Electrophysiology simulations using Physics-Informed Neural Networks

Adam Jakobsen
Department of Physics, University of Oslo
(Dated: June 12, 2023)

We present an exploration of the application of Physics-Informed Neural Networks for electrophysiology simulations. Utilizing in silico EP data, the simulations were conducted in a one-dimensional cable domain, a two-dimensional square domain, and two-dimensional cardiac magnetic resonance image (MRI) based geometries. The optimized PINNs' performance was assessed through evaluating the loss across training points in randomly sampled batches and calculating the Root Mean Square Error (RMSE) for unseen data. The results indicated promising potential for PINNs in EP simulations, achieving an RMSE of 8.8×10^{-3} for the 1D problem, and an RMSE of 7.9×10^{-4}. However, we noted some degree of overfitting and the limitation of the models in accurately capturing behavior in specific regions as well as a lack of ability to extrapolate beyond training data on an MRI-based 2D geometry. Future work should focus on additions to the mathematical model employed, as well as refining the model to increase its accuracy and generalizability.

I. INTRODUCTION

The regular rhythm of the heartbeat is regulated by a complex electrical conduction system. However, under pathological conditions, this system can malfunction, leading to life-threatening situations like arrhythmia and sudden cardiac arrest. Therefore, cardiac electrophysiology simulations tailored to individual patients can prove useful in predicting and treating such conditions. These simulations, though, demand a significant amount of computational power, necessitating the need for more cost-effective computational methods.

Deep learning models have surfaced as a proficient method to tackle Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs), mitigating the limitations of conventional numerical methods. Specifically, Physics-Informed Neural Networks (PINNs) are a class of neural networks designed to take advantage of the inherent physical equations governing a system. The hybrid learning approach of the model, incorporating both data and pre-existing knowledge of the physical system, has been shown to yield heightened accuracy, diminished computational cost, and faster convergence rates[1].

In this project, we produce in silico electrophysiology(EP) data used to train and test PINNs. Initially, the simulations were performed in a one-dimensional cable domain and a two-dimensional square domain inspired by the work demonstrated by C.H.Martin et al.[2]. The work presents EP-PINNs, a method for accurate action potential simulation and EP parameter estimation. Eventually, we progressed to two-dimensional cardiac magnetic resonance image (MRI) based geometries and develop a model for patient-specific EP simulations using PINNs. First, we introduce the relevant Electrophysiology background and the mathematical models used to simulate action potentials. Then we describe the methodology for generating the data and the PINN methodology, and finally we present and discuss the results, and link our findings with existing literature.

[1] https://github.com/Adamjakobsen/FYS5429

II. BACKGROUND

A. The heartbeat

The heartbeat relies on a collective, systematic contraction of cardiac muscle cells, known as cardiomyocytes.

Cardiomiocytes are excitable, which means they can respond to electrical stimuli. At rest, these cells maintain ionic concentrations that differ from their surroundings. Because ions carry electrical charge, a potential difference between the inside and outside of the cell arises, known as the transmembrane potential (TP). When electrical stimuli are applied to excitable cells, the TP rises. If the stimuli is strong enough, the cell's conductive properties change, leading to a flow of positive ions into the cell. This results in the TP increasing from a negative value to a positive value or close to zero, a process referred to as the depolarization of the membrane. After depolarization, the potential goes back to its resting value, a process called repolarization. The entire cycle consisting of depolarisation followed by repolarisation is referred to as an action potential (AP)[3].

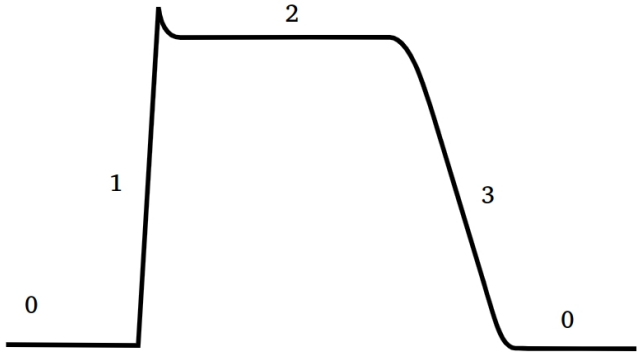

FIG. 1: The action potential of ventricular cardiomyocytes. **0:** Resting potential.**1:** Depolarization. **2:** Plateau. **3:** Repolarization.

