[MAS480C] Final project

- Estimation of electric field using PINNs -

20180564 – 전기정

Introduction [Problem selection[1] / Description of problem]

Almost all of the particles in real worlds have some electric charge, and the electric field E is induced by this electric charge q. In communication engineering, analyzing such electric field on the air or some medium of transmission line is crucial for reducing the error in communication system, for example, when we are in an elevator, the cellular phone does not work well due to 'Electromagnetic shielding' effect, which makes the electric field to be zero inside the elevator and leads to no connection in the elevator. Also, in nuclear engineering, Nuclear fusion uses magnetic field (which correlate strongly with electric field) to enclose 0.1 billion Celsius degree plasma in a vacuum chamber. In such fields, Methods to control electric field (or magnetic field) finely is a very hot research topic nowadays.

For this final project, we will analyze basic electric field using 'Physical Neural Informed Networks' (called PINNs) to grasp how to visualize such field (especially electric field, here) using 'Gauss's law', which states how the electric field should be given under certain electric charge distribution.

In a classical way, the state-of-the-art method to visualize electric field is Finite Element Method (FEM) or Finite Difference Method (FDM) **[1].** Fortunately, these methods always guarantee the convergence when the step size h goes to zero when solving equations via Gauss's law, since the electric field is unique under certain electric charge distribution. But, one of the problems is that those methods become computationally intractable as the grid (or mesh) becomes finer in 3 or 4-dimension case[2], and unfortunately the most crucial part in visualizing electric field using above classical methods is to make fine grid (mesh) as possible as we can so as to capture abrupt change phenomena of electric field. Furthermore, as the environment get complex, the differential equation that governs electric field becomes ill-posed and the solution becomes extremely non-linear.

To avoid this problem, engineers use some trick that focus on main parts of the environment. The idea of this trick is to make sparse grid for unimportant part (such as inside the conductive object or remote part of electrically highly concentrated part) and make fine grid for important part (such

---

[1] There is no previous project or code corresponding to this final project (up to my finding)

[2] The highest dimension to be analyzed is up to 4 in electric field (x, y, z, t), t = time

as neighborhood of electric charge). However, this method does not work when the electric charge is evenly distributed within the environment, which are very common cases in real worlds. (It is hard problem since we need to focus every part of environment).

Up to my findings, there is one trial (this year) to visualize the magnitude of electric field with respect to certain axis (and magnetic field also) using PINNs **[2]** and they demonstrated that PINNs method can be comparable to FEM method in relative L2 error sense, but using small computational cost compared to FEM by some experiments. Also using real data, they succeeded in finding important physics constant ϵ_R(electric permittivity) and μ_r(magnetic permeability) which determine crucial property of materials, by solving inverse problem using PINNs with high accuracy.

Since the PINN method is considered as the game changer on this field nowadays, for this final project, we will visualize some electric field under basic environment using PINNs, and find what are the problems we will encounter and what we get as advantages when we used PINNs.

Mathematical background for this problem [Problem formulation in mathematics]

From elementary physics, we may hear the Coulomb's law, which states that the electric field due to a stationary point charge is given by:

$$\boldsymbol{E}(\boldsymbol{r}) = \frac{q}{4\pi\epsilon_0} \cdot \frac{\boldsymbol{e_r}}{r^2}$$

where $\boldsymbol{e_r}$ = the radial unit vector, r = radius ($i.e:|\boldsymbol{r}|$), ϵ_0 = electric constant, q = charge of the particle (here, the charge q is assumed to be located at the origin)

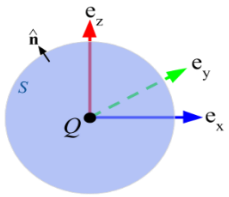


Figure 1. Gauss' law for electric field

And using some mathematical technique on physics **[3]**, we can get the following equation, called differential form of Gauss's law:

$$\nabla \cdot \boldsymbol{E}(\boldsymbol{r}) = \frac{\rho(\boldsymbol{r})}{\epsilon_0}$$

where $\rho(\boldsymbol{r})$ = charge density at point $\boldsymbol{r}$[3] and ∇ is divergence operator

If we replace spherical coordinate into cartesian coordinate, above equation can be re-written as follows in 3-dimensional case:

$$\nabla \cdot \boldsymbol{E}(x,y,z) = \frac{\rho(x,y,z)}{\epsilon_0}$$

Also if we neglect all the constant terms to simplify the question, and focus on visualization of target solution, we need to solve the following :

$$\nabla \cdot \boldsymbol{E}(x,y,z) = \rho(x,y,z) \iff \frac{\partial E^x(x,y,z)}{\partial x} + \frac{\partial E^y(x,y,z)}{\partial y} + \frac{\partial E^z(x,y,z)}{\partial z} = \rho(x,y,z)$$

where $\boldsymbol{E}(x,y,z) = \left(E^x(x,y,z), E^y(x,y,z), E^z(x,y,z)\right) \in \mathbb{R}^3$

And our goal is to find electric field $\boldsymbol{E}(x,y,z)$ given $\rho(x,y,z)$! To solve this using PINNS, let's construct the Neural network architecture first and define loss function secondly.

Step 1) Design of Neural network architecture

Note that this problem is somewhat different in usual partial differential equation since $\boldsymbol{E}(x,y,z)$ is vector in $\mathbb{R}^3$. So, the output of neural network should be 3-dimensional vector in $\mathbb{R}^3$ when the input is also 3-dimensional vector in $\mathbb{R}^3$.

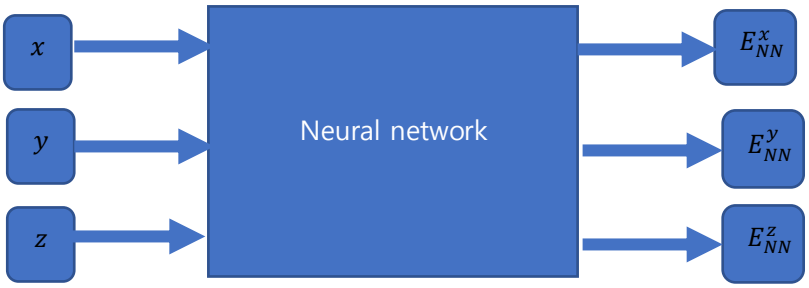


Figure 2. Neural network architecture

The above figure is simplified input – NN – output relation which we will use on this project.

In detail, we can use simple DNN composed of several linear layers and activation functions (we can try also self-adaptive activation function).

