Deep learning the assembly parameters of growing cytoskeletal networks

Kros, Maury F.

Award date:
2019

Technische Universiteit Eindhoven

Deep learning the assembly parameters of growing cytoskeletal networks

Maury Kros
June 2019

Supervisor:
Kees Storm

Abstract

The eukaryotic cell, the building block for humans, is a widely studied system both experimentally and theoretically. In this report, we pave the way for the use of machine learning techniques to investigate cytoskeletal development within the eukaryotic cell. The underlying motivation is that if we can show that machine learning techniques can be used to describe the spatial evolution of simulated data, then perhaps this can be done for experimental data as well. Using the open source simulation engine cytosim, a naive model is simulated of fibers that can nucleate and crosslink at the cell surface. Increase in fiber length in the cytosim model is achieved by nucleation at the plus-ends of polarized fibers and the formation of new fibers that attach to existing fibers at the point of attachment. For this cytosim model a theoretical model is proposed which is then fitted to the network. The model can successfully describe the evolution of the total fiber length as function of time as well as the number of active plus-ends as function of time. The DeepMoD neural network, as of yet fails, to capture the cytosim model well. This has brought to light that DeepMoD, though it can deal with partial differential equations well, struggles to correctly predict underlying ordinary differential equations, and points to promising directions for additional research to improve the performance of DeepMoD.

Contents

1 Introduction

Over the last decade data-driven discovery methods and in particular neural networks have experienced a great boom in research. This is largely due to technological advances such as improved data storage and computational power paired with a decrease in its associated costs. While neural networks in itself are hardly a novel concept, its application in physics is fairly new. It has already proven to be a potent tool in model fitting and solving numerical equations in various domains of physics such as temperature modelling [5], image processing in fluorescence microscopy [12] and atomistic material modelling [10].

A neural network is *trained* to find the correct parameters for the physical problem of which it uses the trainingdata. A neural network that finds parameters of physical equations is called a *physics informed neural network*, or PINN for short. Trainingdata can be gathered in multiple ways, both experimentally and by simulation. Here the training data is retrieved from simulations of an eukaryotic cell using an open source simulation engine called cytosim.

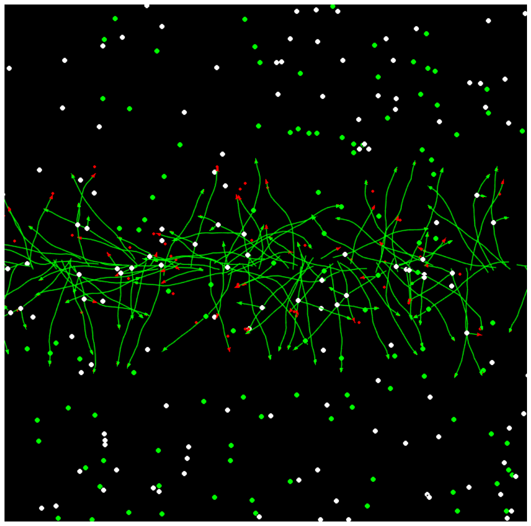


Figure 1: A screenshot taken from a cytosim simulation. Growth of filaments are simulated that nucleate from the surface. The simulation is axis symmetric.

The aim of this research is *proof of concept*. That is, known machine learning techniques are applied to a spatiotemporal dataset generated using cytosim. Then the parameters of the equations that describe the biophysical problem found by machine learning should be consistent with those parameters that the model in cytosim uses as input. Here we simulate the growth of the cytoskeleton of a cell (see figure 1), it is a simple model with only few parameters. If it can be shown that a PINN can be used to find the parameters of the system for this simple system then hopefully this paves the way for more complicated and realistic models.

2 Theory

In this section we explore all relevant theory. First, we briefly sketch the situation from a biological point of view. It might not be the focus of this report, however, for almost every problem a computer solves there is a real world problem that is central to it. More importantly, and in slightly more detail, the physics on which the software is based that generates data is reviewed as well as the workings of the applied machine learning techniques.

2.1 Biological context

The origin of this report lies within the fundamental structure of life for humans, the eurkaryotic cell. Cell mechanics is a highly investigated and broad subject with research done both in and outside of the cell. Research is done on both macroscopic and microscopic cellular processes alike, ranging from maintenance of the shape, cell motility, adhesion and the transduction of mechanical signals into biochemical signals leading to biological responses [8]. Here the focus is on the cytoskeleton, that is, the network of dynamic and interlinked protein filaments encapsulated between the cell membranes and nucleus within a cell. The cytoskeleton is comprised of three components: microtubili, intermediate filaments and actin filaments. Each with distinct mechanical characteristics. Microtubuli are polymerized filaments of α- and β- tubulin in a helical arrangement, forming a hollow cylinder. Intermediate filaments constitute a superfamily of proteins of over 50 different members.

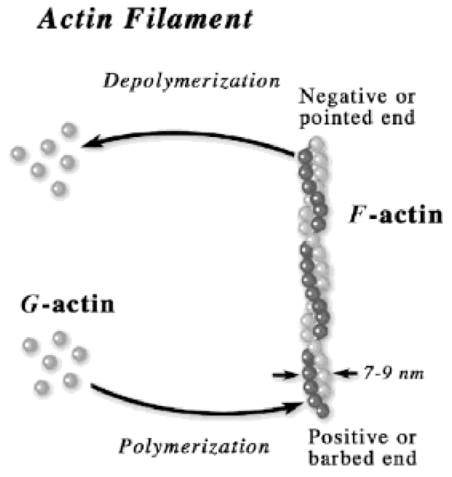


Figure 2: Schematic overview of an actin filament. An actin filament is polarized fiber built up out of monomers. From here the term fiber is used when referred to actin filaments.

In this report the focus is on actin filaments of which a schematic overview is displayed in figure 2 above. Actin filaments make up 1 to 10 percent of all the protein in most cells are the percentage is even higher in muscle cells.

2.2 Cytosim

For the generation of data *cytosim* is used. Cytosim is an open source simulation engine that is based on Brownian dynamics [9]. It offers users a toolkit to work with various cellular structures in a virtual cell without having to learn the underlying code which is developed in C++. Cytosim has already seen various applications such as investigation of contractile disordered cytoskeletal networks [1], actin filament organization [6] and the effect of actin-myosin interaction on the cell cortex [14].

2.2.1 Equation of motion

The physics behind cytosim simulations have already been described in much detail [9], here a brief review is given of the relevant equations. In simulation every object is described by a collection of points. Fibers of length L consist of $p+1$ equidistant points distance L/p apart from each other. Growth of fibers can introduce new points which will be added under the minimization of $|\rho_{seg} - \frac{L}{\rho_{seg}}|$ where ρ_{seg} is the segmentation length specified in the model. A vector $\boldsymbol{x}$ of size Nd collects the points for a model with size N in dimension d. The equation of motion is the Langevin equation given by:

$$d\boldsymbol{x} = \mu F(\boldsymbol{x}, t)dt + dB(t). \tag{2.1}$$

Mobility coefficients are contained within the matrix μ. Forces acting upon the points at time t are contained in matrix $F(\boldsymbol{x}, t)$ which is size Nd. And lastly $dB(t)$, also of size Nd, summarizes the random molecular collisions leading to Brownian motions. This term actually introduces randomness to simulations since values are different for every simulation. For $dB(t)$ cytosim uses a pseudo random number generator (PRNG) for which the random seed is calculated from clock time at initialization.

