University of Bergen
Department of Informatics

Deep Learning Approach To Gene Network Inference

Author: Daniel Hammerstad Johannessen
Supervisors: Tom Michoel and Susanna Röblitz

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

May, 2023

Abstract

Gene regulatory network(GRN) inference remains a challenging problem in the field of bioinformatics. GRN contain valuable information needed to get a deeper understanding of the regulatory network. This could lead to advances in disease treatment or help drug discovery. Our approach to solving the problem of GRN inference is to train multilayered perceptrons (MLPs) to recreate the dynamics of the biological function. With the trained models able to module the dynamics, we hope to extract the underlying relationships between the species through feature attribution algorithms. We apply our method to a regulatory network for cell apoptosis and a network regulating the T-cell response to a pathogen.

Acknowledgements

First and foremost, I would like to thank my supervisors Susanna Röblitz and Tom Michoel for their guidance, bright ideas and invaluable feedback throughout the project and writing of the thesis. Without this, the thesis would not have been possible.
I would also like to thank family and friends for their support. A special thanks to my parents for all of their encouragement during my studies.
Finally, I would like to thank Lea for her unwavering support and for always believing in me.

Daniel H Johannessen
Wednesday 31st May, 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the field of bioinformatics, the concept of gene regulatory networks (GRN) has become the favored choice for describing the complex and dynamic relationship between genes and their products[33]. Their use enables us to improve our knowledge about the mechanism of diseases[23] and could lead to advancements in drug discovery[9]. However, the process of inferring these GRNs remain a great challenge, and a huge effort has been made in developing a solution. The existing methods cover a wide range of different approaches[33].

While GRN is great at describing the relationships between genes and transcription factors, mathematical models have been used to model the dynamic behaviour of the system[23]. These mathematical models have the ability to simulate highly complex biological processes or biological systems. However, at a great cost. Simulations can demand a large amount of computational power and time, which is not optimal for real-time predictions. Machine Learning (ML) models have the potential to replace these, as they have great abilities as function approximators. Once trained, ML models are both time-efficient and cost-efficient in terms of computational demand[14].

Thus, through our work we would like to answer the following:

1. Is it possible to train a set of multilayer perceptrons (MLP) to simulate the dynamical behaviour of a biological process using gene expression data?
2. Can we infer a suitable network topology from the trained models using feature attribution algorithms?

In the process of answering these two questions, we were introduced to another challenge. For some systems, multiple steady states are possible. This made learning the dynamics of the system difficult as the machine learning models would generalize poorly and overfit to some of the steady states. This prompted the question if a classifier could correctly classify the model's state. We, therefore, defined another question:

3. Is it possible to train an ML classifier to correctly identify the biological state of a system using initial values?

In our work, we used a set of independently trained MLP to simulate the dynamics for given biological systems. The models are trained using simulated gene expression data. We tested their predictive ability by only providing the initial concentration for the involved species. We then compared the models predictions against the true dynamical behaviour.

We also propose a method for inferring a topology for a GRN. Having already trained a set of MLP able to recreate the dynamical behaviour, we utilized different feature attribution algorithms to infer possible network topologies. The feature attribution algorithms would provide a score representing the importance of each feature for the output of the model. By using different thresholds on the feature attribution scores for determining whether an edge should be included or not, we produced multiple possible network topologies. We then compared the proposed networks against the true network and used precision-recall and roc-curves as a measure of performance.

Finally, we used a small set of different classifiers to test whether it was possible to recognize the state of the system. The models used were Logistic Regression and two different MLPs. The different models were evaluated by their accuracy and by examining their confusion matrices.

Chapter 2

Background

2.1 Systems biology

Systems biology is a field within biology that uses a holistic approach to understand the complexity of biological systems. Rather than reducing the system into individual objects and analysing each individual object's properties and function separately, systems biology attempts to view the system as a whole and investigate its dynamics and structure[19, 18]. A biological systems structure can be biochemical pathways or a network of interacting genes. In contrast, the dynamics of the system explain how the system develops over time[25]. Any development made in the field of systems biology could contribute to a better understanding of biological processes in the human body. A better understanding of these could lead to advancements in drug discovery and treatment of diseases[24, 2].

Mathematical modelling of a biological system is a useful tool, whether the system represents individuals in a population or the number of molecules present in a reaction. While there exist many different methods for doing this, one of the more popular methods includes an ordinary differential equation(ODE) representation of the system[26]. We can do this on the following form:

$$\frac{d}{dt}X(t) = -k_1 X(t) \tag{2.1}$$

This equation describes the change of concentration of molecule or species X at time t. In this instance, X at time t is decaying at a rate of k_1 proportional to the current

concentration of X at time t. This equation only describes a single species and to extend this into describing a whole system, we can use a system of ODEs' instead.

$$\begin{aligned}\frac{d}{dt}X_1(t) &= f_1(X_1(t), X_2(t), ..., X_n(t)) \\ &\vdots \\ \frac{d}{dt}X_n(t) &= f_n(X_1(t), X_2(t), ..., X_n(t))\end{aligned}$$

This set of equations not only describes the change of concentration within the individual species but also gives information on their relationships.

Sensitivity analysis is a tool used to study mathematical models, such as ODE models. Explicitly it is used to understand how uncertainty in the model's output can be attributed to the uncertainty in the model's input[38]. Sobol sensitivity analysis is a variance-based global sensitivity analysis and can be used for non-linear models[39]. Sobol sensitivity analysis computes both total-order sensitivity and first-order sensitivity. Total-order sensitivity represents the variance in output caused by an input parameter and its interaction with other input parameters. The first-order sensitivity describes only the variance in the output caused by a single input parameter. Sobol sensitivity analysis computes the sensitivities by ranging the input parameters over the whole input space[46].

2.1.1 Gene regulatory networks

In our bodies and other life forms, biological functions are the processes responsible for the individual's ability to function and survive. Biological functions happening within cells, called cellular functions, are driven by proteins and their interactions[49]. Proteins are large biomolecules that consist of chains of amino acids. The production of proteins occurs through a complex process called gene expression. Through gene expression, information stored in DNA is transformed into functioning proteins. This transformation undergoes two major operations called transcription and translation[11]. The first, transcription is responsible for the creation of the intermediate product called messenger RNA (mRNA). Transcription begins when an RNA polymerase binds to the DNA strand and reads the coding information into mRNA. In the latter process, translation, the mRNA is transformed into a sequence of amino acids. The translation is initiated by a ribosome binding to mRNA. The ribosome synthesizes the protein product according to the information stored in the mRNA[11].

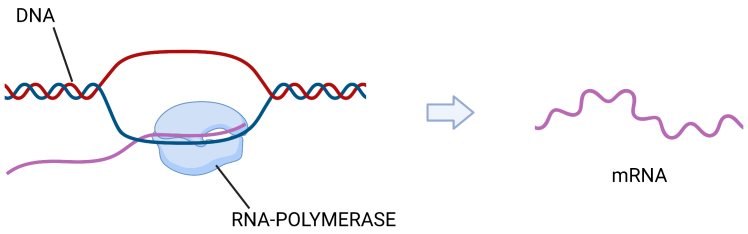


Figure 2.1: Transcription of DNA into mRNA.

Figure 2.2: Translation of mRNA into protein.

Gene expression needs to be regulated to avoid producing unnecessary amounts of protein or to avoid producing proteins at the wrong time. Regulation can happen through a variety of different processes, but the most common way is through transcription factors (TF)[30]. TF are proteins that bind to the regulatory regions of the DNA. The binding of TF changes the regulatory region where RNA polymerase binds to the DNA strand. Dependent on the type of TF, the regulatory region changes to either be able or unable to bind to RNA polymerase. A TF that changes the regulatory region into a binding site for RNA polymerase, is called an activator. On the contrary, a repressor is a TF that changes the regulatory region such that it does not allow binding to RNA polymerase[29].

In an attempt to simplify the description of the interaction of genes and their products, the concept of Gene Regulatory Networks (GRN) was developed [40]. An important advantage of using a GRN representation is that they can often be represented using a graph or network topology. By using graphs it is easier to visualize the relationship between the genes. To avoid unnecessary confusion and complexity, genes and their product, proteins, are simplified to be represented in the same node. The relationship between genes is depicted by the edges. Figure 2.3 shows an example of a GRN with five genes and their relationships.

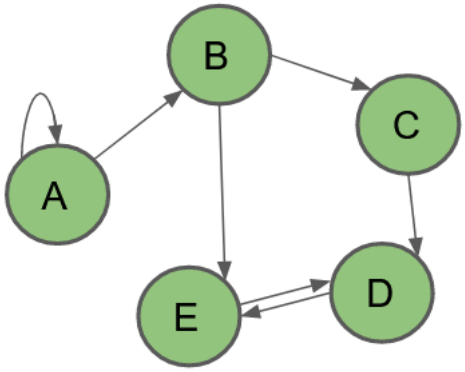


Figure 2.3: Example of gene regulatory network.

Inferring GRNs remains a challenging problem within the field of systems biology. Researchers have developed several methods, with a large variety of approaches to solve this problem.[54]

2.1.2 Examples of biological functions and biological systems

In this section, we will first take a look at a few general examples of the type of biological functions and systems. Then we will present two different types of systems we will study in this thesis.

Adaptation

Adaptation is one of the most widely studied biological functions and can be found in many biological systems. The adaptation process can be broken down into two parts, the first being a transient response to a change in an input signal. The second part is the recovery stage where the system adapts to the change in signal and is restored to its initial level[19, 43]. A perfect adaption happens when the system is able to return to the exact same level as before the change in signal[50]. From research, the modelling of such a function involves three nodes, one input (signal) node, one hidden node (gene 1), and one output (gene 2) node. From this, two possible network topologies arose. The first is a negative feedback loop with a buffer node, while the second is an incoherent feed-forward loop with a proportioner node[32].

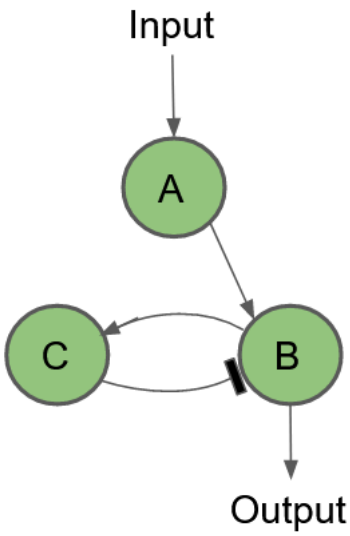


Figure 2.4: Example of adaptation network with negative feedback loop.

Oscillations

An important component of many dynamical cellular processes is the ability of the system to exhibit oscillating behaviour[28]. A system containing both positive and negative feedback loops will result in oscillations. The positive feedback loop is responsible for constructing a bistable system. The negative feedback loop will make the system jump in between these two steady states, therefore creating the oscillating behaviour. While the oscillating behaviour is possible for a system with only negative feedback, this system will eventually come to a steady state through damped oscillations[50].

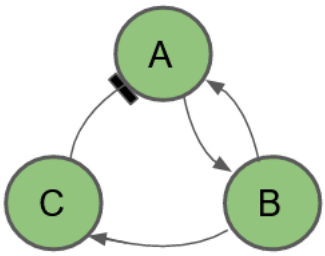


Figure 2.5: Example of oscillatory network with a positive and a negative feedback loop.

Cell apoptosis

Cell apoptosis is an important biological process in which the cells are programmed to die in a controlled fashion. The process can be regulated in several ways, but it is

primarily regulated through transient signals in a caspase-mediated network[3]. Caspase is an enzyme responsible for the destruction of cells[48]. The caspase network consists of 3 different species, where Casp-3 activity is regulated by Casp-8 and XIAP. Casp-3 and Casp-8 are pro-apoptotic factors and XIAP is a pro-survival factor. Casp-8 activates Casp-3, while XIAP inhibits Casp-3 activation. The total amount of complexes and species in the network is 9, while one is not affecting the regulation, and is therefore excluded from the regulatory network[3]. The full overview of the GRN can be viewed in figure 2.6and the corresponding system of ODE's modelling of the network can be found in the appendix.

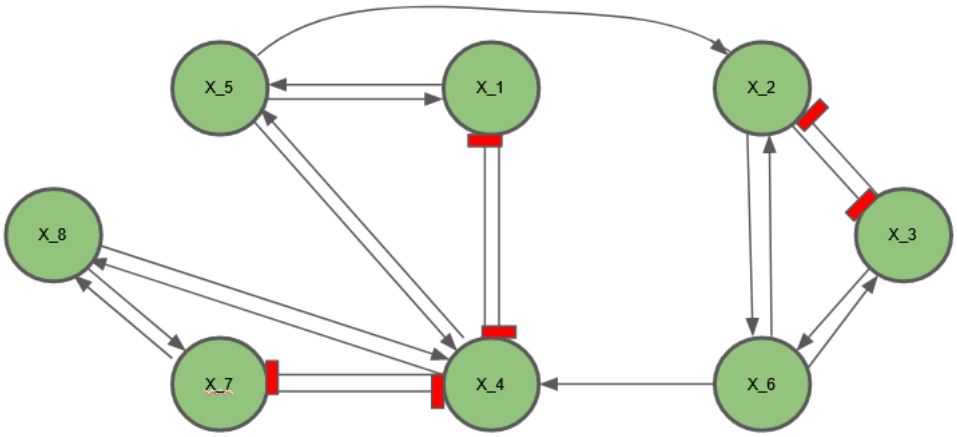


Figure 2.6: The cell apoptosis regulatory network, with 8 species, Casp8(x_1), Casp8*(x_2), Casp3(x_3), Casp3*(x_4), Casp8-Casp3*(x_5), Casp8*-Casp3(x_6), XIAP(x_7) and XIAP-Casp3*(x_8). Activated caspase is denoted by "*", while complexes are denoted by "-" between them.

Ontogeny of nascent CD8 memory T-cells

T-cells are a vital part of the immune system and crucial to fighting any harmful cells that have entered its host body. They are responsible for the adaptive ability of the immune system to handle new infections or diseases[5]. Due to the different stages in the life cycle of a T-cell, it can provide both a short-term response and long-term protective ability. The introduction of a pathogen or a tumor will activate naive T-cells to differentiate into effector cells, capable of eliminating any cell affected by a pathogen or a tumor. With time, the effector cells will differentiate into memory cells, providing long-term protection against the same antigen[53]. Through phenotype investigation 4 possible different phases have been identified, naive (N), early effector (E), late effector (L), and memory (M). The

regulatory network consists of 4 different cell phases and a pathogen (P). The pathogen activates the differentiation of N into E, and E into L. E and L activate the degradation of pathogen [10].

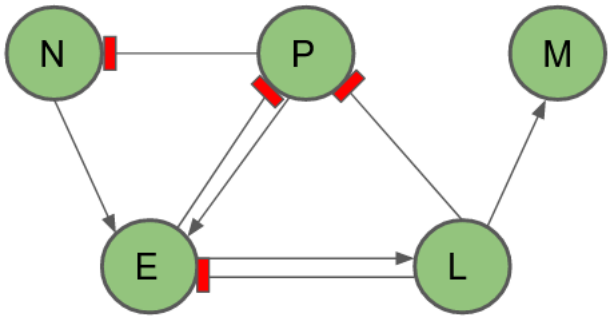


Figure 2.7: The regulatory network for T-cell stages and the pathogen.

2.2 Machine learning

Machine learning is a interdisciplinary field within computer science and statistics[20]. The aim of machine learning is to teach machines to perform a set of tasks using experience in the form of data. The learning part happens through a phase called training. The training mimics the way we humans learn, by repetition of experiences and outcomes. After each new experience, the machine uses this to alter its prediction[13]. Machine Learning has proven to be a powerful and versatile tool, as applications range from performing everyday tasks to more complex problems such as protein-folding[21]. Especially the last two decades have seen a renewed interest and rapid development in the field. This can mostly be contributed to hardware upgrades and the ability to store large amounts of data.

