Determining the Material Parameters of the Arterial Wall of a Mouse - A comparison of computational approaches

Bharath Narayanan

***Abstract*—Given geometric data from synchrotron images, and finite element simulations, we compare three numerical techniques that can be used to determine the material properties of the arterial wall of a mouse.**

The first technique combines Artificial Neural Networks with standard optimization algorithms. The second is a straightforward Matlab optimization routine with finite-difference estimates of the gradients. The third uses Matlab's optimization routine while supplying semi-analytical gradients.

We find that the semi-analytical method works better but with more complex models, the artificial neural network based surrogate model might prove more efficient.

I. Introduction

A. Motivation

It has been documented that the presence of aortic aneurysms in mice and humans is related to the material characteristics of the wall of the aortic artery. Thus far, in most studies, the aortic wall has been treated as a homogeneous entity. However, the advancement in imaging technology has allowed us to observe the arterial walls in mice using synchroton imaging. The wall consists of media and adventitia. The scans reveal the presence of 5 distinct layers within the media, 3 **lamellar layers** primarily composed of elastin and 2 **interlamellar layers** made of smooth muscle, elastin, and collagen. As a long term goal, we would like to determine the material properties of each of these layers.

B. Problem Statement

At the start of the project, synchrotron-based images were available of the arterial wall of a mouse, that is expanded to **eight** different pressure levels under controlled conditions (ex-vivo). The goal is to use these scans in conjunction with the finite element software FEAP to be able to determine the material properties of the layers of the wall.

II. Process Flow : From Synchrotron Imaging scans to an optimization problem

This section, with the aid of figures 1 and 2, details the process of converting the available images into an optimization problem.

A. Images to Simulations

- The scans are first converted into a locus of points in 3D. These points are divided into slices along the longitudinal axis. Each slice contains a set of points that describe it.
- This 3D point cloud is then converted into a set of NURBs control points by projecting the points onto a cylinder of the same approximate diameter and height as the artery.
- The number of points used to discretize the geometry is kept the same throughout each simulation and geometry.
- The first scan is used as a base and fed into the software FEAP. Under the application of material properties, simulations are conducted at various pressures and the resulting geometries are once again stored as NURBs control points.

The following assumptions are made regarding the material models for the FEAP simulations.

- The materials follow a non-linear Arruda-Boyce hyperelastic model [1] which describes their behaviour with the following inputs:
  - Young's modulus **Y**, which describes the stiffness.
  - Poisson's ratio ν, which describes expansion in perpendicular axes under uniaxial load.
  - Power law parameter **m**, which determines the non-linear behaviour at different pressures.
- The lamellar, and the intelamellar layers have the same material properties respectively. This gives rise to **2 materials** having **3 parameters** each, i.e. **6 parameters** or features in total.

B. Simulation Outputs to Errors

- Since the simulated final geometries, as well as the geometries outputted from the scan itself are acquired by projecting onto the same cylinder in the same orientation, each NURBs control point of the simulated geometry can be compared with the control point of the scanned geometry at the same index.
- This allows us to calculate the 'error' at each pressure level as follows.

$$e_i = \sum_{j=1}^{N_{cp}} |\mathbf{r}_j^i - \hat{\mathbf{r}}_j^i|_{L2} \tag{1}$$

where $\mathbf{r}$ is the simulated coordinate and $\hat{\mathbf{r}}$ is the target coordinate obtained from the scanned geometry. N_{cp} is

the total number of control points. e_i here refers to the error at a given pressure level.

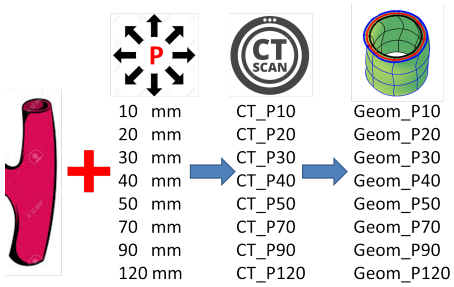

Fig. 1. The process flow from the ex-vivo artery to the geometries at various pressure levels. **Step 1:** The artery is attached to a catheter which can pump in the requisite pressure. **Step 2:** At various pressures of inflation, the entire artery is scanned using synchrotron imaging. **Step 3:** The scans are then converted into a set of NURBs control points which are used to describe the geometries.

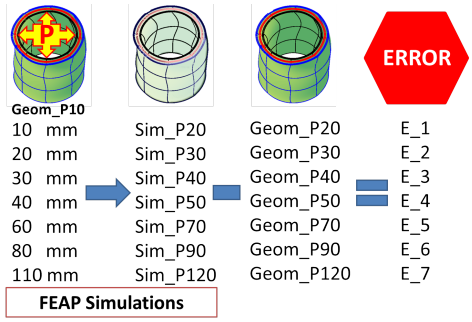

Fig. 2. The process flow that uses the geometry at 10mm Hg of pressure to obtain the various errors at the other pressure levels. **Step 1:** The geometry is fed into the software FEAP. **Step 2:** For a given **trial set of material parameters**, 7 different pressures are applied to obtain 7 different simulation geometries. **Step 3:** These simulated geometries are then compared with the geometries obtained from the scans to provide an error estimate. **Step 4:** The error is the average L2 norm of the distance between the simulated and target NURBs control points.