B. Mathematical model

Cardiac electrophysiology models serve as a valuable tool for understanding the propagation of APs. These models usually take the form of a reaction-diffusion system. The monodomain model is a PDE that describes the spatial and temporal evolution of the electric potential, V, across the cell membrane:

$$\nabla \cdot (\sigma_m \nabla V_m) = -\chi C_m \frac{dV}{dt} + \chi I_{ion}, \tag{1}$$

where V_m is the transmembrane potential, σ_m the diffusion tensor defined by the local conductivities, χ is the surface to volume ratio,C_m is the capacitance of the cell membrane and I_{ion} is the ionic current.

The most straightforward representation of the AP describes it as an excitation wave, followed by a refractory zone. The Aliev-Panfilov model uses two state variables to model the ionic current across the cell membrane and is given by

$$I_{ion} = -kV(V-a)(V-1) - VW, \tag{2}$$

and

$$\frac{dW}{dt} = (\epsilon + \frac{\mu_1 W}{V+\mu_2})(-W - kV(V-b-1)), \tag{3}$$

where V is an adimensional variable that diffuses across neighboring cells, and W is a non-diffusible variable, called a recovery variable.

Combining the monodomain equation with the Aliev-Panfilov model and using rescaled units yields the following coupled PDE and ODE

$$\frac{dV}{dt} = \nabla \cdot (\sigma_m \nabla V) - kV(V-a)(V-1) - VW, \tag{4}$$

and

$$\frac{dW}{dt} = (\epsilon + \frac{\mu_1 W}{V+\mu_2})(-W - kV(V-b-1)), \tag{5}$$

In the case of isotropic and homogeneous conditions, e.g. the conductions are the same all over the domain and in every direction, the diffusion term in Equation (4) becomes $D\nabla^2 V$, where D is a constant.

III. METHOD

A. Data

1. 1D cable and 2D planar wave

In the pursuit of reproducing the research presented in the article, we adhered to the methodologies described therein for the generation of in silico cardiac electrophysiological (EP) data, which served as the training and evaluation dataset for our implementation of EP-PINNs.

The simulations were initially conducted in two geometries, mirroring the original study: one in a one-dimensional domain, and the other in a two-dimensional domain. As in the original study, we use a central finite difference to approximate the second derivatives and 4th order Runge Kutta iterative scheme for the time evolution. In both cases, no-flux Neumann boundary conditions are applied.

2. MRI based geometry

In addition to the 1D cable and the 2D squared domain, we extend the study into the realm of anatomical accuracy by employing an MRI-based geometry of the heart. The code has been provided by Gabriel Balaban, and includes the creation of a triangular mesh and electrical simulations using finite elements implemented in the Cardiac Arrhythmia Research Package (CARP). This extension to the original methodology provides a more realistic representation of the cardiac EP signal propagation.

B. PINNs

The general idea behind PINNs is that we can construct a loss function such that when minimized, the governing equations are satisfied by incorporating their residuals into the loss function. In the approach first suggested by Raissi et al.[4], a fully connected neural network (FCNN) is used to represent a target function. In this case, the neural network receives the spatial and temporal coordinates as input and outputs the function values at the input points (x_i, y_i, t_i). The neural network has two outputs $\tilde{V}(x_i, y_i, t_i, \theta)$ and $\tilde{W}(x_i, y_i, t_i, \theta)$, where θ is the set of parameters to be optimized in order to get an

accurate representation of V and W. This formulation has the advantage that we can take advantage of automatic differentiation during backpropagation to compute the derivatives with respect to the input.