Type 1: normal DNN

Firstly if we use normal activation function, one suggested model will be following:

$$(x,y,z) \rightarrow Linear(3,100) \twoheadrightarrow Linear(100,50)$$

---

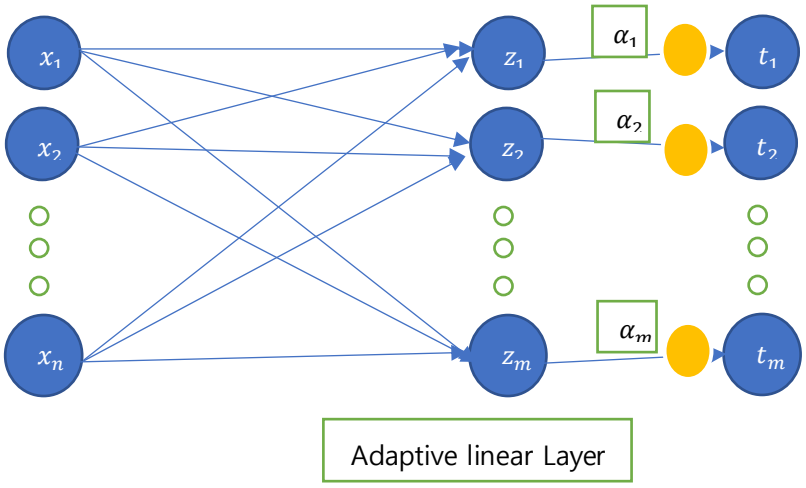
[3] Charge density $\rho(\boldsymbol{r}) = \frac{dQ}{dV}(\boldsymbol{r})$, where Q is total charge on system, dV is the volume element in 3D (or surface / line element in 2/3D), Simply, it is density of charge (q) like density of mass (m).

$$\twoheadrightarrow linear(50,30) \twoheadrightarrow Linear(30,10) \twoheadrightarrow Linear(10,3) \rightarrow \left(E_{NN}^{x}(x,y,z), E_{NN}^{y}(x,y,z), E_{NN}^{z}(x,y,z)\right)$$ [4]

where $\twoheadrightarrow$ represent forward pass with $tanh$ activation function and $\rightarrow$ represent forward pass without activation function.

Type 2: DNN with self-adaptive activation function (local version)

To use self-adaptive activation function with linear layer, the linear layer should be modified as following:



where x_i = input of linear layer $i = 1, \dots n$, $n = input\ dimension$, z_j = output of linear layer $j = 1, \dots, m$, $m = output\ dimension$, α_k = weight of adaptive activation function, $k = 1 \dots m$ and t_j = output of activation function $j = 0, \dots, m$, and yellow circle represents activation function $(tanh)$

After forward pass of normal linear layer finishes, we multiply individual learnable parameter α_k to make normal activation function to be adaptive activation function.

Then, we can suggest following DNN model with adaptive activation function:

$$(x,y,z) \rightarrow AdaptiveLinear(3,100) \twoheadrightarrow AdaptiveLinear(100,50)$$
$$\twoheadrightarrow AdaptiveLinear(50,30) \twoheadrightarrow AdaptiveLinear(30,10)$$
$$\twoheadrightarrow AdaptiveLinear(10,3) \rightarrow \left(E_{NN}^{x}(x,y,z), E_{NN}^{y}(x,y,z), E_{NN}^{z}(x,y,z)\right)$$

[4] Note: This architecture seems to have enough model capacity to approximate current problem (higher capacity turns out to degrade the performance by some experiments)

Step 2) Define the loss function

Recall the original differential equation, we need to solve:

$$\frac{\partial E^x(x,y,z)}{\partial x}+\frac{\partial E^y(x,y,z)}{\partial y}+\frac{\partial E^z(x,y,z)}{\partial z}=\rho(x,y,z)$$

Firstly, we need to minimize the residual part loss and it can be written as following:

$$L_{rd}=\frac{1}{N_{rd}}\sum_{i=1}^{N_{rd}}\left(\frac{\partial E_{NN}^x}{\partial x}\left(x_i^{rd},y_i^{rd},z_i^{rd}\right)+\frac{\partial E_{NN}^y}{\partial y}\left(x_i^{rd},y_i^{rd},z_i^{rd}\right)+\frac{\partial E_{NN}^z}{\partial z}\left(x_i^{rd},y_i^{rd},z_i^{rd}\right)-\rho(x_i,y_i,z_i)\right)^2 \ (residual\ loss)$$

Next, the Neural network outputs should satisfy boundary conditions. So, the boundary loss can be written as following:

$$L_{bd}=\frac{1}{N_{bd}}\sum_{i=1}^{N_{bd}}\left(E_{NN}^x\left(x_i^{bd},y_i^{bd},z_i^{bd}\right)-E^x\left(x_i^{bd},y_i^{bd},z_i^{bd}\right)\right)^2+\left(E_{NN}^y\left(x_i^{bd},y_i^{bd},z_i^{bd}\right)-E^y\left(x_i^{bd},y_i^{bd},z_i^{bd}\right)\right)^2$$

$$+\left(E_{NN}^z\left(x_i^{bd},y_i^{bd},z_i^{bd}\right)-E^z(x_i^{bd},y_i^{bd},z_i^{bd})\right)^2\ (boundary\ loss)$$

where N_{rd} = the number of residual points in dataset, N_{bd} = the number of boundary points in dataset.

By summing up the L_{rd} and L_{bd}, we can get total loss function as follows:

$$L=L_{rd}+L_{bd}\ \ (total\ loss)$$

Usage of Sci-ML method [description of scientific machine learning method]

To summarize and review all the material we've learned during the course, we will try all the method introduced in class as possible as we can, and find some advantages and disadvantages during experiments.

1) Batch training

The loss function given on above section calculates all the data in datasets. And update the model parameter once in a one epoch. This method turns out to be stable, but shows slow convergence to optimal point (also possibly to the bad local minimum point) and degrade generalization performance **[4]**. Alternatively, we can use batch training, which divide dataset by batch_size and update the model_parameters several times in one epoch. This method has advantage to somewhat avoid convergence to bad local minimum point, and shows faster convergence than using all the dataset.

Under batch training, we can rewrite the loss function as follows:

$$L_{rd} = \frac{1}{B_{rd}} \sum_{i=1}^{B_{rd}} \left(\frac{\partial E_{NN}^{x}}{\partial x}(x_i^{rd}, y_i^{rd}, z_i^{rd}) + \frac{\partial E_{NN}^{y}}{\partial y}(x_i^{rd}, y_i^{rd}, z_i^{rd}) + \frac{\partial E_{NN}^{z}}{\partial z}(x_i^{rd}, y_i^{rd}, z_i^{rd}) - \rho(x_i, y_i, z_i) \right)^2$$

$$L_{bd} = \frac{1}{B_{bd}} \sum_{i=1}^{B_{bd}} \left(E_{NN}^{x}(x_i^{bd}, y_i^{bd}, z_i^{bd}) - E^{x}(x_i^{bd}, y_i^{bd}, z_i^{bd}) \right)^2 + \left(E_{NN}^{y}(x_i^{bd}, y_i^{bd}, z_i^{bd}) - E^{y}(x_i^{bd}, y_i^{bd}, z_i^{bd}) \right)^2 + \left(E_{NN}^{z}(x_i^{bd}, y_i^{bd}, z_i^{bd}) - E^{z}(x_i^{bd}, y_i^{bd}, z_i^{bd}) \right)^2$$

where B_{rd} = batch_size of residual data loader, B_{bd} = batch_size of boundary data loader

2) Random Sampling training

In this problem, we need to make our own data to visualize electric field, hence, it is crucial to think how to select the data points from boundary points and residual points, and the random sampling training suggests to pick residual points and boundary points randomly in residual domain and boundary domain.