C. The Final Optimization Problem

We would like to find the optimum set of **six** material parameters that minimize the errors at the **seven** pressure levels, In mathematical terms, the function to be minimized can be described as follows.

$$f(\mathbf{x}) = \sqrt{\sum_{i=1}^{7} e_i(\mathbf{x})}^{2} \tag{2}$$

where e_i refers to each of the 7 errors (outputs). The objective function f is the function that needs to be minimized with respect to the material parameters represented by the vector $\mathbf{x}$. In addition to the function f, an optimization problem would also need the gradients of the function, with respect to the inputs, which are provided as follows:

$$\frac{df}{d\mathbf{x}} = \sum_{i=1}^{n} \frac{df}{de_i} \frac{de_i}{d\mathbf{x}} \tag{3}$$

The first component of the gradient is obtained easily through simple algebra. It is the ease or difficulty in calculation of the second component that will determine our method for solving this problem.

D. Possible Approaches - The case for ANN

The optimization problem described above can be solved in one or more of the following ways:

- **Optimization with gradients using Finite Differences**: This is the default method followed by Matlab's interior-point and SQP algorithms. Each function evaluation would need $n_{param} + 1$ simulations in order to obtain the gradients $\frac{de_i}{d\mathbf{x}}$ using finite differences. This does not scale well with more parameters and a finer grid.
- **Optimization with semi-analytical gradients**: A modification of the first option is to analytically provide gradients. However, this involves the inversion of matrices (or solving them using linear solvers).
- **Artificial Neural Networks**: This approach is to use large volumes of data. Given a design space, we conduct hundreds of independent simulations simultaneously. The data is then used to train a *surrogate model*. Since the training process also yields the gradients of the outputs with respect to the inputs, the *optimization process* is also much faster. This approach, though new, has been successfully used to replace finite element simulations [2][3][4].

In this project, our goal is to compare the three different methods used to solve the optimization problem.

III. Artificial Neural Network (ANN) Model

A. What is an ANN?

An artificial neural network uses a series of layers and connections to model various systems. It uses large sets of training data to create a model that can predict outputs given a set of inputs. The most basic type is known as the Multi-Layer Perceptron model which is modelled on the behaviour of a neuron as shown in figure 3. The structure of a neural

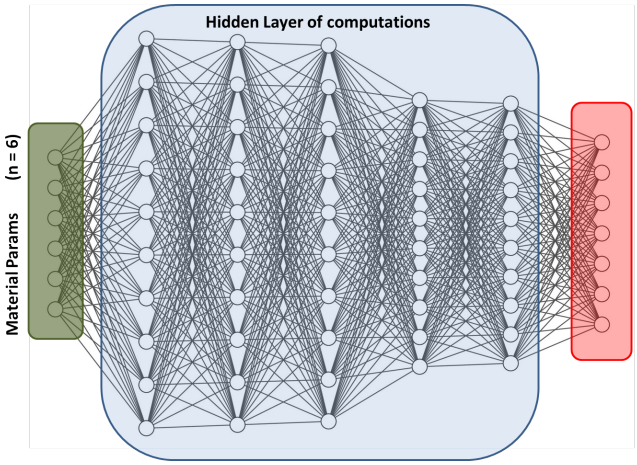

Fig. 3. A sample MLP neural network with 5 hidden layers.

network is defined as follows:

- **The input layer**: This is the initial layer which contains all the inputs (input neurons).
- **The output layer**: This is the final layer that contains one or more outputs (output neurons).
- **The hidden layers**: Between the input and output layers, we can have one or more hidden layers. Each layer has multiple neurons which are connected to the neurons in the preceding and subsequent layers. The number of hidden layers, and the number of neurons in each layer, *cannot* be determined analytically.
- Each neuron j at a layer l receives inputs from each neuron i in the previous layer $l-1$. It then combines these inputs in a linear fashion, with each input being multiplied by a weight $w_{i,j}$. This is shown in the equation below:

$$z_j^l = \sum_{i=1}^{N} w_{i,j}^l x_i^{l-1} \tag{4}$$

  where N is the number of neurons in the previous layer, $l-1$.
- An **activation function** ϕ acts on this combination to form the value at the current node, x_j.

$$x_j^l = \phi(z_j^l) \tag{5}$$

  The choice of activation function can be important. In general, the rectified linear unit, RELU, and the logistic function, SIGMOID, are found to work well. Once again, there is no clear choice between the two and we use trial and error to determine the best option.
- At the final layer, i.e. the output layer, each output is the linear combination of the previous layer's neurons without the imposition of an activation function.
- For a given initial set of weights, the neural network predicts the outputs for a given set of inputs.
- Then, a **loss function** (mean-squared-error, mean-absolute-error, etc.) is calculated using the error between the predicted output and the actual output corresponding to the sample.
- The gradients of this loss function with respect to the different weights are then calculated and a new set of weights is determined.
- This is then continued until the loss function is deemed to be small enough.