Machine learning is typically divided into 4 different types, supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning[34]. Supervised learning uses labelled data to train and test the machine learning model. Labelled data consist of a set of input features and a ground-truth output value(s). In the training phase, the model uses the input features to make a prediction. Then the difference between the prediction and the ground-truth variable is calculated using a loss function. Based upon the model algorithm used, the information captured by the loss function is used to update the model and change the predictive pattern until a optima is reached[16]. Unsupervised learning, on the other hand, uses unlabelled data for training. These types

of models are trying to recognize patterns or groupings of the data without the help of a target attribute[4]. Semi-supervised learning is an approach which utilizes a smaller part of labelled data and a larger part of unlabelled data. This approach to machine learning is often favoured when labelling data is either expensive or difficult.[51] Reinforcement learning is a branch of machine learning that uses experiences and reward signals to train an agent to perform a given task. Given some observational data (state), an agent performs an action and is given a reward signal as a response to that action in that state. By giving appropriate rewards for each state, the agent should be able to perform the task after a training period[22].

2.2.1 Neural networks

A concept within machine learning that has experienced continuous interest for the last decades is neural networks (NN)[1]. A NN is a structured network of interconnected neurons. The networks have a layered structure, where the number of layers(depth) and number of neurons in each layer(width) are predetermined hyperparameters of the model[15]. Information is passed on layer by layer and connections are only found between neurons from consecutive layers. Each NN must be comprised of one input layer, one output layer, and at least one intermediate layer referred to as a hidden layer. This gives the smallest possible NN a depth of at least 3. The data is fed into the input layer of the network and is transferred through each layer by the activation of each neuron. The output layer is responsible for presenting the model's prediction and is the result of all activations in the last hidden layer[6]. In the next paragraphs, we will have a more in-depth look at how these networks make a prediction and how they are trained.

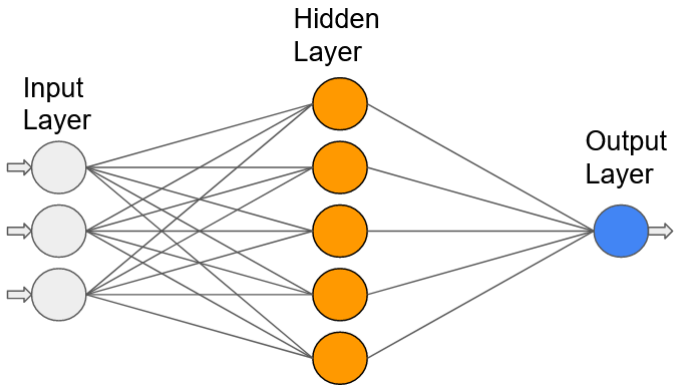


Figure 2.8: Example of a feedforward neural network.

Forward pass through a neural network

The input is passed through the neural network with a series of nonlinear transformations performed by neurons in each layer[17]. Each neuron performs a two-part operation for computing the activation Z. In the first operation, a is computed as a weighted sum over all k incoming signals x_i to the current neuron. The connections between neurons, each have an individual parameter w_i called weight. A term b_0, called bias is then added to the weighted sum to complete the first operation[6].

$$a = \sum_{i}^{k} w_i x_i + b_0 \tag{2.2}$$

In the second operation, the weighted sum is transformed using a non-linear activation function g, and we get the resulting activation Z for the neuron[6]. Often used activation functions include Rectified Linear Unit (ReLU), Sigmoid and Tanh. Where ReLU is used for hidden units in the intermediate layers, while the other two are mostly used for output layers[41].

$$Z = g(a) \tag{2.3}$$

For an intermediate neuron, the activation Z is fed as an input signal to neurons in the next layer where the process is repeated. If the neuron is part of the output layer, the activation Z is the model's prediction.

Backpropagation

While the previous section describes how neural networks perform predictions from input values, let us go into detail about how neural networks learn. In order to change a model's prediction, we need to update the model's weights. The goal of training is to update these weights in a way such that the predictions become closer to the true labels. We utilise a cost function to decide how good the model's prediction is. We define a cost function, $L(\theta)$, for a regression problem:

$$L(\theta) = \sum_{x,y} ||y - N(x, \theta)||^2 \tag{2.4}$$

The Greek letter θ denotes the neural network parameters, namely weights and bias. x and y are the features and labels of the model, respectively. $N(x, \theta)$ refers to the models prediction based on features, x, and the current parameters, θ. Here we define the cost function as the sum of squared errors. However, the choice of cost function is usually dependent on the type of problem and preference of the developer. Now, the problem becomes minimizing this cost function by the change of parameters. We can formally define the problem as:

$$argmin_{\theta} L(\theta) \tag{2.5}$$

The use of the error from the forward propagation to sequentially update the parameters of the layers in a backwards direction is known as backpropagation. Backpropagation works by computing the partial derivative of the cost function with respect to the weights. It does so in a backwards direction, propagating the error from the end of the network to the front. The derivatives can be calculated using the chain rule of differentiation[37]. For each node in the computational graph, we get:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial Z} \frac{\partial Z}{\partial x} \tag{2.6}$$

The computed gradients are then used in a process called gradient descent. The goal of gradient descent is to optimize the cost function by using the gradients[36]. Updating parameters using gradient descent can be defined as:

$$\theta_{t+1} = \theta_t + \eta_t \nabla L(\theta_t) \tag{2.7}$$

Parameters are denoted by θ and the learning rate is denoted by ∇. t denotes the current time step. It is important to note that gradient descent does not guarantee the global optima, only a local minima. This is due to the cost function not being convex[12].

2.2.2 Machine learning metrics

In this section, we will define and present some of the metrics later used to evaluate the performance of the machine learning models.

Precision

Precision is a performance metric often used for classification tasks. It represents the proportion of correctly guessed positive instances among the total guessed positive instances[35]. It is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

Recall

Recall is another performance metric used for classification tasks. It represents the proportion of retrieved positive instances among the total number of positive instances[35]. It is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

Accuracy

Accuracy is also a performance measure used for classification tasks. It represents the total number of correct predictions among all possible predictions[35]. Formally, it can be defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.10}$$

2.2.3 Model interpretability

One major critique of Neural Networks is the lack of explainability. They are often viewed as "black box" functions, where they are able to map an input to a reasonable output. However, understanding how they are able to make these predictions and what the model is looking for when making a decision, is still not entirely clear[31]. To overcome this critique an effort has been made in the area of model interpretability[8]. Captum is a code library that has collected some of the algorithms that have been developed for gradient-based learning in particular[27]. In the rest of this section, we view some of these algorithms in more detail.

Integrated gradients

Integrated gradients is a gradient-based approach to find the feature attribution. The method computes the integral of the gradients with respect to inputs along the path from the baseline to the inputs. The development of the method is driven by fulfilling two axioms: sensitivity and implementation variance. Sensitivity states that if two inputs, x and x', only differ for feature i, and the prediction of the model is different, then the feature attribution for feature i must be non-zero. Implementation variance states that two models predicting equal outputs for all inputs, despite being implemented very differently, must have the same attributions[47].

Saliency

Saliency is a simpler approach to computing attribution. The method returns the gradient of the output with respect to input[45].

DeepLIFT

DeepLIFT is a backpropagation-based approach to quantify the importance of each feature in model prediction. The algorithm does this by trying to explain the difference of the output to the "reference" output in terms of differences of the inputs to their "reference" inputs. The "reference" input is usually chosen to be some neutral baseline. The algorithm utilizes the difference between the activation in neurons to the "reference" activation in the same neurons, as contribution scores[44].

Feature permutation

Feature Permutation is a perturbation-based approach to finding the attributions. The method starts with individually permuting each feature in the testing set and then computing the difference (loss) in the output of the model for the different sets of input values. The feature values are randomly permuted within a batch. The idea is simply if a change in one feature leads to a big increase in loss, the feature must be important. Likewise, a small change in loss suggests that the feature is not important[7].

Feature ablation

Feature ablation is also a perturbation-based approach, and follows the same idea as feature permutation. However, rather than using a randomly permuted set for each feature, the method only requires a baseline or reference for each feature. The difference in output is computed when changing between the reference value and input value [27].

2.3 Related work

2.3.1 Systems biology informed deep learning for inferring parameters and hidden dynamics

In the work by Yazdani et al.[52], they are interested in developing an algorithm for parameter inference and the prediction of a system's hidden dynamics. As mentioned earlier, using ODEs to model a biological system is frequently used in systems biology. With this, unknown parameters are introduced and inferring these are the objective for Yazdani et al. In addition, they aim to achieve this using a smaller number of observable variables. To be able to infer the parameters and the hidden dynamics, they utilize a deep neural network(DNN), with the system of ODEs incorporated into the neural network(NN). To incorporate the ODEs into the NN, they use a custom loss function with three terms. Two of the terms are related to the difference between the model prediction and the measurements of each species. The last term is based on the ODE model.

Developing an algorithm that can predict the hidden dynamics of a biological system is also of interest to us, but one difference remains. In our work, we do not want to infer the parameters of ODEs used in modelling the system. Instead, we want to develop an algorithm that can infer a GRN topology.

2.3.2 Finding gene network topologies for given biological function with recurrent neural network

Shen et al.[42] displays an alternative approach to the GRN inference by using Recurrent neural networks (RNN). Their methods train an RNN to model the dynamics for the

biological function of interest. The trained RNN is investigated using a developed link knock-out method to infer possible network topologies. The method involves systematically perturbing the different inputs to the neural network and studying the change in the model's dynamic behaviour. The last part of the method focuses on exploring the biological feasibility of the proposed network topologies. This is done by checking if the inferred network can still model the desired biological function when its edges are expressed through Hill functions. The approach is applied to four different types of biological functions, adaptation, controlled oscillations, pattern formation and a set of 10-node cellular automata.

This approach differs from ours by the choice of machine learning model, and the method of inferring the gene network topology. The choice of a machine learning model does not make a difference in the case of capturing the dynamics of the desired function. Both RNN and MLP have been proven able to simulate the hidden dynamics of systems modelled by ODEs. In contrast to their approach how to infer the network topologies, our approach is based on several attribution-based algorithms. Additionally, our method compares the proposed network topology to the true topology. However, it does not investigate the biological feasibility beyond this comparison.

Chapter 3

Methods

3.1 Data generation

For any project involving machine learning algorithms, the data used is highly important. The quality of the data should be high, and errors due to inaccuracy in measurement should be absent. Missing values are also a common problem that decreases the quality of data. To train our model, we are using gene expression data, and even though real-world data is available, it would not meet the quality criteria. Real-world gene expression data is typically noisy, has a shorter time frame and has few sampled data points. On the basis of this, we are using controlled simulated data. In the next section, we will take a closer look at how the data is simulated.

To generate time series data with the concentration of each species, we use the solution to the system of ODEs as a baseline. The solution to the system of ODEs is solved using the ODE solver, ODEint, from the SciPy library. From the solution, evenly distributed data points from $t_0 = 0$ to $t_n = t_{end}$ are sampled for each species. The t_{end} are dependent on the problem and the time scale might be minutes or hours. To add some variance in the data, Gaussian noise is added to each data point. The noise is sampled from a normal distribution with mean, $\mu = 0$, and standard deviation $\sigma = 0.05 * std_x$, where std_x is the standard deviation for each species involved. The noise added corresponds to measurement noise and not biological noise. It is added after each data point is generated and that way the error is not propagated to the next data point.

$$y_i = f_i(x_t, p) + 0.05\epsilon, \epsilon \ N(0, \sigma^2) \tag{3.1}$$

In machine learning, we need to generate 3 different data sets, one for training, one for validation and one for testing. The training set includes 500 trajectories with t_{end-1} time points, which gives a total sample size of 500*t_{end-1}. Also, as this is a supervised machine learning problem each data set must be divided into features (X) and labels (Y). The features of the data set are the concentration level of all species at time t, and the corresponding labels are the concentration levels at time $t + 1$. Since we are predicting the concentration level of a species in the next time step, the feature data set does not include the last time step as we would be predicting outside of the time frame. Similarly, the labels do not include the first time step as we have no previous time step to make a prediction from.

3.2 Network architecture and training

3.2.1 Neural network

From section 2.1, we remember that in systems biology, modelling a biological function can be done using a system of ODEs. Each ODE will describe the change in the level of concentration for a species, based on the current level of concentration of all species involved in the network of genes, and some defined parameters. In our work, the idea is that each ODE can be modelled using a neural network. Neural networks have proven to be great function approximators. We will use fully connected feed-forward neural networks called multilayer perceptrons (MLP). Consequently, a biological function with n species will need n individual MLPs. Each MLP will take a vector of size n as input and will output a single value vector representing one of the species.

$$N_i(X_t, \theta) = X_{i,t+1} \tag{3.2}$$

Here N_i represents the MLP modelling species i, X_t is a vector containing the concentration of all n species at time t. p is the adjustable parameters of the neural network corresponding to weights and biases. The output $X_{i,t+1}$ is the concentration level of species i at time $t + 1$.

The architecture of each MLP is consisting of 5 layers in total, where 3 of them are hidden layers and one input- and output layer. The number of nodes in the input layer is

dependent on the number of species in the gene regulatory network. The hidden layers are comprised of 128 nodes, and the output layer has 1 node. The output of each node in the network is transformed using the non-linear ReLU activation function. Backpropagation is done using the Adam Optimizer. The network was implemented using the PyTorch library.

3.2.2 Training procedure

An important step in the training procedure is pre-processing the data. Here we do feature scaling by standardizing the features along each dimension. This step helps the models to converge faster in training, and it might help the models' prediction ability. Standardization is defined as:

$$z_i = \frac{x_i - \mu_i}{\sigma_i} \tag{3.3}$$

In equation (3.3) z_i is the scaled features for dimension i. x_i is the unscaled features for dimension i. While μ_i and σ_i are the mean and standard deviation for dimension i, respectively.

The training of each model is done in succession, and therefore each model is trained separately. Each model is trained for 15 epochs, where the training data is randomly shuffled at the start of each epoch. This is done as another precaution to avoid overfitting to the training data. The input data to the model is divided into batches of eight samples per batch. This way, the model updates its parameters after seeing eight samples, which reduces the time used in training. The training was completed using a PC with 2 core CPU and a clock speed of 2.50 GHz. The total training time was around 20 minutes for the first system and around 40 minutes for the second system.

3.2.3 Network inference

The first step in our method of doing network topology inference is to use the different feature attribution algorithms. The algorithms are implemented in a package called Captum, which is developed for neural networks implemented in PyTorch. The feature attribution scores are calculated separately for each model. The input vectors to the

attribution algorithms are the prediction of the models when given the initial values. For all attribution algorithms that require a baseline or reference, the zero vector is chosen. The final attribution scores are the aggregate over all input vectors divided by the L1 norm. This gives all attribution scores within the same range and comparable to each other.

Once we have the attributions scores for all algorithms and all trained models, we can infer possible network topologies. We do this by comparing the attribution scores to a threshold value. If the attribution score is greater than the threshold value, we add an edge from the current feature to the current model we are inspecting. In our case, we want to infer the presence of an edge only, and not the relationship for each edge. Therefore we use the absolute value of the attribution score for comparison. We repeat this procedure for the different algorithms and subsequently get 5 different network topologies for each threshold value. By using a range of different threshold values we are able to create precision-recall plots and ROC curves. The threshold values range from 0 to 1.

From the precision-recall and ROC curves, we choose the best-performing algorithms. We then compare the attribution scores to the results of Sobols sensitivity analysis. Sobols sensitivity analysis is performed using the SALib library in Python. The analysis is performed on the ODE modelling of the system, not the trained machine learning models. We fix all parameters in the ODEs and perform the analysis by permuting the input variables. From the results we only use the first-order sensitivities. Finally, we present the best-performing graph network topology.

3.2.4 Classification problem

For the bi-stable system, we are also interested in if it is possible to train a machine learning model to predict the correct state of the system after 60 hours have passed. Since there are two possible states we can use a binary label and the problem becomes a classification problem. To investigate this, we created a different data set and changed some hyperparameters in our MLP. For this classification, we only need one model.