2.2.2 Cytosim model

During simulations a simple cytosim model is used. Fibers start uniformly distributed at the x-axis with negligible length at a random orientation. *Crosslinkers* can attach to a fiber and initiate growth of a new fiber at the point of attachment under an angle of 70°. So growth of a fiber only occurs at the plus-ends or at attachment points. *Cappers* can bind to plus-ends of fibers to halt growth, if they detach they can bind again to plus-ends. In figure 3 it it can be seen how this all actually translates to a simulation.

2.2.3 Simulation

Central to any simulation are the timestep *ts*, the number of simulations *sim* and the number of frames *frames*. The *ts* determine the timestep that is used in equation 2.1 for calculating the motion of the points within a simulation. The total running time of a simulation is

$$t_{tot} = sim \cdot ts. \tag{2.2}$$

The number of frames is important for the actual output of the model. Suppose a simulation is ran with $ts = 0.001s$, $sim = 1000$ and $frames = 10$. This yield a simulation with $t_{tot} = 1s$ and the output is given for every frame i at time $t_i = \frac{t_{tot}}{frames} \cdot (i-1)$ for $i = 1, 2 \ldots (frames - 1)$. So

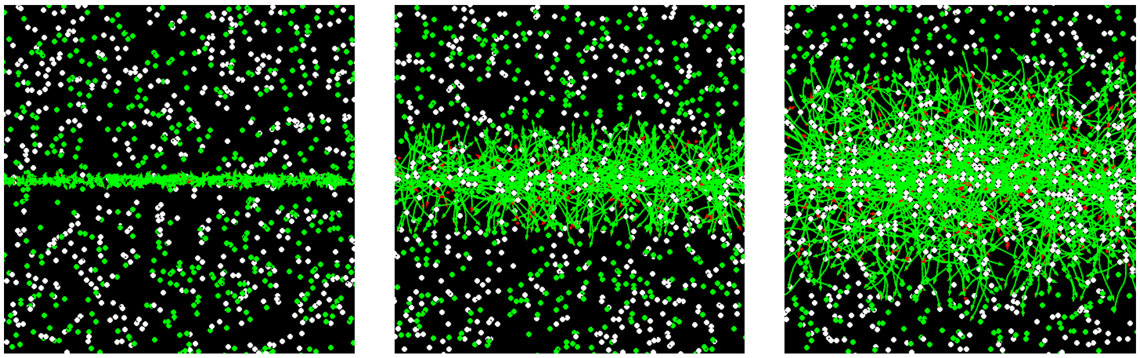


Figure 3: Above 3 screenshots displayed of a simulation in cytosim. Fibers are displayed as green strings while crosslinkers are both green and white dots. This is because in cytosim a crosslinker is formed by constructing a particle that consists of two other particles, a nucleator in white and a activator in green. The plus-ends of fibers are red or green. A red plus-end has stopped growing and is effectivly capped, a green plus-end is an active, growing plus end. Free red dots can also be seen, these are cappers.

output can be retrieved at time $t_1 = 0s, t_2 = 0.1s,\ \ldots\ , t_{10} = 0.9s$.

Cytosim has its own binaries for retrieving the actual data using the command *./report*. Particularly interesting is the output of the fibers since this is the basis for creating a spatio-temporal dataset. For full options on the report command see cytosim the documentation. Here it is used to extract a *.txt* file of the fiber points and their positions per frame.

In this report the word model is often used. Now, briefly the nomenclature is discussed so that it is clear when a reference is made to the cytosim model or any other model. When a model is mentioned, a physical model is meant (e.g. equation **??**). Cytosim model refers to models in cytosim evidently. All cytosim model parameters can be found in the script in appendix 6.1 yet only few are ever varied in simulations. Hence a simulation can be denoted

$$\Omega(ts, sim, frames, cl, fib), \tag{2.3}$$

where ts is the timestep, sim the number of simulation steps, $frames$ the number of frames, cl the number of crosslinkers and fib the number of initial fibers. This allows for a brief notation. Suppose a simulation is ran with $ts = 0.001$, $sim = 1000$, $frames = 10$, $cl = 250$ and $fib = 100$ this is denoted $\Omega(0.001, 1000, 10, 250, 100)$.

2.3 Theoretical description model

For the theoretical description of the outcome of a cytosim simulation a model is suggested. The suggested model is a system of two ordinary differential equations based on conservation of particles, equations 2.4 and 2.5. Here the change of total fiber length dN/dt is the product of the polymerization rate k_p and number of active ends N^+:

$$\frac{dN}{dt} = k_p N^+. \tag{2.4}$$

The change of active plus ends dN^+/dt is equal to the branching rate k_b at which crosslinkers attach to fibers to initiate growth of a new fiber and the total fiber length N, minus the product of the capping rate k_c and the number of active plus ends:

$$\frac{dN^+}{dt} = k_b N - k_c N^+. \tag{2.5}$$

An additional benefit of this simple model is that the exact solutions can be found. If it turns out that there is no similarity between the cytosim results and the exact solutions one can hardly expect to obtain valid results from machine learning techniques. More on this in section 4.2.

2.4 Neural Networks

The concepts on which neural networks are based were pioneered in the 1940's [7, 4] and originate from the study of neural connections in the brain, hence the name. How they work has been well documented and can be found nowadays in various textbooks [3, 15, 13]. Here, the free online book written by Michael Nielsen is used to describe how neural network work (see: neuralnetworksanddeeplearning.com). Also, the liberty was taken to adapt the notation used in this book.

2.4.1 Neural Network Mechanisms

Neural networks are formed by multiple layers of so called *neurons* that transform input into output (see figure 4). The output a of a neuron is determined weighted input z which is related to the inputs $x_1, ..., x_n = X$, a weighing matrix w and bias b according to

$$z = wX - b. \tag{2.6}$$

z is then mapped to value between 0 and 1 using an *activation function* σ

$$a = \sigma(z) = \sigma(wX - b). \tag{2.7}$$

Various functions can be used, and have been used to map the weighted input z such as the sigmoid-, softplus- and tanh function. It is the actvation function that introduces non linearity to the system.

What a neural network then tries to find is the correct weights and biases in each layer such that the output of the neural network matches the desired output. In order to evaluate the difference between the predicted and the desired output we define the cost function

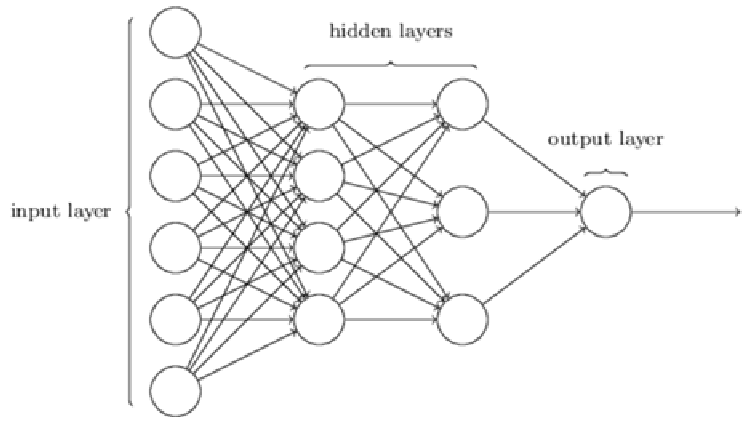


Figure 4: Here a generic neural network is displayed. A neural network in essence is a collection of many connected neurons. The first layer is called the input layer and the last layer is the output layer. Any layer in between is part of the hidden layers.

$$\mathcal{L} = \frac{1}{2n} \sum_i |y_i - a_i^L|^2. \tag{2.8}$$

Which is a function of the number of samples n, the desired output of sample i, y_i. a_i^L is the activation which using expression 2.7 we can write for a layer l:

$$a^l = \sigma(w^l a^{l-1} + b^l). \tag{2.9}$$

Equation 2.8 is the mean squared error which has the benefit that it is smooth so that differentiation is easier. However, a cost function need not to be equal to the mean squared error and later on it turns out that by this cost function the physics can be introduced into the neural network by feeding it the relevant physical equations directly into the cost function. More on this in section 2.4.3.