1. Automatic differentiation

Training a neural network involves determining the optimal weights and biases for each node in the network. Central to this process is the backpropagation algorithm, which is a specific case of Automatic Differentiation (AD). The iterative procedure leverages the chain rule to update the network's weights and biases with the goal of minimizing the discrepancy between predicted and actual outputs. Considering the fact that the neural network represents a compositional function, then AD applies the chain rule repeatedly to compute the derivatives. The advantage of AD over other methods like finite difference, becomes more apparent for a large input space, as it only needs one forward and backward pass to compute the derivatives with respect to the inputs[1].

In a neural network, forward propagation is the process of passing the input data through the network to generate the output. We start by sending the data from each input node into every node in the first hidden layer, where weights and biases are applied to the data. The outputs from layer $l \in \{1, 2, ..., L\}$, denoted by $\mathbf{z}^l \in \mathbb{R}^{N_l}$ then has an activation function f applied to it. Then, it is passed to the next layer where the procedure is repeated.

In vector-matrix notation, we can write this as

$$\mathbf{z^l} = W^l a^{l-1} + \mathbf{b^l}$$

where $a^l = f(\mathbf{z}^l)$, $W \in \mathbb{R}^{N_l \times N_{l-1}}$is a matrix containing the weights, $\mathbf{b}^l \in \mathbb{R}^{N_l}$ is a vector containing the biases and N_l denotes the number of neurons in layer l.

For simplicity, consider a neural with one input, x, one hidden layer, and one output y. The forward propagation then yields

$$\begin{aligned} \mathbf{z}^1 &= W^1 x + \mathbf{b}^1 \\ y &= z^2 \\ &= W^2 a^1 + b^2 \end{aligned}$$

Then through backpropagation, we can find the derivative of the output with respect to the input as follows

$$\begin{aligned} \frac{\partial y}{\partial x} &= \frac{\partial y}{\partial a^1}\frac{\partial a^1}{\partial x} \\ &= \frac{\partial y}{\partial a^1}\frac{\partial a^1}{\partial z^1}\frac{\partial z^1}{\partial x} \\ &= W^2 \frac{\partial f(z^1)}{\partial a^1} \end{aligned}$$

Keeping in mind that this requires the activation function to be differentiable, the procedure can easily be expanded to accommodate more hidden layers.

2. Loss function

We can define residuals of Equations (4) and (5) in terms of the neural network and its derivatives

$$R_V = -\frac{\partial \tilde{V}}{\partial t} + \nabla \cdot (D\nabla \tilde{V}) - k\tilde{V}(V-a)(\tilde{V}-1) - \tilde{V}\tilde{W}$$

$$R_W = -\frac{\partial \tilde{W}}{\partial t} + \left(\epsilon + \frac{\mu_1 \tilde{W}}{\tilde{V}+\mu_2}\right)(-W - k\tilde{V}(\tilde{V}-b-1)),$$

The loss function $\mathcal{L}$ can thus be defined as a combination of different components which ensure both data fitting and physical consistency, balancing the objectives of supervised learning and unsupervised learning

$\mathcal{L}_{data}$ represents the supervised learning part, ensuring that the network's outputs $\tilde{V}$ are close to the data V_i. $\mathcal{L}_{R_V}$ and $\mathcal{L}_{R_W}$ are the residuals, representing the unsupervised learning part, enforcing the network to learn solutions that satisfy the governing physics encapsulated in the PDEs. $\mathcal{L}_{BC}$ enforces the boundary conditions, ensuring that the derivative of the network's output $\tilde{V}$ with respect to the boundary normal $\vec{n}$ at the boundary points (x_k, y_k, t_k) is close to zero. $\mathcal{L}_{IC}$ enforces the initial conditions, ensuring that the network's output $\tilde{V}$ at the initial time t_0 is close to the initial potential V_0.

