Comparison between Multiple Random Forest Regressors and Auto-Encoder Predictions on Neurite Network Concentration

Yuxuan Yu
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA
yuxuany1@andrew.cmu.edu

Kuanren Qian
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA
kuanrenq@andrew.cmu.edu

December 13, 2019

Abstract

Neurons have remarkably complex geometry in their neurite networks. So far, the way which materials are transported in complex geometry and how the neuron function works are not yet clear. Answering this question is important to understand the physiology and disease of neurons. Here we have developed a platform based on machine learning for the simulation of material transport in neural networks. We model the transport process using reaction-diffusion-transport equations. We solved the Navier-Stokes equations to find the velocity field of material transport. Then we solved the transport equations. With our machine learning model, we predict concentration on neurite network in one basic models of network geometry: a Y-shape single neurite. In addition, the robustness of our machine learning is illustrated by using the simulation of material transport as a dataset. Our study shows that using machine learning model, the process of neurite-network material transport prediction can speed up significantly, up to $6,000$ time.

1 Introduction

Neurons exhibit remarkable complexity and diversity in their geometry, mainly due to the morphology of their hyper branched neuritic networks(Fig.1) [12, 6, 8]. Geometry is essential for the function of individual neurons and the formation of neuronal circuits [10, 15], but it presents a challenge for transport since the synthesis and degradation of materials in neurons takes place mainly in the body cellular [19, 18]. To survive and continue provide the function, neurons must constantly transport a variety of essential materials through their complex neural networks to meet their metabolic and functional needs [7, 11]. Currently, we cannot answer the question of how materials are transported within complex networks of neurites. Answering this question is essential to understand the physiology and pathophysiology of neurons, since they depend crucially on the material transport process [14, 17]. Answering this question is also essential to control the application transport process such as targeted delivery [2].

To answer the question of how materials are transported in neural networks, computer simulation is essential as it forms the basis for mathematical modeling and transport process analysis. However, since neurite networks consist of neurons, and the neurons' structures can be extremely complex and diverse. As a result, the material transport simulation in a neurite network is, therefore, computationally expensive and very time-consuming. The machine learning algorithm, on the contrarily, predicts solutions based on weights learned from existing data and does not need sequential equation calculation as a finite element method. Therefore, ML shows excellent potential in supplementing existing finite element simulation.

In this study, we modeled the transport process with a set of reaction-diffusion-transport equations [21, 9, 16]. However, the complex geometry of the network of neurites posed considerable challenges to solve these partial

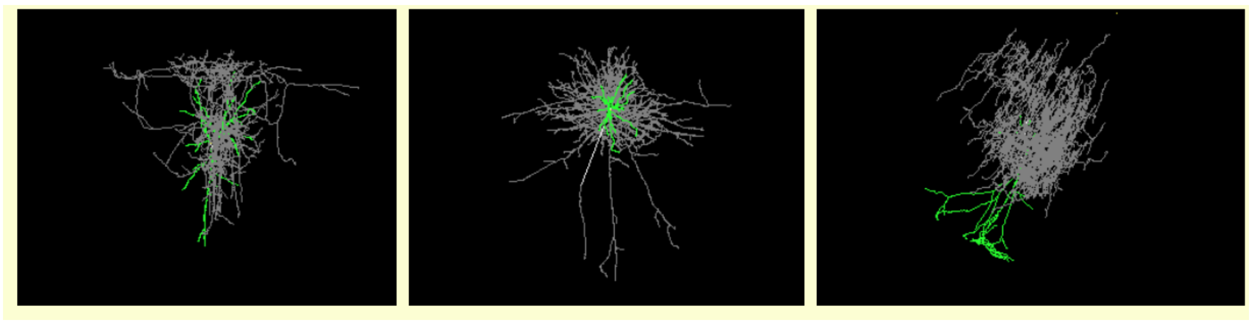

Figure 1: Examples of complex neuronal geometry.

differential equations (PDE). It is well known that conventional numerical techniques based on finite elements have significant challenges of precision and efficiency when solving PDEs within the long, thin and branched geometry of neural networks, such as computationally expensive and very time-consuming [3, 20, 4]. To overcome the challenges, we have implemented two machine learning algorithm based on multiple random forest regressors or auto-encoder to predict the concentration of transport of materials within the complex geometry of neural networks. To derive a physically realistic velocity field required for the solution of the generalized 3D transport model, we solved the Navier-Stokes equations using the multiscale residual approach based on variation [1]. Then we apply it to the machine learning platform. In summary, our study shows that using multiple random forest regressors, the process of neurite-network material transport simulation can speed up significantly, up to $6,000$ time. The rest of the paper is organized as follows. Section 2 introduces the construction of the geometry of the neurites. Section 3 presents the random forest regressor algorithm . In Section 4, auto-Encoder algorithm is tested to compare the result with random forest regressor algorithm. Section 5 concludes and proposes future work.