Ultimately, minimizing the cost function is that which yields the results for a neural network. A common technique to find the minimum of the cost function is *gradient descent*. It is not uncommon to have weights and biases in the millions or even billions therefor finding the minimum is not done in the classical way. That is, it is not done by setting all derivatives equal to zero and finding the appropriate values since this is too complicated.

Suppose we have a function $f(x)$ that is to be minimized with respect to x. Then, given an initial position x_n, gradient techniques find the minimum by following the direction of the steepest descent. Given that a function increases maximally in a point in the direction of it's gradient at that point, x_n is iteratively updated:

$$x_{n+1} = x_n - \eta \nabla f(x_n). \tag{2.10}$$

Here η is the stepsize. Gradient descent techniques converge to a minimum. If the function is convex, such that it has no local minima, there is only one outcome possible. Still, the questions remains how to find the derivatives of the cost function. This is discussed in the upcoming section.

2.4.2 Minimizing the cost function

Here the *back propagation* algorithm is introduced. The aim here is to understand how a cost function C is changed by the weights and biases of the network. Ultimately it is about finding the partial derivatives $\partial C/\partial w_{jk}^l$ and $\partial C/b_j^l$. In these derivatives w_{jk}^l is the weight of the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer and ${b_j^l}^{th}$ the j^{th} bias in the l^{th} layer. The error δ_j^l of neuron j in layer l is defined as

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}. \tag{2.11}$$

Here δ^l is the vector of errors associated with layer l. Back propagation provides a way to compute δ^l for every layer and relating them to parameters of interest $\partial C/\partial w_{jk}^l$ and $\partial C/b_j^l$. Before we see how the back propagation algorithm works, the equations that it uses are evaluated. It comes down to 4 equations. First the equation for the error in the output layer is

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L). \tag{2.12}$$

The term $\partial C/\partial a_j^L$ is a measure of the change in cost function as function of the j^{th} output activation. The term $\sigma'(z_j^L)$ is a measure for the change in activation function σ at z_j^L. Notice that equation 2.12 is a rewritten form of equation 2.11 using the chain rule. For the use in back propagation equation 2.12 is written in matrix form:

$$\delta^L = \nabla_a C \odot \delta'(z^L). \tag{2.13}$$

Secondly, an equation is needed that relates the error δ^l to the error in the next layer δ^{l+1}:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^L). \tag{2.14}$$

Here $(w^{l+1})^T$ is the transpose of the weight matrix of layer $l+1$. Suppose that the error δ^{l+1} is known of layer $l+1$, then applying $(w^{l+1})^T$ can be interpreted as moving the error backwards through the network, providing a measure of error in layer l. Then by taking the Hadamarkt product $\odot\sigma'(z^L)$ gives the error δ^l in the weighted input layer l through the activation function. Together equations 2.13 and 2.14 can give the error δ^l for any layer. Equation 2.13 gives δ^L, then after applying equation 2.14 error δ^{L-1}. Then, equation 2.14 is applied again and again to find δ^{L-2}, δ^{L-3} and so one until the error of every layers if found.

Thirdly, an equation is needed for the rate of change of the cost with respect to any bias in the network.

$$\frac{\partial C}{b_j^l} = \delta_j^l \tag{2.15}$$

Finally there is an equation for the rate of change of the cost with respect to any weight in the network.

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{2.16}$$

The back propagation algorithm is described for a single point. Equations It consists of 5 steps:

1. **Input:** activation a^1 for the input layer is set according to equation 2.7.
2. **Feedforward:** for every layer $l = 1, 2, 3 \dots L$ compute $a^l = \sigma(w^l a^{l-1} + b^l)$.
3. **Output error:** compute the vector $\delta^L = \nabla_a C \odot \delta'(z^L)$.
4. **Back propagation of the error:** For each $l = L-1, L-2 \dots 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \delta'(z^L)$
5. **Output:** the gradient of the cost function by $\partial C / \partial w^l_{jk} = a^{l-1}_k \delta^l_j$ and $\partial C / b^l_j = \delta^l_j$

2.4.3 Physics Informed Neural Networks

A neural network is transformed to a physics informed neural network (PINN) by altering the cost function. Suppose that by either simulation or experimentation on a certain physical system a spatio-temporal dataset $\gamma(\{x, t\})$ is retrieved such that the system, for unknown constants θ_1, θ_2 and θ_3, obeys:

$$\partial_t \gamma = \theta_1 \gamma + \theta_2 \gamma + \theta_3 \nabla \gamma. \tag{2.17}$$

Then, rewriting equation 2.17,

$$f = \theta_1 \gamma + \theta_2 \gamma + \theta_3 \nabla \gamma - \partial_t \gamma, \tag{2.18}$$

it can directly be inserted in the cost function of the neural network. Subsequently, for a PINN, an extra term is added to the cost function 2.8, resulting in a new cost function:

$$\begin{aligned} \mathcal{L}_{pinn} &= \frac{1}{2n} \sum_i |\gamma_i - a^L_i|^2 + \frac{\kappa}{n} \sum_i |\theta_1 \gamma_i + \theta_2 \gamma_i + \theta_3 \nabla \gamma_i - \partial_t \gamma_i|^2, \\ \mathcal{L}_{pinn} &= \mathcal{L}_{mse} + \frac{\kappa}{n} \sum_i |f_i|^2. \end{aligned} \tag{2.19}$$

Here κ is a constant that sets the strength of the contribution of the second term in comparison to first term. Now the physics are encoded within the network. By inserting f into the cost function physically feasible solutions can be found by training the network. The cost function 2.19 is a function of the weights, biases and parameters θ_1, θ_2 and θ_3, these are inferred by training the network. Hence, equation 2.17 coupled with the inferred coefficients gives the underlying model for the dataset $\gamma(\{x,t\})$.

Evidently, equation 2.17 is a generic equation so that the concept of a PINN can be illustrated easily. Actual physical equations have been employed successfully such as the Navier-Stokes equation [11].

A quick glance at 2.19 also reveals two major shortcomings of PINNs. First, it requires some prior knowledge of the system so that a model can be proposed to transform the neural network to a PINN by altering the cost function. Secondly, given the data and the proposed model, the PINN retrieves coefficients without actually providing information about why these are the coefficients. In a sense, it is a purely mathematical construct that provides no further information about casuality.

