Solution of PDEs using neural networks
Individual project III

Miskei Ferenc István

22 December, 2022

1 Introduction

Solving partial differential equations (PDEs) is an essential field in applied sciences such as physics, chemistry and engineering. However, finding exact solutions to PDEs that emerge in applications is often impossibly difficult, and we therefore must resort to the use of numerical methods.

A relatively new approach to solving PDEs is the application of neural networks (NNs or neural nets), which has a number of benefits. Multilayer feedforward neural networks can be used as universal approximators to functions defined on compact subsets of finite dimensional real spaces (see [6], [1]). Furthermore, once a neural net is trained to give an approximate solution to a PDE using some input parameters (such as a boundary condition, source function values, etc.), the solution of further PDEs of the same type is reduced to evaluating the NN on different parameters.

Of course, one obvious issue with training a neural network to solve a PDE is that a large number of already solved PDEs of the same type would be required for the training phase, which could render the whole procedure useless. A new idea within the established framework is the use of Green functions of the differential operator instead. [3]

This project builds upon a previous one that focused on Laplace's equation with Dirichlet boundary condition on a nontrivial compact polygon utilizing the Green functions of the two dimensional Laplacian (Δ). [2] Based on this, so far we have managed to generalize said approach further to a different problem: Poisson's equation with Dirichlet type boundary condition, and another one (Laplace's equation with non-linear boundary condition) is currently being implemented.

2 Introducing the method: Laplace's equation with Dirichlet boundary condition

2.1 Definition of the setup

The starting point of this project is the following working algorithm.
Let $\Omega \subset \mathbb{R}^2$ be an open set, as shown in Figure 1, and $g \in C(\partial\Omega, \mathbb{R})$. Consider the following boundary value problem:

$$\begin{cases} \Delta u(\mathbf{x}) = 0, & \text{if } x \in \Omega, \\ u(\mathbf{x}) = g(\mathbf{x}), & \text{if } x \in \partial\Omega. \end{cases} \tag{1}$$

We are interested in the values $\{u(\mathbf{x}_i)\}_{i=1}^{I}$, where $X = \{\mathbf{x}_i\}_{i=1}^{I}$ are in this case the $I = 3$ 'dark orange' points inside the domain. To approximate $\{u(\mathbf{x}_i)\}_{i=1}^{I}$, define the following auxiliary sets:

$$\{Y = \mathbf{y}_j\}_{j=1}^{J} \subset \operatorname{ext}\Omega \quad Z = \{\mathbf{z}_k\}_{k=1}^{K} \subset \partial\Omega$$

"border training points" and "circumference training points".