In essence, we are solving an optimization problem to determine the optimum values of all the weights that minimize the error between our predictions and the actual outputs seen at each sample. In addition to the number of hidden layers and neurons, we also need to decide on the following optimization parameters.

- **Optimizer**: This is the method used for determining the gradients in the optimization. For most cases, a variation of a stochastic gradient method is used.
- **Batch-size**: This determines the number of samples to be used for determining the gradient. In stochastic gradient descent, a smaller subset of the total dataset is used for determining the gradients. This is done to save time and memory, especially when datasets get too large.
- **Learning rate**: Once a step direction is determined, the learning rate tells the optimizer how far to proceed along the given direction.
- **Epochs**: This is the total number of iterations to be performed before stopping the optimization process. It obviates the need for another stopping criterion.

In general, we do not use all the data for training. This is done to ensure that we can avoid overtraining. For example, if we have an extremely large number of weights, it is quite likely that we can attain the exact outputs for each set of inputs. However, this network would then be 'over-trained' and perform poorly for any sample that is outside this training set.
As a best practice, we divide the dataset into **K** parts. We then use K-1 out of these for training and 1 for testing to see how the model performs for data that is not part of the training set. This is then repeated **K** times, with each part forming a test set once. This method is known as **K-fold validation**.

B. Generating Data for the ANN

The first step towards creating a robust Neural Network model is creating the sample space. Since we have complete control of the simulation process, it is up to us to choose the design space within which we want to generate samples. For this, we need to decide the bounds, the number of samples, as well as the sampling method.

1) Bounds: The bounds of the design space are described below:

- The choice of **Y** bounds was done based on a time constraint. When Y_{lam} (Young's modulus of the lamellar layer) is set to 50 KPa, and $Y_{interlam}$ (Young's modulus of the interlamellar layer) is set to values below 180 kPa, the simulation does not converge unless we use a smaller step size. We also ran a few test simulations to see at what values of Y_{lam} and $Y_{interlam}$ the simulated artery expands to a diameter larger than the largest pressure geometry. We found this to be at around 100 kPa. We used this information to set the bounds in table I. Note that the larger value of the Y lower bound corresponds to the interlamellar layer. In addition to these heuristic methods, the bounds we use correspond well with the average properties of the arterial wall reported by Deng et al [5], and Le et al [6]. They calculated values in the range of a few hundred kilo-Pascals.
- Poisson's ratio ν for most materials lies below 0.5 since 0.5 represents an incompressible material. The lower bound is set to 0.2 as an initial guess.
- The bounds for parameter **m** were determined after reading the paper by Bram Trachet and Joris Bols [7]. He uses a value of 1.01 for his λ_m parameter which is equivalent to $\frac{1}{m^2}$. Note that his parameter is used in a homogeneous *abdominal* artery wall model. Considering that the artery in our case is in the aorta, our **m** value need not necessarily be the same.

TABLE I
BOUNDS FOR THE DESIGN SPACE. WE HAVE TWO MATERIALS WITH THREE PARAMETERS EACH.

Parameter	Lower Bound	Upper Bound
Y (kPa)	50/180	400
ν	0.2	0.49
m	0.5	1.5

2) Number of samples: If we decide to do a grid search with even 3 points for each parameter, we would end up with 3^6 or 729 simulations. We decide to go for a conservative number of 500 simulations to see if this is sufficient to train our algorithm. In theory, we could extend this to an extra order of magnitude but we might fill the design space to an extent where the power of ANNs is not clearly projected.

3) Distribution of samples: We distribute the 500 samples in the design space using a **Latin-Hyper Cube** sampling system (Fig. 3 in Appendix). This is found to work better than using a Monte-Carlo distribution of points [8]. The simulations took about 5 hours to obtain, using 5 parallel batches. The post-processing of data from the simulations was done using Matlab. This yielded **500 samples** of **6 features** (the material parameters), and **7 outputs** (the 7 L_2 norm errors for each simulation).

4) Standardization using min-max: The different material parameters have values that are orders of magnitude apart, which makes training on the raw data infeasible. We decide to standardize the data using a min-max standardization. However, instead of detecting the min and max values automatically, we provide them ourselves using the bounds described above.

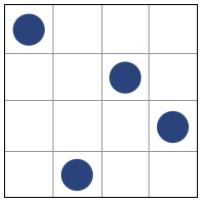

Fig. 4. Latin-Square (2-D) sampling. Notice how in each row and each column, there is only one sample. This is a more efficient extension of the monte-carlo algorithm.

C. Hyperparameter Tuning - Choice of Network

The successful creation of a surrogate ANN model depends on many parameters of the network that need to be chosen. We conduct hyperparameter tuning for the ANN by studying the impact of various hyperparameters on our chosen metrics. Since ours is a regression model that will later be used for optimization, we require a high fidelity model. Hence, we use the median and maximum errors as metrics. This is in contrast with the usual mean-squared-error used for most other problems that stem from random, natural processes. The reason for this is that, a system with low average errors might still have significant outliers. If this is the case, then the model cannot be trusted in certain ranges. Tracking the maximum error ensures that our worst case scenario is still within an acceptable limit. In the interest of time, we use two-fold cross validation in this stage, instead of a ten-fold cross validation. This would require only 2 runs per parameter choice instead of 10.