If the method guarantees good results even in PINNs, this somewhat becomes fascinating property of PINNs in the sense that random sampling in visualization of electric field is problematic in classical method such as FEM (as mentioned in introduction)

3) Self-adaptive activation function

Local idea of this part is explained on above section, the detailed idea of this is to adopt some learnable scaler parameter and several activation function and use them simultaneously. For example, we can take an adaptive activation function as follows:

$$\phi(x, \delta, s) = \delta_1 \cos(s_1 x) + \delta_2 \tanh(s_2 x) + \delta_3 ReLU(s_3 x) + \cdots + \delta_k sigmoid(s_k x)$$

where $\delta = (\delta_1 \ldots, \delta_k)$, $s = (s_1, \ldots s_k)$ are learnable parameters for some $k \in \mathbb{N}$ and replace usual activation function (such as $tanh$) to this activation function ϕ.

For this project, we do not use combination of several activation function[5], instead simply use learnable parameter α_i such that[6] :

$$\text{activation function } (z_i) \rightarrow \text{activation function } (\alpha_i \cdot z_i)$$

where z_i is the ith output after linear layer.

[5] We left this implementation as the appendix at the end of this report.

[6] This method is similar to the infant version of generalized adaptive-activation function (using combination of several activation unction) **[5]**

For recent PINNS paper, many of them adopts self-adaptive activation function and observed that it can break through bad local minimum efficiently **[6]**, and we will check whether the effect can be appeared on our problem.

4) Self-adaptive weight for loss

Recall that we calculate total loss as $L = L_{rd} + L_{bd}$, but during training, one of L_{rd} or L_{bd} becomes very big compared to the other, which leads to unfair optimization only focused on the points whose corresponding loss is huge. To prevent this phenomenon, one heuristic is to adopt learnable parameter vector $\lambda_{rd} = (\lambda_{rd,1}, \dots, \lambda_{rd,N_{rd}})$, and $\lambda_{bd} = (\lambda_{bd,1} \dots \lambda_{bd,N_{bd}})$ such that

$$L = \frac{1}{B_{rd}} \sum_{i=1}^{B_{rd}} \boldsymbol{\lambda_{rd,i}} \left(\frac{\partial E_{NN}^{x}}{\partial x}(x_i^{rd}, y_i^{rd}, z_i^{rd}) + \frac{\partial E_{NN}^{y}}{\partial y}(x_i^{rd}, y_i^{rd}, z_i^{rd}) + \frac{\partial E_{NN}^{z}}{\partial z}(x_i^{rd}, y_i^{rd}, z_i^{rd}) - \rho(x_i, y_i, z_i) \right)^2$$
$$+ \frac{1}{B_{bd}} \sum_{i=1}^{B_{bd}} \boldsymbol{\lambda_{bd,i}} [\left(E_{NN}^{x}(x_i^{bd}, y_i^{bd}, z_i^{bd}) - E^{x}(x_i^{bd}, y_i^{bd}, z_i^{bd}) \right)^2$$
$$+ \left(E_{NN}^{y}(x_i^{bd}, y_i^{bd}, z_i^{bd}) - E^{y}(x_i^{bd}, y_i^{bd}, z_i^{bd}) \right)^2$$
$$+ \left(E_{NN}^{z}(x_i^{bd}, y_i^{bd}, z_i^{bd}) - E^{z}(x_i^{bd}, y_i^{bd}, z_i^{bd}) \right)^2]$$

And, train $10k$ iteration under following adversarial optimization:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla_{\theta} L$$
$$\lambda^{(k+1)} = \lambda^{(k)} + \eta \nabla_{\lambda} L$$

where θ = learnable model parameter, $\lambda = (\lambda_{rd}, \lambda_{bd})$ and η = learning rate

After $10k$ training, fix the values λ_{rd} and λ_{bd}, and perform original training using loss function L defined by right above.

However, taking learnable vector λ_{rd} and λ_{Bd} and training them may be intractable as the data becomes huge (for example, here, we use model with 7022 parameters, if we consider λ_{rd} and λ_{bd}, with $N_{bd} = 10^8$, $N_{rd} = 10^8$, then the number of parameters to be learned becomes greater than $2 \cdot 10^8$)

To simplify this problem, we considered two scalar learnable parameters λ_{rd}, λ_{bd} such that $L = \lambda_{rd} L_{rd} + \lambda_{bd} L_{bd}$ and perform the heuristic above for this project.[7]

[7] Unfortunately, this does not work well on our problem. Alternatively, we give another heuristic to overcome this problem on later section.

Experiment project [Data generation / Training-Optimization technique / Numerical simulations and results]

Part1 – Data generation and problem setting [2-dimensional conductors setting]

From the basic physics, it is well known that electric field cannot exist within the conductive object, and assume there are two charged plates on left ($-1 \le x \le -0.8$) and right sides ($0.8 \le x \le 1$) with charge density $\rho(x,y) = 1, -1$, respectively. Also, futher assume there is a big conductive circle at the center $0.4^2 \le x^2 + y^2 \le 0.5^2$, which have induced charge density due to charged plates as follows:

$$\rho(x,y) = \begin{cases} -200x^2 & x \le 0,\ \ 0.4^2 \le x^2 + y^2 \le 0.5^2 \\ 200x^2 & x >= 0,\ \ 0.4^2 \le x^2 + y^2 \le 0.5^2 \end{cases}$$

Then, due to left and right two charged plate, the boundary condition should be as follows ::

$$Boundary\ condition: \begin{array}{c} E^x(x,y) = 1,\ E^y(x,y) = 0 \ \ when\ \ x = -1,\ \ -1 \le y \le 1 \\ E^x(x,y) = 1,\ E^y(x,y) = 0 \ \ when\ \ x = 1,\ \ -1 \le y \le 1 \\ E^x(x,y) = E^y(x,y) = 0 \ \ when\ \ x^2 + y^2 \le 0.4^2 \end{array}$$

And the problem we need to solve can be summarized as given as follows:

$$\frac{\partial E^x(x,y)}{\partial x} + \frac{\partial E^y(x,y)}{\partial y} = \rho(x,y)$$

$$where\ \rho(x,y) = \begin{cases} -1 & 0.8 \le x \le 1,\ \ -1 \le y \le 1 \\ 1 & -1 \le x \le -0.8,\ \ -1 \le y \le 1 \\ -200x^2 & x \le 0,\ \ 0.4^2 \le x^2 + y^2 \le 0.5^2 \\ 200x^2 & x \ge 0,\ \ 0.4^2 \le x^2 + y^2 \le 0.5^2 \\ 0 & otherwise \end{cases}$$