2 Data generation

To represent the complex and branched geometry of neurite networks, we create the geometry with the TREES toolbox [5]. Before generating the mesh SWC file, we first clean up the data to make sure there are only 13 vertices and the centerlines in the geometry forms Y shape. Then we break down the geometry of the structure of the neurite network into many segments. We call the connected mid-lines of the segments as a skeleten and the division of a neurite into two branches as a bifurcation (Figure 2(a)) . We take each segment as a pipe so that we can describe its position and shape with two endpoints and the associated diameters. We generate the hexahedral control mesh for each segment by sampling the quadrilateral mesh of the cross section along the skeleton. The control network is required to control the generation of splines for a more accurate and smooth representation of the neurite network geometry. Using the skeleten scan method, we create control networks for one simple model of the neurite network geometry: a single Y shape neurite (Figure 2(b)). The rest of the section presents each part in detail.

2.1 Tree Structure Design Generator

We use a parametric script to batch-generate different tree structure designs for FEA and data collection. The script takes several design parameters as constraints and generates individual tree structures by randomizing the node locations. The nodes are labeled and connected by edges (Figure 2(a)). The results are topologically consistent while being geometrically and behaviorally varied.

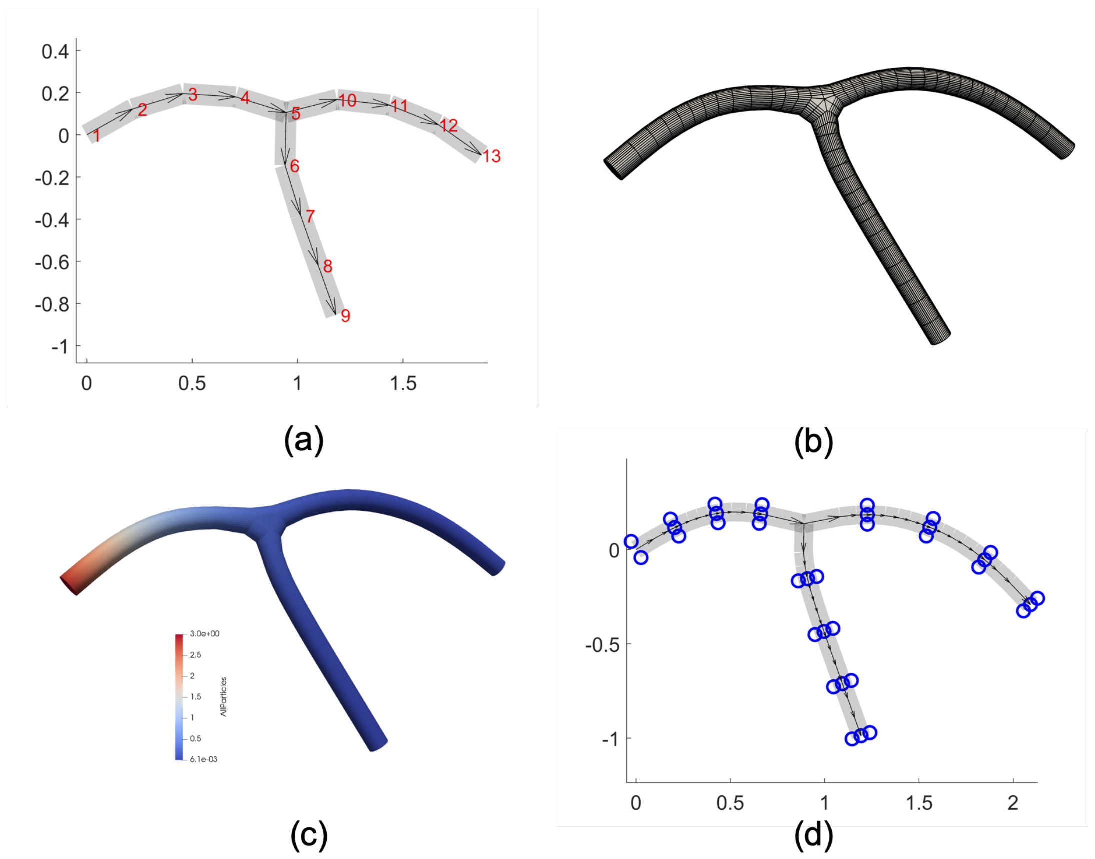

Figure 2: Data generation overview. (a) Neuronal tree structure; (b) Hexahedral control mesh; (b) FEA simulation result for data collection; (b) Feature representation.