Similar to the other problem, we need to generate data using simulation. The data consist of the initial values for two species as features, and a binary label indicating if the system is in cell survival or cell death state after 60 hours. The initial values were sampled using Latin hypercube sampling. Latin hypercube sampling is a method for random sampling from a multidimensional distribution. In order to label the data, we

need to simulate the concentration levels for the full time course for all species. Together with the ODEsolver in SciPy and the set of sampled initial values was used to simulate the dynamics of each species. To label the data, we observed the concentration level of species 4, activated caspase 3, casp3*. The concentration level of casp3* determines which state the system has reached. The absence of casp3* means that there is nothing left to start the degradation of cells. Therefore, the system is in a cell death state if $x_4 > 0$ and a cell survival state if $x_4 = 0$. Finally, before training, the data sets were balanced with an equal representation of each state.

For the classification problem, we trained 3 different classifiers. One logistic regression model, one MLP classifier using Scikit-learn and one MLP classifier using PyTorch. The logistic regression uses the standard implementation by scikit-learn with L2-loss as the penalty term. The MLP from scikit-learn consists of 3 layers, one hidden layer with 100 nodes, one input and one output. The activation function used is ReLU and the ADAM optimizer is used for updating the weights. L2 loss is used as the loss function. The self-implemented MLP classifier in PyTorch uses the same activation function and optimizer. The network architecture is different and is comprised of 5 layers, 3 hidden layers with 8 nodes each, 1 input layer and one output layer. Both the logistic regressor and the PyTorch MLP classifier are trained for 100 epochs, while the scikit-learn MLP classifier is trained for 300 epochs.

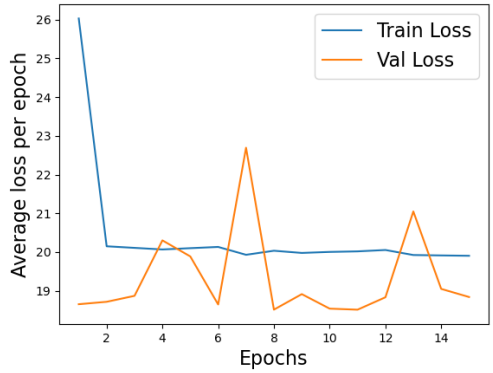
Chapter 4

Results

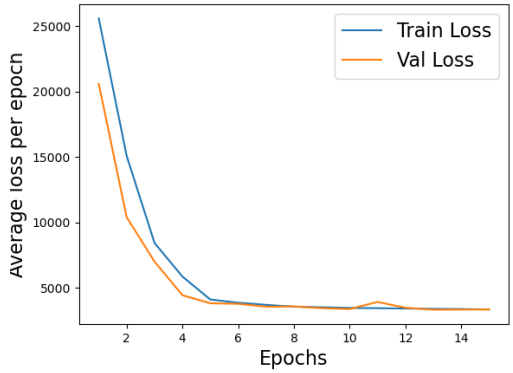
4.1 System with one steady state

In this section, we present the results related to the single steady-state system modelling the ontogeny of naive T-cells. We start by examining the recorded training loss and validation loss for our set of MLPs. Following this, we will look at the models' ability to recreate the dynamics for each species, given only the initial values. We will then present the results of our method for doing network topology inference. We start by looking at the different feature attribution algorithms' performance using precision-recall curves and ROC curves. Next, we look closer at the attribution scores for the individual species and compare them to the first-order sensitivities from Sobol sensitivity analysis. Finally, we look at the best-performing inferred graph topology.

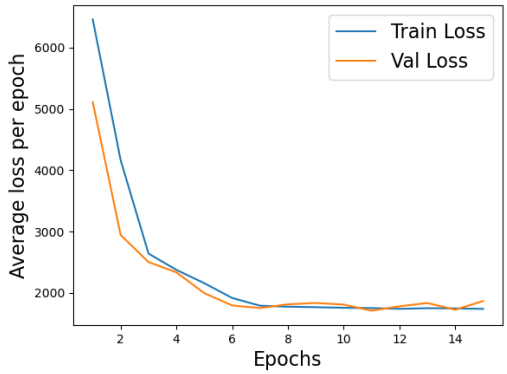
4.1.1 Training results



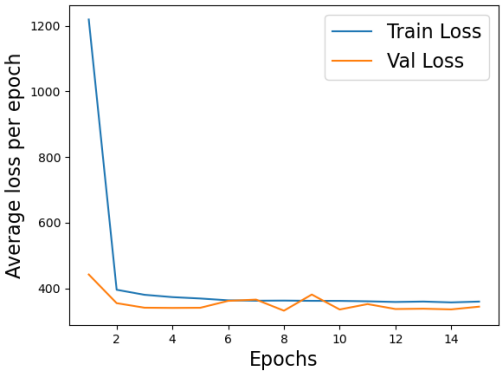
(a) Training and validation loss for species 1



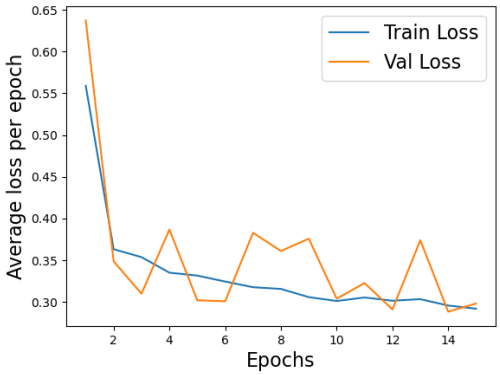
(b) Training and validation loss for species 2



(c) Training and validation loss for species 3



(d) Training and validation loss for species 4

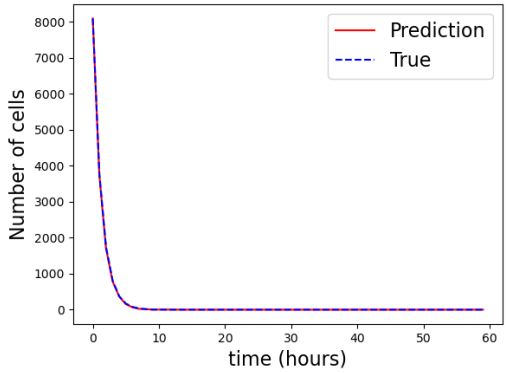


(e) Training and validation loss for species 5

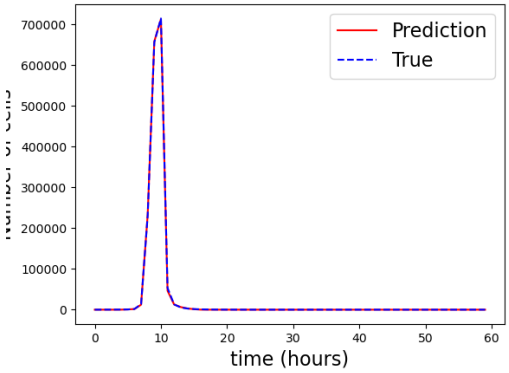
Figure 4.1: Recorded training loss and validation loss for single steady state trained model. Along the x-axis, we find the number of epochs the model has been trained for. The y-axis expresses the average loss recorded for each epoch.

Figure 4.1 show the recorded training loss and validation loss for all models trained. Both losses are shown as the mean recorded for that epoch. We observe that the losses for models 2,3, and 4 all converge to the same value for both training and validation. This indicates that the models are generalizing well to the training and validation data. For models 1 and 5, the validation loss does not converge to the same value as the training loss. The validation loss for model 1 is in general smaller than the training loss, while it is the opposite for model 5, where the training loss is in general smaller than the validation loss. In addition, the training loss for model 5 does not appear to have converged after 15 epochs and is still decreasing.

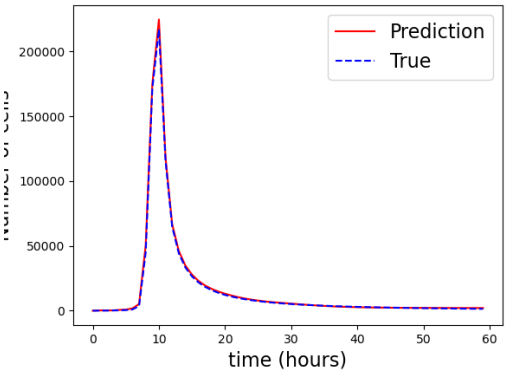
4.1.2 Dynamics



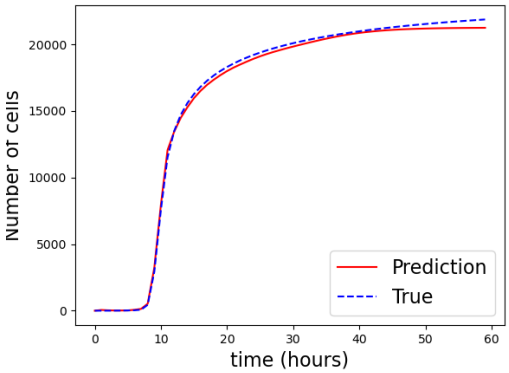
(a) Model 1 prediction for species 1 (Naive Cells)



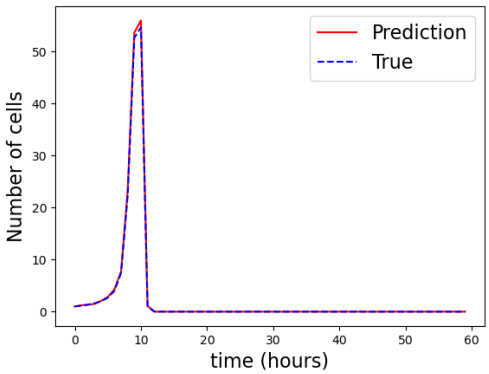
(b) Model 2 prediction for species 2 (Early Effector Cells)



(c) Model 3 prediction for species 3 (Late Effector Cells)



(d) Model 4 prediction for species 4 (Memory Cells)

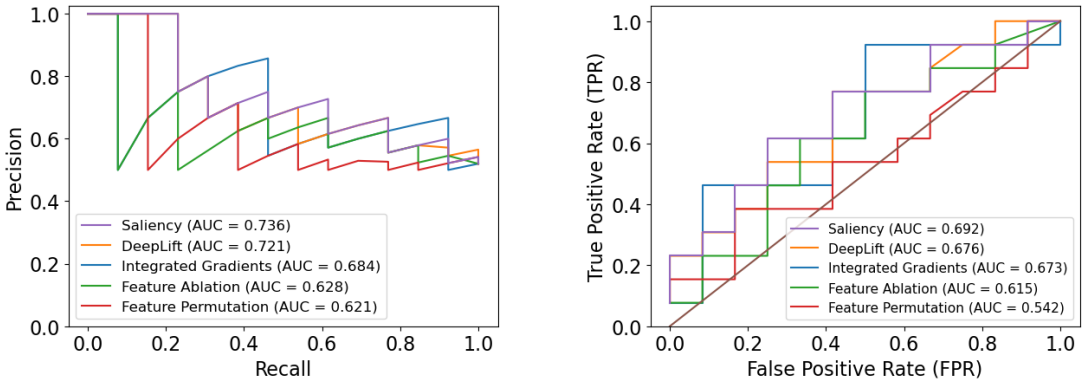


(e) Model 5 prediction for species 5 (Pathogen)

Figure 4.2: Trained models prediction of the dynamics, when only given the initial values. Red whole lines are the models prediction, while the striped blue lines are the "true" simulated trajectories from ODE model.

In figure 4.2 we see the five models predictions of each species dynamics over a 60 hour time period compared to the ODE solution. The trained models are only given the initial value and then use the output as the input value for the next prediction. This is then repeated until the 60-hour mark is reached, where one hour equals one time step. From the figure we can observe that all models are able to capture the dynamics for all species. There is only a slight deviation close to the end for species number 4, shown in d). The true dynamics indicate that the number of memory cells will continue to increase throughout the whole time frame. In contrast, the trained model predicts that a steady state is reached sometime after 50 hours have passed.

4.1.3 Network topology inference



(a) Precision-Recall Curves

(b) ROC Curve

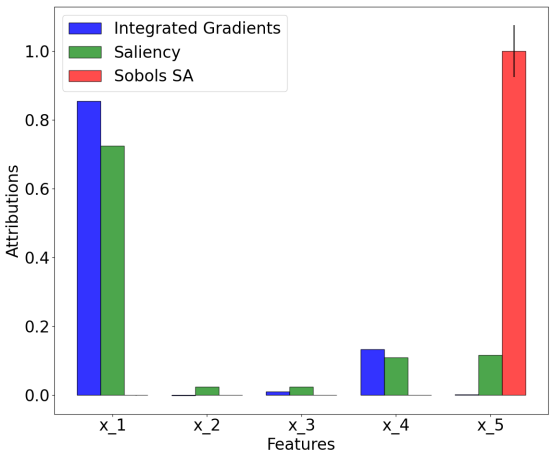
Figure 4.3: Precision-Recall and ROC Curve for Single-Steady state model.

From figure 4.3a we can observe the precision-recall (PR) curves for the different feature attribution algorithms. PR curves are determined by the precision score and the recall score evaluated at different thresholds. The optimal result would be an algorithm achieving both high precision and high recall at the same threshold value. Integrated Gradients and saliency are able to achieve the highest scores of precision and recall.

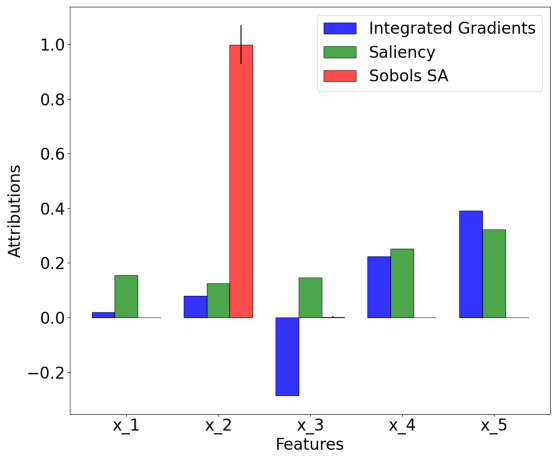
Figure 4.3b shows the Receiver Operating Characteristics (ROC) curves for the feature attribution algorithms. A ROC curve is determined by the true positive rate and false positive rate evaluated at different thresholds. Their performance can then be compared using the area under the curve (AUC) score. A higher AUC score indicates a better performance. From the figure we can observe that saliency is the algorithm that performs best with an AUC score of 0.692. Following closely are the DeepLift and Integrated

Gradients with AUC scores of 0.676 and 0.673 respectively. On the other hand, feature permutation performed close to a random assignment of edges.

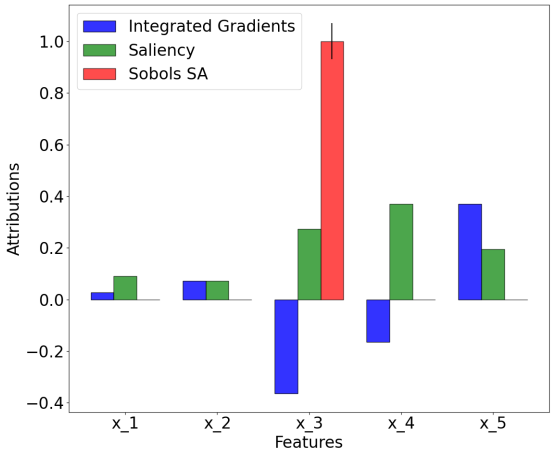
From the results of PR curves and ROC curves, we chose the two best-performing feature attribution algorithms and compared them to the Sobol sensitivity analysis. The comparison shows a clear discrepancy between the sensitivity analysis and the two feature attribution algorithms. Although more in agreement, there is also a difference between Integrated Gradients and saliency algorithms. For comparison, we do not set a specific threshold of when to include the influence. Instead, we look at each species (each bar plot) and its attribution scores separately and compare the attribution scores relative to each other for the same species.