Also, the loss can be defined as follows :

$$L_{rd} = \frac{1}{N_{rd}} \sum_{i=1}^{N_{rd}} \left(\frac{\partial E_{NN}^x}{\partial x}(x_i^{rd}, y_i^{rd}) + \frac{\partial E_{NN}^y}{\partial y}(x_i^{rd}, y_i^{rd}) - \rho(x,y) \right)^2 \ (residual\ loss)$$

$$L_{bd} = \frac{1}{N_{bd}} \sum_{i=1}^{N_{bd}} \left(E_{NN}^x(x_i^{bd}, y_i^{bd}) - E^x(x_i^{bd}, y_i^{bd}) \right)^2 + \left(E_{NN}^y(x_i^{bd}, y_i^{bd}) - E^y(x_i^{bd}, y_i^{bd}) \right)^2 \ (boundary\ loss)$$

$$L = \lambda_{rd} L_{rd} + \lambda_{bd} L_{bd} \ (total\ loss)$$

where $\lambda_{rd} = 1,\ \lambda_{bd}$ = 100 (found by manual inspection using heuristic described in later section)

Now, we generate data for residual parts and boundary parts as follows:

First, we took $N_{bd} = 100\ (left\ boundary) + 100(right\ boundary) + 200(center\ boundary) = 500$.

where left boundary point sets = $\{(-1,k) \in \mathbb{R}^2 |\ k \in linspace(-1,1\ 100)\}$ and right boundary point sets = $\{(1,k) \in \mathbb{R}^2\ | k \in linspace(-1,1,100)\}$, the center boundary point sets = $\{(x,y) \in \mathbb{R}^2 \mid x = r_k cos\theta_k,\ y_k = r_k sin\theta_k\ \},\ , where\ r_k \sim uniform\ ([0,0.4]),\ \theta_k \sim uniform\ ([0,2\pi]), k = 1, \dots 200$

And, the boundary point sets = union of left / right / center boundary points.

Secondly, we took $N_{rd} = 10000\ (uniformly\ on\ domain) + 5000\ (center\ concentrated) = 15000$.

where uniform residual point set = $\{(i,j) \in \mathbb{R}^2 \mid i \in linspace(-1,1,100),\ j \in linspace(-1,1,100)\}$ and center concentrated point set is given as follows : $\{(x,y) \in \mathbb{R}^2 \mid x = r_k cos\theta_k,\ y_k = r_k sin\theta_k\}, where\ r_k \sim uniform\ ([0,0.5]),\ \theta_k \sim uniform\ ([0,2\pi]), k = 1, \dots 5000\}$

Thus, the residual point sets = union of uniform residual / center concentrated point sets.

Note that, by taking the dataset like above, we are taking the near-field dataset which commonly used as a dataset in any experiments in electric field analysis. (Recall that all the important phenomenon happens around near-by field area)

Part2 – Training/Optimization related Techniques

Step 1) Initialization

Our experiment setting will be given as following:

Model: Type-1 DNN / Type-2 DNN with (local) self-adaptive activation function

Optimizer: SGD with momentum 0.9 / Adam / AdamW[8]

Learning rate scheduler: CosineAnnealing (with T_max = the number of epochs)[9]

Batch_size = 32 (for boundary points) / 10 (for residual points) with shuffling for both cases[10]

Self-adaptive weights: $\lambda_{rd} = 1,\ \lambda_{bd} = 100$

Crucial Note (heuristic for this problem): Those values are not found by adversarial optimization mentioned above section, Unfortunately, that heuristic does not work well on this problem. (Even if we initialize λ_{rd} ,λ_{bd} several pair of values as (1,10), (1,100), …, the value of λ_{bd} will keep increase up to almost 10^8 until $10k$ iterations, which does not make sense.) And, it seems that this issue happened due to abrupt changes in near-field area induced by boundary conditions and charge density function $\rho(x,y)$. As we can expect from the problem, there should be (almost) no change except for near-field area. In other words, the electric field should be $E^x(x,y) \cong 1$, $E^y(x,y) \cong 0$ in

---

[8] AdamW is a revised version of Adam by decoupling weight decay from the gradient update to resolve generalization performance drop of Adam on weight decay / L2 regularization environment

[9] A type of LR scheduler which control LR following the cosine function, by setting T_max = the number of epochs, the LR goes to 0 at the end.

[10] Note that the boundary points dataloader and the residual points dataloader circulate simultaneously. In fact, 1 epoch corresponds $50\ iters = \frac{N_{bd}}{10}$ (since $50 = \frac{N_{bd}}{10} < \frac{N_{rd}}{32} \cong 312.5$)

non-near field area, furthermore the electric field should be vanished inside the circle, which makes training really hard. From these conditions, the almost all of the loss are come from residual loss especially on the area: $0.4^2 \leq x^2 + y^2 \leq 0.5^2$, implying the loss from boundary will be neglibile by the nature of the problem. This is why I think the adversarial optimization to get λ_{rd}, λ_{bd} failed for this problem. Alternatively, we can average the L_{rd} and L_{bd} during training (10k iterations without adversarial optimization) and set the $\lambda_{rd} = 1$ and $\lambda_{bd} = \frac{\mathbb{E}[L_{rd}]}{\mathbb{E}[L_{bd}]} \cong 100$. Fortunately, this heuristic helped greately to get desired result on this problem (This will be covered next section)

Step 2) Learning rate search

To find reasonably good learning rate, we need to grasp how the order of learning rate should be. Hence, we grid-searches for learning rate $\lambda \in \{1, 10^{-1}, 10^{-2}, .. 10^{-6}\}$ for each optimizer using 500 epochs to find best learning rate which minimizes the loss as follows:
(The values written are averaged loss on last 50 epochs)

OPT / LR	SGD	Adam	AdamW	OPT / LR	SGD	Adam	AdamW
1	Inf	66.415	75.721	1	Inf	87.353	79.703
10^{-1}	Inf	64.544	89.272	10^{-1}	Inf	94.664	86.202
10^{-2}	57.983	43.909	35.431	10^{-2}	inf	63.768	73.133
10^{-3}	47.200	35.742	25.091	10^{-3}	60.711	9.803	7.878
10^{-4}	29.151	67.527	60.446	10^{-4}	79.798	71.068	54.407
10^{-5}	52.890	65.646	53.606	10^{-5}	26.817	65.114	70.994
10^{-6}	50.049	54.579	43.391	10^{-6}	58.815	135.469	108.390

Table 1. LR grid search result of Type–1 (left) / Type-2 (right) DNN with respect to Loss

Also, further finer grid search gives that the best learning rate / optimizer pair for Type-1 DNN and Type-2 DNN are ($\lambda = 1.6 \times 10^{-3}$, AdamW) / ($\lambda = 8.9 \times 10^{-4}$, AdamW), respectively.