2.2 FEA modeling and simulation

We assume each segment as a pipe so that we can describe its location and shape with two endpoints and their associated diameters. We generate the hexahedral control mesh for each segment by sweeping the quadrilateral mesh of the cross-section along with the skeleton (Figure 2(b)). Simulation data comes from Neuron material transport simulation (Figure 2(c)).

2.3 Feature extraction

Completed FEA results are used for feature extraction to collect data for training models. In each case, the result is described by the chosen 35 nodes (coordinate z=0), which have additional information (time t, concentration c) to describe the design (Figure 2(d)). The chosen 35 nodes are obtained by interpolating a set of points at fixed distances along the centerline and boundary in space.

3 Machine learning algorithms

In this study, we developed two machine learning algorithms and compared their accuracy and performance based on generated data. The first implemented machine learning algorithm uses multiple random forest regressors to learn information on each node from data, and it is very fast and accurate. The second implemented machine learning algorithm uses convolution neural network to construct a encoder to predict a one-channel matrix from three-channel input matrix. Though capable of handling data with complex features, encoder structure proved to perform poorly with simplified data in this study.

3.1 Random forest regressor

The first machine learning algorithm developed here consists of 35 individual random forest regressors. Each regressor is responsible for learning and predicting the concentration value at a specific node based on x position, y position, and time variable. After learning all simulation data, predictions of concentration value from 35 random forest regressors are combined to create a mesh grid consists of 35 nodes. Figure 3 below illustrates the structure of the developed machine learning algorithm. The input data for random forest regressor algorithm is a $m \times n$ matrix, where m is the total number of simulations and n is the sum of all features. Each row starts with 35 x position information from 35 nodes, then followed by 35 y position information, and end with 1 time variable. The output for each regressor is one concentration value.

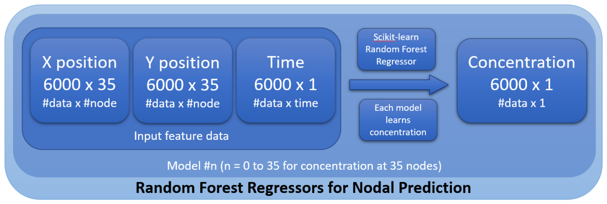
X position
6000 x 35
#data x #node
Y position
6000 x 35
#data x #node
Time
6000 x 1
#data x time
Input feature data
Scikit-learn Random Forest Regressor
Each model learns concentration
Concentration
6000 x 1
#data x 1
Model #n (n = 0 to 35 for concentration at 35 nodes)
Random Forest Regressors for Nodal Prediction

Figure 3: Multiple random forest regressor structure. This method implements 35 random forest regressors and each regressor learns concentration value on one node.

3.2 Auto-Encoder CNN

The second machine learning algorithm developed uses the auto-encoder theory. Similar to the proposed random forest regressor algorithm, the auto-encoder algorithm uses the same data, but layering x position, y position, and time into a 3-layer matrix as input. The x and y matrix layers are constructed with position information from mesh data, the time matrix layer is a uniform value matrix filled with same time variable from specific simulation data. hen, with Pytorch[13] package, the following convolution neural network structure is constructed, shown in figure 4. The algorithm takes 3 x 6 x 6 matrix as an input and conducts calculations shown in the figure to obtain a 6 x 6 matrix, representing learned concentration. Then, the computed concentration matrix is compared with simulated concentration value using mean square error, and the loss is sent back for iterative optimization.