2.5 Deepmod

DeepMoD is a deep learning based model discovery algorithm that seeks partial differential equations (PDEs) underlying a spatio-temperal dataset [2] (the explanation of DeepMod presented in this section is largely based on the arguments presented in that paper). It performs particulary well for noisy datasets. It has already proven to be able to correctly predict PDEs for various equations such as the Burgers' equation and the Keller-Segel equations. The key concepts are discussed here. This subsection can be interpreted as a summary of it's original paper, to which we refer for more details.

The problem at hand is that for a dataset $u(\{x,t\})$ DeepMoD wishes to solve:

$$\partial_t u(x,t) = \mathcal{F}(u, u_x, uu_x, u_{xx}, ...). \tag{2.20}$$

Hence the problem boils down to finding the correct PDEs underlying a spatio-temporal dataset $u(\{x,t\})$. The task of finding the correct PDEs is approached by writing down the task as a regression problem:

$$\partial_t \boldsymbol{u} = \boldsymbol{\Theta}\xi. \tag{2.21}$$

Here $\partial_t \boldsymbol{u}$ is a column vector of size n containing the time derivatives of each sample, Θ is a matrix containing a *library* of polynomial and spatial derivative functions given by:

$$\Theta = \begin{bmatrix} 1 & u(\{x,t\}_0) & \cdots & u^2 u_{xx}(\{x,t\}_0) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & u(\{x,t\}_n) & \cdots & u^2 u_{xx}(\{x,t\}_n) \end{bmatrix}$$

Here it can also directly be seen that this matrix contains more elements for PDEs. If two systems have the same order of derivatives, a system that is only described by its spatial derivatives has less columns than a system that is described by both its spatial- and time derivatives. ODEs therefor are more restricting than PDEs.

ξ is the coefficient vector for which the aim is to find a sparse representation. This means that the coefficient vector ξ has many zeros in it, the sparsity is not be confused with data sparsity! This regression task is solved using *Lasso*, a sparsity promoting regression method within a neural network.

A densely-connected feed-forward neural network is employed that takes the spatio-temporal coordinates of the problem such that from input $\{x, t\}$ it outputs an approximation of u at $\{x, t\}$, $\hat{u}$. Now, the cost function differs from a cost function that is applied in regular PINNs since it contains a term $\mathcal{L}_{L_1}$ that ensures sparsity of the coefficient vector ξ. The cost function is given by:

$$\mathcal{L} = \mathcal{L}_{mse} + \mathcal{L}_{regression} + \mathcal{L}_{L_1}. \tag{2.22}$$

The first term on the right, $\mathcal{L}_{mse}$, is the mean squared error (MSE) of the output of the neural network $\hat{u}$ with respect to the dataset $u(\{x, t\})$,

$$\mathcal{L}_{mse} = \frac{1}{n}\sum_{i=1}^{n} |u(\{x, t\} - \hat{u}_i)|^2. \tag{2.23}$$

The second term on the right,

$$\mathcal{L}_{regression} = \frac{1}{n}\sum_{i=1}^{n} |\Theta_{ij}\xi_j - \partial_t \hat{u}_i|^2, \tag{2.24}$$

performs regression to find the coefficient vector ξ. The final term on the right represents a L_1 regularizer on ξ, which is given by:

$$\mathcal{L}_{L_1} = \lambda \sum_{i=2}^{m} |\xi^i|. \tag{2.25}$$

Here λ is a constant that determines the strength of the regularization. DeepMoD updates the coefficient vector ξ alongside the weights and biases of the neural network. The terms in Θ are computed from the output of the neural network. The combination of $\mathcal{L}_{mse}$ and $\mathcal{L}_{regression}$ constrain the network in a way such that it converges to the right solution. Since the MSE converges before ξ does, a convergence criterion is established based on the convergence of ξ:

$$max(\frac{\partial \mathcal{L}}{\partial \xi_i}\frac{||\partial_t u||}{||\Theta_i||}) < tol. \tag{2.26}$$

This convergence criterion states that the maximum value of the gradient of the loss function with respect to the coefficients must be smaller than a given tolerance value. Given that there is no guarantee that a coefficient vector ξ that satisfies 2.26 is retrieved, the network is trained until the specified tolerance value is reached or if a maximum amount of iterations has been reached.
If training the neural network is executed successfully, the sparse coefficient vector ξ is retrieved. Still, this is not yet the true sparse vector representation. Even after regularization, most terms will be non-zero so the small coefficients must be tresholded. Each term in equation 2.20 has a different dimension hence they are made dimensionless using:

$$\partial_t u \to \frac{\partial_t u}{||\partial_t u||}, \quad \Theta \to \frac{\Theta}{||\Theta||} \quad \text{and} \quad \xi \to \xi\frac{||\Theta||}{||\partial_t u||}. \tag{2.27}$$

Here $||\Theta||$ is the norm of each column of Θ and $||\partial_t u||$ is the norm of the time-derivative vector. Components of ξ will typically be $\mathcal{O}(1)$ as a result of this transformation. Then, the network is trained one final time without L_1 pentalty and with the regression term only containing terms selected in the first cycle. Ultimately this result in an unbiased estimate of the coefficients of the PDEs underlying the physical problem at hand.

3 Methods

3.1 Collecting data

Cytosim outputs plain *.txt* files. While nearly everything that happens within a model can be retrieved as output in the form of *.txt* file, implementation of the data such that it can be used as trainingdata for a PINN requires a bit more effort.

For the cytosim model that is used in this research the output that cytosim gives for the fiber points and end points of fibers are sufficient to retrieve a spatio-temporal dataset that can be used as a training set for a neural network. All fiber points and end points are given using the commands *./report fiber:points* and *./report fiber:ends* respectively. In appendix 6.2 it can be seen what the output looks like.

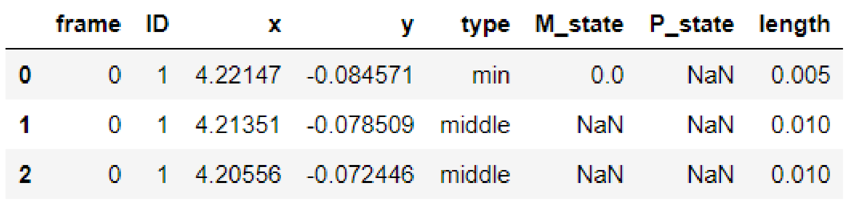
3.2 Data preparation

After the model data has been collected it must be parsed such that it is suitable for deep learning. This is done using the python language in *jupyter notebooks*. Full credit goes to Gert-Jan Both for writing this code. The code together with the appropriate amount of comments is displayed in appendix 6.3. If one is familiar with python, a brief scan through the code might already be sufficient to comprehend the data parsing process.

The parsing basically comes down to creating one *pandas* dataframe using three functions: *points_parser*, *ends_parser* and *merger*. This can be seen as three steps that are executed in order to form the dataframe.

1. First, *points_parser* is used to for the fiber points. They are split per frame and put in a dataframe.
2. In the second step *ends_parser* is used for the fiber ends. These are also split per frame and put in a dataframe.
3. Finally, as the name suggests, *merger* is used to combine the dataframes from step 1 and 2 into one big dataframe. This contains all points for all frames and contains informationa about the polarity of a point.