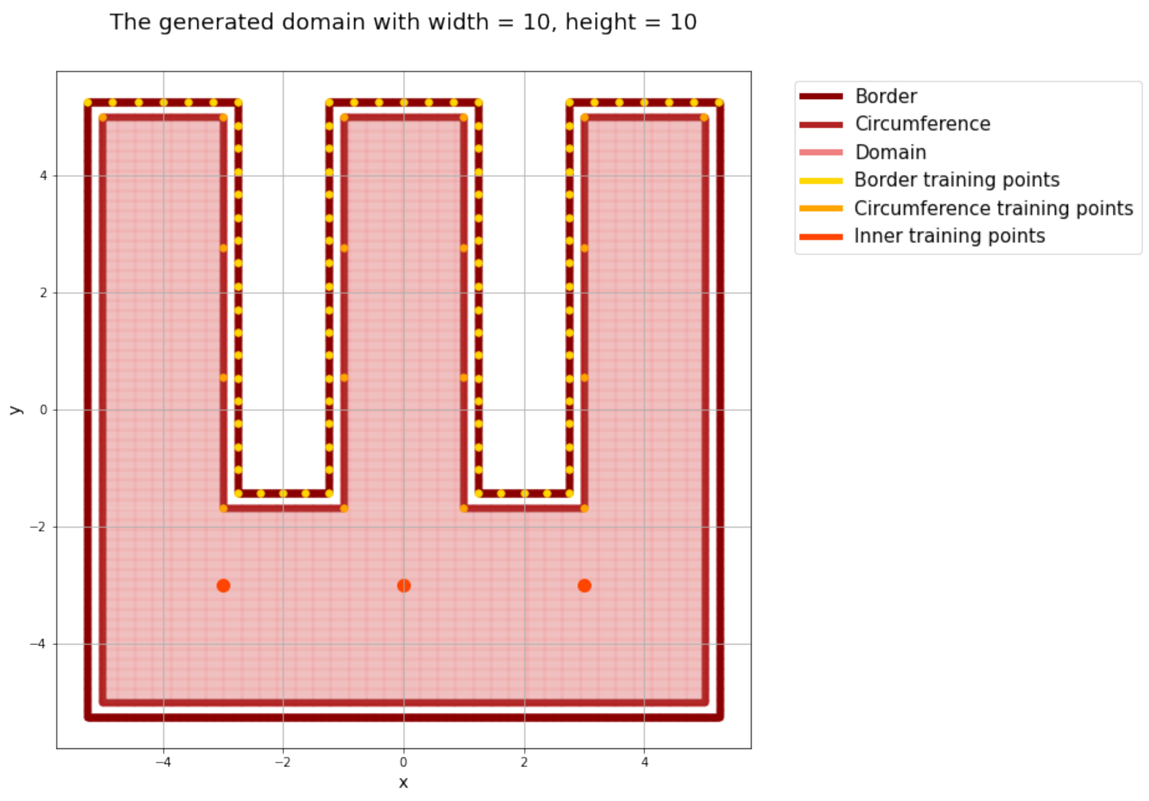


Figure 1: Domain of the first problem

The Green function for the two dimensional Laplacian operator is $G(\mathbf{x}, \mathbf{y}) = -\frac{1}{2\pi} \ln \|\mathbf{x} - \mathbf{y}\|_2$. Now, let us train a neural network – whose architecture will be specified later – such that

$$\text{NN} : \underbrace{\big(G(\mathbf{z}_k, \mathbf{y}_j)\big)_{k=1}^{K}}_{\in \mathbb{R}^K} \mapsto \underbrace{\big(G(\mathbf{x}_i, \mathbf{y}_j)\big)_{i=1}^{I}}_{\in \mathbb{R}^I} \quad (\forall \mathbf{y}_j \in Y). \tag{2}$$

Then, we define the numerical approximation as

$$\tilde{u} \colon \mathbb{R}^K \to \mathbb{R}^I, \quad (\tilde{u}(\mathbf{x}_i))_{i=1}^{I} := \text{NN}\left((g(\mathbf{z}_k))_{k=1}^{K}\right) \tag{3}$$

What this means is that we shift J copies of the Green function such that their singularities are outside of the domain Ω, evaluate them at the boundary points $\{\mathbf{z}_k\}_{k=1}^{K}$ and the inner points of interest $\{\mathbf{x}_i\}_{i=1}^{I}$. The NN is trained such as to return the function values in the inner points given the function values on the boundary.

In another words, the NN learns to predict the values inside the domain given the values on the boundary, which is really the question in (1).

This is essentially an interpolation problem, where we impose $\Delta u = 0$ by only picking functions that already satisfy this criterion (specifically the Green functions of Δ). An interesting question for further research – perhaps as part of my complete thesis – is how the inclusion of other harmonic functions in this setup influence the properties of the solutions obtained this way.

2.2 The architecture of the NN and results

[2] approached the above described problem as follows. The NN trained as in (2) will clearly learn to approximate a specific linear map from $\mathcal{L}(\mathbb{R}^K, \mathbb{R}^I)$. [2] sets up an NN without any kind of biases or non-linearities. Not using any non-linearities is a sensible step, given that (1) is a linear boundary value problem. The exclusion of biases of course means that a homogeneous Dirichlet type boundary condition implies that $u \equiv 0$. This is of course true in the case of (1).

All the tests were done with $I = 3$, $J = 95$, $K = 18$, train-test split ratio = 0.9, batch size = 32 (which is $\approx 15\%$ of the total number of data) and loss function = MSE. Further parameters of the NN include the optimizer method (and its own specific parameters), learning rate and number of epochs.

Their approach achieves the following numerical estimation properties with respect to these three variables: choice of opitimizer, learning rate and number of epochs.

The exact values to be estimated are $u(\mathbf{x}_1) = -21, u(\mathbf{x}_2) = 0$ and $u(\mathbf{x}_3) = 39$, and the relative error is calculated as:

$$e_p = \frac{\|\big(u(\mathbf{x}_1) - \tilde{u}(\mathbf{x}_1), u(\mathbf{x}_2) - \tilde{u}(\mathbf{x}_2), u(\mathbf{x}_3) - \tilde{u}(\mathbf{x}_3)\big)\|_p}{\|\big(u(\mathbf{x}_1), u(\mathbf{x}_2), u(\mathbf{x}_3)\big)\|_p}$$

The following table includes the results of some of the experiments:

optimizer	learning rate	epochs	e_1	e_2	e_∞
Adam	0.1	1,000	0.7358	0.7345	0.7851
Adam	0.01	1,000	8.4109×10^{-2}	7.8732×10^{-2}	7.3069×10^{-2}
Adam	0.01	5,000	6.9011×10^{-2}	7.0638×10^{-2}	7.5214×10^{-2}
Adam	0.01	10,000	7.9641×10^{-2}	7.2514×10^{-2}	7.5266×10^{-2}
Adam	0.001	1,000	100+	100+	100+
Adam	0.001	5,000	7.005×10^{-3}	7.285×10^{-3}	7.6783×10^{-3}
Adam	0.001	10,000	5.032×10^{-3}	4.342×10^{-3}	4.018×10^{-3}
SGD	0.001	1,000	1.2421	0.9722	0.6726
SGD	0.001	5,000	1.5755×10^{-2}	1.4383×10^{-2}	1.4402×10^{-2}
SGD	0.001	10,000	1.1992×10^{-2}	1.21523×10^{-2}	1×10^{-2}

Of course, there are much faster and more accurate deterministic algorithms to determine the coefficients of 52×3 a matrix.