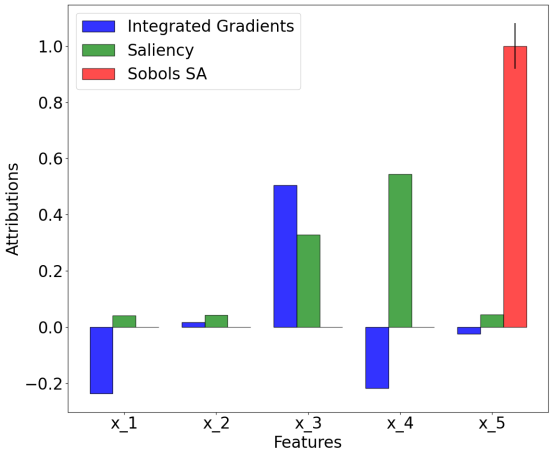
(a) Feature attribution scores for species 1



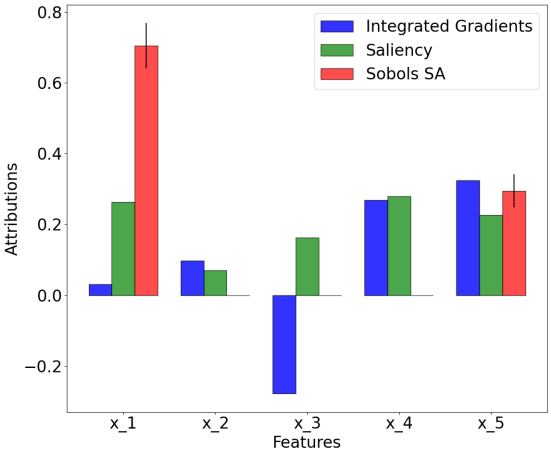
(b) Feature attribution scores for species 2



(c) Feature attribution scores for species 3



(d) Feature attribution scores for species 4



(e) Feature attribution scores for species 5

Figure 4.4: Bar plot of Captum feature attributions and sobol sensitivity analysis. The figure compares the level of influence each of the species has on each other, according to Integrated Gradients and saliency, to the results of the Sobols sensitivity analysis.

Both Integrated Gradients and saliency are confident in a self-influence for species 1,

naive cells. In contrast, the sensitivity analysis shows a clear influence from species 5, pathogen, on the naive cells. Taking into account what we know about the biological system and the mathematical modelling of the system, there is a relation between the naive cells and the pathogen. Pathogen will directly influence the number of naive cells, as it triggers the immune system to start differentiating naive T-cells into effector cells. We also know that the number of naive cells that die, is a product of the death rate and the current number of naive cells. As with earlier results, we do not consider whether the relation is activating or inhibiting, only if it is present or not.

For species 2, early effector cells, saliency does not show any distinct influence from any of the species. The attribution scores are similar for all species, but the scores show the most influence from species 5, pathogen. Integrated Gradients show the most confidence in influence from the pathogen, as well as some influence from species 3, late effector cells. The sensitivity analysis indicates a self-influence for early effector cells. Although the algorithms and the sensitivity analysis indicate different influences, they are all present in the biological system. Pathogen drives the differentiation of naive cells into early effector cells and the proliferation of early effector cells. Late effector cells affect the death rate of early effector cells. And again, early effector cells are self-influenced when it comes to cell degradation.

Integrated Gradients and Sobol sensitivity analysis indicate a self-influence for species 3, late effector cells. In addition, Integrated gradients also show an influence from the pathogen on late effector cells. Saliency attribution scores hint at a relationship between species 4, memory cells and late effector cells. Compared to the mathematical modelling of the system, we only find the self-influence as the only real edge. Using this method, we would fail to find the differentiation of early effector cells into late effector cells.

From figure 4.4 d) we observe that Integrated Gradients suggest an influence from late effector cells on memory cells, while saliency suggests a self-influence. The sensitivity analysis shows an influence from pathogen on memory cells. Both from the mathematical modelling of the system and prior knowledge about the ontogeny of T-cells, an influence from late effector cells on memory cells looks plausible. Self-influence is also present for memory cells. However, there is no direct relationship between the pathogen and memory cells.

Figure 4.4 e) show for the first time that the sensitivity analysis indicates more than one influencer for pathogen. Sensitivity analysis indicates both self-influence and influence from naive cells. Saliency does not regard one of the species in particular to have

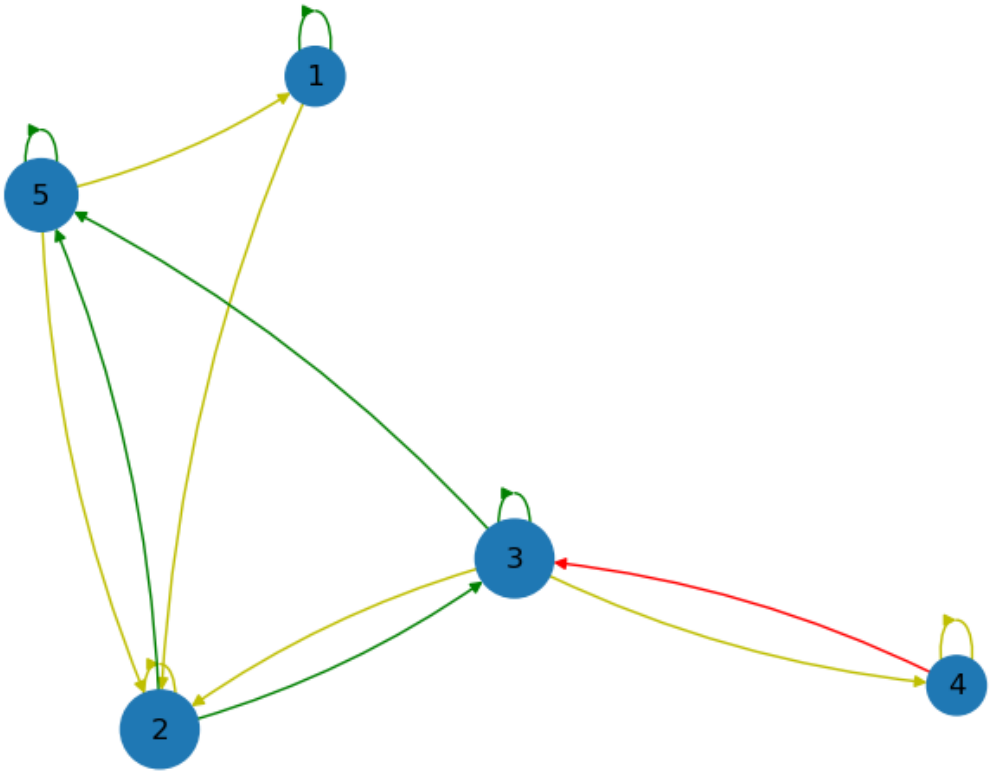


Figure 4.5: Inferred topology for the single steady state system. Green edges are correct edges, red are wrong, and yellow are missing edges.

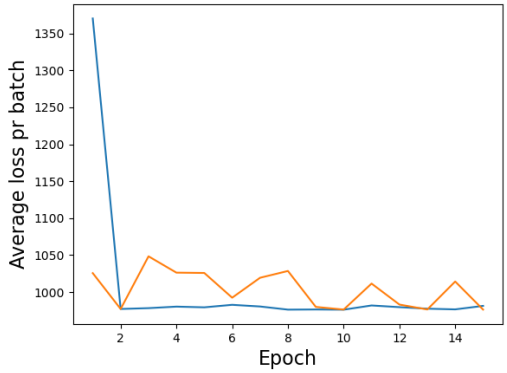
attribution towards pathogen. While Integrated Gradients favors self-influence and influence from both early and late effector cells. However, all of the attribution scores would need a lower threshold to be counted. Self-influence, early effector and late effector cells are all present relationships in the modelling of the system.

Figure 4.5 show an inferred topology with missing edges. The correctly inferred edges are colored green, while the wrongly guessed edges are red. The yellow edges indicate edges present in the true graph, but missing in the inferred graph topology. The observed graph topology was the graph with the highest precision with a recall above 0.4. We observe only one wrongly inferred edge, seven missing edges, and five correctly inferred edges. The precision of the inferred topology is 0.857 and recall of 0.416. The feature attribution algorithm used to make this topology was integrated gradients.

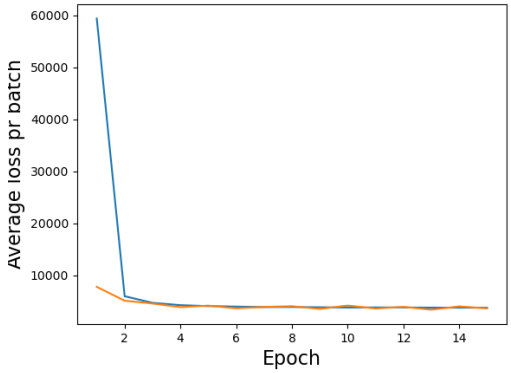
4.2 Bi-stable system

In this section, we will examine the results for the set of models trained on simulated gene expression data from the cell apoptosis system. We trained one set of models for each of the two states the models can be in. We start by looking at the training and validation loss acquired during training, then we show the models ability to recreate the dynamics of the system. We then move on to the results related to the process of inferring possible topologies.

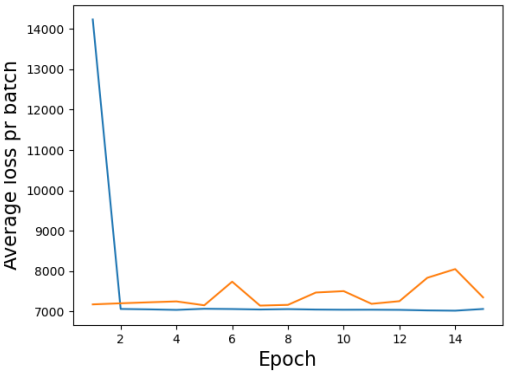
4.2.1 Training results



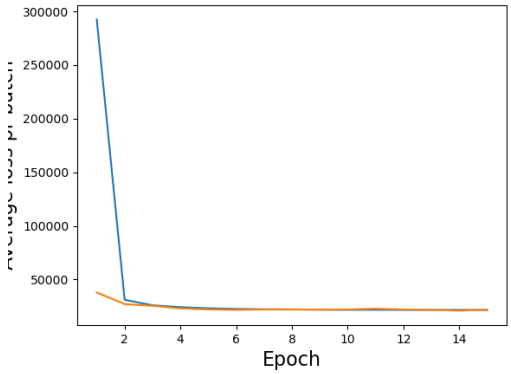
(a) Training and validation loss for species 1



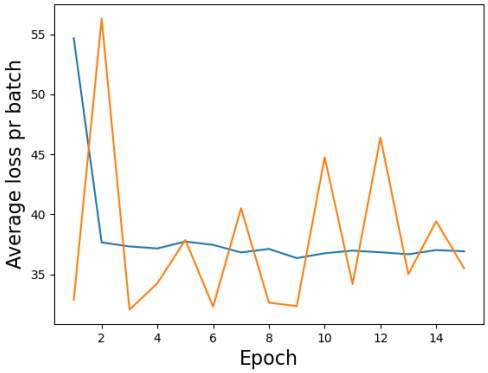
(b) Training and validation loss for species 2



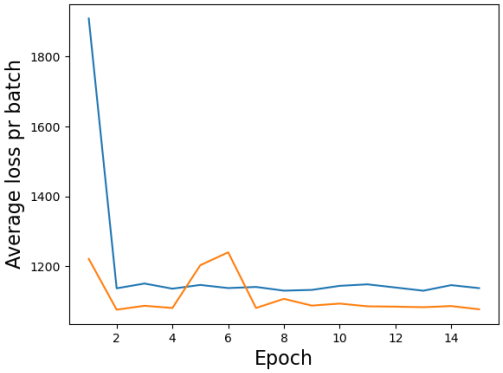
(c) Training and validation loss for species 3



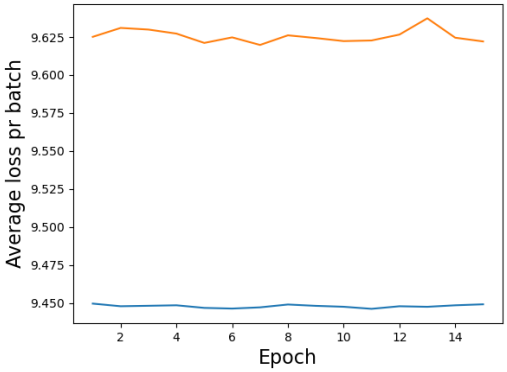
(d) Training and validation loss for species 4



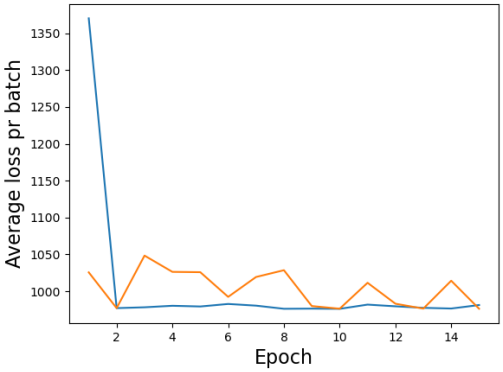
(e) Training and validation loss for species 5



(f) Training and validation loss for species 6



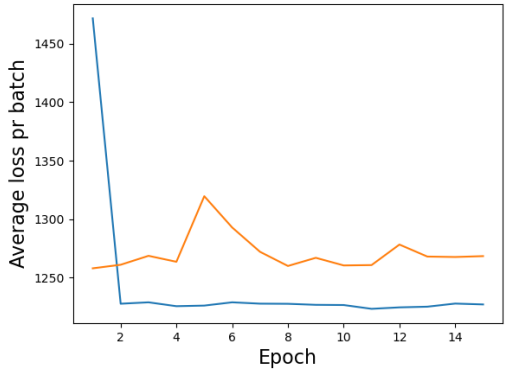
(g) Training and validation loss for species 7



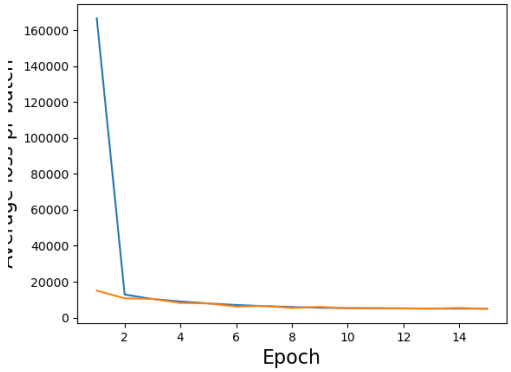
(h) Training and validation loss for species 8

Figure 4.6: Training and validation loss for model trained on cell death data.

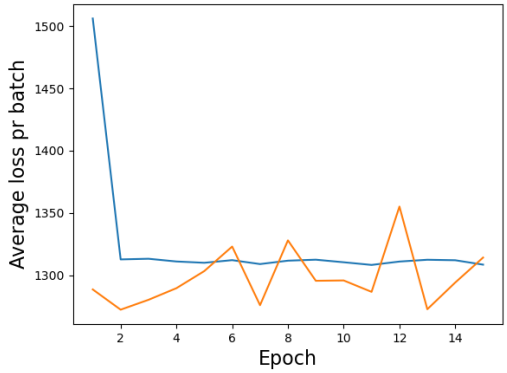
From figure 4.6 we can observe the training and validation loss for the MLP models trained on cell death data. The training and validation loss for species 2,3, and 8 converge to the same level. The results also show that for the remaining species, the training loss converges, which would indicate that the models are adapting well to the training data. The validation loss for species 7 converges to a higher level than the training loss, which could imply the model overfitting to the training data and generalising poorly on the validation data. However, in this case, the relative error, $d = \frac{v_{loss} - t_{loss}}{t_{loss}}$, is quite small, $d \approx 0.02$, and overfitting might not be the case. The validation loss for species 1, 3, and 5 does not converge. Lastly, we observe the validation loss for species 6 converges to a lower level than the training loss. This might happen for a number of reasons, such as less variance and an "easier" validation set than a training set.