Under 1000 epoch and best LR / optimizer pair, the loss trajectory plot will be as following:

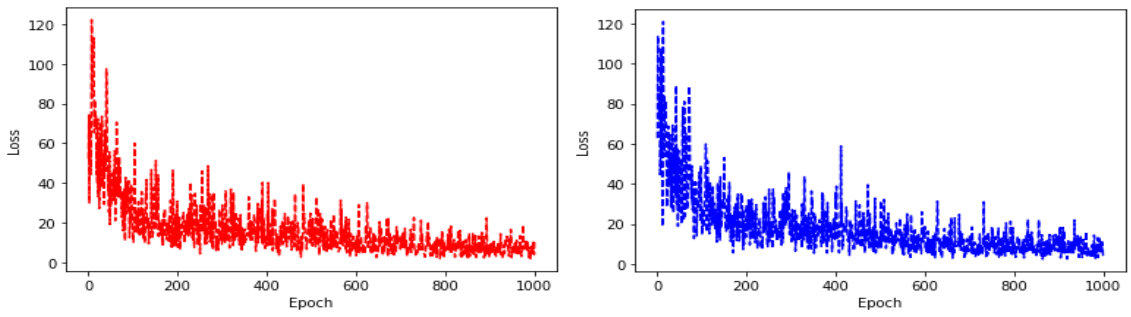


Figure 3. Loss trajectory of Type-1 (left) / Type-2 (right) DNN under best LR/optimizer pair

where averaged last 50 epoch loss = 5.604 (Type-1 DNN) / 4.356 (Type-2 DNN)

As we can check, it seems that adaptive activation function of our version does not overwhelm the performance of Type-1 DNN (but still better a little bit).

Part3 – Training/Optimization related Techniques

Unfortunately, we have no closed form solution for this problem, however we actually know how the vector field should be under this circumstance (using numerical analysis technique), and the estimated electric vector field should be similar as the following:

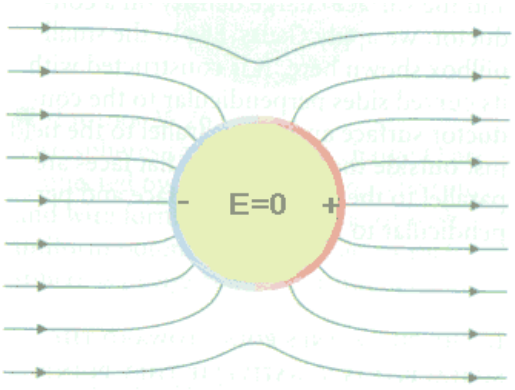


Figure 4. Target solution

Now, under best setting (Type-2 DNN with ($\lambda = 8.9 \times 10^{-4}$, AdamW) pair), we have following result:

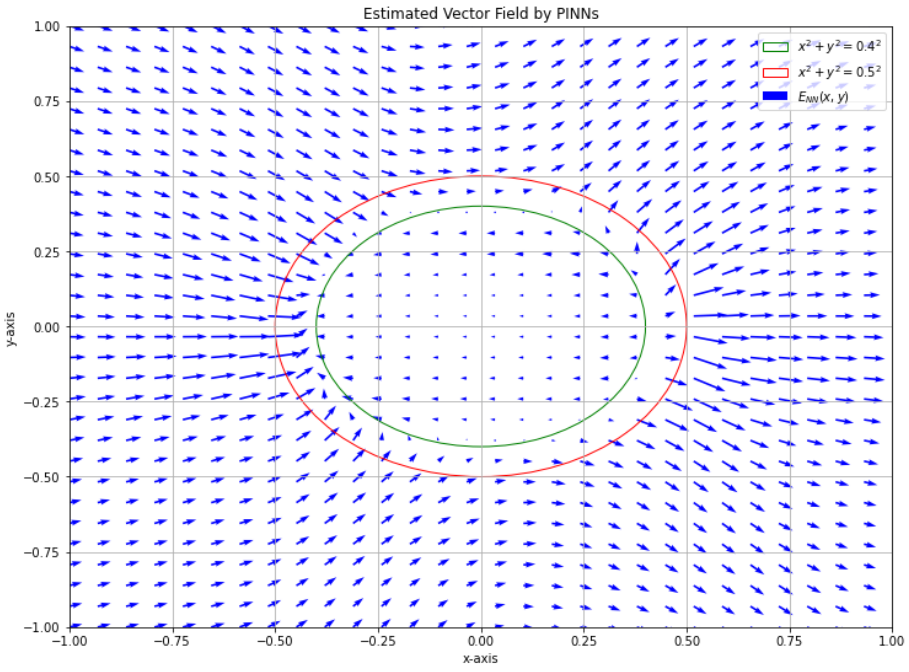


Figure 5. Best estimated vector field by PINNS (1000 epochs)

which is very similar to target solution shape except for the left / right – upper/lowermost area and the in-between circle area[11]. However, if we remove (or add) our SCI-ML technique one by one, the estimated vector field starts to collapse for some case, but not for all.

1) **Remove the self-adaptive activation function** (Type-1 DNN with ($\lambda = 1.6 \times 10^{-3}$, AdamW))

After we remove self-adaptive activations and replace them into normal activation function ($tanh$), the best result still looks great as the best loss of Type-1 DNN was comparable to best loss of Type-2 DNN. The result is given as follows:

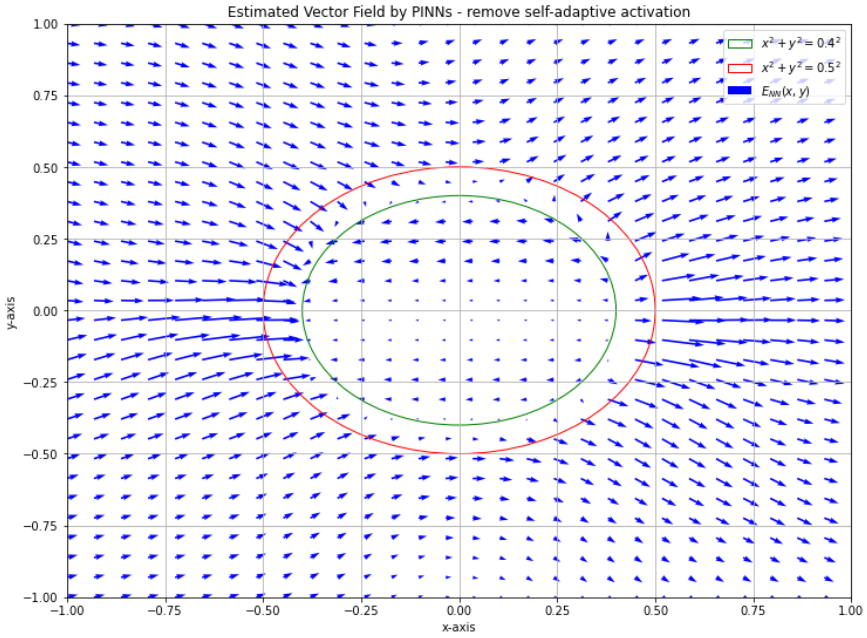


Figure 6. Best estimated vector field by Type-1 DNN (1000 epochs)

2) **Remove the self-adaptive weight for loss (set $\lambda_{rd} = 1, \lambda_{bd} = 1$)**

If we remove process to find self-adaptive weight for loss ($\lambda_{rd}, \lambda_{bd}$), it falls into some weird problem, that is: Even though the loss is comparable to best case (averaged last 50 epoch loss = 4.858), the estimated vector field is totally different compared to what we expect (Figure 7). This problem is mainly due to different difficulties between residual points and boundary points, if we inspect the loss values L_{rd} and L_{bd} carefully, the mean value of L_{rd} is tends to be 100 times of mean value of L_{bd}, without knowing this fact, it is highly likely