$$\begin{aligned} \mathcal{L} =& \alpha_d \mathcal{L}_{data} + \alpha_{R_V}\mathcal{L}_{R_V} + \alpha_{R_W}\mathcal{L}_{R_W} \\ &+ \alpha_{BC}\mathcal{L}_{V_{BC}} + \alpha_{IC}\mathcal{L}_{IC} \\ =& \frac{\alpha_d}{N}\sum_{i=1}^{N}\left(V(x_i, y_i, t_i) - V_{GTi}\right)^2 + \\ & \frac{1}{N_f}\sum_{j=1}^{N_f}\left(\alpha_{R_V} R_V(x_j, y_j, t_j)^2 + \alpha_{R_W} R_W(x_j, y_j, t_j)^2\right) \\ & + \frac{\alpha_{BC}}{N_b}\sum_{k=1}^{N_b}\left(\frac{\partial V}{\partial \vec{n}}(x_k, y_k, t_k)\right)^2 \\ & + \frac{\alpha_{IC}}{N_0}\sum_{l=1}^{N_0}\left(V(x_l, t_0) - V_0\right)^2 \end{aligned}$$

- α_d, α_{R_V}, α_{R_W}, α_{BC}, α_{IC} are hyperparameters that weight the importance of each term in the loss function. They balance the contributions from different parts of the loss to the total loss $\mathcal{L}$.
- N, N_f, N_b, N_0 are the numbers of respective samples and are used to average the loss over these samples.

3. Optimization

Adaptive Moment Estimation (Adam) is an optimization algorithm and has become the default choice for

Algorithm 1 Adam with L2 regularization and Adam with decoupled weight decay (AdamW)[5]

initialise $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0}$, learning rate η,
first moment vector $\boldsymbol{m}_{t=0} = 0$,
second moment vector $\boldsymbol{v}_{t=0} = 0$,
schedule multiplier $c_{t=0} \in \mathbb{R}$,
Initialise momentum factors β_1, β_2
repeat
$t \leftarrow t + 1$
$\nabla f_t(\theta_{t-1})$ ▷ Select batch and compute gradient
$\boldsymbol{g}_t \leftarrow \nabla f_t(\theta_{t-1}) + \lambda\theta_{t-1}$
$\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$ ▷ Update first moment
$\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$ ▷ Update second moment
$\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1-\beta_1^t)$ ▷ decay rate to the power of t
$\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1-\beta_2^t)$ ▷ decay rate taken to the power of t
$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - c_t(\alpha \hat{\boldsymbol{m}} / (\sqrt{\hat{\boldsymbol{v}}} + \epsilon) + \lambda\boldsymbol{\theta}_{t-1}$
untilStopping criterion
return Optimal parameters $\boldsymbol{\theta}_t$

training neural networks. Adam combines the benefits of two other extensions of stochastics gradient descent, namely AdaGrad (Adaptive Gradient Algorithm) and RMSProp (Root Mean Square Propagation). The AdaGrad optimisation method adaptively scales the learning rate for each individual parameter such that each update of the parameters is focused in the direction of smaller gradients. To keep updating in the direction of the smallest oscillations the accumulation of the gradients is kept in a gradient accumulation vari- able which is updated every iteration. RMSprop, on the other hand, is an adaptation of AdaGrad where a fixed decay rate is introduced to change the gradient accumulation variable into a weighted moving average. Adam improves on this by also keeping in memory the gradient with a smaller decay rate than the second moment.

Weight decay and L2 regularization are two ways of reducing overfitting. The two methods can be made identical for Stochastic Gradient Descent (SGD) by redefining the weight decay factor based on the learning rate. However, it has been shown that this equivalency does not hold for Adam. Specifically, when coupled with adaptive gradients, L2 regularization results in weights or gradient magnitudes being less regulated than they would be with weight decay[5]. The authors, therefore, propose an improvement of Adam called AdamW by decoupling the weight decay from the optimization steps.