1) Number of hidden layers: We decide to use 2 hidden layers since we have a small number of features and outputs, lest we end up overtraining. It seems that for most purposes, 2 hidden layers suffice [9].

2) Number of neurons per layer: In our initial configuration, we set the batch size, epochs, activation function, optimizer and learning rate to 10, 2000, sigmoid, Adam and 0.001. We vary the number of neurons in each layer from 5 to 40. We decide to use this range based on a combination of factors: rule-of-thumb methods, trial and error, trade-offs between keeping the number of neurons low to prevent overfitting and good metrics. The Tables II and III (along with I, II in the Appendix) show the results of tests done to determine the optimal number of neurons per hidden layer. Our final choice is 40 neurons in the first layer and 30 neurons in the second layer, which we use in our remaining experiments.

3) Batch size and Epochs: The batch size determines the number of samples used at a time to determine the gradient used for the descent direction. We vary the batch size, keeping the other hyperparameters unchanged. Our results indicate that a smaller batch size of 10 gives the least error metrics. We also

TABLE II
MAXIMUM ERROR WITH DIFFERENT NUMBER OF NEURONS IN TWO LAYERS. COLUMN AND ROW INDICATE NEURONS IN FIRST AND SECOND LAYER RESPECTIVELY.

Max Error (%)	5	10	15	20	30	40
5	60.019	58.751	47.510	49.423	47.269	–
10	51.854	45.618	47.385	43.481	41.627	–
15	47.186	49.497	46.587	43.043	44.327	–
20	49.843	49.992	48.614	45.954	42.564	–
30	55.596	48.395	44.940	52.712	43.849	–
40	–	–	40.516	51.551	**32.683**	38.979

TABLE III
MEDIAN ERROR WITH DIFFERENT NUMBER OF NEURONS IN TWO LAYERS. COLUMN AND ROW INDICATE NEURONS IN FIRST AND SECOND LAYER RESPECTIVELY.

Median Error (%)	5	10	15	20	30	40
5	0.714	0.427	0.538	0.449	0.440	–
10	0.533	0.466	0.489	0.360	0.245	–
15	0.436	0.411	0.397	0.211	0.329	–
20	0.393	0.437	0.365	0.303	0.177	–
30	0.347	0.239	0.100	0.135	0.189	–
40	–	–	0.081	0.079	**0.074**	0.087

look at the variation in loss(MSE) with number of epochs for each batch size and find that a) a size of 10 has the fastest convergence b) 1500 epochs is sufficient for convergence. The loss behavior plot is in the Appendix.

4) Activation function: The fundamental choice was among the Sigmoid, Tanh and Relu functions. The Sigmoid and Tanh can capture non-linearities present in the system while the Relu model can avoid vanishing gradients. We conduct a set of runs, varying only the activation function. We choose Sigmoid as our final activation function, though there was little difference between the Tanh and the Sigmoid. For the final output layer, we do not use any special function as we do not have a classification problem (which might need Softmax or Sigmoid).

5) Optimizer and learning rate: For the stochastic gradient descent, we use the Adam optimizer, which seems to be recommended for general use. This is because it not only looks at the gradient but also at the momentum. In other words, the update at each iteration is also based on the gradient at the previous iteration, thereby providing a better estimate of the descent direction and step size [10][11]. We study the loss convergence for various learning rates and find that even though a rate of 0.01 converges faster and might lead to shorter training time, a rate of 0.001 provides the least errors at higher number of cross-validation folds (plot in Appendix).

6) Loss weights: One of the features of Keras is the ability to specify loss weights for each specific output. This is just a scalar multiplied to each output. It is seen in Section VI that the addition of these loss weights proves to be quite useful in ensuring that the last output in particular is modelled sufficiently accurately in terms of the maximum error.

D. Final ANN Model

The final model can be recreated by following the choices outlined in table IV. We do not use regularization techniques, such as dropout, since our experiments did not show a significant difference between training and test errors. The results of 10-fold cross validation with the final choices is shown in V. Changing the activation to Tanh or using epochs of 1500 or 2000 prove to have little effect on the overall performance. To be on the safe side, the model used for optimization uses 2000 epochs.

NOTE: We found that using 50 neurons in the first layer provided for a better result further down the chain, in the optimization part. Hence we have updated this value. However, we have not included it in the test matrix in tables II and III.

TABLE IV
TUNED HYPERPARAMETERS OF THE FINAL MODEL.

Hyperparameter	Value	Space
Optimizer	Adam	Adam, SGD, Adamax
Learning Rate	0.001	0.0001 .. 0.1
Hidden layers	2	2
Neurons per layer	50 (layer 1), 30 (layer 2)	5 .. 40,50 (each layer)
Activation function	sigmoid	sigmoid, tanh, relu
Epochs	1500	1500, 2000
Batch Size	10	10 .. 100

TABLE V
METRICS USING FINAL MODEL WITH 10-FOLD CROSS VALIDATION (TRAIN/TEST = 450/50)

Metric	Value (%)
Mean Error	0.236 ± 0.215
Maximum Error	6.050 ± 9.121
Median Error	0.057 ± 0.024

E. Optimization Routine

Once a surrogate model is built, we use the inbuilt gradient functionality in Tensorflow to create an optimization routine. The function to be optimized is the norm of each output vector as shown in equation 2. The second component of the derivative, $\frac{de_i}{d\overrightarrow{x}}$ is provided by tensorflow's inbuild command, **tensorflow.gradients**. We use Sci-Kit Learn's minimize function , coupled with an SQP algorithm, to obtain the minimum within bounds. We chose SQP after trying a few other constrained and bounded optimization algorithms (L-BFGS-B, TNC, trust-constr), and finding them unsatisfactory.