As we can see, the model does in fact seem to converge to the desired solution, even if this convergence is not optimal yet. Manuscript [3] shows that the method would indeed converge.

Theorem 1 *Suppose that the points $Y = \{\boldsymbol{y}_j\}_{j=1}^J$ equally spaced around the domain, and their distance from the domain is above a particular positive constant. Then, the approximation*

$$\tilde{u}(\mathbf{x}) := \sum_{j=1}^{J} a_j G(\mathbf{x} - \mathbf{y}_j) := \sum_{j=1}^{J} a_j G_{\mathbf{y}_j}(\mathbf{x})$$

provides the convergence rate

$$\min_{(a_1,\ldots,a_N)^T \in \mathbb{R}^J} \left\{ \sup_{\mathbf{x}\in\Omega} |u(\mathbf{x}) - \tilde{u}(\mathbf{x})| + \sup_{\mathbf{x}\in\Omega} |\nabla(u(\mathbf{x}) - \tilde{u}(\mathbf{x}))| \right\} \lesssim h, \tag{4}$$

We also have

$$\min_{(a_1,\ldots,a_N)^T\in\mathbb{R}^J} \|u-\tilde{u}\|_{H^1(\Omega)} \lesssim h. \tag{5}$$

One of the goals of this project is to make empirical measurements in the case of different PDEs with similar methods.

3 Poisson's equation with Dirichlet type boundary condition

After our investigation of Laplace's equation, we turn our attention towards one of the more obvious directions of generalization: Poisson's equation. (Of course, we could generalize in the direction of increasing the dimension of the domain as well, or changing the type of the boundary condition, the latter of which we are going to do in Section 4.)

The first remark is of course that we probably cannot omit the use of functions whose Laplacian vanishes everywhere inside the domain, since the linear combination of such functions also has a Laplacian that vanishes everywhere inside the domain.

3.1 Definition of the setup

Let $\Omega \subset \mathbb{R}^2$ be an open set, as shown in Figure 2, $f \in C(\Omega,\mathbb{R})$ and $g \in C(\partial\Omega,\mathbb{R})$. Consider the following boundary value problem:

$$\begin{cases} \Delta u(\mathbf{x}) = f(\mathbf{x}), & \text{if } x \in \Omega, \\ u(\mathbf{x}) = g(\mathbf{x}), & \text{if } x \in \partial\Omega. \end{cases} \tag{6}$$

We are interested in the values $\{u(\mathbf{x}_i)\}_{i=1}^I$, where $X = \{\mathbf{x}_i\}_{i=1}^I$ are the $I = 15$ 'spring green' points in the inside. To approximate these values, let us define the following auxiliary sets:

$$Y = \{\mathbf{y}_j\}_{j=1}^J \subset \operatorname{ext}\Omega \quad Z = \{\mathbf{z}_k\}_{k=1}^K \subset \partial\Omega \quad W = \{\mathbf{w}_l\}_{l=1}^L.$$

Here, the "border distance" is 0.005, meaning that every point in Z is 0.005 units away from $\partial\Omega$. It might appear that the singularities of the Green functions are too close to the boundary this way, but in fact, $-\frac{1}{2\pi}\ln 0.005 \approx 0.8433$, meaning that there is barely any elevation there. The diameter of the domain is $5\sqrt{2}$, so the variance of the Green function over the two most distant points is approximately 1.1546.

The exact solution chosen to be

$$u(x,y) = \underbrace{xy\sin(x+y^2)}_{\Delta(\ldots):=f} + \underbrace{x^5 - 10x^3y^2 + 5xy^4}_{\Delta(\ldots)=0},$$

and the exact solutions at the points of X are

$$\big(u(\mathbf{x}_i)\big)_{i=1}^{15} = \big(-468.34, -420.25, -224.78, 31.685, 215.65, 267.86, 225.76, 149.92,$$
$$73.672, 8.265, -49.81, -111.43, -188.82, -289.09, -398.99\big),$$

which visibly has a significant variance.