[11] Unfortunately, we don't have any accurate solution to be compare with under this certain circumstance. If we have estimated vector field estimated by FEM, then we could use it to check relative L2-error with estimated vector field by PINNs.

for the model to estimate wrong answer at the end. But, as we seen in previous section, the adversarial optimization to get λ_{rd} and λ_{bd} sometimes fails and manual inspection (like here) can be also failed, because highly stochastic movement of L_{rd}, L_{bd} can happen. **Here, I suggest one method which may resolve this problem, the method can be called GraNd score analysis [7] (one another heuristic we can try)**

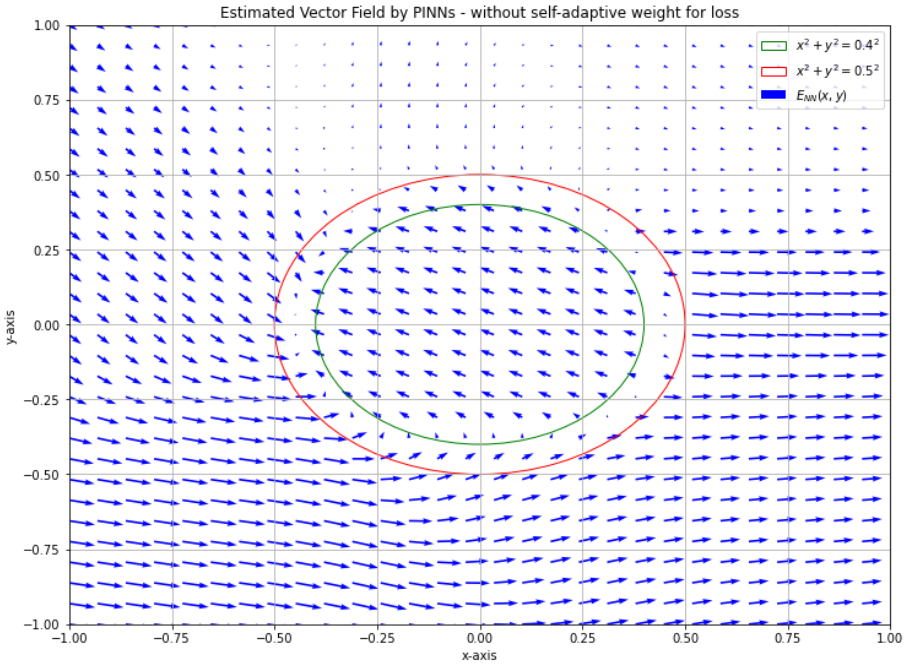


Figure 7. Estimated vector field - without self-adaptive weight for loss

GraNd (Gradient Normed) score of some input x_i is defined to be the expectation of norm of loss gradient with respect to input x_i, where expectation is taken over SGD noise[12]. And this score turned out to be work greatly to distinguish what are the difficult or easy examples (or data points) in perspective of model itself (also it can distinguish some noisy examples too). **Using this score, we can calculate the 'difficulty' of the data points in the perspective of model, and, by weighting more (adopting learnable vector $\boldsymbol{\lambda_{rd}, \lambda_{bd}}$) on loss induced by hard data points, we can reflect the difficulty of this example to the loss function.** (This technique can avoid instability of adversarial optimization or manual inspection to set λ_{rd}, λ_{bd}).

[12] SGD noise simply means any noise that can be occurred due to initialization (such as order of batch_shuffling, weight initialization …). To calculate GraNd score in practical, we usually prepared several same models and train them respectively with same environment, but different SGD noise. After training, we average (MC approximation) the GraNd scores from individual models and use it to grasp what are the difficult or easy data points inside the dataset.

3) Perform random sampling training

Obviously, random sampling training will be best possible method we can take when there is no information on what are the important data points. Fortunately, if we randomly sample data points on residual domain, we still get good results as following: (Also, the averaged last 50 epoch loss (=3.520) becomes smaller than the best case under our near-field data)

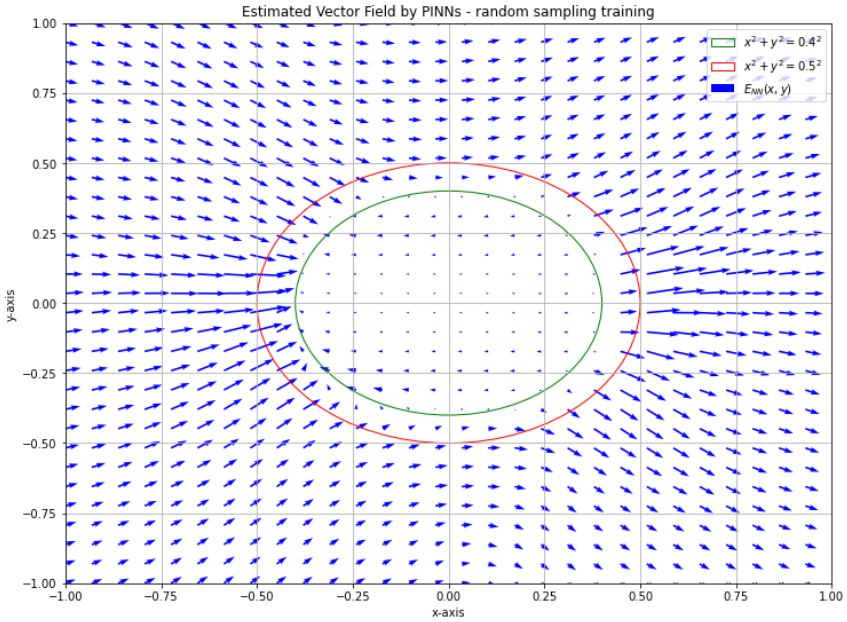


Figure 8. Estimated vector field using random sampling training

This property becomes great advantage of using PINNs in analyzing electric field, which overcomes the critical drawback of classical method mentioned in the introduction.

To sum up the results, the PINNs shows great ability to estimate electric field under this problem even though there are some issues related with SCI-ML techniques (self-adaptive weight for loss). Also, in terms of random sampling training, it demonstrated that PINNs can sometimes overcome random sampling problem happened in classical numerical analysis. To the extension of this project, we can estimate 3-dimensional electric field on some complicated environment[13] (such as the electric field nearby thick communication line underneath the ocean) or even 4-dimensional (x, y, z, t) electric field varying on time t.

[13] Note that PINNs may not estimate the electric field well in complex environment, we left this as one of appendix at the end of this report.