IV. DIRECT OPTIMIZATION USING FINITE DIFFERENCE GRADIENTS

An alternative to using the ANN based surrogate model in section III is to run a simple optimization problem directly using Matlab's fmincon algorithm. The **bounds** specified are

exactly the same as in table I. The algorithm used is **interior-point**. The objective function to be minimized is that specified in 2. The algorithm evaluates the function and then determines the next location by calculating the gradient of the objective function f, with respect to the input material parameters $\mathbf{x}$, by using finite differences. If the number of features is n, then, in addition to the simulation needed for the function evaluation at the current point, the algorithm conducts a further n (forward FD) or $2n$ (centred FD) simulations to approximate the gradients. This method has the following positives and negatives:

- **+ve** : There is no need for any calculations on our side. We just feed in the function that calculates the objective function when provided the material data.
- **-ve** : It becomes computational very expensive as the number of features and the resolution of the mesh increase.
- **-ve** : The gradient approximations are not very accurate, especially if the function is non-linear. This would mean a larger number of steps taken before we reach the optimum value.

V. Direct Optimization using Semi-Analytical Gradients

As a faster and less expensive alternative to the gradient evaluation in section IV, we develop a semi-analytical method where many components of the gradient are determined using **matrices that are already computed for the simulation in FEAP**. The basic components of this method are outlined below:

- The gradient of the error at pressure i w.r.t. the input material parameters in vector $\mathbf{x}$ is calculated as follows:

$$\frac{de_i}{d\mathbf{x}} = \frac{de_i}{d\mathbf{u}}\frac{d\mathbf{u}}{d\mathbf{x}} \tag{6}$$

  where $\mathbf{u}$ represents the vector of displacements of each degree of freedom in the model.
  Note : In equation 1, we have used $\mathbf{r}$ for the error calculation instead of $\mathbf{u}$. This is not a mistake. The $\mathbf{r}$ refers to the coordinate at each control point. Each $\mathbf{r}_j$ at control point j is a vector $[u_j^1, u_j^2, u_j^3]$. The vector $\mathbf{u}$ is the assembly of all the degrees of freedom at all the control points $j = 1...n_{cp}$.
- The first component of the derivative can be obtained easily from the equation given by 1. The second component is obtained by considering the nonlinear system of equations that the FEAP simulation is actually solving :

$$f(\mathbf{x}, \mathbf{u}) = 0 \tag{7}$$

- The total derivative of the non-linear system with respect to the material parameters should be zero. Expanding the total derivative using a chain rule provides a relationship between the material and displacement ($\mathbf{u}$) vectors.

$$\frac{D\mathbf{f}}{D\mathbf{x}} = 0 \tag{8}$$

$$\implies \frac{df}{d\mathbf{x}} + \frac{df}{d\mathbf{u}}\frac{d\mathbf{u}}{d\mathbf{x}} = 0 \tag{9}$$

- Finally, using equations 8, we can reframe equation 6 as:

$$\frac{de_i}{d\mathbf{x}} = -\frac{de_i}{d\mathbf{u}}\left(\frac{df}{d\mathbf{u}}\right)^{-1}\frac{df}{d\mathbf{x}} \tag{10}$$

- The term $\frac{df}{d\mathbf{u}}$ is the tangent stiffness matrix used by FEAP. This can be outputted easily at each pressure level.
- The term $\frac{df}{d\mathbf{x}}$ is the only part that now cannot be obtained analytically. However, this is just the derivative of the **residual** f with respect to the material parameters. This can be obtained easily via finite differences and a FEAP restart file. The important thing to note here is that we only need to **assemble** a new f for the finite difference calculation, instead of conducting **whole new simulations**. This is an operation of a few seconds as opposed to a few minutes.

VI. Results

A. ANN Model

1) Impact of Loss weights: In table VI, we can see the impact of loss weights (**LW**) on the error percentages. Despite having a marginally poorer performance in the first 4 error outputs, the advantage of the loss weights is seen in the maximum error at higher pressure values. These were obtained for a test-train split of 0.2.