Now, the task of the numerical approximation is to find a map A, for which

$$A\Big(\big(f(\mathbf{w}_l)\big)_{l=1}^L, \big(g(\mathbf{z}_k)\big)_{k=1}^K\Big) \approx \big(u(\mathbf{x}_i)\big)_{i=1}^I.$$

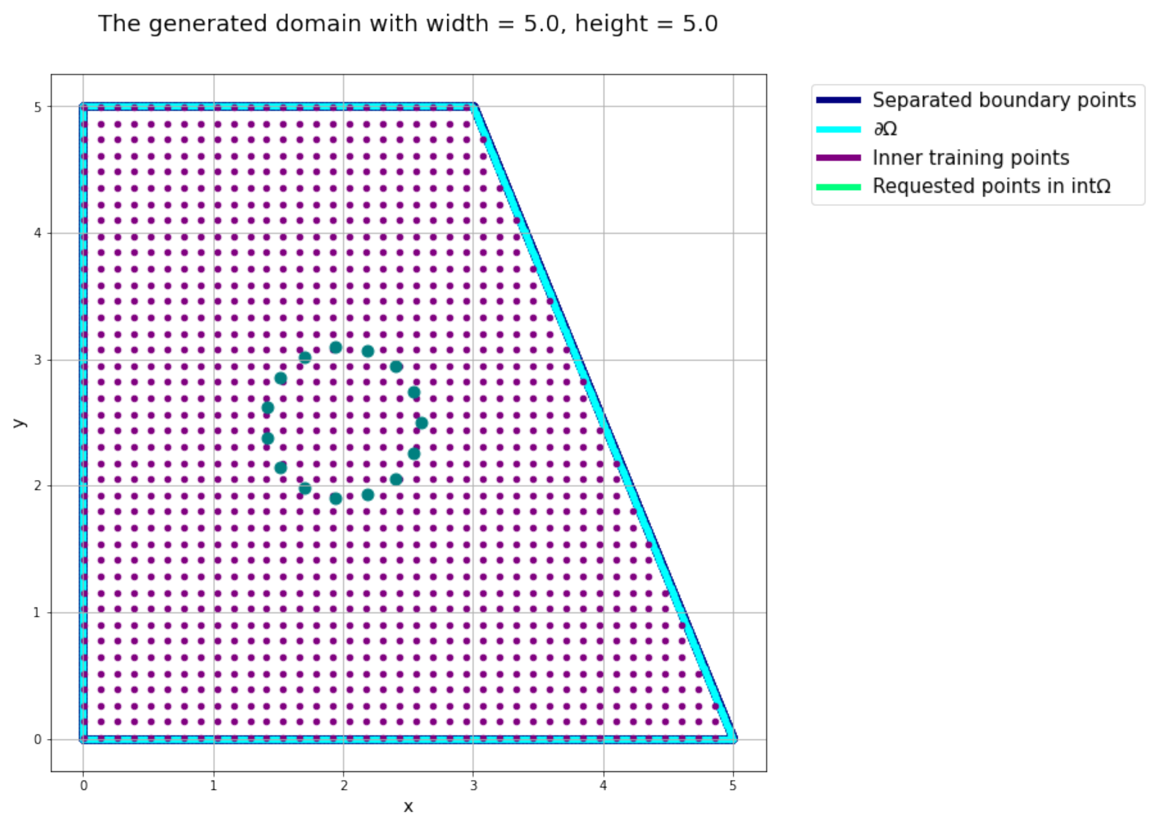


Figure 2: Domain of the second problem

What this means is that given the source values $f(\mathbf{w}_l)$ in the interior of the domain and the boundary values $g(\mathbf{z}_k)$, the mapping A is able to predict the function values $u(\mathbf{x}_i)$ in the interior points.

Now, it is clear that due to the linearity of (6), any solution can be decomposed as $u = u_1 + u_2$, where u_1 and u_2 satisfy

$$\begin{cases} \Delta u_1(\mathbf{x}) = f(\mathbf{x}), & \text{if } x \in \Omega, \\ u_1(\mathbf{x}) = 0, & \text{if } x \in \partial\Omega. \end{cases} \quad (7) \qquad \begin{cases} \Delta u_2(\mathbf{x}) = 0, & \text{if } x \in \Omega, \\ u_2(\mathbf{x}) = g(\mathbf{x}), & \text{if } x \in \partial\Omega. \end{cases} \quad (8)$$

This suggests that we might obtain a solution by decomposing A into $A(\mathbf{u}, \mathbf{v}) = A_1(\mathbf{u}) + A_2(\mathbf{v})$ as well, where A_1 approximates u_1 and A_2 approximates u_2. Of course, both A_1 and A_2 would be linear maps (from different spaces) to $\mathbb{R}^I$.

Although this remark has the opportunity to reduce the numerical costs in the case of linear PDEs, our ultimate goal is to give a method that converges in the case of non-linear PDEs also. Perhaps we might chase this line of thought in the future (maybe as part of my complete thesis). For now, we build the NN such that its application should generalize easier to non-linear PDEs.