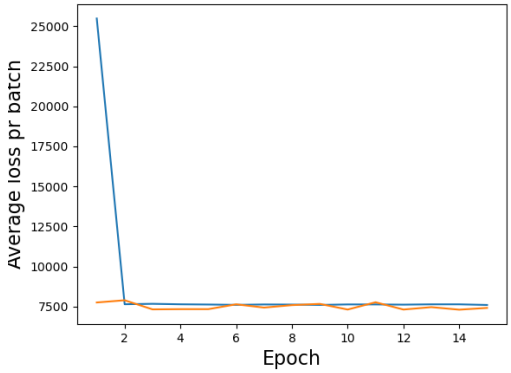
(a) Training and validation loss for species 1



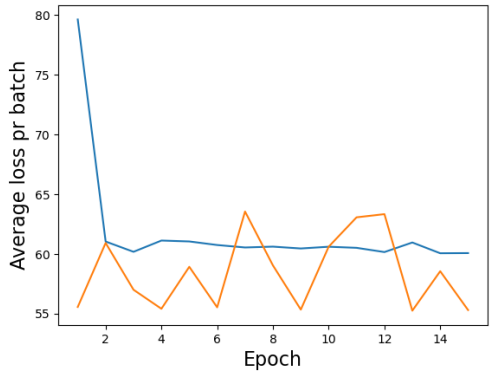
(b) Training and validation loss for species 2



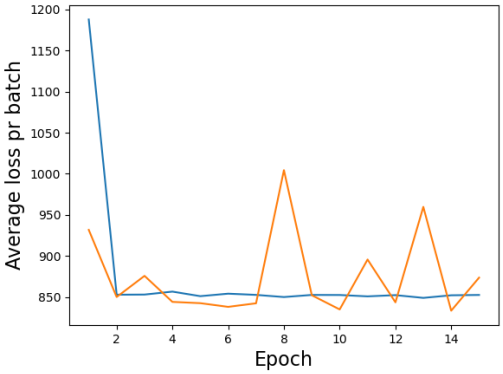
(c) Training and validation loss for species 3



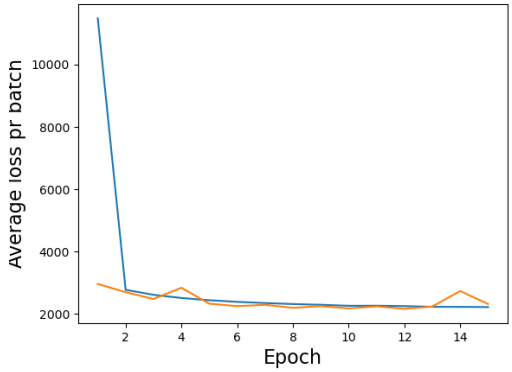
(d) Training and validation loss for species 4



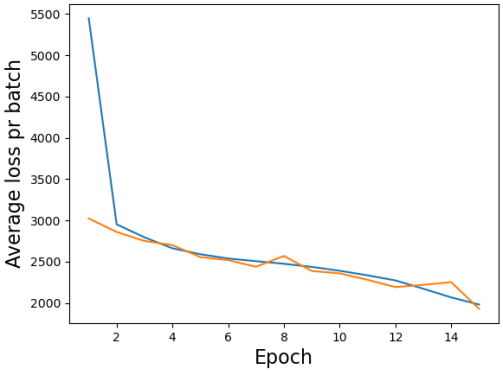
(e) Training and validation loss for species 5



(f) Training and validation loss for species 6



(g) Training and validation loss for species 7

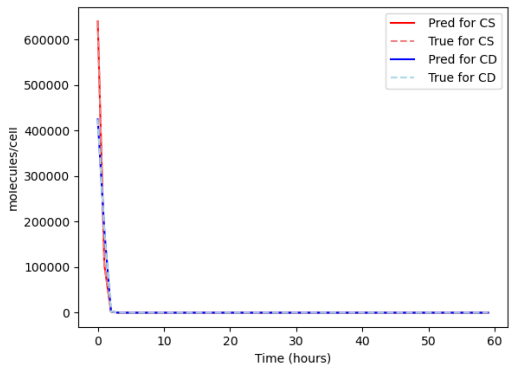


(h) Training and validation loss for species 8

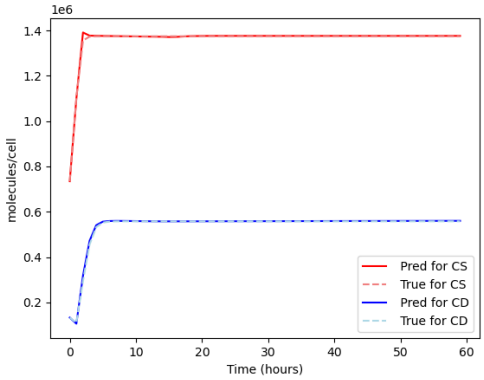
Figure 4.7: Training and validation loss for model trained on cell survival data.

Figure 4.7 shows the recorded training and validation loss for the MLP models trained on cell survival data. For species 2,4, and 7, both training and validation loss converge to the same level. In contrast, the training and validation loss for species 1,3,5,6 and 8 does not converge to the same level. We note that the training loss converges for all of these species. The validation loss converges to a higher level than the training loss for species 1 and 8, while the remaining species validation loss does not converge to a specific level.

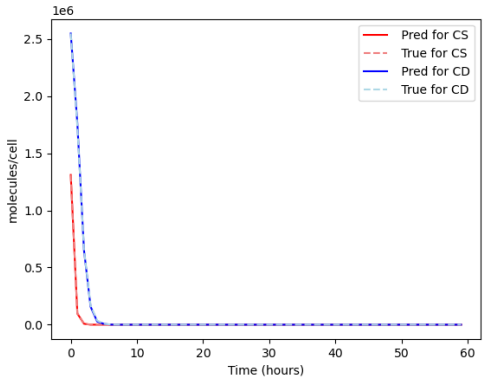
4.2.2 Dynamics



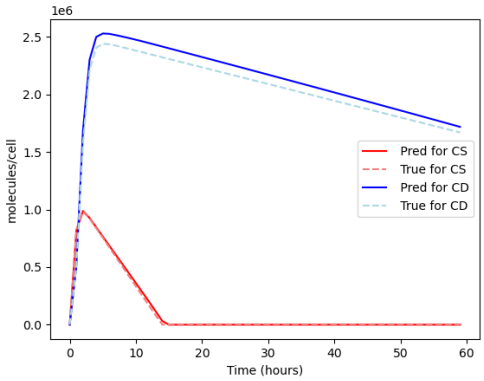
(a) Prediction values vs true values for species 1



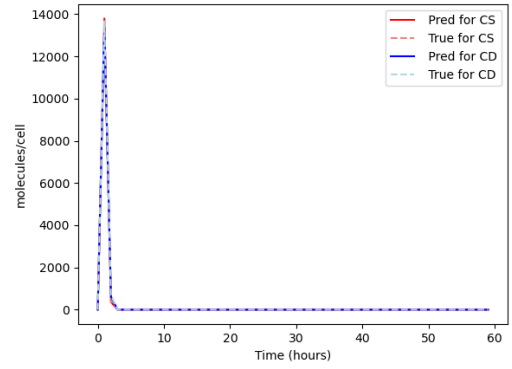
(b) Prediction values vs true values for species 2



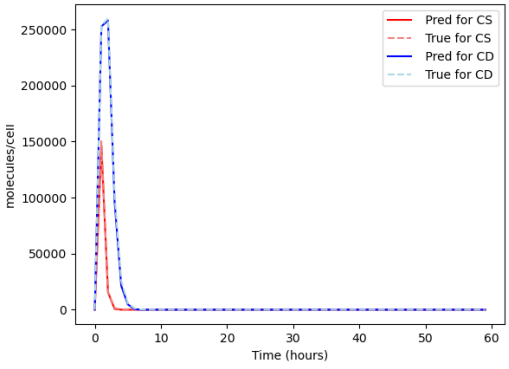
(c) Prediction values vs true values for species 3



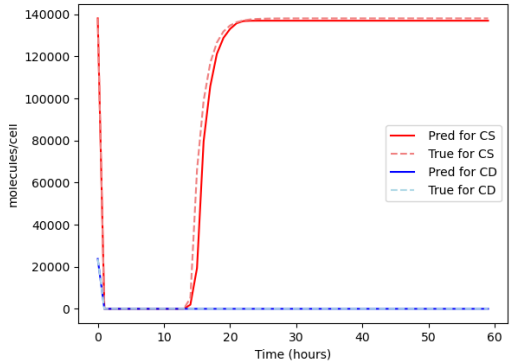
(d) Prediction values vs true values for species 4



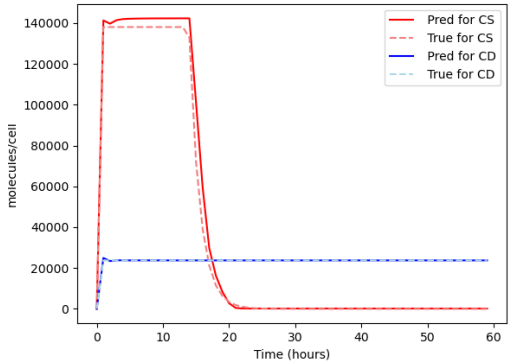
(e) Prediction values vs true values for species 5



(f) Prediction values vs true values for species 6



(g) Prediction values vs true values for species 7

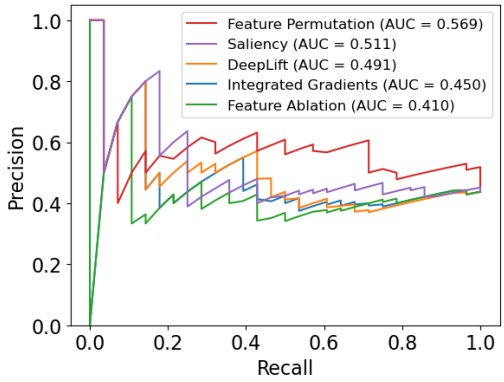


(h) Prediction values vs true values for species 8

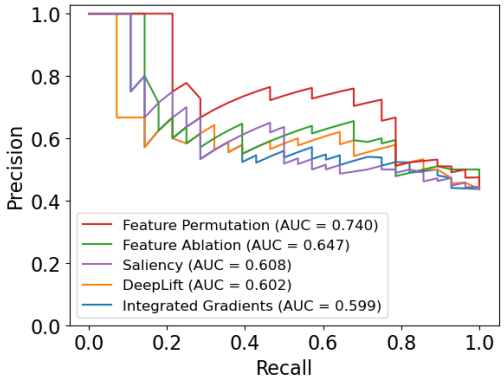
Figure 4.8: Predicted Dynamics Compared To True Dynamics For Cell Apoptosis Models.

The results show that the set of 8 MLP models is, in general, able to capture the dynamics of the biological system for both states. For the models trained on cell survival data, there is a slight deviation in the prediction for species 7 and 8. The trained model for species 7 is slightly off in predicting the increase in the concentration level. Figure 4.8 shows that the trained model slightly overestimates the concentration level of species 8 after the initial increase. The model also fails to capture the "smooth" start of the decrease. In the prediction of the dynamics for cell death, the models capture the dynamics close to perfect. The exception is for species 4, where the trained model overestimates the concentration level from five hours into the process, until the end.

4.2.3 Network topology inference

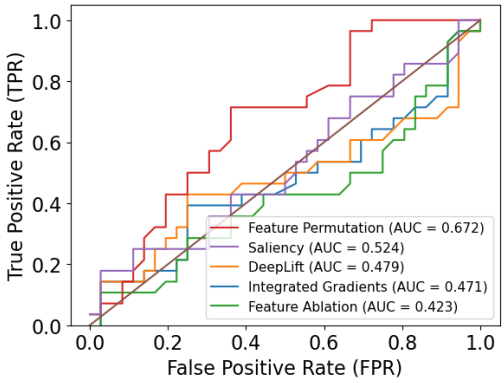


(a) Precision-Recall curves for cell death

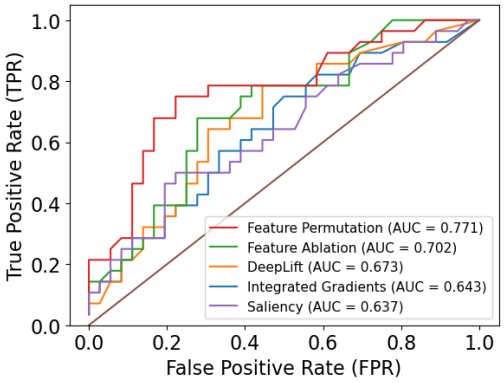


(b) Precision-Recall curves for cell survival

Figure 4.9: Precision-Recall Curves.



(a) ROC-curves for cell death



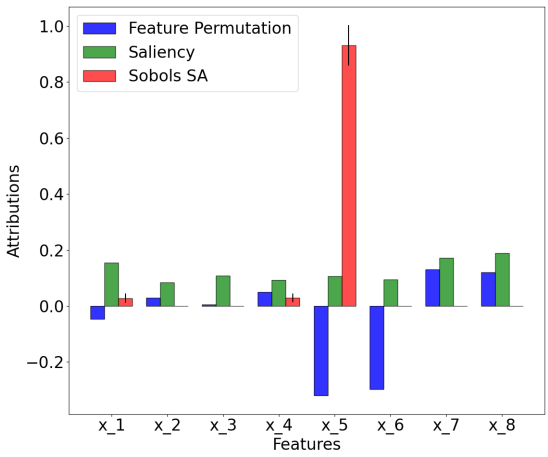
(b) ROC-curves for cell survival

Figure 4.10: ROC Curves.

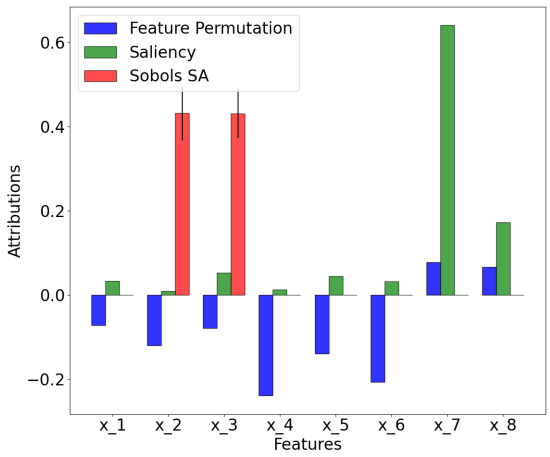
From both the PR-curves and ROC-curves in Figures 4.9 and 4.10, we can observe a clear distinction in performance for the models trained on the two different states. For the models trained on cell death data, the different attribution algorithms did not achieve high precision and recall. The results from the ROC-curve also shows a bad performance, as several of the attribution algorithms achieved a lower AUC score than a random selection of edges. The best performing network topologies were inferred using saliency and feature permutation.

Better performance was recorded by attribution algorithms applied to the models trained on cell survival data. Especially the graphs inferred from the attribution scores

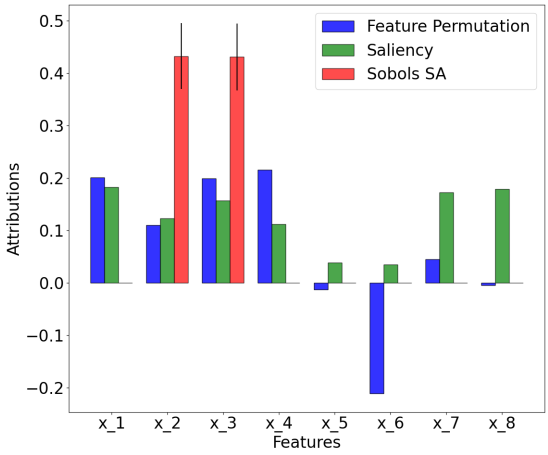
of feature permutation achieved a reasonable high precision and recall scores. Feature permutation also recorded the highest AUC score with AUC = 0.771. The other perturbation method, feature ablation also performed well in terms of the AUC score.