Following the steps above a dataframe that holds all fiber points as can be seen in figure 5 is obtained. This dataframe is that which ultimately enables us to find results. First, it allows investigations of the effects of variation in model parameters (see figures 6 and 7). Also, it prepares the data so that it can be used for the deep learning described in sections 2.4.3 and 2.5.



	frame	ID	x	y	type	M_state	P_state	length
0	0	1	4.22147	-0.084571	min	0.0	NaN	0.005
1	0	1	4.21351	-0.078509	middle	NaN	NaN	0.010
2	0	1	4.20556	-0.072446	middle	NaN	NaN	0.010

Figure 5: A screenshot of a dataframe that can be retrieved in a jupyter notebook using the data parsing code. The first four terms speak for itself. The *type* can indicate a middle, plus or minus segment of a fiber. The M_state is 0.0 for minus ends, since there is no nucleation on the minus ends of fibers. For middle- or plus ends it is NaN. Similarly, the P_state only exists for plus ends, it is 1.0 for growing ends and 4.0 for inactive ends. For middle- and minus ends it is NaN. The *length* gives the length of point.

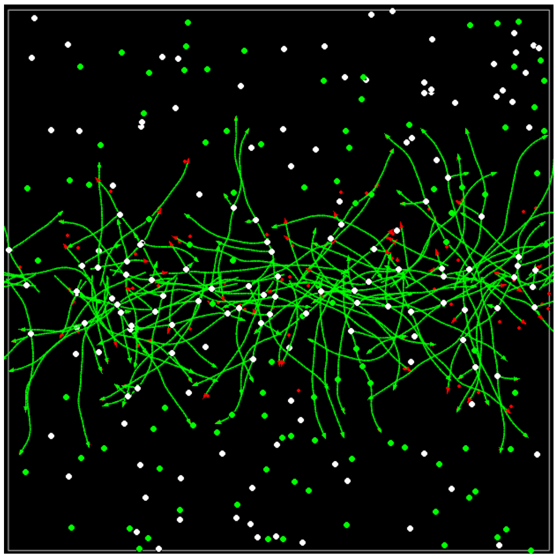


Figure 6: A screenshot from a simulation in cytosim at $t = 0.4s$. For this particular simulation there are 100 fibers initiated at the center. The simulation takes 1 second, so $simul = 100$ and $timestep = 0.1s$. In figure 7 it can be seen that in python the same image can be reconstructed.

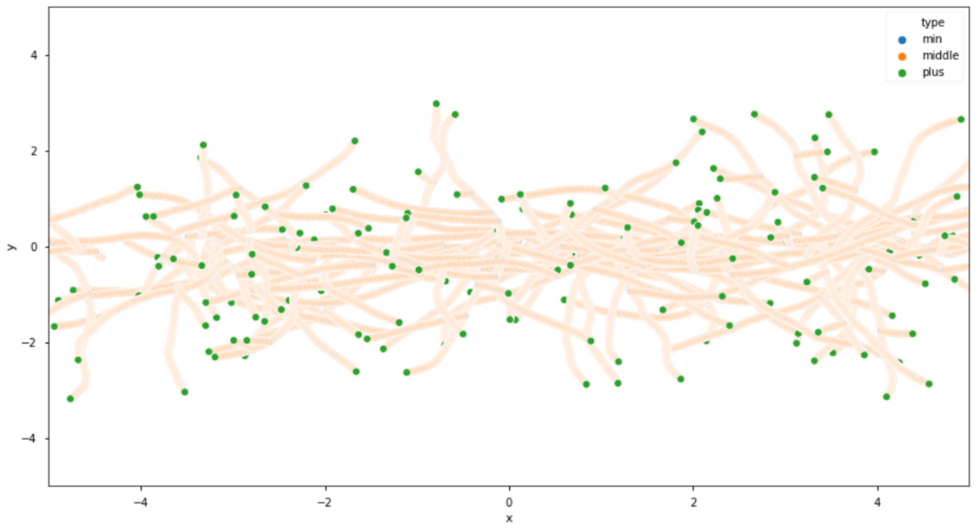


Figure 7: spatial distribution of fiber points reconstructed using the data parsing code provided in appendix 6.3

4 Results

4.1 Cytosim model behaviour

First some basic results of the model are evaluated. The aim is to describe the behaviour qualitatively in order to demonstrate that the cytosim model behaves as expected.

Recall that a simulation can be denoted Ω as described in section 2.2.3. Given that a relative simple model is simulated expected behaviour is observed. By constructing histograms it is confirmed that there axial symmetry as displayed in figure 8. Furthermore it can be seen in figure 9 that fibers seems to be uniformly distributed in the x-direction.

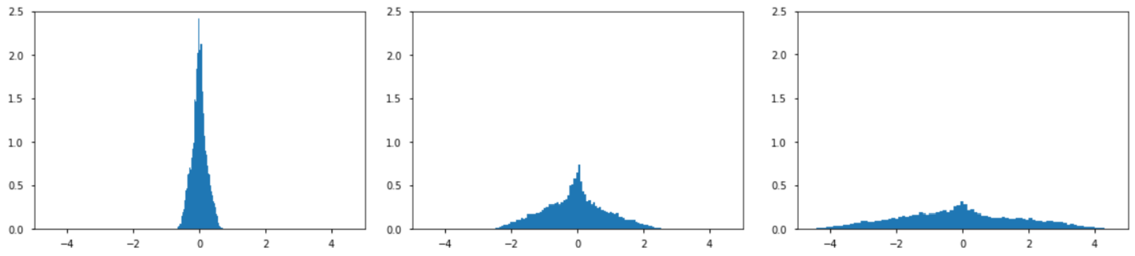


Figure 8: Three normalized histograms are displayed for a simulation $\Omega(0.01, 100, 10, 500, 200)$ on frame 1, 5 and 10 (left to right). In section 3.2 it can be seen how model output can be prepared. These histograms take as input for all points their y-position, which is the distance from the x-axis in the cytosim model. Symmetry around the origin is observed in the histogram implying axial symmetry around the x-axis in the cytosim model.

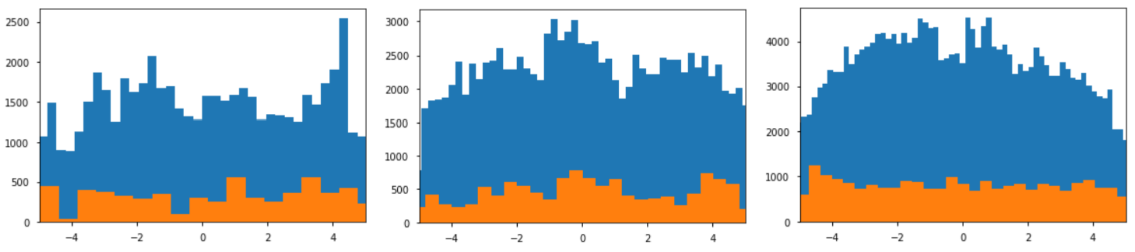


Figure 9: Three histograms are displayed for three different simulations $\Omega(0.01, 100, 10, 250, 100)$, $\Omega(0.01, 100, 10, 500, 200)$ and $\Omega(0.01, 100, 10, 1000, 400)$ (left to right). The histograms are from all data points in their respective simulations of the x-coordinate. The orange histogram is from frame 1 and the blue from frame 10. The histograms are more or less uniform and it is can be seen that as the number of initial fibers and crosslinkers increase, uniformity increases too.