3.2 The architecture of the NN and preliminary results

Now, we take the input-output pairs as before:

$$\mathrm{NN}: \underbrace{\left(\mathbf{0}, G(\mathbf{z}_k, \mathbf{y}_j)_{k=1}^{K}\right)}_{\in \mathbb{R}^{L+K}} \mapsto \left(G(\mathbf{x}_i, \mathbf{y}_j)\right)_{i=1}^{I} \quad (\forall \mathbf{y}_j \in Y), \tag{9}$$

however, we also need to take input-output pairs for which the Laplacian in the inside is nonzero. Our choice fell on the inverse multiquadric radial functions of the form $\Psi(r) := (1+r^2)^{-1/2}$, as suggested by [5]. Here, $\Delta_{(x,y)}\psi(r) = \frac{2+r^2}{(1+r^2)5/2}$. It is important to note that this function has no singularities at all, and only has a total variation of 1, while its Laplacian has a total variation of 2.

The input-output pairs belonging for this radial basis function take the form

$$\mathrm{NN}: \left(\left(\Delta\psi(\|\mathbf{w}_l - \mathbf{y}_j\|_2)\right)_{l=1}^{L}, \left(\psi(\|\mathbf{z}_k - \mathbf{y}_j\|_2)\right)_{k=1}^{K}\right) \mapsto \left(\psi(\|\mathbf{x}_i - \mathbf{y}_j\|_2)\right)_{i=1}^{I} \quad (\forall \mathbf{y}_j \in Y), \tag{10}$$

At first, we picked a relatively small number of points: $K = 302$ boundary points and $L = 312$ inner training points. Since this will mean a total number of $I \cdot (L+K) = 9{,}210$ parameters in the model, and we will attempt to find $A \in \mathcal{L}(\mathbb{R}^{L+K}, \mathbb{R}^{I})$ it made sense to try and match the size of the training data to around 9,000. Therefore, we picked 32 separated boundary points for every boundary points, plus two extra points where the gap between consecutive elements of Z would have been too large: $J = 9662$.

Attempting the same NN parameters as the one that resulted in the most succesful (one dense linear layer, learning rate $= 0.001$ no bias, 10,000 epochs, batch size $= 3,000$ – which is about 15% of the training data set) as in the first case yields a very noisy learning curve and poor numerical results (about 50% relative error)