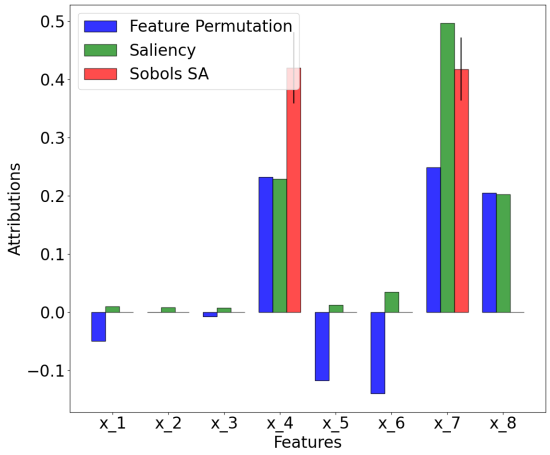
(a) Feature attribution scores and first-order sensitivities for species 1



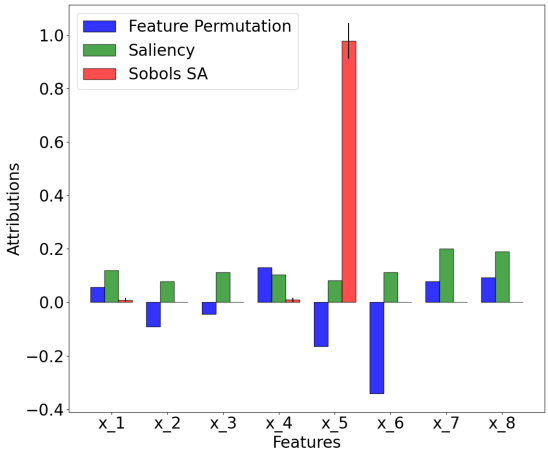
(b) Feature attribution scores and first-order sensitivities for species 2



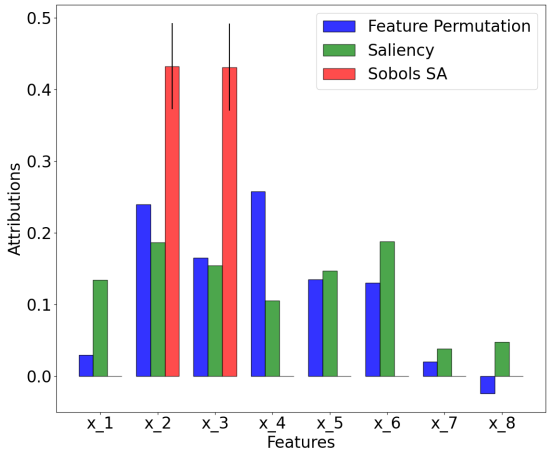
(c) Feature attribution scores and first-order sensitivities for species 3



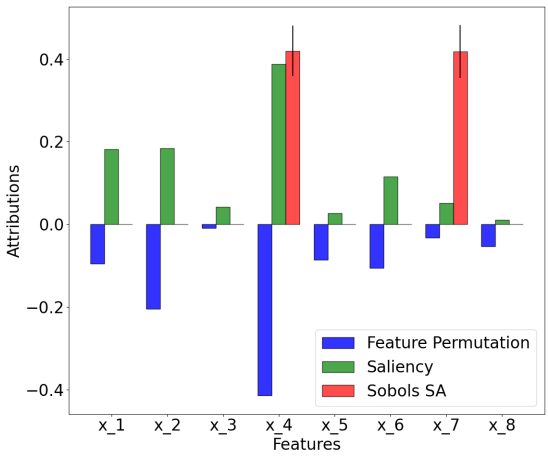
(d) Feature attribution scores and first-order sensitivities for species 4



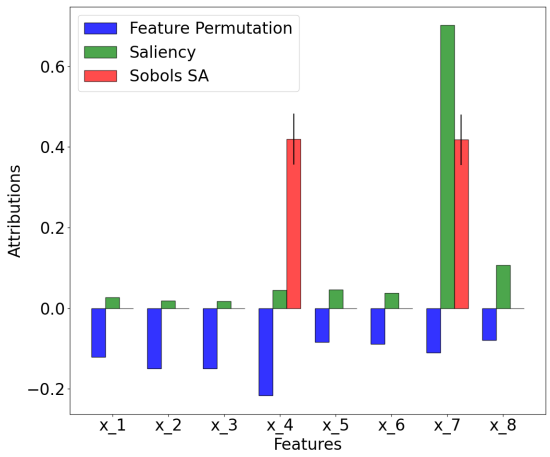
(e) Feature attribution scores and first-order sensitivities for species 5



(f) Feature attribution scores and first-order sensitivities for species 6



(g) Feature attribution scores and first-order sensitivities for species 7



(h) Feature attribution scores and first-order sensitivities for species 8

Figure 4.11: Bar plot of sensitivity analysis, saliency and feature permutation.

The bar plots in figure 4.11 show the different attribution scores for the two best-performing attribution algorithms. The attribution scores were obtained for the models trained on cell death data. Their attribution values are compared to the first-order sensitivity computed using Sobol sensitivity analysis. The results show a clear contrast between the results from the sensitivity analysis and the attribution scores computed for all species. In addition, the two attribution algorithms disagree for a number of species.

The first bar plot shows that the saliency did not favor any influence from the other species on casp8(x_1). The attributions scores for feature permutation indicated a similar influence from casp8-casp3*(x_5) and casp8*-casp3(x_6) onto casp8(x_1). The first-order sensitivity showed a clear influence from casp8-casp3*(x_5) onto casp8(x_1), and smaller influences from casp3*(x_4), and self-influence. From the mathematical modelling of the system, there exists a relationship between casp8(x_1) and itself, casp3*(x_4) and casp8-casp3*(x_5). This relationship comes from casp8(x_1) being an initiator caspase and while not being activated it can form a complex casp8-casp3*(x_5) with casp3*(x_4). We do not find any evidence to support any influence from casp8*-casp3(x_6) which feature permutation indicated.

For the second species casp8*(x_2), the results in the second bar plot showed a disagreement between all three methods. Saliency expressed a singular influence from XIAP(x_7), while feature permutation signalled influence from both casp3*(x_4) and casp8*-casp3(x_6). Sensitivity analysis seemed confident in an equal influence from casp3(x_3) and a self-influence. The mathematical model of the system exhibits an influence from species casp8-casp3*(x_5), casp3(x_3), casp8*-casp3(x_6) and self-influence on casp8*(x_2). Casp8*(x_2)

is activated when casp8-casp3*(x_5) dissociates, and casp8*(x_2) associates with inactive casp3(x_3) to form the complex casp8*-casp3(x_6). Saliency did not capture any of the relationships, while feature permutation captured one correct relationship out of two predicted.

The third bar plot shows the attribution scores for the third species casp3(x_3). Attribution scores from saliency did not indicate a strong influence from any of the other species. However, they showed that casp8-casp3*(x_5) and casp8*-casp3(x_6) have less influence than the rest. Feature permutation also did not show any strong influence from the other species, but an influence from casp8-casp3*(x_5), XIAP(x_7) and XIAP-casp3*(x_8) seemed unlikely, based on the attribution scores. The first-order sensitivities expressed an equal influence from casp8*(x_2) and a self-influence. From prior knowledge about the regulatory system, casp3(x_3) is only influenced by casp8*(x_2), casp8*-casp3(x_6)and itself. This comes from casp3(x_3) being activated by casp8*(x_2) when forming a complex casp8*-casp3(x_6). Both feature attribution algorithms performed poorly here, feature permutation correctly guessed all influences, but also guessed two non-existent relationships. Saliency on the other hand failed to recognize the influence of casp8*-casp3(x_6).

The results for species 4, casp3*(x_4), show the strongest agreement between the attribution scores and first-order sensitivities. All three favored a self-influence for casp3*(x_4). Saliency also expressed additional influence from XIAP(x_7) and XIAP-casp3*(x_8), with XIAP(x_7) being the most likely. Feature permutation also expressed influence from XIAP(x_7) and XIAP-casp3*(x_8), but also showed attribution from casp8-casp3*(x_5) and casp8*-casp3(x_6) is greater than the rest of the species. The first-order sensitivities expressed the additional influence by XIAP(x_7). From the modelling of the system, casp3*(x_4) is affected by casp8*-casp3(x_6), casp8(x_1), casp8-casp3*(x_5), XIAP(x_7), XIAP-casp3*(x_8), and itself. Casp3*(x_4) is one of the central components of the system as it is responsible for the degradation of the cell causing cell death. Casp3*(x_4) can either combine with casp8(x_1) to form casp8-casp3*(x_5) or be inhibited by XIAP(X_7) forming the intermediate XIAP-casp3*(x_8) complex where tagging for ubiquitination happen. Casp3*(x_4) is also a product of the dissociation of casp8*-casp3(x_6), where casp3(x_3) is activated. Feature permutation correctly guessed 5 out of 6 influences. However, the attribution scores for casp8-casp3*(x_5) and casp8*-casp3(x_6) were relatively low and a threshold including these might include a number of false relationships as well. Saliency correctly guessed 3 out of 6 influences.

The computed attribution scores for both attribution algorithms did not show any particular influence from the other species on species 5, casp8-casp3*(x_5). The only

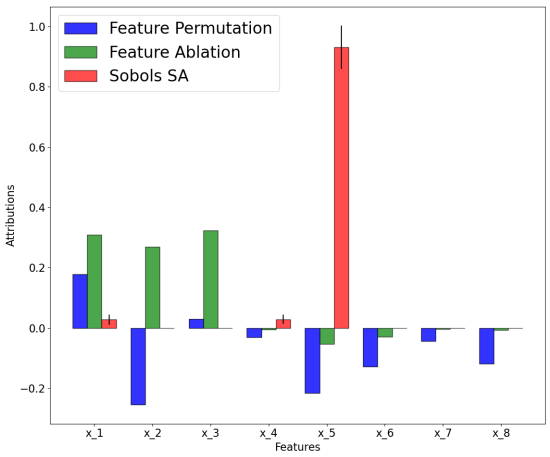
exception is the attribution score from feature permutation for casp8*-casp3(x_6). The first-order sensitivity showed a self-influence only. The mathematical modelling of the system shows a relationship between casp8-casp3*(x_5) and casp8(x_1), casp3*(x_4) and itself. These relationships stem from casp8-casp3*(x_5) complex being formed by the association of casp8(x_1) and casp3*(x_4). Both feature attribution algorithms performed poorly and neither recognized any of the relationships in the regulatory system.

The sixth bar plot shows that saliency again did not favor any attribution from the other species onto species 6, casp8*-casp3(x_6). The attribution scores were lowest for XIAP(x_7) and XIAP-casp3*(x_8), while casp8*-casp3(x_6) and casp8*(x_2) had the highest. Feature permutation and sensitivity analysis both regarded a larger attribution from casp8*(x_2) to casp8*-casp3(x_6). Featured permutation also indicated an influence from casp3*(x_4), while sensitivity analysis showed an influence from casp8*(x_2) and casp3(x_3). From prior knowledge about the regulatory system, casp8*-casp3(x_6) is influenced by casp8*(x_2), casp3(x_3) and itself. These relationships come from casp8*-casp3(x_6) being formed as an intermediate complex where casp3(x_3) gets activated when associated with casp8*(x_2). Both saliency and feature permutation indicated the influence by casp8*(x_2), but did not manage to separate the attribution of casp3(x_3) from many of the other species. Feature permutation also expressed an influence from casp3*(x_4), but this relationship does not exist. Saliency correctly guessed self-influence as well.

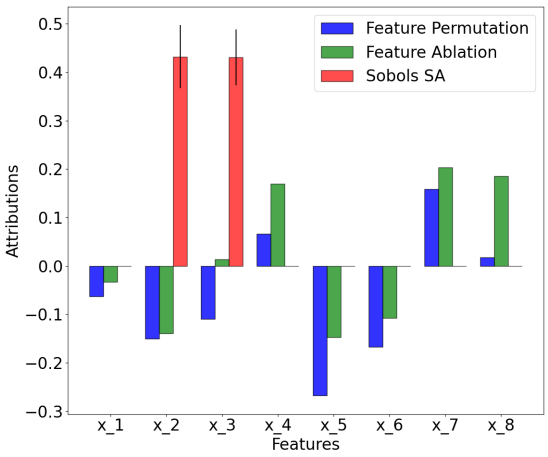
The bar plot for species 7, XIAP(x_7) shows a clear discrepancy between the two attribution algorithms in terms of positive and negative attribution. However, the absolute values of the attribution were mostly similar. All three methods expressed an influence from casp3*(x_4) onto XIAP(x_7). The sensitivity analysis indicated a self-influence as well. The mathematical modelling of the system showed influence on XIAP(x_7) by casp3*(x_4) XIAP-casp3*(x_8) and self-influence. XIAP(x_7) is pro-survival of the cell and does this by forming an XIAP-casp3*(x_8) complex with casp3*(x_4). XIAP(x_7) tags casp3*(x_4) for ubiquitination, a process which eliminates unwanted proteins. None of the feature attribution algorithms correctly guessed either the self-influence or XIAP-casp3*(x_8) influence. Saliency and feature permutation both correctly scored the attribution from casp3*(x_4). However, both of them also indicated an influence from casp8*(x_2), which is wrong. Saliency also expressed casp8(x_1) as a potential influence, which is also wrong.

The last bar plot shows that saliency expressed a strong influence from XIAP(x_7) onto species 8, XIAP-casp3*(x_8). Saliency indicated that all other species had a small or no influence. Feature permutation did not express any influence in particular, although there was a slightly higher attribution score for casp3*(x_4). The first-order sensitivities revealed

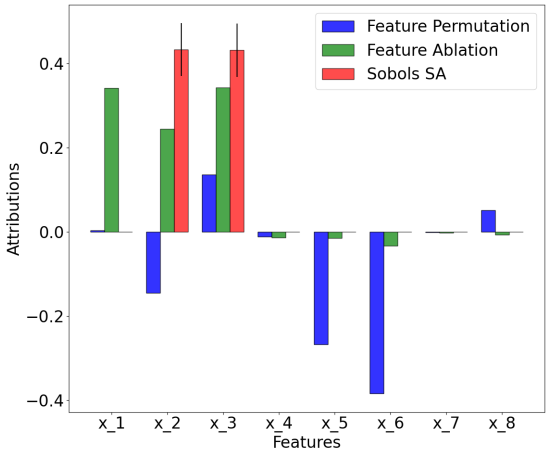
an equal attribution from casp3*(x_4) and XIAP(x_7). From the mathematical modelling of the system, we know that XIAP-casp3*(x_8) is regulated by casp3*(x_4), XIAP(x_7) and itself. As described in the paragraph above, XIAP-casp3*(x_8) is the complex where casp3*(x_4) is marked for degradation. Saliency correctly indicated influence by XIAP(x_7), but failed to show influence from casp3*(x_4) and self-influence. Feature permutation on the other hand, barely indicated a higher influence by casp3*(x_4) than any of the other species and failed to recognize the self-influence and influence by XIAP(x_7).