4.2 Model fitting

To recapitulate, in section 2.3 the suggested model was proposed via equations 2.4 and 2.5:

$$\frac{dN}{dt} = k_p N^+,$$

$$\frac{dN^+}{dt} = k_b N - k_c N^+.$$

Using some technical computing system such as *mathematica* the model can be solved using the initial conditions $N(0) = 0$ and $N^+(0) = n_0$. Also, it is assumed that $\{k_c, k_b, k_p, n_0\} > 0$. For $N(t)$ and $N^+(t)$ the following results are obtained:

$$N(t) = \frac{2\text{Exp}(-\frac{k_c}{2}t)k_p n_0 \text{Sinh}(\frac{1}{2}t\sqrt{k_c^2 + 4k_b k_p})}{\sqrt{k_c^2 + 4k_b k_p}}, \tag{4.1}$$

$$N^+(t) = \text{Exp}(-\frac{k_c}{2}t)n_0(\text{Cosh}(\frac{1}{2}t\sqrt{k_c^2 + 4k_b k_p}) - \frac{k_c \text{Sinh}(\frac{1}{2}t\sqrt{k_c^2 + 4k_b k_p})}{\sqrt{k_c^2 + 4k_b k_p}}) \tag{4.2}$$

Here, t is the time and n_0 is the initial number of active plus-ends. The proposed model consists of ordinary differential equations instead of partial differential equation(s). In section 3.2 it can be seen how data was collected for every fiberpoint and endpoint, for every frame. If grouped by frame for a simulation $N(t)$ and $N^+(t)$ can be retrieved. With the true values known, a non linear fit with equations 4.1 and 4.2 is performed. The results are displayed in figures 10, 11 and 12.

In figure 10 it can be seen that equation 4.1 describes the evolution of total fiber length well. However, it seems that equation 4.2 does not seem to describe the evolution of plus ends well. But, both figures 10 and 11 are constructed using one simulation only. If more simulations are ran and averaged, it can be observed that the fit works better as displayed in figure 12.

4.3 DeepMoD

Unfortunately no results from DeepMoD can be displayed due to technical difficulties. DeepMoD as of yet does not work well ordinary differential equations. It was found that for ordinary differential equations, the elements of the sparse regression vector ξ do not go to zero, or not go to zero fast enough. With ordinary differential equation DeepMoD struggles to drop terms.

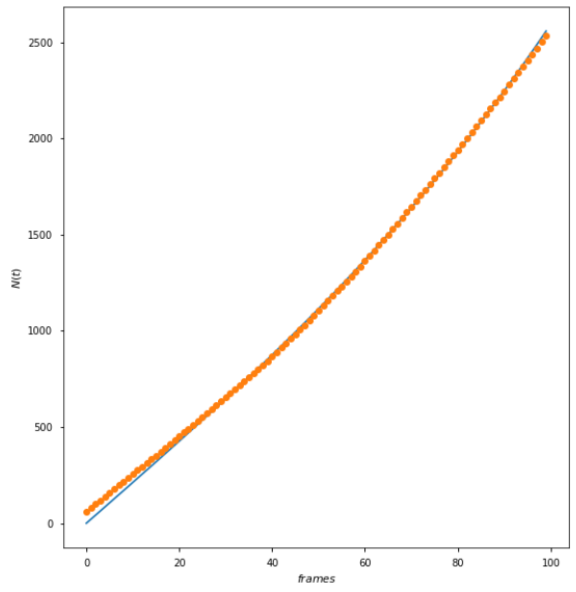


Figure 10: Here the total fiber length is plotted against the time in frames for a simulation $\Omega(0.01, 100, 100, 1000, 400)$. Orange dots are the actual data points while the blue line is the fit.

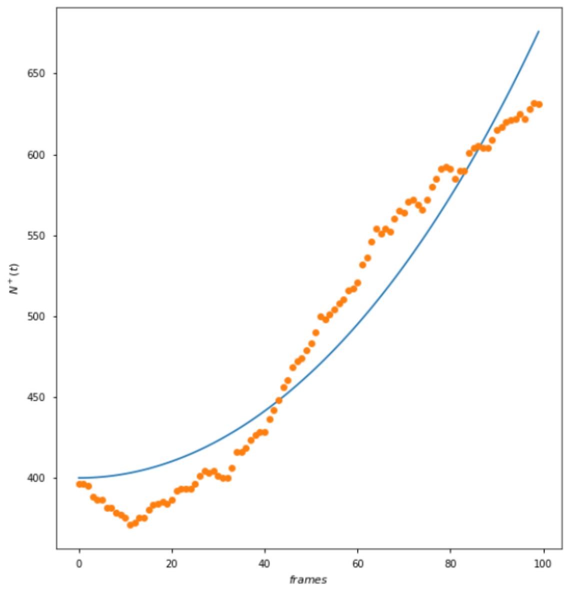


Figure 11: Here the number of active ends is plotted against the time in frames for a simulation $\Omega(0.01, 100, 100, 1000, 400)$. Orange dots are the actual data points while the blue line is the fit.

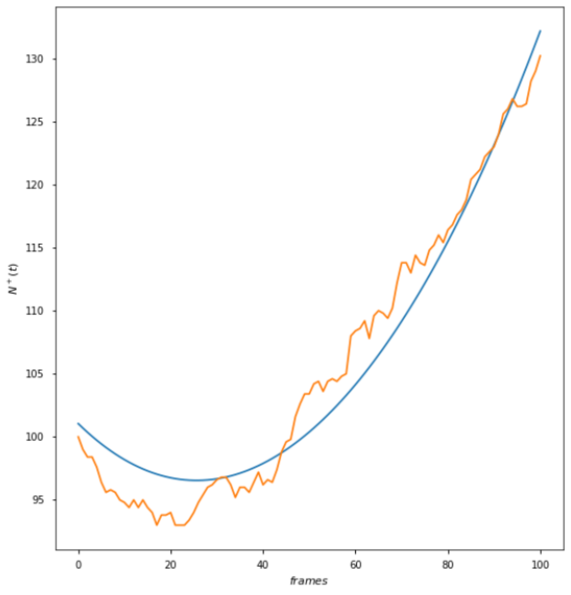


Figure 12: Here in orange the average for 5 simulations $\Omega(0.01, 100, 100, 1000, 400)$ is plotted. In blue the fit is plotted

5 Conclusion and outlook

This report consists of two parts: data generation using cytosim, and machine learning using DeepMoD. Cytosim exhibits desirable behaviour in the context of data generation. First of all, data is essentially near infinite with the bottle neck being processing power. Secondly, data is reproducible. While each simulation actually gives slightly different output for the same cytosim model, they can never differ too much. Also, if simulations are averaged it can be seen that the suggested model describes the evolution of total fiber length and active plus ends well. Finally, it is relatively easy to work with cytosim output since the files are exported as .txt files that are straightforwardly parsed for further processing.

The machine learning results were not as anticipated. We have found, that DeepMoD struggles to correctly predict underlying ordinary differential equations, and in particular does notably worse than it performs with partial differential equations. We conjecture that the reason for this is that a system based on ordinary differential equations has fewer degrees of freedom as discussed in section 2.5. DeepMoD uses the spatial and temporal derivatives to regularize the cost function but fails to do so for the system of coupled ordinary differential equations in this report.