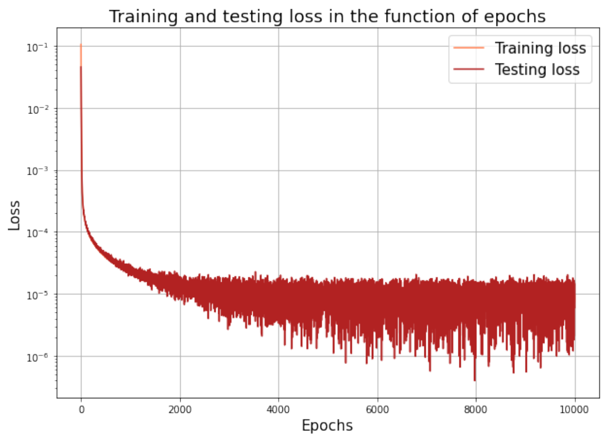


Figure 3: First trial on Poisson's equation

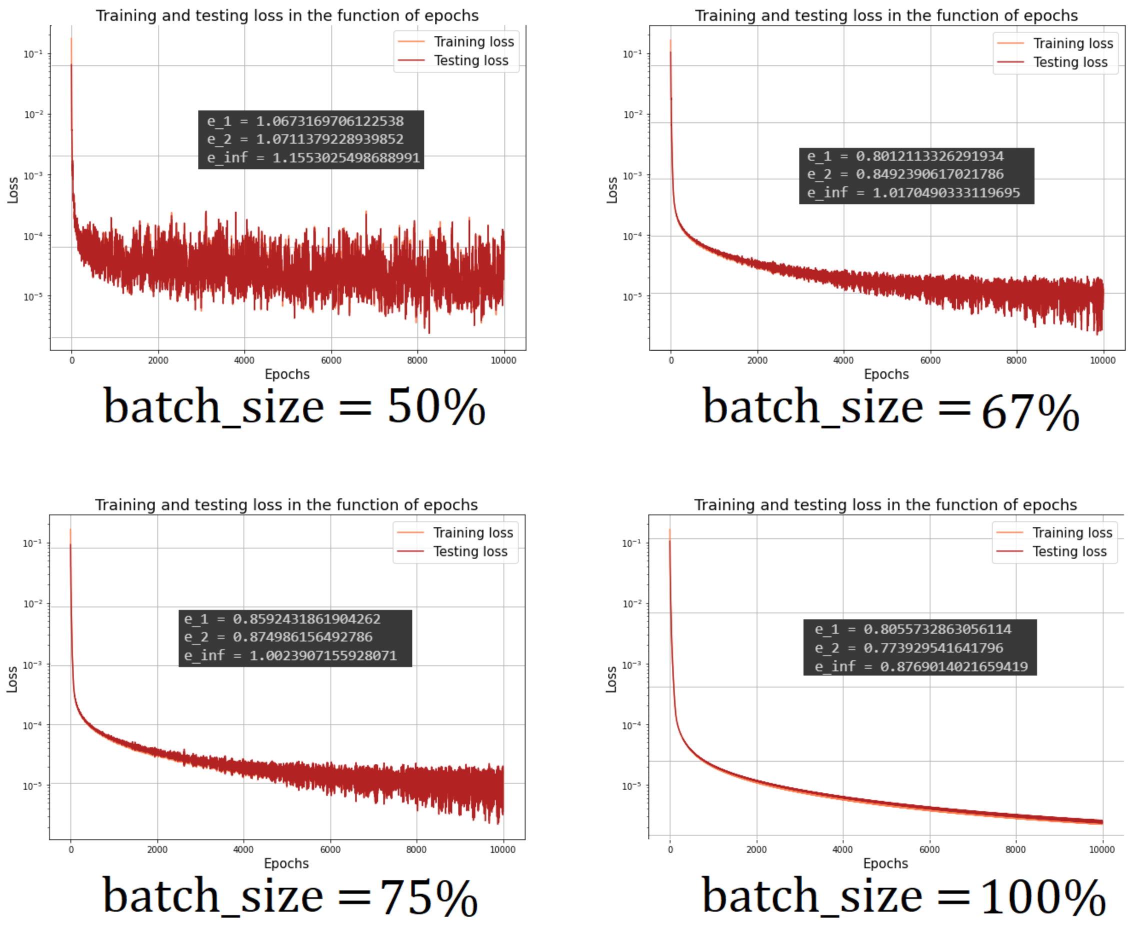
There are directions in which one might try to improve the result of this experiment:

- increasing batch size to decrease oscillations,
- changing the architecture of the NN to enable more parameters, thus increasing the complexity of the model.

- increasing the size of the training data to improve convergence speed,
- increasing the number of separated boundary points and decreasing the border distance to perhaps enable a better overall attainable accuracy.

3.3 Increasing the batch size

The next few figures show results of experiments with the exact same parameters only with the batch size changed. Note that each of these experiments have been tried a few times (3-4), and these are typical results in each case.



```
e_1 = 1.0673169706122538
e_2 = 1.0711379228939852
e_inf = 1.1553025498688991
```

```
e_1 = 0.8012113326291934
e_2 = 0.8492390617021786
e_inf = 1.0170490333119695
```

```
e_1 = 0.8592431861904262
e_2 = 0.874986156492786
e_inf = 1.0023907155928071
```

```
e_1 = 0.8055732863056114
e_2 = 0.773929541641796
e_inf = 0.8769014021659419
```

Figure 4: Increasing the batch sizes

It appears that increasing the batch size has a noticeable effect at smoothing the learning curves, and slightly decreasing the time necessary for the learning process to be completed. Moreover, it does seem like that there is a slight tendency of decreasing accuracy with increasing the batch size, however, this tendency cannot be deemed significant, as such high relative error rates are unacceptable anyway.

Considering the apparent slight decrease in accuracy, however, we will be attempting to eliminate the roughness of the learning curves by further decreasing the learning rate, while setting the batch size at about 50% of the total data set.

3.4 Increasing the number of measurement points and the complexity of the architecture

At this high levels of relative error, it is exceedingly difficult to say anything definitive with respect to the accuracy of the model. Therefore, we now turn our attention to improving the accuracy as much as possible. This will be done on two fronts:

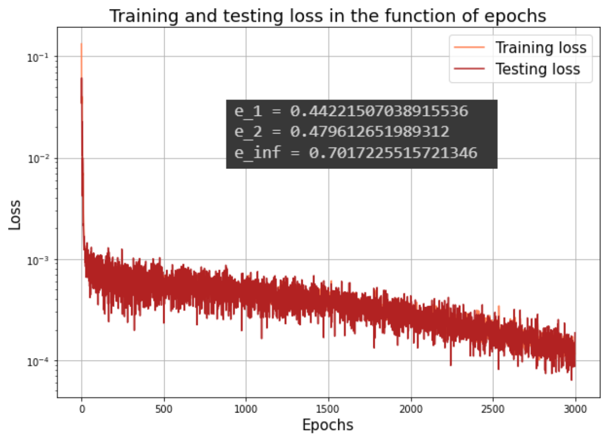
- increasing J and decreasing the border distance,
- increasing the complexity of the model: including another hidden layer, including biases.