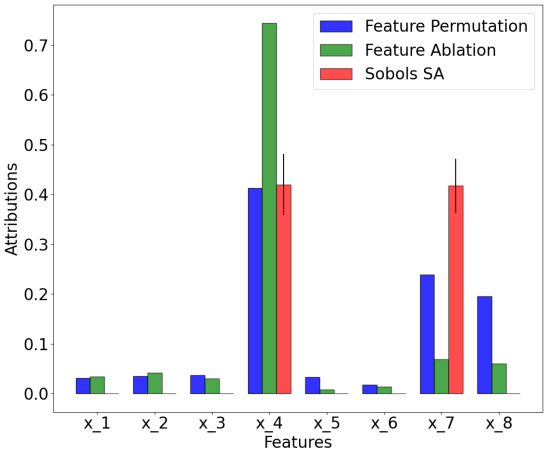
(a) Feature attribution scores and first-order sensitivities for species 1



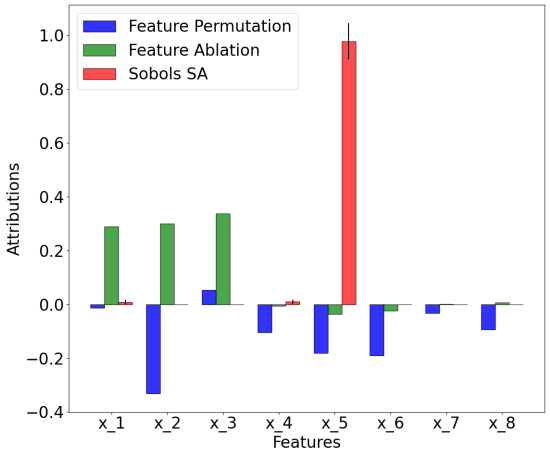
(b) Feature attribution scores and first-order sensitivities for species 2



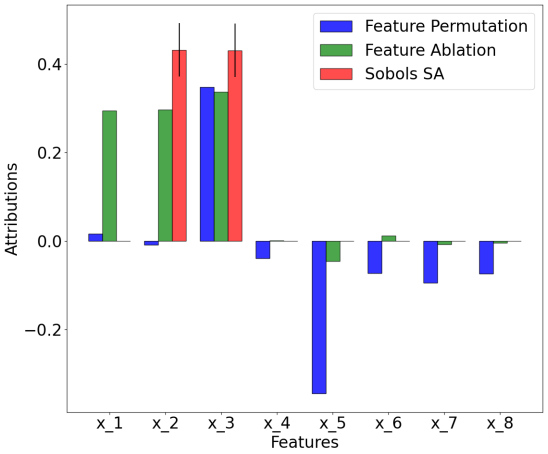
(c) Feature attribution scores and first-order sensitivities for species 3



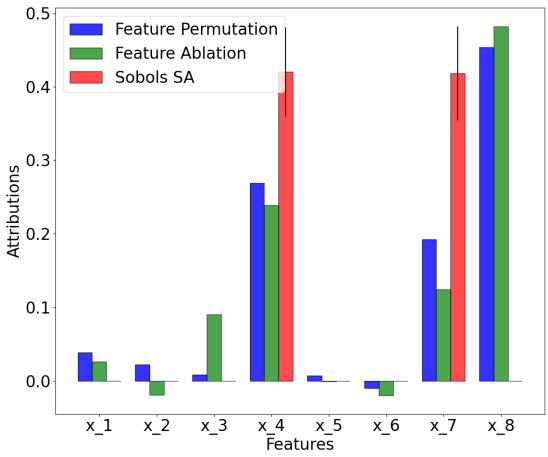
(d) Feature attribution scores and first-order sensitivities for species 4



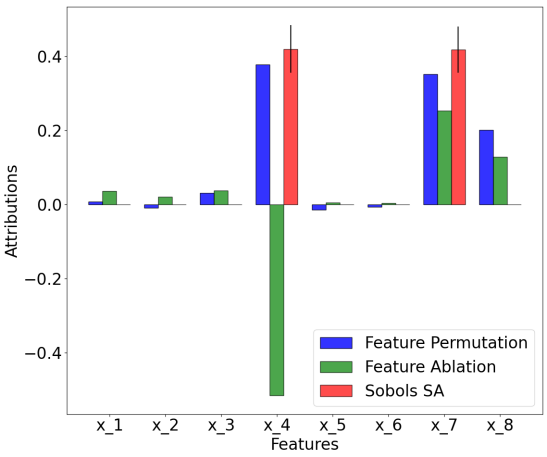
(e) Feature attribution scores and first-order sensitivities for species 5



(f) Feature attribution scores and first-order sensitivities for species 6



(g) Feature attribution scores and first-order sensitivities for species 7



(h) Feature attribution scores and first-order sensitivities for species 8

Figure 4.12: Bar plot of sensitivity analysis, feature ablation and feature permutation.

Figure 4.12 shows the attribution scores computed for the models trained on cell survival data. The attribution scores are again compared to the first-order sensitivities computed from the ODE model of the system. Both cell survival state and cell death state are modelled using the same ODE model. Therefore, the sensitivities will be the same as in figure 4.11.

The first bar plot shows the attribution scores for species 1, casp8(x_1). We can observe that feature ablation expressed an influence from casp8*(x_2) and casp3(x_3), and a self-influence. Feature permutation had the highest attribution scores for casp8(x_1), casp8*(x_2), and casp8-casp3*(x_5). Feature ablation had close to zero attribution scores for casp3*(x_4), XIAP(x_7), and XIAP-casp3*(x_8). Compared to the real relationships described by the mathematical modelling of the system, the self-influence is recognized by both algorithms. In contrast, none of them recognized the influence by casp3*(x_4). Feature permutation correctly expressed influence from casp8-casp3*(x_5). Both wrongly expressed that casp8*(x_2) was an influence. The influence by casp3(x_3), indicated by feature ablation, is also not found in the biological system.

From the bar plot with attribution scores for species 2, casp8*(x_2), we can observe that feature permutation expressed an influence from casp8-casp3*(x_5), while there was almost a non-existent influence from XIAP-casp3*(x_8). Feature permutation also expressed an almost equal attribution from casp3(x_3), casp8*-casp3(x_6), XIAP(x_7) and itself. Feature ablation did not signal a high attribution from any of the species, while the attribution score for casp3(x_3) was close to zero. According to the mathematical modelling of the

system, casp8*(x_2) is influenced by casp3(x_3), casp8-casp3*(x_5), casp8*-casp3(x_6) and itself. Feature permutation correctly indicated all influences, but also included a non-existent influence from XIAP(x_7). Feature ablation on the other hand failed to express any of the relationships for casp8*(x_2). The highest attribution score was by XIAP(x_7) and XIAP-casp3*(x_8), which is non-existent in the regulatory system.

The feature permutation attribution scores calculated for species 3, casp3(x_3), were the highest for species casp8-casp3*(x_5) and casp8*-casp3(x_6), while they were almost zero for casp8(x_1), 4 and XIAP(x_7). Feature ablation expressed a self-influence and influence from casp8(x_1) and casp8*(x_2). The other species had close to zero influence based on the attribution scores. The attribution scores for feature ablation did to some degree concur with the first-order sensitivity. From the mathematical modelling, casp3(x_3) is only influenced by casp8*(x_2), casp8*-casp3(x_6) and itself. Feature permutation correctly expressed all of these influences, but the influences by casp8*(x_2) and self-influence receive a lower attribution score than the non-existent influence from casp8-casp3*(x_5). Feature ablation recognized both self-influence and the influence by casp8*(x_2), but fail to recognize influence from casp8*-casp3(x_6). In addition, feature ablation also expressed influence from casp8*(x_2), which is not a real influence.

For species 4,casp3*(x_4), feature ablation strongly signalled self-influence. The rest of the attribution scores were relatively low. Feature permutation also expressed self-influence, although not as strongly as feature ablation. XIAP(x_7) and XIAP-casp3*(x_8) were the only remaining species to show influence towards casp3*(x_4). From the modelling of the system, casp3*(x_4) is affected by casp8*-casp3(x_6), casp8(x_1), casp8-casp3*(x_5), XIAP(x_7), XIAP-casp3*(x_8), and itself. Feature permutation recognized 3 out of the 6 relationships, while feature ablation recognized only self-influence.

The fifth bar plot shows a discrepancy between the two perturbation-based algorithms. Both agreed on the influence from casp8*(x_2) on casp8-casp3*(x_5) and the absence of influence from XIAP(x_7). Feature ablation expressed influence from casp8(x_1) and casp3(x_3), in addition to the mentioned influence. Feature permutation had increased attribution scores for casp8-casp3*(x_5) and casp8*-casp3(x_6) compared to the rest. The mathematical modelling of the system shows a relationship between casp8-casp3*(x_5) and casp8(x_1), casp3*(x_4) and itself. Feature ablation correctly expressed influence from casp8(x_1), but failed for the other two. Additionally, it incorrectly expressed influence from casp8*(x_2) and casp3(x_3).

The sixth bar plot shows the attribution scores for species 6, casp8*-casp3(x_6). All three agree on relatively high attribution for casp3(x_3), while only feature ablation and

the first-order sensitivity values an attribution from casp8*(x_2). Feature ablation also indicated an influence from casp8(x_1). In contrast, feature permutation expressed an influence from casp8-casp3*(x_5). From prior knowledge about the regulatory system, casp8*-casp3(x_6) is influenced by casp8*(x_2), casp3(x_3) and itself. Feature ablation correctly expressed influence by casp8*(x_2) and casp3(x_3), but it failed to express self-influence. Feature ablation also incorrectly indicated an influence by casp8(x_1). Feature permutation only recognized influence from casp3(x_3) and incorrectly expressed influence from casp8-casp3*(x_5).

For species 7, XIAP(x_7), almost all three agreed on high attribution from casp3*(x_4), XIAP-casp3*(x_8) and itself. The first-order sensitivity differed from the other by not showing any contribution from XIAP-casp3*(x_8) towards XIAP(x_7). The mathematical modelling of the system showed influence on XIAP(x_7) by casp3*(x_4), XIAP-casp3*(x_8) and self-influence. Feature permutation and feature ablation both recognized all three influences correctly by giving them the highest attribution scores.

The last bar plot shows the attribution scores for species 8, XIAP-casp3*(x_8). Both feature attribution algorithms agree on the attribution scores for most species. The highest scores were for casp3*(x_4), XIAP(x_7), and self-influence. Sensitivity analysis results showed most attribution from casp3*(x_4) and XIAP(x_7) only. From the mathematical modelling of the system, we know that XIAP-casp3*(x_8) is regulated by casp3*(x_4), XIAP(x_7) and itself. Again, both algorithms recognized the three influences by giving them the highest attribution scores.

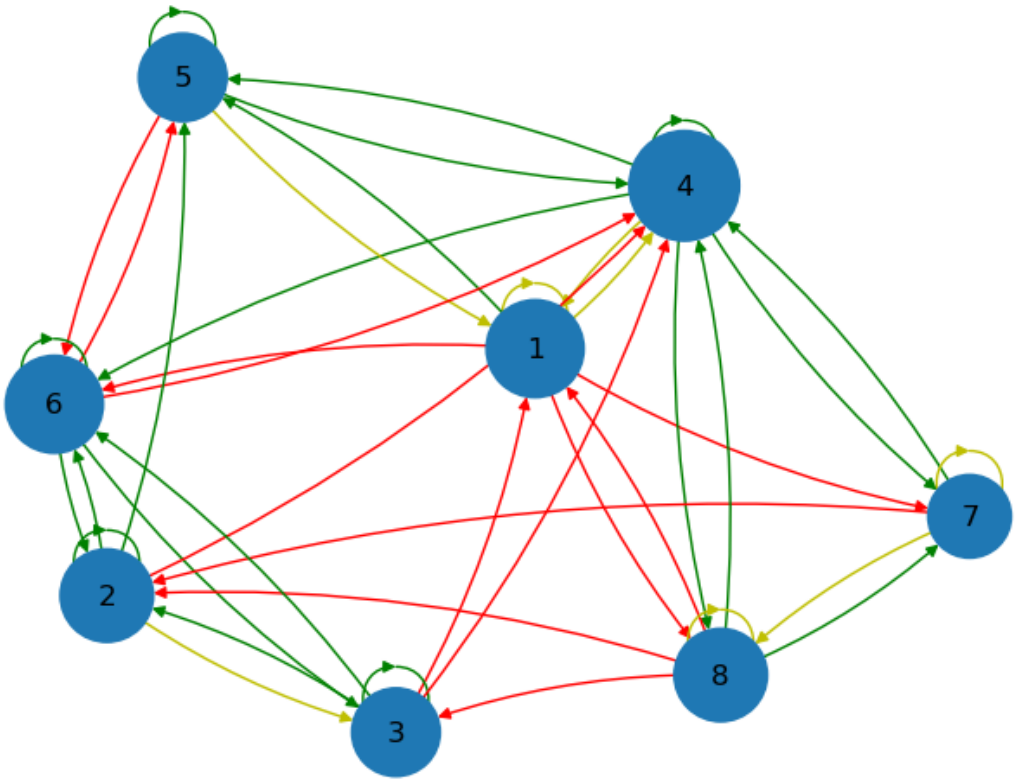


Figure 4.13: Inferred topology for cell death state.

Figure 4.13 show an inferred graph topology from the models trained on cell death state. The observed graph achieved the highest precision with a recall above 0.5. Although there was more than one graph achieving the same precision with this criteria, we chose the one with the highest recall. We can observe a high number of both correctly and wrongly inferred edges. The number of missing edges is relatively small. The precision for the network topology was P 0.606 and the recall score was R = 0.7143. In the true graph, there are a total of 28 edges, while in the inferred graph topology there are 33 edges.

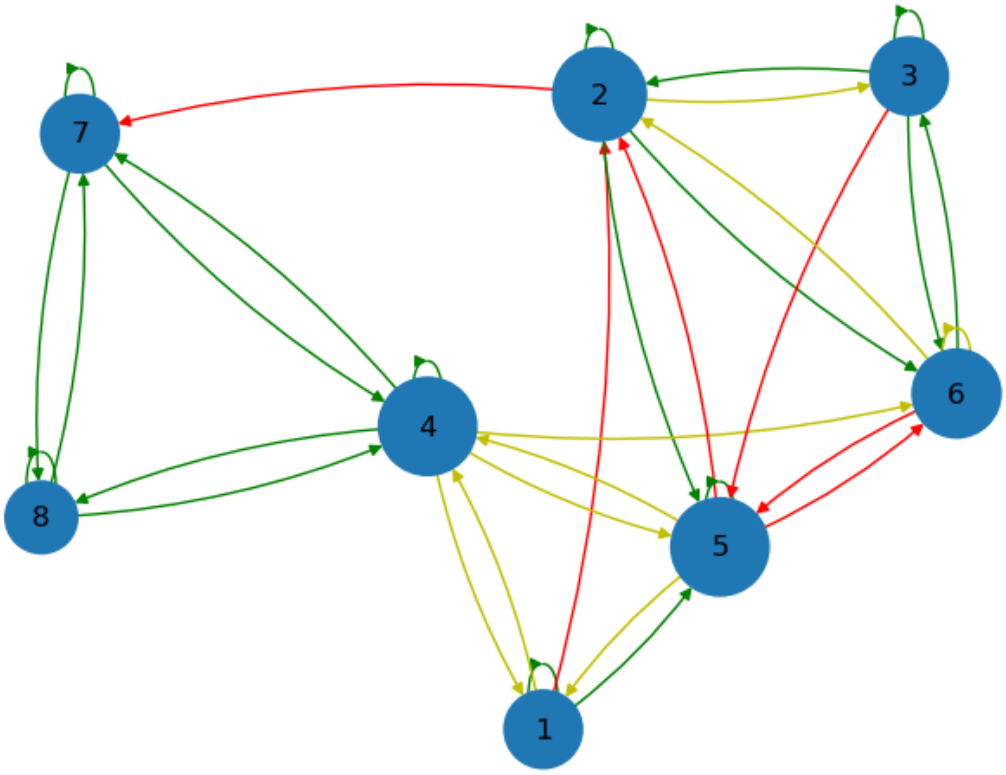


Figure 4.14: Inferred topology for cell survival state.

Figure 4.14 show an inferred graph topology from the models trained on cell survival state data. The graph topology achieved the highest precision and recall for a graph with a recall above 0.5. Of the 28 real edges in the true graph, the inferred model correctly guessed 19. The total number of wrong edges was 6. The inferred network topology achieved a precision score P = 0.760 and a recall score R = 0.679. The network captured 7 out of 8 self-loops.