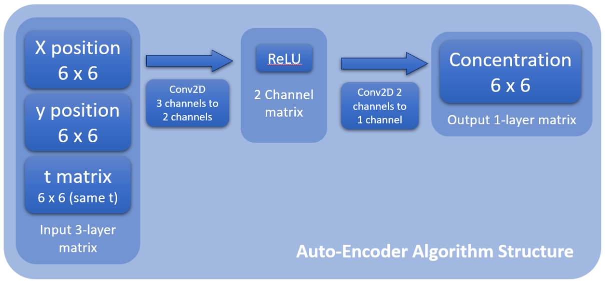

Figure 4: Auto-encoder method structure.

4 Results and performance

The performance of random forest regressor algorithm and auto-encoder algorithm are evaluated using mean square error. For random forest regressor, 6000 data sets are split 70/30 as training data and test data. Predicted concentration values of 35 random forest regressor models are compared with corresponding concentration values from test data using mean square error method. Each model produces one mean square error and the maximum mean square error is found to be 5.177e-05.

Figure 5 and 6 show the predictions of concentration values made by the random forest regressor algorithm on two neuron models with 35 nodes. Model 0 is the same model shown in figure 2, but with simplified nodes.

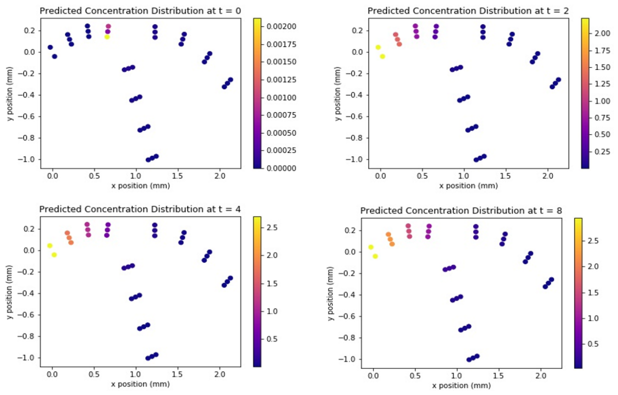

Figure 5: Random forest regressors' prediction on geometry 0.

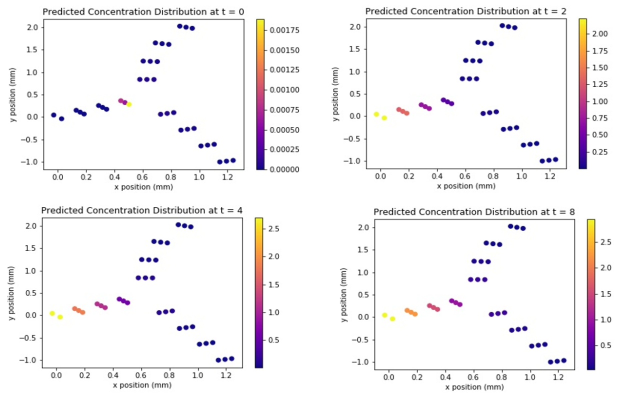

Figure 6: Random forest regressors' prediction on geometry 1.

The Auto-encoder algorithm, contrary to random forest regressor, is difficult to reduce error and requires extensive computational resource. Even after 5000 epochs conducted, the means square error is still above $9e-3$. One of the causes for this implementation problem is the over-simplification of simulation data. The neuron simulation data used in this study is heavily simplified to reduce computational cost. However, the simplification process losses too much hidden information from the data, and therefore causing the algorithm to perform poorly.

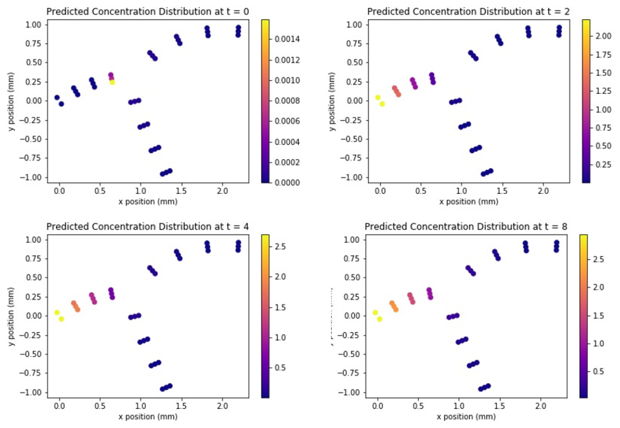

Figure 7: Random forest regressors' prediction on geometry 35.

Another cause for this problem is the unoptimized structure of convolution neural network. The current structure takes a 3-channel matrix as input and predicts a 1-channel matrix. This implemented structure does not take nodal connection into consideration and only one ReLU operation is conducted in the process. As a result, the algorithm is performing poorly with generated data.