Furthermore, we decrease the learning rate to 0.0001 in order to attain smoother learning curves. In the following experiments, the data has the following parameters:

- border distance = 0.0005 ($\max_{\mathbf{z}\in Z,\mathbf{y}\in Y} |G(\mathbf{z},\mathbf{y})| = -\frac{1}{2\pi}\ln 0.0005 \approx 2.42$)
- $J = 38{,}577$ separated boundary points, ($\mathrm{dist}(\mathbf{z}_j, \mathbf{z}_{j+1}) \approx 2^{-11} \approx 0.000488 \approx$ border distance,
- $K = 1{,}206$ boundary points, and
- $L = 1{,}264$ inner training points.

Given these parameters, we have $2J = 77{,}154$ input-output pairs from $\mathbb{R}^{2470} \times \mathbb{R}^{15}$, which is a 37,050 dimensional space of linear maps. At face value, this would mean that we have an overdetermined system, which could be problematic if we only include 37,050 parameters in the NN. However, this over abundance of data points will come in handy in the case of more complicated NN structures.

With these remarks in mind, we will be using another hidden layer with $L + K = 2{,}470$ neurons. Moreover, we also include a potential bias in both layers. The increasing running time requires that we use a smaller number of epochs. Here, we will be training the model for 3,000 epochs. The first preliminary result is shown in figure 3.4.

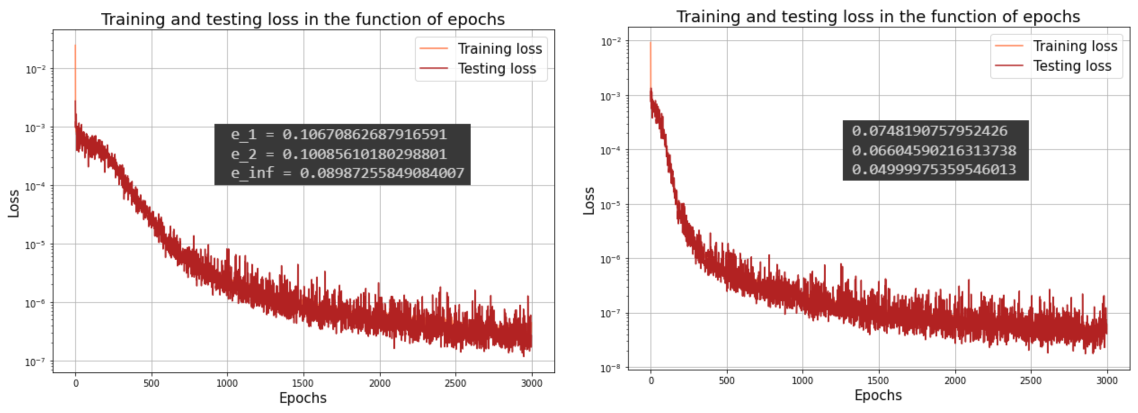


The roughness of the learning curve is consistent with the batch size of 50%, however it is worth pointing out two facts:

1. this model achieved the same 'accuracy' in just 3,000 epochs
2. here seems to be a qualitative difference in the shape of the learning curve (downward convex decline) suggesting that teaching the model further would result in much higher accuracy.

At this point, the model has 6,140,420 parameters and is trained on 77,154 data points. Therefore increasing the number of epochs further or decreasing the batch size results in experiments that have a very long running time, however, we give the result of two such further experiments: see figure for these two experiments.

Both experiments were run with the same NN architecture (one hidden layer with 2,470 neurons, with biased linear activation before and and an unbiased linear activation after the hidden layer, and the learning rate is still 0.0001). The left plot shows a case of 3,000 epochs with batch size 2,048, while the right hand side plot shows the results of an experiment run for 3,000 epochs with a batch size further decreased to 512.



```
e_1 = 0.10670862687916591
e_2 = 0.10085610180298801
e_inf = 0.08987255849084007
```

```
0.0748190757952426
0.06604590216313738
0.04999975359546013
```

Figure 5: picking batch sizes of 1,024 and 512 respectively

These two experiments indicate that a convergence theorem similar to 1 might hold in the case of Poisson's equation also. However, further experiments with different border distance values and $h = \mathrm{dist}(\mathbf{y}_j, \mathbf{y}_{j+1})$ values need to be done in order to investigate the nature of this convergence.

Research questions for the Thesis:

1. What NN structure and optimization method is best suitable for solving these kinds of problems? How can we obtain methods that have a more sensible running time – both for further experiments and application purposes.
2. What is the nature of the convergence hinted at by the above experiments? (Further experiments and perhaps even theoretical work to be done here.)
3. Can we show – either empirically, or theoretically – a better than $\mathcal{O}(h)$ rate of convergence in the case of Laplace's equation?
4. How might we further generalize the class of problems for which this method applies? One such setup is described in the last subsection.
5. How can we use the results obtained here in sensible applications? Can we perhaps use a trained NN to perform time-steps in a time-dependent setup?