Another popular optimization algorithm, is Limited-memory Broyden–Fletcher–Goldfarb–Shanno(L-BFGS). The L-BFGS algorithm is an iterative method for solving non-linearly approximated optimization problems. The algorithm is particularly useful when dealing with optimization problems with a large number of variables, which makes it ideal for machine learning. Unlike the original BFGS algorithm that requires storing a dense $n \times n$ approximation to the Hessian matrix (where n is the number of variables), L-BFGS stores only a few vectors that implicitly represent the approximation. This makes the memory requirements drastically smaller, hence the "Limited-memory" in L-BFGS. The algorithm's memory efficiency and speed make it a preferred choice for large-scale optimization. For a detailed description of the algorithm, the reader is referred to [6].

4. *Architecture and training*

The PINNs are implemented in Python using Pytorch for the 1D case and DeepXDE[1] for the 2D domains. DeepXDE is a library for scientific machine learning which provides a high-level API, facilitating the creation of PINNs. The number of hidden layers and neurons greatly depends on the problem at hand. A general representation of the architecture is illustrated in Figure 2.

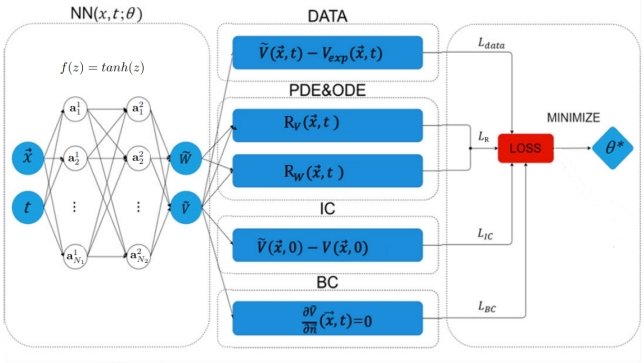

FIG. 2: Neural network architecture. Source: Adaptation of Figure 2 from [2]

For the 1D case we train with both Adam and AdamW as optimization methods for comparison. In the 2D rectangular case, we use a training scheme consisting of initially training with Adam for only $\mathcal{L}_{data}$, followed by the whole loss function, and finally L-BFGS, as suggested in [1]. For the MRI-based geometry, the whole training is done using AdamW.

5. *Evaluation*

In order to evaluate the performance, we record the loss across training points in randomly sampled batches and evaluate the model on equally sized batches randomly sampled from unseen data. Additionally, we evaluate the final optimized models across all test points by calculating the RMSE as done in [2]:

$$\text{RMSE} = \sqrt{\frac{1}{N_{test}} \sum_{i=0}^{N_{test}} (\tilde{V}(i) - V_{test}(i))^2}$$

IV. RESULTS AND DISCUSSION

A. 1D cable

The optimal hyperparameters for the 1D problem can be found in Table I along with an RMSE of 8.8×10^{-3}, computed for the entire test set. Figure 5 Shows how the model's performance improves over time. We see the test loss being consistently higher than the training loss with large fluctuations, which suggests some degree of overfitting because the model performs better on better on the training set and struggles to generalize to the unseen test data.

TABLE I: 1D problem Neural Network Parameters and best model RMSE on test data.

Layers	Neurons	λ	η	Optimizer	$RMSE_{test}$
4	32	0.01	0.005	AdamW	8.8×10^{-3}

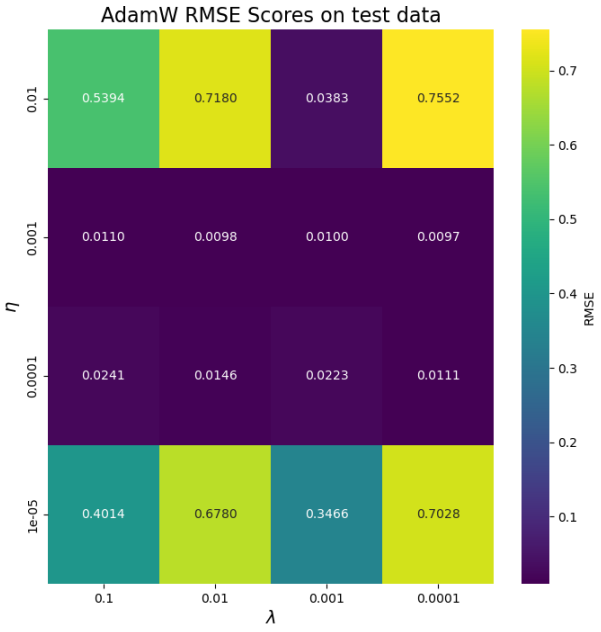

FIG. 3: RMSE of the test data computed using different values for the learning rate η and weight decayλ. The training was done using the AdamW optimizer, 20000 epochs and default coefficients $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$.