5 Conclusion

Based on analysis conducted on the developed machine learning algorithms that use random forest regressors and convolution neural networks, machine learning algorithm is vastly more efficient and faster if the simulation target is specific and the data is organized.

With a straight-forward random forest regressor algorithm, the machine learning model is x5647 faster. To simulate One model, finite element analysis took around 864s to finish, but random forest regressor finished it in 0.00015s after training. The difference in performance showcases that machine learning has great potential in boosting fundamental research.

However, the Auto-encoder machine learning algorithm has a underwhelming performance in the test, with a mean square error of $9e-3$ after over 5000 epochs. The concluded cause is the combination of over-simplified input data and unoptimized convolution neural network structure. Due to the time constraint for this study, the Auto-encoder algorithm did not have sufficient time to be optimized and generating data again will take a long time.

In the future, given enough time and resource, we will generate more complex data with more features and construct an optimized Auto-encoder structure that takes more features of the simulation into account. Also, a more powerful application of this machine learning algorithm on neurons is to extend geometry model to complex neurite network structures, instead of predicting a single neuron.

Acknowledgment

We want to thank Angran Li, a Ph.D. student from the Department of mechanical engineering, for his efforts in developing the simulation algorithm for neurite-network material transport. We would also like to thank Changlin Jiang, a machine-learning teaching assistant, for his constructive inputs on algorithm development. We are also exceedingly grateful for professor Amir Barati Farimani for his instruction on machine-learning algorithms.

Contribution

Yuxuan Yu worked on generating randomized geometry information, data processing, and random forest regressor. Kuanren Qian worked on the implementation of random forest regressor and auto-encoder. Angran Li worked on generating simulation data.

References

[1] Y. Bazilevs, V. M. Calo, J. A. Cottrell, H. Tjr, A. Reali, and G. Scovazzi. Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 197(1):173–201, 2007. 2

[2] R. B. Bird, W. E. Stewart, and L. E. T. phenomena. *A Wiley-Interscience Publication*. Wiley, 2007. 1

[3] S. A. Brown, I. I. Moraru, J. C. Schaff, and L. L. V. Neuron. a strategy for merged biochemical and electrophysiological modeling. *Journal of Computational Neuroscience*, 31(2):385–400, 2011. 2

[4] J. A. Cottrell, H. Tjr, and B. Y. I. Analysis. *Toward Integration of CAD and FEA*. Wiley, 2009. 2

[5] H. Cuntz, F. Forstner, A. Borst, and M. Husser. One rule to grow them all: a general theory of neuronal branching and its practical application. *PLOS Computational Biology*, 6(8):1–14, 2010. 2

[6] X. Dong, K. Shen, and B. HE. Intrinsic and extrinsic mechanisms of dendritic morphogenesis. *Annual Review of Physiology*, 77:271–300, 2015. 1

[7] N. Hirokawa and R. Takemura. Molecular motors and mechanisms of directional transport in neurons. *Nature Reviews Neuroscience*, 6(3):201–214, 2005. 1

[8] K. Kalil and D. E. B. management. mechanisms of axon branching in the developing vertebrate cns. *Nature Reviews Neuroscience*, 15(1):7–18, 2014. 1

[9] M. T. Lazarewicz, S. Boer-Iwema, and A. GA. Practical aspects in anatomically accurate simulations of neuronal electrophysiology. In *Computational Neuroanatomy*, pages 127–148. 2002. 1

[10] M. London and H. M. D. computation. Annu. *Rev. Neurosci*, 28:503–532, 2005. 1

[11] C. I. Maeder, K. Shen, and H. CC. Axon and dendritic trafficking. *Current Opinion in Neurobiology*, 27:165–170, 2014. 1

[12] R. Parekh and A. GA. Neuronal morphology goes digital: a research hub for cellular and system neuroscience. *Neuron*, 77(6):1017–1038, 2013. 1

[13] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. 4

[14] B. Ruozi, D. Belletti, L. Bondioli, A. De Vita, F. Forni, M. Vandelli, and G. Tosi. Neurotrophic factors and neurodegenerative diseases: a delivery issue. *Int Rev Neurobiol*, 102:207–247, 2012. 1

[15] I. Segev and M. London. Untangling dendrites with quantitative models. *Science*, 290(5492):744–750, 2000. 1