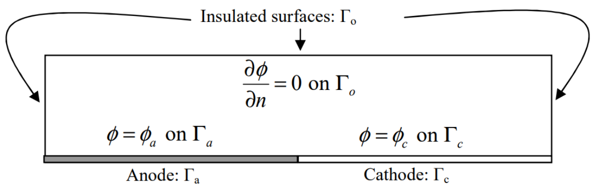
4 Laplace's equation with non-linear boundary condition

So far we have only applied our method to linear PDEs. However, the great potential of the universal approximator property – as compared to the traditional methods – lies in the application of this method to non-linear PDEs. Therefore one further problem modelling steel corrosion in concrete – suggested by [4] – for which we wish to apply this method, and draw both empirical and theoretical conclusions is the following.

Let $\Omega = \left(-\frac{W}{2}, \frac{W}{2}\right) \times \left(-\frac{H}{2}, \frac{H}{2}\right) \subset \mathbb{R}^2$ be a rectangle, and let us partition $\partial\Omega$ as follows:

$$\partial\Omega := \begin{cases} \Gamma_a := \left(-\frac{W}{2}, -\frac{W}{2} + L_a\right) \times \left\{-\frac{H}{2}\right\}, \\ \Gamma_c := \left(-\frac{W}{2} + L_a, \frac{W}{2}\right) \times \left\{-\frac{H}{2}\right\}, \\ C := \left\{\left(\frac{W}{2}, \frac{H}{2}\right), \left(-\frac{W}{2}, \frac{H}{2}\right), \left(-\frac{W}{2}, -\frac{H}{2}\right), \left(\frac{W}{2}, -\frac{H}{2}\right)\right\} \\ \Gamma_0 := \partial\Omega \setminus (\Gamma_a \cup \Gamma_c \cup C), \end{cases}$$

shown below by figure 1 of [4]. Here, W and H denote the width and height of the domain, and L_a denotes the length of the cathodic boundary. C is the set of corners where we do not impose boundary conditions, Γ_a, Γ_c and Γ_0 denote the anodic, cathodic and insulating boundary respectively.



Insulated surfaces: Γ_o
$\frac{\partial\phi}{\partial n} = 0$ on Γ_o
$\phi = \phi_a$ on Γ_a
$\phi = \phi_c$ on Γ_c
Anode: Γ_a
Cathode: Γ_c

Figure 6: setup of the problem as depicted in [4]

The governing equation – using the paper's notation – is

$$\begin{cases} \Delta\phi(\mathbf{x}) = 0 & (\mathbf{x} \in \Omega), \\ \phi(\mathbf{x}) = \phi_a & (\mathbf{x} \in \Gamma_a), \\ \phi(\mathbf{x}) = \phi_c & (\mathbf{x} \in \Gamma_c), \\ \frac{\partial}{\partial \mathbf{n}}\phi(\mathbf{x}) = 0 & (\mathbf{x} \in \Gamma_c), \end{cases}$$

where ϕ_a and ϕ_b are set constants, and $\mathbf{n}$ is a normal vector to the domain in the given point ($\mathbf{i}$, $\mathbf{j}$ or $-\mathbf{i}$) depending on which side of Γ_0 the point $\mathbf{x}$ is. Here, ϕ is the potential function of $\mathbf{E}$ (the electric vector field).

Another variation of the same problem is when there is a non-homogeneous Neumann type boundary condition on Γ_c as well (depending non-linearly on ϕ), which corresponds to a current on cathodic boundary. In later stages of the project (in the thesis), we wish to implement the above described method to these problems as well.

Two particularly challenging aspect of the second problem is how to account for the non-homogeneous Neumann type boundary condition, and how to verify the method itself.

An idea for the first problem is to measure the difference on the boundary by which a particular function (i.e. the Green functions) fail to satisfy the boundary condition, and in the numerical estimation, we try to force this difference to 0. The second problem is yet to be addressed.

References

[1] Csáji Balázs Csanád. “Approximation with Artificial Neural Networks”. In: *MSc Thesis, Eötvös Loránd Tudományegyetem, Természettudományi Kar* (2001).

[2] Haffner Domonkos and Izsák Ferenc. “Solving the Laplace equation by using neural networks”. In: URL: `https://m2.mtmt.hu/api/publication/32625402`.

[3] T. Hieu Hoang, Ferenc Izsák, and Gábor Maros. “Approximation properties of fundamental solutions: a three-dimensional study with Sobolev norms”. In: 2022.

[4] Ge Ji and O. Isgor. “On the numerical solution of Laplace’s equation with nonlinear boundary conditions for corrosion of steel in concrete”. In: (Jan. 2006).

[5] R. Schaback. *A Practical Guide to Radial Basis Functions.* URL: `http://num.math.uni-goettingen.de/~schaback/teaching/sc.pdf`. (accessed: 15.12.2022).

[6] Kurt Hornik, Maxwell Stinchcombe, Halbert White. “Multilayer Feedforward Networks are Universal Approximators”. In: *Neural Networks* Vol. 2 (1989), pp. 359–366.