TABLE VI
Impact of loss weights - Error %

Output weight	**E-1** 0.1	**E-2** 0.1	**E-3** 0.1	**E-4** 0.5	**E-5** 0.6	**E-6** 0.7	**E-7** 1.0
Mean(LW)	0.12	0.13	0.14	0.14	0.19	0.17	0.22
Mean	0.05	0.07	0.06	0.08	0.13	0.12	0.18
Med.(LW)	0.09	0.1	0.09	0.09	0.12	0.11	0.15
Med	0.04	0.06	0.04	0.05	0.06	0.06	0.07
Max(LW)	0.77	0.94	1.56	1.53	1.5	2.1	1.78
Max	0.54	0.71	1.05	1.28	3.24	4.24	4.65

2) Optimization using the surrogate model: The optimization routine, using the ANN surrogate model, converges to a minimum function evaluation of 52.41, yielding a feature vector as follows:

- **Lamellar Properties**: E = 50 kPa, ν = 0.2, m = 0.74
- **Inter-lamellar Properties**: E = 330.85 kPa, ν = 0.44, m = 1.5

Once this is obtained, we run a simulation using the same features and compare the results of each of the 7 outputs with those predicted by the optimizer-surrogate model combination (See Table VII). The mean difference in predicted and actual values is around 0.1% which is within an acceptable range. In order to ensure that this is indeed the minimum value within the entire dataset, we also check the minimum norm of the errors in the entire training set and find it to be 52.8788. If our minimum function value was larger than this, it would have meant that the entire system was incapable of finding even the minimum value within the provided dataset.

TABLE VII
Optimization result

Output	**E-1**	**E-2**	**E-3**	**E-4**	**E-5**	**E-6**	**E-7**
Sim.	12.51	13.47	19.05	19.45	19.48	23.25	27.10
Optim.	12.50	13.47	19.08	19.46	19.43	23.18	27.41
% Diff.	0.07	0.008	0.2	0.07	0.27	0.3	1.13

3) Robustness of the entire tool chain: A neural network can provide results that are *different* even when fed the *same data*. This can then influence the outcome of our optimization routine. In order to investigate the robustness of our entire tool chain, we have conducted 3 different training runs followed by the optimization process. The results are in table VIII. It can be seen that the outputs are fairly close to each other. However, as Table IX shows, the parameters ν and **m** that yield these values can be quite different. It can be observed that the Poisson's ratio is especially volatile and fluctuates a lot between runs.

TABLE VIII
Comparison of 3 different runs - Outputs of function minima

Output	**E-1**	**E-2**	**E-3**	**E-4**	**E-5**	**E-6**	**E-7**
Run-1	12.50	13.47	19.08	19.46	19.43	23.18	27.41
Run-2	12.52	13.55	19.34	19.78	19.85	23.46	27.90
Run-3	12.29	13.27	18.78	19.27	19.51	23.29	27.30

TABLE IX
Comparison of 3 different runs - Properties of Lamellar and Inter-lamellar layers

Output	**Lamellar**			**Inter-Lamellar**		
	Y (kPa)	ν	**m**	**Y(kPa)**	ν	**m**
Run-1	50	0.2	0.74	330	0.44	1.5
Run-2	50	0.38	0.984	394	0.297	1.5
Run-3	50	0.49	0.872	306	0.45	1.5

B. Optimization with in-build Finite-Difference Gradients

The results obtained by using Matlab's interior-point method, with gradients approximated using finite differences, are shown in table X. Note that each function evaluation requires at least 7 simulations. The optimizer reaches the function value of 52.41 predicted by the ANN-based optimization (sec. VI-A2) in **2 hours and 34 minutes**. The algorithm was stopped after 16 iterations in order to provide a comparison with the semi-analytical method in section VI-C.

TABLE X
Functional evaluations using Finite-Difference Gradients.

Iter	**F-count**	**f(x)**	**Optimality**	**Step-size**	**Time (s)**
0	7	58.869	1.48E+08		1992.857312
1	16	55.230	2.61E+07	5.54E-03	4016.841834
2	23	53.592	1.69E+07	7.93E-01	5495.904916
3	31	52.882	2.46E+07	1.26E-01	7318.828766
4	**39**	**52.174**	**4.64E+06**	**5.93E-02**	**9250.484152**
5	47	51.996	2.72E+07	2.37E-02	11138.59503
6	54	51.946	2.16E+06	1.27E-01	12776.42802
7	61	51.840	7.48E+05	5.32E-02	14421.50157
8	68	51.780	5.74E+06	4.18E-02	16039.20439
9	75	51.788	2.47E+06	9.78E-03	17711.38578
10	82	51.773	1.55E+06	1.70E-02	19391.56054
11	89	51.744	1.10E+06	2.74E-02	21066.58439
12	96	51.724	9.92E+05	1.53E-02	22712.34176
13	103	51.716	9.88E+05	4.85E-03	24390.39135
14	110	51.709	9.90E+05	5.37E-03	26015.08432
15	117	51.705	1.02E+06	1.90E-03	27694.15831
16	127	51.705	1.02E+06	8.33E-04	30066.79669

The final material properties are:

- **Lamellar Properties**: E = 89.7 kPa, ν = 0.457, m = 1.28
- **Inter-lamellar Properties**: E = 238 kPa, ν = 0.474, m = 1.38

The final objective function value is 51.705.

C. Optimization with Semi-Analytical Gradients

The results of the semi-analytical gradient based optimization are shown in table XI. The iteration in bold is where the function evaluation is lower than the value of 52.41 *predicted* by the ANN (sec. VI-A2) using the optimum material parameters. We can see that the SA gradient method reaches the

TABLE XI
FUNCTIONAL EVALUATIONS USING SEMI-ANALYTICAL GRADIENTS.