[16] B. M. Slepchenko and L. LM. *Chapter One - Use of Virtual Cell in Studies of Cellular Dynamics*. Academic Press, 1–56, 2010. 1

[17] D. Smith and R. Simmons. Models of motor-assisted transport of intracellular particles. *Biophysical Journal*, 80(1):45–68, 2001. 1

[18] O. Steward and S. EM. Compartmentalized synthesis and degradation of proteins in neurons. *Neuron*, 40(2):347–359, 2003. 1

[19] S. A. Swanger and B. GJ. Dendritic protein synthesis in the normal and diseased brain. *Neuroscience*, 232:106–127, 2013. 1

[20] H. Tjr, J. A. Cottrell, and B. Y. I. analysis: Cad. finite elements, nurbs, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39):4135–4195, 2005. 2

[21] G. A. Truskey, F. Yuan, and K. DF. Transport Phenomena in Biological Systems, 2004. 1

Supplementary material

Tree structure design generator code (Matlab)

```
clc
clear
close all
A1=clean_tree (sample2_tree, 20);
A1=delete_tree (A1, [8]);

X_add(1)=(A1.X(1)+A1.X(2))/2;
X_add(2)=(A1.Y(1)+A1.Y(2))/2;
X_add(3)=(A1.Z(1)+A1.Z(2))/2;
X_add_2(1)=(A1.X(3)+A1.X(2))/2;
X_add_2(2)=(A1.Y(3)+A1.Y(2))/2;
X_add_2(3)=(A1.Z(3)+A1.Z(2))/2;
% A1=insert_tree (A1,[1 2 X_add 1 1]);
% A1=insert_tree (A1,[1 2 X_add_2 1 2]);
A1=root_tree (A1);
A1=root_tree (A1);

A1.X(3)=A1.X(4);
A1.Y(3)=A1.Y(4);
A1.Z(3)=A1.Z(4);
A1.X(2)=X_add(1);
A1.Y(2)=X_add(2);
A1.Z(2)=X_add(3);
A1.X(4)=X_add_2(1);
A1.Y(4)=X_add_2(2);
A1.Z(4)=X_add_2(3);
A1.Z(:)=0;
A1=scale_tree (A1, [1/25 1/20 1]);
A1=sort_tree (A1,'s');
A1.R(:)=2;
A1.D(:)=0.1;
% repair_tree (A1);
xplore_tree(A1);
% swc_tree (A1, 'test.swc');
%
```

```
      str=sprintf('%04d',file_ID);
      dirname=[foldername,'/job',str,'.swc'];
      swc_tree (A1_new, dirname);
      file_ID=file_ID+1;
   end
end
```

Tree structure design generator code - remove bad branch (Matlab)

```matlab
clc
clear
close all
start_trees
file_ID=1;
foldername=sprintf('skeleton_swc_checked');

for loopi=1:1369
   str_original=sprintf('job%04d.swc',loopi);
   A1=load_tree(str_original);
   A1=sort_tree (A1,'s');

   % xplore_tree(A1);
   branch_1=[];
   branch_1=[branch_1,A1.X([5:9])];
   branch_1=[branch_1,A1.Y([5:9])];
   branch_1=[branch_1,A1.Z([5:9])];
   branch_2=[];
   branch_2=[branch_2,A1.X([5,10:13])];
   branch_2=[branch_2,A1.Y([5,10:13])];
   branch_2=[branch_2,A1.Z([5,10:13])];

   if sum(sqrt(sum((branch_1-branch_2).^2,2))>0.2)==4
       A1.R(:)=2;
         str=sprintf('%04d',file_ID);
          dirname=[foldername,'/job_checked_',str,'.swc'];
          swc_tree (A1, dirname);
          file_ID=file_ID+1;
   end
end
```