Appendix (some extra failed case and implementation of generalized self-adaptive activation function)

1) Generalized self-adaptive activation function

Consider the previous discussion about self-adaptive activation function. We set the one example of generalized self-adaptive activation function as follows:

$$\phi(x, \delta, s) = \delta_1 \cos(s_1 x) + \delta_2 \tanh(s_2 x) + \delta_3 ReLU(s_3 x) + \cdots + \delta_k sigmoid(s_k x)$$

To experience how this can affect on our problem, we set our generalized self-adaptive activation function as follows: (ELU: Exponential Linear Unit function)

$$\phi(x, \delta, s) = \delta_1 ReLU(s_1 x) + \delta_2 \tanh(s_2 x) + \delta_3 sigmoid(s_3 x) + \delta_4 ELU(s_4 x)$$

And, we grid-searched the best learning rate parameter (500 epochs) under the AdamW optimizer with same other environments as before. (The values written are averaged loss on last 50 epochs)

One thing that we can observe that the model with generalized activation function achieves somewhat insensitivity with respect to learning rate compared to the normal model as following: (The values written are averaged loss on last 50 epochs)

OPT / LR	AdamW
1	54.671
10^{-1}	32.508
10^{-2}	7.634
10^{-3}	5.699
10^{-4}	60.404
10^{-5}	61.101
10^{-6}	62.207

Table 2. LR grid search result with respect to loss for generalized self-adaptive activation

And, further fine grid-search gives the best averaged loss on last 50 epochs (= 4.690) when $\lambda = 0.002$, which is similar compared to previous best model (averaged loss on 50 epochs = 4.356). Although the loss does not decrease drastically, it still shows good results as we can see (figure 9).

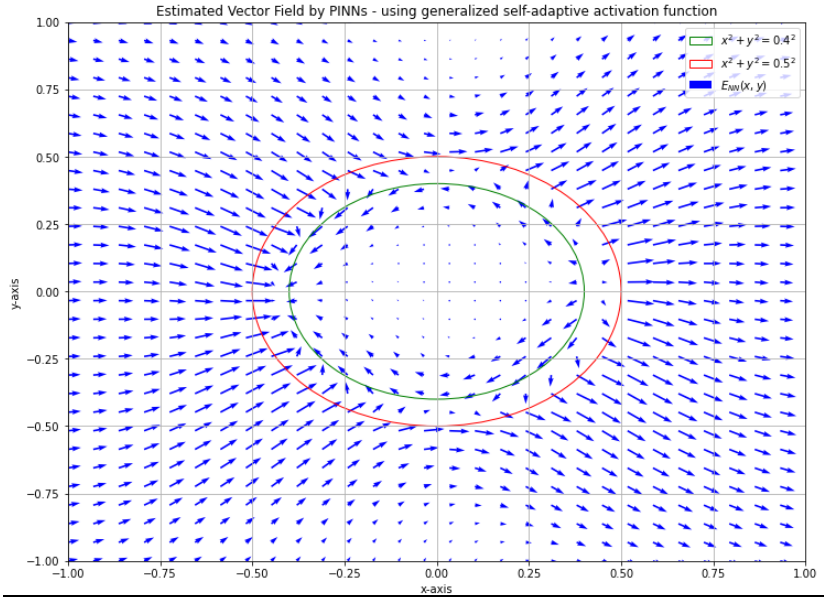


Figure 9. Best estimated vector field using generalized self-adaptive activation function

2) Hard problem for PINNs (2-dimensional opposite sign point charge case)

To find some hard case, we made some well-known problem in visualizing electric field as follows :

$$Boundary\ condition: \begin{matrix} E^x(x,y)=0,\ E^y(x,y)=0 \quad when\ \ (x-0.5)^2+y^2 \le 0.1^2 \\ E^x(x,y)=0,\ E^y(x,y)=0 \quad when\ \ (x+0.5)^2+y^2 \le 0.1^2 \end{matrix}$$

And the problem we need to solve is given as follows:

$$\frac{\partial E^x(x,y)}{\partial x}+\frac{\partial E^y(x,y)}{\partial y}=\rho(x,y)$$

$$where\ \rho(x,y)=\begin{cases} 30 & 0.1^2 \le (x-1)^2+y^2 \le 0.2^2 \\ -30 & 0.1^2 \le (x+1)^2+y^2 \le 0.2^2 \\ 0 & otherwise \end{cases}$$

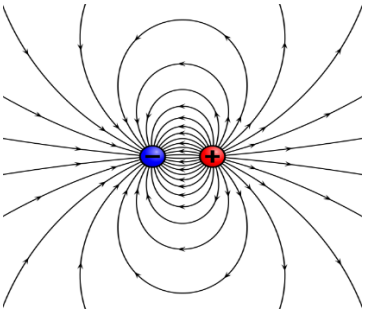


Figure 10. Target solution

We can see the target solution is extremely complex compared to our previous experiment. Using the same setting as before, but with different best learning rate, the best estimated electric field will be given as follows:

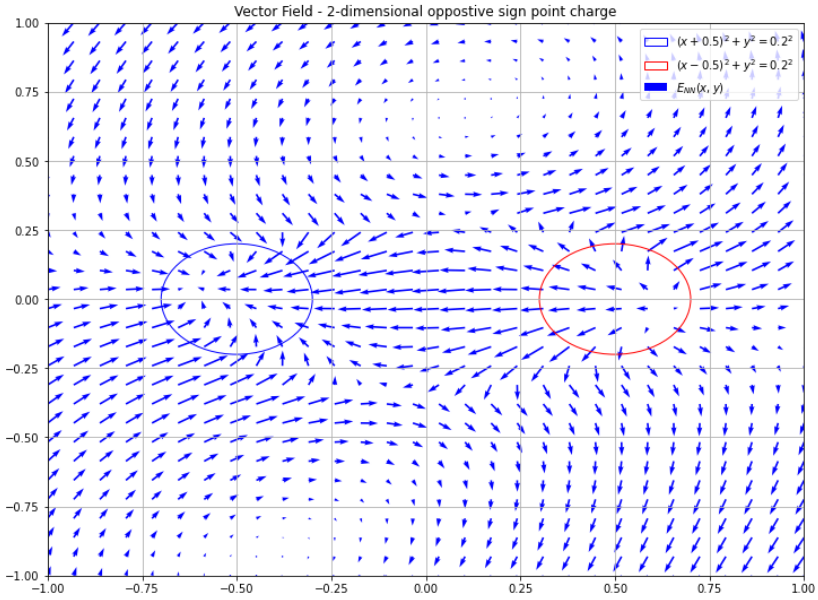


Figure 11. Best estimated vector field using generalized self-adaptive activation function[14]

It resembles the target solution, but not enough. This would happen due to the extreme non-linearity of the target solution, which directly related with difficulty of problem. If we do not use self-adaptive activation function, the results are not good as the previous one as follows:

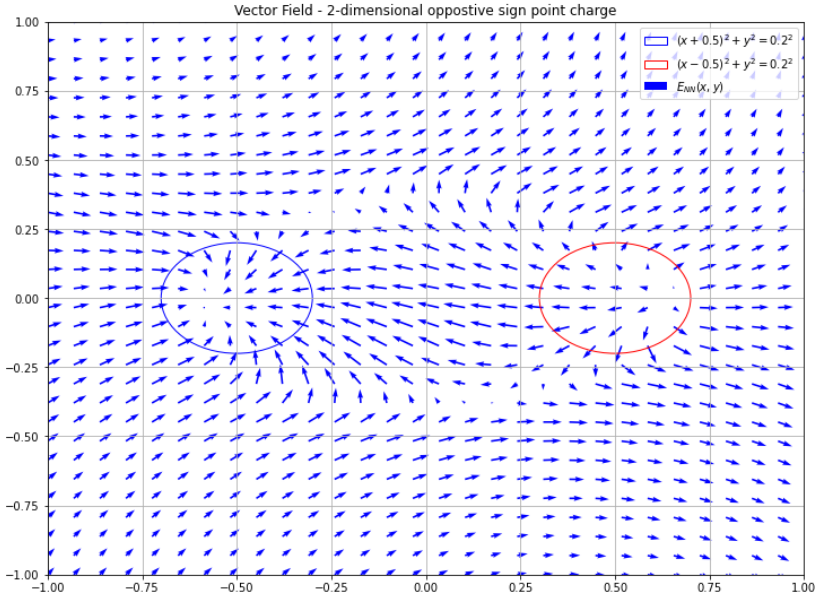


Figure 12. Best estimated vector field using Type-1 DNN

[14] Here, we used generalized self-adaptive activation and all the SCI-ML technique introduced above except for random sampling to get best result as possible as we can. The detail setting will be given as following: ($\lambda_{rd} = 1, \lambda_{bd} = 100$, epoch = 1000, LR = 0.001, LR_scheduler = cosineannealing (T_max = epoch) Optimzier: AdamW, Model: DNN with generalized self-adaptive activation, Datatset: neaf-field data)

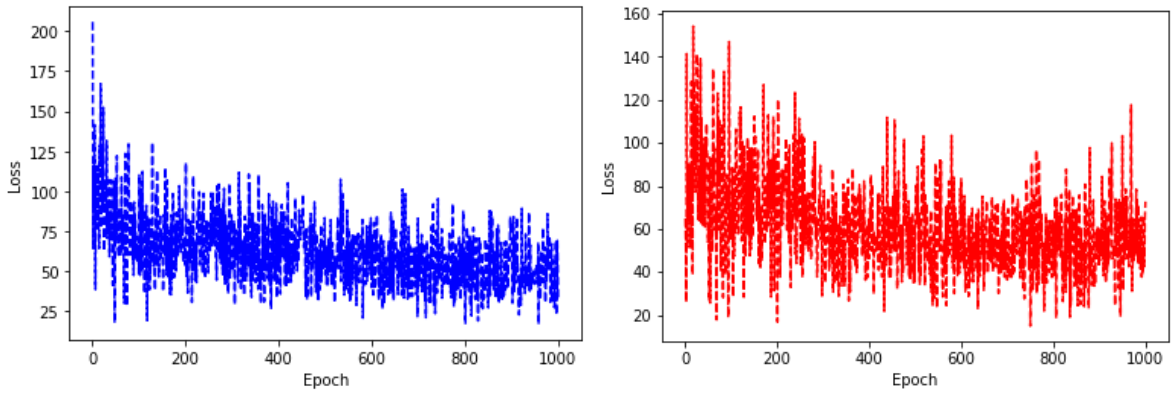


Figure 13. Loss trajectory of two DNN (left: generalized self-adaptive activation / right: Type-1 DNN)

As we can see from the loss trajectory of DNN with generalized self-adaptive activation function and Type-1 DNN, the problem is much more difficult than the previous one **(the averaged last 50 epoch loss are 50.35 (generalized self-adaptive activation), 61.29 (Type-1 DNN), respectively)**. Also, as the result suggests, this is the right problem where generalized self-adaptive activation function can play a role compared to Type-1 DNN, and the difference of averaged loss of last 50 epoch between them shows a good advantage of using self-adaptive activation function.

Additionally, if we run 5000 epochs with the same setting written in comment 15, we get following estimated vector field: **(with averaged last 50 epoch loss = 30.52)**

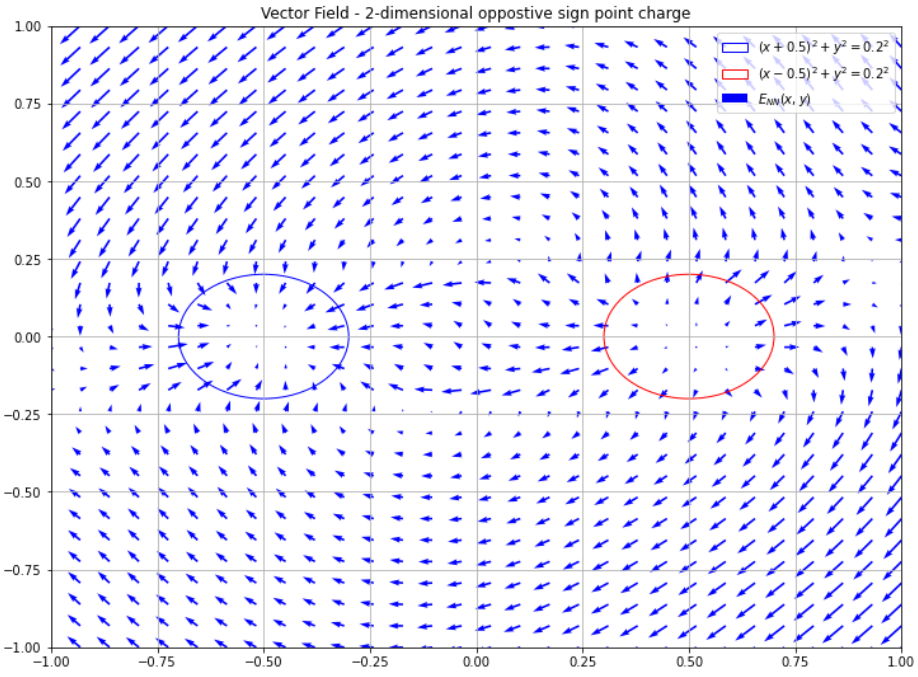


Figure 14. Estimated vector field using generalized self-adaptive activation (5000 epochs)

Although it is hard problem for PINNs, the estimated vector field can be very close to target solution.

Reference

[1] Shallal, Abidaoun & Alshaibi, Maather & Ibrahim, & Hasan, Mohanad & Fleh, Saad. (2015). ESTIMATION AND PLOT OF ELECTRICAL FIELD USING FINITE DIFFERENCE METHOD.

[2] Yuyao Chen and Luca Dal Negro, "Physics-informed neural networks for imaging and parameter retrieval of photonic nanostructures from near-field data", APL Photonics 7, 010802 (2022)

[3] Wikipedia contributors. (2022, November 20). Gauss's law. Wikipedia. https://en.wikipedia.org/wiki/Gauss's_law

[4] Lin, T., Stich, S. U., & Jaggi, M. (2018). Don't Use Large Mini-Batches, Use Local SGD. CoRR.

[5] Jagtap, A. D., Kawaguchi, K., & Karniadakis, G. E. (2020). Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 476(2239).

[6] Hu, H., Liu, A., Guan, Q., Li, X., Chen, S., & Zhou, Q. (2021). Adaptively Customizing Activation Functions for Various Layers. arXiv.

[7] Paul, M., Ganguli, S., & Dziugaite, G. K. (2021). Deep Learning on a Data Diet: Finding Important Examples Early in Training. Advances in Neural Information Processing Systems (Vol. 34, pp. 20596–20607).