Iter	F-count	f(x)	Optimality	Step-Size	Time (s)
0	1	58.869	1.49E+08		467.94434
1	4	55.230	2.49E+07	5.47E-03	1922.789478
2	5	53.626	1.62E+07	7.78E-01	2436.565702
3	7	52.947	2.75E+07	1.37E-01	3476.453458
4	9	52.859	5.40E+07	4.54E-02	4521.581669
5	**10**	**52.094**	**3.76E+07**	**1.38E-01**	**5041.03577**
6	12	52.001	8.74E+06	8.35E-02	6141.814817
7	14	51.918	3.03E+07	8.61E-02	7190.015427
8	15	51.836	5.16E+06	1.07E-01	7721.143276
9	16	51.806	5.63E+05	4.56E-02	8249.961671
10	17	51.778	1.82E+06	4.07E-02	8775.145602
11	18	51.751	3.37E+05	2.56E-02	9326.403406
12	19	51.733	4.89E+05	1.21E-02	9880.986283
13	20	51.721	8.09E+05	7.81E-03	10406.05615

ANN predicted result in **1 hour and 24 minutes**.
The optimum material parameters found using the semi-analytical gradient method are:

- **Lamellar Properties**: E = 90.67 kPa, ν = 0.4516, m = 1.2620
- **Inter-lamellar Properties**: E = 238.25 kPa, ν = 0.4745, m = 1.3815

The final objective function value is 51.721.

VII. DISCUSSION

A. Comparison of the methods

1) Predictive Abilities: We compare the predictions of the different methods with known values of the Young's Modulus and the Poisson's ratio.

- The study by Asawinee et al. [12] reports values in the range of 46 - 113 kPa for the lamellar region and 136 to 350 kPa for the interlamellar region. The Poisson's ratio for the materials is also expected to be in the range of 0.4.
- **The ANN surrogate model** (see table IX) predicts Young's modulii of 50 kPa for the lamellar regions and around 300 - 390 kPa for the interlamellar regions. The Poisson's ratio, however, varies quite a lot (0.2 - 0.49) between each run.
- The **finite-difference gradient** method predicts **Y** values of 89.7 KPa and 238 kPa, and ν values of 0.457 and 0.474, for the lamellar and the interlamellar regions respectively.
- The **semi-analytical gradient** method predicts values of 90.67 kPa and 238.25 kPa respectively for the lamellar and interlamellar regions. The Poisson's ratios are within the expected ranges of 0.45 - 0.49.
- It can be seen that that the semi-analytical and the finite-difference gradients agree quite well with each other. This provides further validation of our semi-analytical formulation.

It is important to note that while the semi-analytical and finite-difference based optimizers will always provide the same results, given the same initial conditions, there is no *implicit guarantee* of the same with ANNs. This can be observed clearly in tables VIII and IX.

2) Performance:

- **The ANN surrogate model** required around five hours of data collection. While the training and optimization of the model itself took less than ten minutes, each new system would require hyper-parameter tuning which could require a day of work.
- The **finite-difference gradient** method takes about two and a half hours to reach the same value as the ANN model. However, as the mesh gets refined and the number of parameters increases, it might prove to be extremely expensive computationally.
- The **semi-analytical gradient** method takes less than an hour and a half to minimize the function to the value attained by the ANN model. This is almost twice as fast as the finite-difference gradient method. The changes required to the system for additional parameters is minimal. One major issue is that each time a mistake is made, the entire system needs to be rerun, making prototyping very time consuming. However, once the gradient method is validated, no further work needs to be done.

While the ANN can scale much better with more parameters, it might be more time consuming to validate the robustness of the model. The analytical and FD gradients have a sound mathematical backing and hence need not be tested each time. They, however, cannot scale very well even when given unlimited resources.

3) Robustness: We believe it is necessary to provide a short note on robustness. A robust system is one which can be trusted to provide consistent results and whose predictions can be trusted even outside a certain domain of values. The robustness of the semi-analytical gradient system comes from our ability to mathematically derive the gradients and then validate them (using a finite difference check). Matlab's own code uses only finite differences for the gradients and hence there is little need to verify its robustness. For ANNs, the verification is a bit more complex. It must be pointed out that it **is** possible to build a robust ANN system. However, this requires thorough tuning of hyperparameters and numerous tests of its performance outside of its training set. It is difficult to determine, apriori, if this extra work offsets the advantages of rapid acquisition of data.

4) Verdict: Considering the advantages and disadvantages of all the three systems, we would recommend using the semi-analytical gradient method over the other two. It combines the robustness of the finite-difference gradient method, with the

speed and accuracy of using an analytical formulation for the complex parts of the gradients. The finite-difference approach might become prohibitively more expensive as the mesh is refined and more material parameters are added.
While the ANN is quite powerful, its results might need to be verified before they can be relied on completely. Perhaps, it can be used in cases where it is extremely difficult to obtain the semi-analytical gradients since it is definitely more scalable than using the finite-difference gradient method.