Feature extraction (Matlab)

```
clc
clear
close all
Matrix_total=[];
for loopi=1:100
   id=sprintf('%04d/skeleton_smooth.swc',loopi);
   if isfile(id)
      M = dlmread(id,'',6,0);
      branch_1=M([1,5:M(2,7),2],3:4);
      branch_2=M([2,M(2,7)+1:M(3,7),3],3:4);
      branch_3=M([2,M(3,7)+1:M(4,7),4],3:4);

      [pt_1,dpt_1] = interparc(5,branch_1(:,1),branch_1(:,2),'spline');
      [pt_2,dpt_2] = interparc(5,branch_2(:,1),branch_2(:,2),'spline');
      [pt_3,dpt_3] = interparc(5,branch_3(:,1),branch_3(:,2),'spline');

      dpt_1_normal=[dpt_1(:,2),-dpt_1(:,1)]./repmat(vecnorm(dpt_1,2,2),[1,2]);
      pt_1_up=pt_1+0.05*dpt_1_normal;
      pt_1_down=pt_1-0.05*dpt_1_normal;

     dpt_2_normal=[dpt_2(:,2),-dpt_2(:,1)]./repmat(vecnorm(dpt_2,2,2),[1,2]);
      pt_2_up=pt_2+0.05*dpt_2_normal;
      pt_2_down=pt_2-0.05*dpt_2_normal;


     dpt_3_normal=[dpt_3(:,2),-dpt_3(:,1)]./repmat(vecnorm(dpt_3,2,2),[1,2]);
      pt_3_up=pt_3+0.05*dpt_3_normal;
      pt_3_down=pt_3-0.05*dpt_3_normal;

    point_observed=[];

      point_observed=[point_observed;[pt_1(:,1),pt_1(:,2) ]       ] ;
      point_observed=[point_observed;[pt_3(2:end,1),pt_3(2:end,2) ]     ] ;
      point_observed=[point_observed;[pt_2(2:end,1),pt_2(2:end,2) ]     ] ;
      point_observed=[point_observed;[pt_1_up(1:end-1,1),pt_1_up(1:end-1,2) ] ] ;
      point_observed=[point_observed;[pt_1_down(1:end-1,1),pt_1_down(1:end-1,2) ] ] ;
      point_observed=[point_observed;[pt_2_up(2:end,1),pt_2_up(2:end,2) ] ] ;
      point_observed=[point_observed;[pt_2_down(2:end,1),pt_2_down(2:end,2) ] ] ;
      point_observed=[point_observed;[pt_3_up(2:end,1),pt_3_up(2:end,2) ] ] ;
    %       plot(pt_2(:,1),pt_2(:,2),'b-o','MarkerSize', 20)
    %       hold on
    %       plot(pt_1_up(1:end-1,1),pt_1_up(1:end-1,2),'b-o','MarkerSize', 20)
    %       hold on
    %       plot(pt_1_down(1:end-1,1),pt_1_down(1:end-1,2),'b-o','MarkerSize', 20)
    %     hold on
    %       plot(pt_2_up(2:end,1),pt_2_up(2:end,2),'b-o','MarkerSize', 20)
    %       hold on
    %       plot(pt_2_down(2:end,1),pt_2_down(2:end,2),'b-o','MarkerSize', 20)
    %     hold on
    %       plot(pt_3_up(2:end,1),pt_3_up(2:end,2),'b-o','MarkerSize', 20)
    %       hold on
    %       plot(pt_3_down(2:end,1),pt_3_down(2:end,2),'b-o','MarkerSize', 20)
  end
end
```

