolutional filters of a CNN to be from the family of circular harmonics. Proof is in appendix.
- They use filters expressed in polar form, having a rotation order, radial profile and phase offset term, the two latter of which are learned during training. Note that this means that the filters are complex-valued.
- They use cross-correlation, which is an alternative for convolution.
- They claim a better interpretability of the feature maps.
- They have a Tensorflow implementation ready on their website.

5. *"On the generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups"*

- Gives a rigorous, theoretical treatment of convolution and equivariance in NNs with respect to the action of any compact group.
- Topology is the branch of mathematics concerned with the properties of a geometric object that are preserved under continuous deformations. A compact group is a topological group whose topology is compact being closed and bounded.
- Defines a general formulation of a convolution for a compact group and claims that a FFNN is equivariant to the action of any compact group if and only if each of its layers implements a general form of their convolution. If this condition is met, the NN is called a G-CNN, with G representing the compact group.
- Interestingly, the claim that if a network is fully equivariant, the network must be convolutional with respect to the specific compact group.
- Mentions an example of NN learning from graphs, and specifically so-called Message-Passaging NNs, or MPNNs, claiming that is also a generalized CNN.

6. *"Hamiltonian Neural Networks for solving differential equations"*

- Paper by Sondak, Protopapas, Mattheakis and Dhogra
- They want to solve ODE's governing dynamical systems. The model is data-free and unsupervised.
- Hamiltonian mechanics is a theory developed as a reformulation of classical mechanics and predicts the same outcomes as non-Hamiltonian classical mechanics
- They embed the Hamiltonian equations into the loss function of the NN, therefore conserving energy. Also, they use the sine as activation function, resulting in less iterations to reach the same performance than the sigmoid. This has 'global support similar to the Fourier Series' and introduces periodic, and empirically considered as equal, minima, which makes it easier to converge to a local minimum.