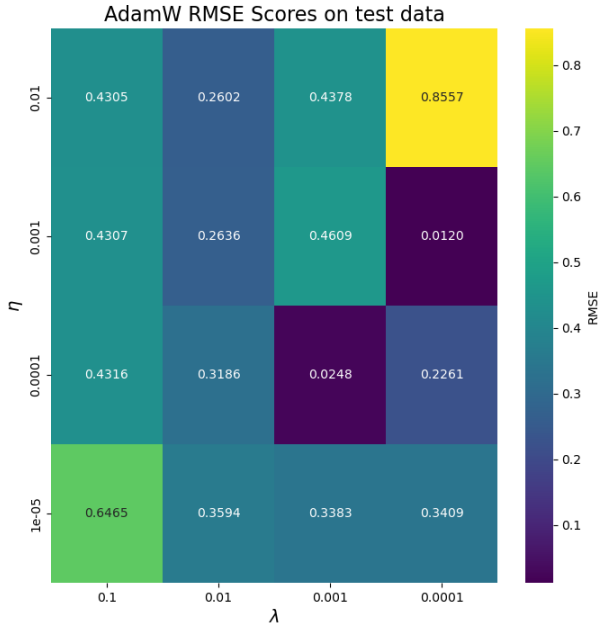

FIG. 4: RMSE of the test data with the computed using different values for the learning rate η and weight decayλ. The training was done using the Adam optimizer, 20000 epochs and default coefficients $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$.

In Figures 4 and 3, we see grid searches for the optimal hyperparameters λ and η using Adam and AdamW respectively. The results indicate that the weight decay parameter significantly impacts performance when using AdamW compared to Adam. This could suggest that AdamW benefits more from weight regularization, potentially leading to better generalization on unseen data.

B. 2D planar wave

Table II provides the optimal hyperparameters found for the 2D problem. The model has five hidden layers with 60 neurons, weight decay $\lambda = 0.01$, a learning rate $\eta = 0.0005$, and uses Adam as the optimizer for two phases followed by L-BFGS as described in section III. This model achieves an RMSE on the test data of 7.9×10^{-4}.

TABLE II: 2D problem Neural Network Parameters and best model RMSE on test data.

λ	η	Optimizer	$RMSE_{test}$
0.01	0.0005	AdamW and L-BFGS	7.9×10^{-4}

The visualizations in Figure 6 shows a relatively accurate reproduction of the planar wave, with errors mainly

FIG. 5: The total train loss and test loss as a function of epochs. The training was done using the AdamW optimizer, 14000 epochs, learning rate $\eta = 0.005$, weight decay $\lambda = 0.01$,batch size of 512 samples, and default coefficients $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$.

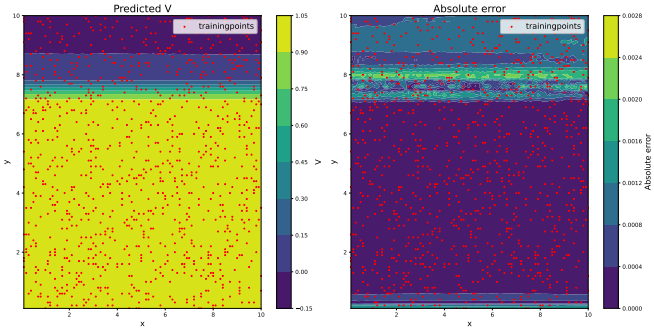

FIG. 6: Visualisation of the prediction to the left and the absolute error between the prediction the ground truth to the right. The red crosses mark the sampled training data for that specific time step.

concentrated around the wavefront, and the top and bottom boundary. This might suggest some limitations of the model in accurately capturing the behavior at these specific regions.