Random Forest Regressors Algorithm (Python)

```
import numpy as np

print('****************************************\n')
print('Reading data ...\n')
data = np.genfromtxt('./Neurite_Network_Simulation.csv', delimiter=',')

x = data[:,0:37].copy()
y = data[:,37:74].copy()
c = data[:,74:111].copy()
t = data[:,111].reshape(-1,1)
d = data[:,112].reshape(-1,1)
k = data[:,113].reshape(-1,1)

print('****************************************\n')
print('Normalizing data ...\n')
# Data Normalization
x = np.delete(x,0,1)
y = np.delete(y,0,1)
c = np.delete(c,0,1)
x = np.delete(x,3,1)
y = np.delete(y,3,1)
c = np.delete(c,3,1)
x_norm = x.copy()
y_norm = y.copy()
c_norm = c.copy()
c_norm_ratio = np.zeros((1,x_norm.shape[1]))

for j in range(x_norm.shape[1]):
   x_col_max = np.max(x_norm[:,j])
   x_col_min = np.min(x_norm[:,j])
   y_col_max = np.max(y_norm[:,j])
   y_col_min = np.min(y_norm[:,j])
   c_col_max = np.max(c_norm[:,j])
   c_col_min = np.min(c_norm[:,j])

   for i in range(x_norm.shape[0]):
      x_norm[i,j] = (x_norm[i,j] - x_col_min)/(x_col_max - x_col_min)
      y_norm[i,j] = (y_norm[i,j] - y_col_min)/(y_col_max - y_col_min)
```

```
print('\n****************************************\n')
print('Completed ! ----------------------------\n')

##############################################################################
# To use this model for prediction, only run the entire code once, and then
# re-run the following code. The following code prompts user to enter geometry
# index, a int value between 0 to 67, and time variable, a positive variable.
# A plt scatter plot will be generated to show all the nodes and the color on
# these nodes represent the predicted concentration

print('****************************************\n')
print('Prediction for user input ----------------------------\n')
import matplotlib.pyplot as plt

print('Please enter geometry index (int val, 0~67) :\n')
# Only these two need to be changed at inputs
index_geometry = int(input())
print('Please enter time (positive val) :\n')
t = float(input())/t_normalize_ratio

x_plot = x[index_geometry*67,:].copy().reshape(-1,1).T
y_plot = y[index_geometry*67,:].copy().reshape(-1,1).T
c_plot = np.zeros((x_plot.shape[1],1)).T

xx = np.hstack((x_plot,y_plot))
xx = np.append(xx,t).reshape(-1,1).T

for i in range(len(a)):
   c_plot[0,i] = a[i].predict(xx)*c_norm_ratio[0,i]

cm = plt.cm.get_cmap('plasma')
fig0 = plt.figure(0)
plot = plt.scatter(x_plot,y_plot,c=c_plot,cmap=cm)
plt.colorbar(plot)
plt.xlabel('x position (mm)')
plt.ylabel('y position (mm)')
plt.show()
```

Auto-encoder Algorithm (Python)

```python
import numpy as np

data = np.genfromtxt('./Neurite_Network_Simulation.csv', delimiter=',')

data_size = data.shape[0]
x = data[:,0:37].copy()
y = data[:,37:74].copy()
c = data[:,74:111].copy()
t = data[:,111].reshape(-1,1)
d = data[:,112].reshape(-1,1)
k = data[:,113].reshape(-1,1)

x = np.delete(x,4,1)
y = np.delete(y,4,1)
c = np.delete(c,4,1)

x_r = np.zeros((6,6,x.shape[0]))
y_r = np.zeros((6,6,y.shape[0]))
c_r = np.zeros((6,6,c.shape[0]))
t_r = np.zeros((6,6,x.shape[0]))
d_r = np.zeros((6,6,y.shape[0]))
k_r = np.zeros((6,6,c.shape[0]))

for i in range(x.shape[0]):
   x_r[:,:,i] = x[i,:].reshape(6,6)
   y_r[:,:,i] = y[i,:].reshape(6,6)
   c_r[:,:,i] = c[i,:].reshape(6,6)
   for j in range(6):
      for l in range(6):
         t_r[j,l,i] = t[i,0]
         d_r[j,l,i] = d[i,0]
         k_r[j,l,i] = k[i,0]

c_z = np.zeros((6,6))

print('************************************')
print('Finished Data Generation.')

import torch
```

```
for k in range(epoch):
   LOSS = 0
   for n in range(train_size):

      optimizer.zero_grad()

      output = model(x_Train[:,:,:,:,n])

      loss = criterion(output, y_Train[:,:,:,n])

      model.zero_grad()

      loss.backward()
      optimizer.step()

      LOSS += loss.item()

   LOSS = LOSS/train_size
   print(k,round(LOSS,6),device)


#################################################################################
# To use this model for prediction, only run the entire code once, and then
# re-run the following code. The following code prompts user to enter geometry
# index, a int value between 0 to 67, and time variable, a positive variable.
# A plt scatter plot will be generated to show all the nodes and the color on
# these nodes represent the predicted concentration

print('*****************************************\n')
print('Prediction for user input -----------------------------\n')
import matplotlib.pyplot as plt

print('Please enter geometry index (int val, 0~67) :\n')
# Only these two need to be changed at inputs
index_geometry = int(input())
print('Please enter time (positive val) :\n')
t = float(input())

x_plot = x_Train[:,0,:,:,index_geometry*67].numpy().copy()
y_plot = x_Train[:,1,:,:,index_geometry*67].numpy().copy()
t_plot = np.zeros((x_plot.shape[1],x_plot.shape[1]))
c_plot = np.zeros((x_plot.shape[1],x_plot.shape[1]))
```