To improve the performance of DeepMoD for ordinary differential equations two suggestions are made. First, extra trajectories can be added in so that an extra constant is added artificially. Also, DeepMoD could first train using only the mean squared error as the cost function. Then, separately, equations that describe the model can be fitted. Using these results, better starting parameters can be taken. Intuitively this can be seen as making an educated guess of where to start the gradient descent.

In hindsight one may cautiously state that within the available time it was a rather ambitious project. In this report only one model was proposed that is comprised of ODEs. Given that DeepMoD struggles to make correct predictions for ODEs in future work PDEs can be used. For example, one could use a continuum approach with spatial as well as time derivatives. Further more, noise can be added to simulation results to test the robustness of DeepMoD for the suggested model(s). Also, from this simple model of fiber growth, the extension can easily be made to a more complex system that bears closer resemblance to an actual eukaryotic cell. Other particles and fibers can be added to capture the effects of molecular motors and microtubili.

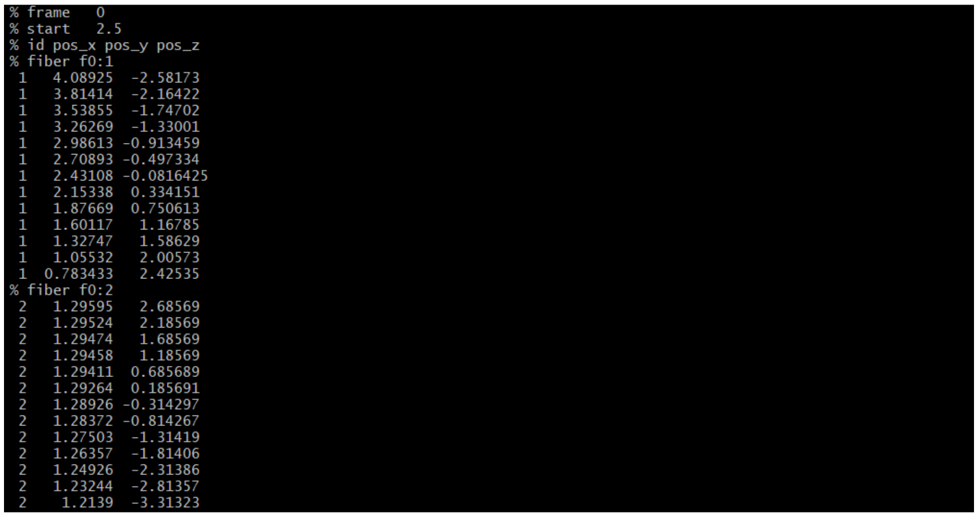
6 Appendices

6.1 Cytosim code

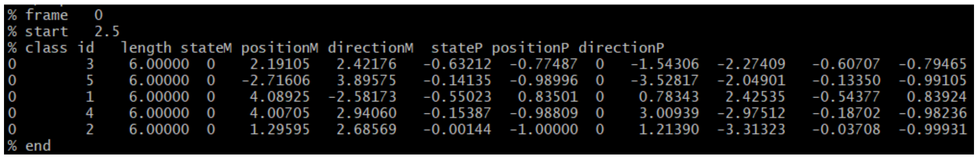
```
set simul branch
{
    time_step = 0.01
    viscosity = 1
}

set space boundary { geometry = ( square 5 5 ) }

new space boundary

set fiber actin
{
    rigidity = 0.02
    segmentation = 0.01

    activity = dynamic
    growing_speed = 5, 0.0
    growing_force = 8.0

    display = { plus_end=5; color=green; line_width=1 }
}

set hand activator
{
    binding = 10, 0.1
    unbinding = 0, 4
    display = ( size=7; color=green; )
}

set hand nucleator
{
    unbinding = 0, 3
    activity = nucleate
    nucleate = 10, actin, ( fiber_length=0.1; single = plus_end_nucleator,
        plus_end; nucleation_angle=1.22; )
    display = ( size=7; color=white; )
}

set couple arp23
{
    hand1 = activator
    hand2 = nucleator
        trans_activated = 1
    diffusion = 1
    stiffness = 1000
    activity = fork
    torque = 1, 1.22  % 1.22 radian is 70 degrees
}
```

```
set hand capper
{
    binding = 1.0, 0.02
    unbinding = 1.0, 100.0

    activity = nucleate
    nucleate = 0.0, actin, { length=0.1 }

    hold_growing_end = 1
    hold_shrinking_end = 0
    track_end = plus_end

    addictive = 1
    display = { color=red }
}

set single plus_end_nucleator
{
    hand = capper
    stiffness = 100.0
    diffusion = 0.1
}

new 100 fiber actin { length = 0.1; single = plus_end_nucleator, plus_end;
    position = rectangle 5 0.1 at 0 0 }

new 250 couple arp23

run 100 simul *
{
    nb_frames = 10
}
```

6.2 Cytosim output



```
% frame   0
% start   2.5
% id pos_x pos_y pos_z
% fiber f0:1
 1   4.08925  -2.58173
 1   3.81414  -2.16422
 1   3.53855  -1.74702
 1   3.26269  -1.33001
 1   2.98613 -0.913459
 1   2.70893 -0.497334
 1   2.43108 -0.0816425
 1   2.15338  0.334151
 1   1.87669  0.750613
 1   1.60117   1.16785
 1   1.32747   1.58629
 1   1.05532   2.00573
 1  0.783433   2.42535
% fiber f0:2
 2   1.29595   2.68569
 2   1.29524   2.18569
 2   1.29474   1.68569
 2   1.29458   1.18569
 2   1.29411  0.685689
 2   1.29264  0.185691
 2   1.28926 -0.314297
 2   1.28372 -0.814267
 2   1.27503  -1.31419
 2   1.26357  -1.81406
 2   1.24926  -2.31386
 2   1.23244  -2.81357
 2    1.2139  -3.31323
```



```
% frame   0
% start   2.5
% class id   length stateM positionM directionM  stateP positionP directionP
0       3    6.00000  0    2.19105   2.42176   -0.63212  -0.77487  0   -1.54306  -2.27409   -0.60707  -0.79465
0       5    6.00000  0   -2.71606   3.89575   -0.14135  -0.98996  0   -3.52817  -2.04901   -0.13350  -0.99105
0       1    6.00000  0    4.08925  -2.58173   -0.55023   0.83501  0    0.78343   2.42535   -0.54377   0.83924
0       4    6.00000  0    4.00705   2.94060   -0.15387  -0.98809  0    3.00939  -2.97512   -0.18702  -0.98236
0       2    6.00000  0    1.29595   2.68569   -0.00144  -1.00000  0    1.21390  -3.31323   -0.03708  -0.99931
% end
```

Figure 13: As example to see the output that cytosim provides, output can be seen for a simple model which has 5 non-interacting fibers that diffuse in space. Output is displayed for 1 fiber in the first frame. The output for the model for cytoskeletal growth is the same with simply more fibers and thus more points. **Top:** A screenshot from the terminal of the output of the fiber points. For every frame and for every fiber, cytosim outputs the fiber points and coupled with their $x-$ and $y-coordinate$. **Bottom:** A screenshot from the terminal of the output of the fiber ends. For both the minus-ends and plus-ends it gives the their $x-$ and $y-coordinate$ aswell as their orientation. Also, the state of the minus-end and plus-end, *stateM* and *stateP* respectively, is given. For this particular simulation it is 0 for both meaning that on both ends their is no nucleation.