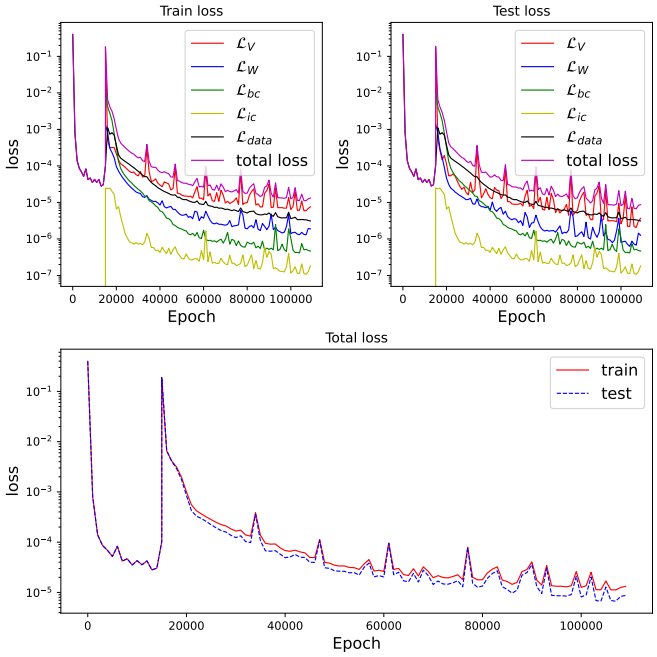

FIG. 7: The total loss along with its constituents as functions of epochs for the 2D planar wave model. The training was done using the Adam optimizer, learning rate $\eta = 0.0005$, weight decay $\lambda = 0.01$, and default coefficients $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$. The model was initially trained For only the data term of the loss function L_{data}, followed by training on the total loss function, and finally a third phase of otimization with L-BFGS.

Figure 7 shows the total loss and its constituents, where we see that the total loss is dominated by the PDE loss $\mathcal{L}_V$. Additionally, we see the training loss being slightly higher than the test loss, which could be due to the use of L2 regularization, considering that the penalty term is only added to the loss function under training and not for evaluation. This also suggests that the model generalizes well to unseen data, which is confirmed by the low RMSE from TableII computed for the test data.

C. MRI based geometry

The model for the MRI-based geometry has five hidden layers, where the first layer has 100 neurons and the remaining layers 50 neurons. The training was done using AdamW with learning rate $\eta = 0.0005$, weight decay $\lambda = 0.01$, and default coefficients $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$. A visualization of the electric wave propagation in an MRI-based 2D geometry can be seen in Figure 8 as snapshots taken at different time points. We see that the predictions respect the dynamics of the system as long as the prediction are done within the timeframe of the data. When trying to extrapolate beyond the data, we see the PINN fails to make physical predictions.

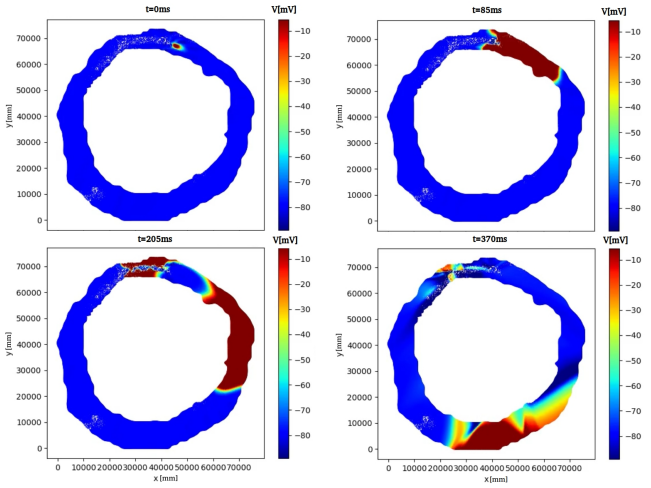

FIG. 8: visualization of the electric wave propagation in an MRI-based 2D geometry.