B. Some notes on using ANN

1) Field dependent model fidelity: Considering the relatively small dimensions of our problem, the ANN model was achieved with little discomfort in the way of hyperparameter tuning. One of the key takeaways is the importance of using the correct metrics to judge performance. Many use cases of ANN deal with classification problems where an accuracy of 90% might be sufficient. However, for a regression problem dealing with inputs from a deterministic simulation, this is extremely unsatisfactory. In addition, having a mean error that is small enough only tells half the story. In order to have a high-fidelity model, we also need to ensure that the maximum errors are within bounds too.

2) Non-trivial nature of the inverse problem: The question may arise as to why we are attempting to recreate the forward problem using ANN and not the inverse problem directly. This might make sense initially as the inverse problem would have 7 inputs and 6 outputs. However, there are two key points to note:

- **Nonlinearity** : The simulation is extremely non-linear. This results in two vastly different material parameter selections converging to two very co-linear output vectors.
- **Out of bounds predictions** : Assuming a trained inverse model, we would then easily input errors of close to 0 and expect to obtain the material parameters. However, these 0 errors would be very far from the training set. The fidelity of such an estimate is questionable.

Hence, we decided to use a forward model and then an optimization using gradients provided by the ANN algorithm. Nevertheless, we tried the inverse problem with various combinations of layers and nodes and obtained a **best** performance of 39 % mean error! This helped validate our choice.

VIII. Conclusion

Our project has validated the use of ANNs for surrogate modelling. With enough data, and tuning, a relatively robust model can be created. However, we also find that, in our case, with the possibility of obtaining semi-analytical gradients, the ANN may not be the best choice.
We would recommend the use of ANNs for simulations where there is no possibility of accessing the matrices used, or where the systems are too complex to be effectively linearized.
The semi-analytical gradient method proves to be robust and efficient for our problem. In order provide a more accurate estimate of the material properties of the medial layers, the following steps need to be taken:

- Use a very refined mesh that can capture all undulations.
- Use a material model that has a few more parameters. It can be seen that the existing model always has a trade-off between the low and high pressure errors. This might be because the Arruda-Boyce model may not sufficiently capture the non-linear behaviour of the collagen in the interlamellar layer.
- Include the adventitia in the simulations. This provides a more realistic scenario of the expansion of the media. Currently, there is no external resistance to the expansion.
- Include the layer specific thickness. Currently, we assume a uniform thickness for all layers. This is physically inaccurate.
- Treat each layer individually. Instead of having 2 materials, we would then have 5.

The material science community has started exploring ways in which computational power might be saved by using the hundreds of simulations, already run, to create robust surrogate models. This report adds to the growing volume of literature that makes such a case. It also, however, advocates the use of mathematically derived algorithms wherever possible, such as in our current problem.

References

[1] R. L. Taylor, "Feap-a finite element analysis program," 2000.
[2] A. Chamekh, H. B. H. Salah, and R. Hambli, "Inverse technique identification of material parameters using finite element and neural network computation," *The International Journal of Advanced Manufacturing Technology*, vol. 44, no. 1-2, p. 173, 2009.
[3] D. Novák and D. Lehkỳ, "Neural network based identification of material model parameters to capture experimental load-deflection curve," *Acta Polytechnica*, vol. 44, no. 5-6, 2004.
[4] O. Midtgarden, "Material Parameter Identification Using Artificial Neural Networks and Genetic Algorithm," 2009, master Thesis at NTNU.
[5] S. Deng, J. Tomioka, J. Debes, and Y. Fung, "New experiments on shear modulus of elasticity of arteries," *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 266, no. 1, pp. H1–H10, 1994.
[6] V. P. Le, J. K. Cheng, J. Kim, M. C. Staiculescu, S. W. Ficker, S. C. Sheth, S. A. Bhayani, R. P. Mecham, H. Yanagisawa, and J. E. Wagenseil, "Mechanical factors direct mouse aortic remodelling during early maturation," *Journal of The Royal Society Interface*, vol. 12, no. 104, p. 20141350, 2015.
[7] B. Trachet, J. Bols, J. Degroote, B. Verhegghe, N. Stergiopulos, J. Vierendeels, and P. Segers, "An animal-specific fsi model of the abdominal aorta in anesthetized mice," *Annals of biomedical engineering*, vol. 43, no. 6, pp. 1298–1309, 2015.
[8] L. Chrisman, "Latin Hypercube vs. Monte Carlo Sampling," http://www.lumina.com/blog/latin-hypercube-vs.-monte-carlo-sampling, 2016, [Online; accessed 01-December-2018].
[9] J. Heaton, "The Number of Hidden Layers," https://www.heatonresearch.com/2017/06/01/hidden-layers.html, 2016, [Online; accessed 01-December-2018].
[10] S. Ruder, "An overview of gradient descent optimization algorithms," http://ruder.io/optimizing-gradient-descent/index.html#adam, 2016, [Online; accessed 10-December-2018].
[11] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, 2017, [Online; accessed 07-December-2018].
[12] A. Danpinid, J. Luo, J. Vappou, P. Terdtoon, and E. E. Konofagou, "In vivo characterization of the aortic wall stress–strain relationship," *Ultrasonics*, vol. 50, no. 7, pp. 654–665, 2010.