6.3 Python script for dataparsing

```
import pandas as pd
import numpy as np


def points_parser(path):
    'Takes in path to points file generated by cytosim, returns dataframe.'

    raw_data = [line.strip() for line in open(path)]

    # Split data into separate frames
    frame_locs = [idx for idx, line in enumerate(raw_data) if 'frame' in line]
    raw_data_per_frame = np.split(np.array(raw_data), frame_locs)

    # Cutting the crap and turning stuff into actual numbers
    data_per_frame = [[line for line in frame if (len(line) != 0) and (line[0]
        != '%')] for frame in raw_data_per_frame][1:] #cut out empty lines,
        first entry and info lines
    data_per_frame_split = [[line.split(' ') for line in frame] for frame in
        data_per_frame]
    data_per_frame_split = [np.array([[float(element) for element in line if
        len(element) != 0] for line in frame]) for frame in
        data_per_frame_split]

    # Combining it all in a nice dataframe
    df_data = [pd.DataFrame(frame, columns=['ID', 'x', 'y']) for frame in
        data_per_frame_split]
    for frame, frame_data in enumerate(df_data):
        frame_data['frame'] = frame
    df = pd.concat(df_data, axis=0, ignore_index=True)

    # Final details
    df = df[['frame', 'ID', 'x', 'y']]
    df['ID'] = df['ID'].apply(lambda x: np.int(x))

    return df


def ends_parser(path):
    'Takes in path to points file generated by cytosim, returns dataframe.'

    raw_data = [line.strip() for line in open(path)]

    # Split data into separate frames
    frame_locs = [idx for idx, line in enumerate(raw_data) if 'frame' in line]
    raw_data_per_frame = np.split(np.array(raw_data), frame_locs)

    # Cutting the crap and turning stuff into actual numbers
    data_per_frame = [[line for line in frame if (len(line) != 0) and (line[0]
        != '%')] for frame in raw_data_per_frame][1:] #cut out empty lines,
        first entry and info lines
    data_per_frame_split = [[line.split(' ') for line in frame] for frame in
        data_per_frame]
    data_per_frame_split = [np.array([[float(element) for element in line if
        len(element) != 0] for line in frame]) for frame in
        data_per_frame_split]

    # Combining it all in a dataframe
    df_data = [pd.DataFrame(frame[:, [1, 2, 3, 4, 5, 8, 9, 10]], columns=['ID',
         'length', 'M_state', 'M_x', 'M_y', 'P_state', 'P_x', 'P_y']) for frame
         in data_per_frame_split]
    for frame, frame_data in enumerate(df_data):
        frame_data['frame'] = frame
    df = pd.concat(df_data, axis=0, ignore_index=True)

    # Final details
    df = df.sort_values(['frame', 'ID'])
    df = df[['frame', 'ID', 'length', 'M_state', 'M_x', 'M_y', 'P_state', 'P_x
        ', 'P_y']]
    df['ID'] = df['ID'].apply(lambda x: np.int(x))
    df['M_state'] = df['M_state'].apply(lambda x: np.int(x))
    df['P_state'] = df['P_state'].apply(lambda x: np.int(x))

    df = df.reset_index(drop=True)

    return df


def merger(points_df, ends_df):
    def fiber_properties(fiber, fiber_ends):
        internal_fiber = fiber.copy()
        # Setting the segment length/weight
        segment_length = fiber_ends.length / (len(fiber.index) - 1)
        internal_fiber['length'] = segment_length

        # Finding the plus and minus segments
        min_idx = fiber[np.isclose(fiber.x, fiber_ends.M_x, atol=1e-5) & np.
            isclose(fiber.y, fiber_ends.M_y, atol=1e-5)].index[0]
        plus_idx = fiber[np.isclose(fiber.x, fiber_ends.P_x, atol=1e-5) & np.
            isclose(fiber.y, fiber_ends.P_y, atol=1e-5)].index[0]

        # Setting all the data
        internal_fiber.loc[min_idx, ['type', 'M_state', 'P_state', 'length']] =
             ['min', fiber_ends.M_state, np.nan, 1/2 * segment_length]
        internal_fiber.loc[plus_idx, ['type', 'M_state', 'P_state', 'length']]
            = ['plus', np.nan, fiber_ends.P_state, 1/2 * segment_length]

        return internal_fiber

    df = points_df.copy()
    df['type'] = 'middle'
    df['M_state'] = np.nan
    df['P_state'] = np.nan
    df['length'] = 0.0
    new_df = [fiber_properties(fiber, fiber_ends) for ((points_idx, fiber), (
        ends_idx, fiber_ends)) in zip(df.groupby(['frame', 'ID']), ends_df.
        iterrows())]
    df_full = pd.concat(new_df, axis=0, ignore_index=True)

    return df_full
```

References

[1] Julio M Belmonte, Maria Leptin, and François Nédélec. A theory that predicts behaviors of disordered cytoskeletal networks. *Molecular systems biology*, 13(9):941, 2017.

[2] Gert-Jan Both, Subham Choudhury, Pierre Sens, and Remy Kusters. Deepmod: Deep learning for model discovery in noisy data. *arXiv preprint arXiv:1904.09406*, 2019.

[3] Nikhil Buduma and Nicholas Locascio. *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms.* " O'Reilly Media, Inc.", 2017.

[4] Donald O Hebb. The organization of behavior; a neuropsycholocigal theory. *A Wiley Book in Clinical Psychology.*, pages 62–78, 1949.

[5] Anuj Karpatne, William Watkins, Jordan Read, and Vipin Kumar. Physics-guided neural networks (pgnn): An application in lake temperature modeling. *arXiv preprint arXiv:1710.11431*, 2017.

[6] Gaëlle Letort, Antonio Z Politi, Hajer Ennomani, Manuel Théry, Francois Nedelec, and Laurent Blanchoin. Geometrical and mechanical properties control actin filament organization. *PLoS computational biology*, 11(5):e1004245, 2015.

[7] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[8] Mohammad RK Mofrad and Roger D Kamm. *Cytoskeletal mechanics: models and measurements in cell mechanics.* Cambridge University Press, 2006.

[9] Francois Nedelec and Dietrich Foethke. Collective langevin dynamics of flexible cytoskeletal fibers. *New Journal of Physics*, 9(11):427, 2007.

[10] GP Purja Pun, R Batra, R Ramprasad, and Y Mishin. Physically informed artificial neural networks for atomistic modeling of materials. *Nature communications*, 10, 2019.

[11] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. *arXiv preprint arXiv:1808.04327*, 2018.

[12] Aurélien Rizk, Grégory Paul, Pietro Incardona, Milica Bugarski, Maysam Mansouri, Axel Niemann, Urs Ziegler, Philipp Berger, and Ivo F Sbalzarini. Segmentation and quantification of subcellular structures in fluorescence microscopy images using squassh. *Nature protocols*, 9(3):586, 2014.

[13] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2016.

[14] Viktoria Wollrab, Julio M Belmonte, Lucia Baldauf, Maria Leptin, François Nédeléc, and Gijsje H Koenderink. Polarity sorting drives remodeling of actin-myosin networks. *J Cell Sci*, 132(4):jcs219717, 2019.

[15] Jacek M Zurada. *Introduction to artificial neural systems*, volume 8. West publishing company St. Paul, 1992.