In Figure 9, we see the training history, which shows a loss dominated by the data term $\mathcal{L}_{data}$. Additionally, we see the losses starting to converge early, indicating that the optimization is stuck at a local minimum.

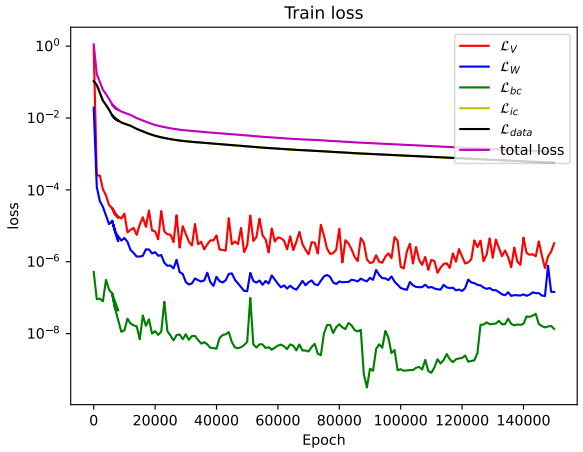

FIG. 9: The total loss along with its constituents as functions of epochs for the 2D planar wave model. Training was done using the AdamW optimizer with $\lambda = 0.01$ and $\eta = 0.0005$

V. CONCLUSION

We have demonstrated the potential of the application of PINNs in EP simulations. In terms of the 1D problem, our results reveal that the AdamW optimizer, as proposed by the authors in their article, shows potential. However, in this case, it also suggests some degree of overfitting due to the test loss consistently being higher than the training loss (as seen in Figure 5). The RMSE achieved was 8.8×10^{-3}, which aligns with the results reported by EP-PINNs for a similar problem, thus affirming the effectiveness of AdamW as an optimizer.

However, the 2D planar wave scenario presents a different case. Our model achieved an impressive RMSE of 7.9×10^{-4} on the test data, surpassing the lowest RMSE reported by EP-PINNs of 5.6×10^{-3}. It is worth noting that the error in our 2D model is concentrated around the wavefront and the boundaries (Figure 6). Despite this, the model appears to generalize well to unseen data, as displayed by the lower test loss compared to the training loss (Figure 7). The choice of optimizer appears to have a significant role in these results, with a combination of Adam and L-BFGS proving most effective for this problem.

The results of the MRI-based 2D geometry provide further insights. The model was able to successfully predict the dynamics of the system within the timeframe of the data. However, it failed when extrapolating beyond the data (Figure 8). This suggests that the model is lacking in terms of learning the physics behind the propagation of the electric wave. This can be explained by the absence of information about the orientation of the myocardial fibers that drive the electric signal propagation. Hence, implementation of fiber orientation is an important aspect for future improvements. Another explanation, not necessarily exclusive, is the existence of a local minimum that the optimization is unable to surpass (as illustrated in Figure 9).

[1] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, SIAM Review **63**, 208 (2021).
[2] C. Herrero Martin, A. Oved, R. A. Chowdhury, E. Ullmann, N. S. Peters, A. A. Bharath, and M. Varela, Frontiers in Cardiovascular Medicine **8** (2022), 10.3389/fcvm.2021.768419.
[3] J. Sundnes, G. T. Lines, X. Cai, B. F. Nielsen, K.-A. Mardal, and A. Tveito, eng*Computing the Electrical Activity in the Heart*, Monographs in Computational Science and Engineering, Vol. 1 (Springer Berlin Heidelberg, Berlin, Heidelberg).
[4] M. Raissi, P. Perdikaris, and G. Karniadakis, Journal of Computational Physics **378**, 686 (2019).
[5] I. Loshchilov and F. Hutter, CoRR **abs/1711.05101** (2017), 1711.05101.
[6] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, SIAM Journal on Scientific Computing **16**, 1190 (1995), https://doi.org/10.1137/0916069.