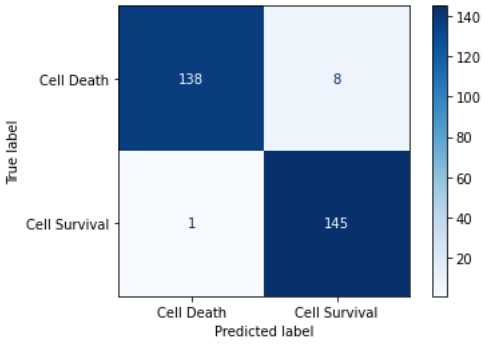
4.3 Classification

In this section we will go through the results of the classification of states from the initial values. We will start by looking at the accuracy of the different ML models, and then look at the confusion matrices for them. The models were tested on a balanced test data set, with a total of 292 samples. The testing data were normalized for both neural networks, while logistic regression used unprocessed data. Logistic regression performed best using non-standardized data.

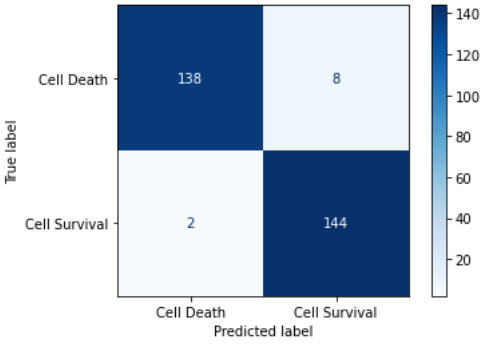
Model	Accuracy
Logistic Regression	0.9281
MLP Classifier Sk-learn	0.962
MLP Classifier PyTorch	0.969

Table 4.1: Model Accuracy on test data.

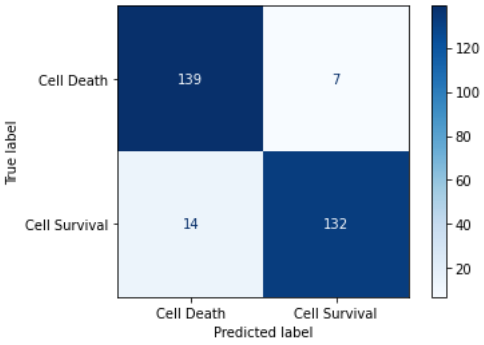
From Table 4.1 we observe that both MLP classifiers received a high accuracy on the testing data. Logistic regression had fewer correct classifications and had lower accuracy than the others. The top-scoring model in terms of accuracy was the MLP classifier implemented in PyTorch.



(a) Confusion matrix for MLP Classifier PyTorch.



(b) Confusion matrix for MLP Classifier SK-learn.



(c) Confusion matrix for Logistic Regressor.

Figure 4.15: Confusion matrix for Classifiers.

From the confusion matrices in figure 4.15 we can observe that the MLP classifiers' performances were almost equal, but the classifier from sklearn got one more wrong prediction of the cell survival state. The classifiers struggled more with predicting cell death state as they both labelled 138 of 146 correctly. The logistic regressor labelled 139 of 146 correctly cell death states, but struggled more with recognizing the cell survival states.

Chapter 5

Discussion

5.1 Choice of biological systems

The choice of biological systems to use in our work was based on a few conditions. The first was that a mathematical model was available, without this we would not be able to generate simulated data for training the ML models. Another condition was that the number of species was relatively small. Lastly, we wanted to test our method for two systems that exhibit different dynamical behaviour and complexity in terms of steady states.

5.2 Training results

The results from the training section were varied. Some of the models were able to generalize well to both training and validation data, with both training and validation loss converging to the same level. Other models looked to overfit the training data, by converging nicely for the training data, while still experiencing variance in the validation error. The variance in performance was observed within the different models for a biological system and was not specific to any of the biological systems. One of the reasons for this behaviour could be the difference in "difficulty" for the data sets. The features were the same for all the models in the same biological system, but the labels were model-specific. This difference in the data sets might lead to some of the models being too complex or not complex enough, while other models have the right complexity. Building on this, all

models were trained for the same number of epochs, and the number of epochs might not be optimal for every model. Obtaining better results in training might be achieved by testing a range of different hyperparameters controlling the model's complexity. Another approach could be implementing an early stopping method or a stopping method, that stops training when the change in training and validation loss reaches a certain threshold.

The varying results from training did not seem to be reflected in the model's ability to predict the species dynamics. There was no direct correspondence between a model performing poorly in training and performing not optimal in predicting the dynamics.

5.3 Models ability to reproduce the dynamical behaviour

The results from this section showed an overall good ability to predict the dynamic behaviour of the systems. Only minor errors were recorded. We cannot observe any noticeable difference in performance related to the two different kinds of systems. The great performance can be attributed to NNs ability to fit almost any kind of data. With our choice of having one MLP network for each species, each species has an equal amount of internal parameters in the MLPs and do not need to share these with the other species. This choice could be contributing to not seeing an increase in error when modelling a system with an increased number of species. Further work in relation to this could be to introduce more variance in the data sets, thereby increasing the robustness of the models. We perturbed the data with 5 % of an error drawn from a Gaussian distribution, $N(0, \sigma^2)$, where the standard deviation σ was calculated for each species in the data. As mentioned in the previous section, training the models with a broad range of hyperparameters could find models capable of obtaining even better predictions. Another benefit to this could be discovering less complex models capable of performing as well as the current models.

The results show that using relatively low complex ML models such as an MLP, can be used as surrogates for high computational demand mechanistic models.

5.4 Network topology inference

After reviewing the results, our method for network topology inference proved to not be consistent. The performance varied between the different sets of models trained on different data. There was no indication that a single attribution algorithm performed best in all three instances. The results showed promise for the graph topologies inferred from models trained on cell survival data, but for the other two, none of the inferred graph topologies showed any promise.

Our method using integrated gradients managed to infer a topology with high precision and a reasonable recall for the single steady-state system. However, after recognizing about half of the true edges, the precision drops quite significantly. This is related to the attribution scoring and threshold values. The precision will drop if the attribution algorithm computes a wrong attribution score, especially giving an existent relation and an almost non-existent attribution score. By wrongly asserting a low attribution score, the resulting topology would need to include a lot of false edges to achieve a high recall score. For example, we can observe that the attribution algorithm assigns a very low score for the pathogen to influence naive cells. In order to include this in the graph almost every other possible edge must be included as well.

The precision-recall curves for the attribution algorithms applied to models trained on cell death data, reveal that the attribution algorithms managed to assign correct scores to a few features only. The precision drops off very quickly for most of the attribution algorithms, and quickly descends into a performance equal to the random assignment of edges.

The best-performing case was for inferred topologies from the models trained on cell survival data. Feature permutation managed to assign feature attribution scores reasonably well. The precision-recall curve proves that it was able to maintain a high precision score while recognizing most edges. By comparing the feature attribution scores for the models trained on different states, we can observe a general trend. In the case of cell survival, the feature attribution algorithms are better at distinguishing the level of attribution for the different features. For cell death, many of the attribution scores are similar.

We present three possible reasons for our method failing to infer suitable network topologies. The first is that the data used to train the models do not contain enough information about the relationship of the species in the system. For the two sets of models

trained on the two states of cell apoptosis, the only difference was the data used in training and this led to two very different results. If we look at figure 4.8, we can compare the difference between the two states. The cell survival data will capture both types of states as cell death is a transient state in the dynamics. Therefore the cell survival data might capture more information about the system and can explain the better performance. The second reason could be that the ML models are not able to capture the correct structure of the system. If one of the models overfits the data, it will capture noise in the training set and lead to wrong information about the system. The last reason could be the feature attribution algorithms failing to explain the model's thinking when making a prediction. Either by the algorithm itself not being able to explain the model or the wrong usage of the algorithm. Wrong, meaning not exploring different baselines or "references" for the feature attribution algorithms that rely on these. As stated by the authors of these algorithms, the choice of baseline is problem-dependent and there is no general optimal solution. Investigating with a set of different baselines could see better inferred network topologies.

5.5 Classification ability

The results showed that classifiers without much tweaking in terms of hyperparameters are able to achieve a reasonable accuracy. All of the tested models managed to correctly classify the state of the system in more than 90% of the test cases. Further work could be performing the classification for other systems than cell apoptosis. Improving the accuracy could also be an area of interest. Our work did not include testing with different hyperparameters, and this could be experimented with.

Chapter 6

Conclusion

In our work, we have further demonstrated that ML models are capable of replacing high-demand computational mechanistic models for simulating the dynamics of a given biological system. In section 2.3 we reviewed two different approaches for solving the same problem. Yazdani et al.[52] used a single DNN with a tailored loss function to incorporate the ODEs into the NN. On the other hand, Shen et al.[42] utilized a single RNN to learn the dynamics of a given biological function. We have shown that a set of n MLPs can mimic the dynamic of an n-species biological system. We applied our method to two different systems, exhibiting two very different dynamics and succeeded in both.

We have also proposed a new method for doing GRN inference. Shen et al.[42] used sensitivity analysis of the network to infer the underlying network topology. Our approach relied on computing the feature attributions for each of the trained MLPs. We used a set of different algorithms for computing the feature attributions. Finally, we used a set of threshold values to determine which edges to include in the final topology. We applied our method to two different systems and achieved varying results. Further work could include exploring different baselines for the feature attribution algorithms or applying our method to different biological systems.

Finally, we proved that it is possible to train an ML classifier to recognize the final state of a bistable system using initial values. Our approach was to use a MLP classifier for the classification problem.

Bibliography

[1] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[2] Alan Aderem. Systems biology: its practice and challenges. *Cell*, 121(4):511–513, 2005.

[3] Bree B Aldridge, George Haller, Peter K Sorger, and Douglas A Lauffenburger. Direct lyapunov exponent analysis enables parametric study of transient signalling governing cell behaviour. *IEE Proceedings-Systems Biology*, 153(6):425–432, 2006.

[4] Mohamed Alloghani, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain, and Ahmed J Aljaaf. A systematic review on supervised and unsupervised machine learning algorithms for data science. *Supervised and unsupervised learning for data science*, pages 3–21, 2020.

[5] Mads Hald Andersen, David Schrama, Per thor Straten, and Jürgen C Becker. Cytotoxic t cells. *Journal of Investigative Dermatology*, 126(1):32–41, 2006.

[6] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[7] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[8] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.

[9] Lian En Chai, Swee Kuan Loh, Swee Thing Low, Mohd Saberi Mohamad, Safaai Deris, and Zalmiyah Zakaria. A review on the computational approaches for gene regulatory network construction. *Computers in biology and medicine*, 48:55–65, 2014.

[10] Fabien Crauste, Julien Mafille, Lilia Boucinha, Sophia Djebali, Olivier Gandrillon, Jacqueline Marvel, and Christophe Arpin. Identification of nascent memory cd8 t cells and modeling of their ontogeny. *Cell systems*, 4(3):306–317, 2017.

[11] Francis Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970.

[12] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27, 2014.

[13] Issam El Naqa and Martin J Murphy. *What is machine learning?* Springer, 2015.

[14] Ioana M Gherman, Zahraa S Abdallah, Wei Pang, Thomas E Gorochowski, Claire S Grierson, and Lucia Marucci. Bridging the gap between mechanistic biological models and machine learning surrogates. *PLOS Computational Biology*, 19(4):e1010988, 2023.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[16] Trevor Hastie, Robert Tibshirani, Jerome Friedman, Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Overview of supervised learning. *The elements of statistical learning: Data mining, inference, and prediction*, pages 9–41, 2009.

[17] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[18] Michael Hecker, Sandro Lambeck, Susanne Toepfer, Eugene Van Someren, and Reinhard Guthke. Gene regulatory network inference: data integration in dynamic models—a review. *Biosystems*, 96(1):86–103, 2009.

[19] Brian P Ingalls. *Mathematical modeling in systems biology: an introduction*. MIT press, 2013.

[20] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[22] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[23] Guy Karlebach and Ron Shamir. Modelling and analysis of gene regulatory networks. *Nature reviews Molecular cell biology*, 9(10):770–780, 2008.

[24] Hiroaki Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002.

[25] Hiroaki Kitano. Systems biology: a brief overview. *science*, 295(5560):1662–1664, 2002.

[26] Edda Klipp, Wolfram Liebermeister, Christoph Wierling, and Axel Kowald. *Systems biology: a textbook*. John Wiley & Sons, 2016.

[27] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, et al. Captum: A unified and generic model interpretability library for pytorch. *arXiv preprint arXiv:2009.07896*, 2020.

[28] Karsten Kruse and Frank Jülicher. Oscillations in cell biology. *Current opinion in cell biology*, 17(1):20–26, 2005.

[29] David S Latchman. Transcription factors: an overview. *The international journal of biochemistry & cell biology*, 29(12):1305–1312, 1997.

[30] Wei-Po Lee and Wen-Shyong Tzou. Computational methods for discovering gene networks from expression data. *Briefings in bioinformatics*, 10(4):408–423, 2009.

[31] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[32] Wenzhe Ma, Ala Trusina, Hana El-Samad, Wendell A Lim, and Chao Tang. Defining network topologies that can achieve biochemical adaptation. *Cell*, 138(4):760–773, 2009.

[33] Daniele Mercatelli, Laura Scalambra, Luca Triboli, Forest Ray, and Federico M Giorgi. Gene regulatory network inference resources: A practical overview. *Biochimica et Biophysica Acta (BBA)-Gene Regulatory Mechanisms*, 1863(6):194430, 2020.

[34] Ravil Muhamedyev. Machine learning methods: An overview. *Computer modelling & new technologies*, 19(6):14–29, 2015.

[35] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[36] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[37] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[38] Andrea Saltelli. Sensitivity analysis for importance assessment. *Risk analysis*, 22(3): 579–590, 2002.

[39] Andrea Saltelli, Paola Annoni, Ivano Azzini, Francesca Campolongo, Marco Ratto, and Stefano Tarantola. Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index. *Computer physics communications*, 181 (2):259–270, 2010.

[40] Thomas Schlitt and Alvis Brazma. Current approaches to gene regulatory network modelling. *BMC bioinformatics*, 8:1–22, 2007.

[41] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[42] Jingxiang Shen, Feng Liu, Yuhai Tu, and Chao Tang. Finding gene network topologies for given biological function with recurrent neural network. *Nature communications*, 12(1):3125, 2021.

[43] Wenjia Shi, Wenzhe Ma, Liyang Xiong, Mingyue Zhang, and Chao Tang. Adaptation with transcriptional regulation. *Scientific reports*, 7(1):1–11, 2017.

[44] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International conference on machine learning*, pages 3145–3153. PMLR, 2017.

[45] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[46] Ilya M Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and computers in simulation*, 55(1-3):271–280, 2001.

[47] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[48] Rebecca C Taylor, Sean P Cullen, and Seamus J Martin. Apoptosis: controlled demolition at the cellular level. *Nature reviews Molecular cell biology*, 9(3):231–241, 2008.

[49] Paul D Thomas. The gene ontology and the meaning of biological function. *The gene ontology handbook*, pages 15–24, 2017.

[50] John J Tyson, Katherine C Chen, and Bela Novak. Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell. *Current opinion in cell biology*, 15(2):221–231, 2003.

[51] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine learning*, 109(2):373–440, 2020.

[52] Alireza Yazdani, Lu Lu, Maziar Raissi, and George Em Karniadakis. Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLoS computational biology*, 16(11):e1007575, 2020.

[53] Ben Youngblood, J Scott Hale, Haydn T Kissick, Eunseon Ahn, Xiaojin Xu, Andreas Wieland, Koichi Araki, Erin E West, Hazem E Ghoneim, Yiping Fan, et al. Effector cd8 t cells dedifferentiate into long-lived memory cells. *Nature*, 552(7685):404–409, 2017.

[54] Mengyuan Zhao, Wenying He, Jijun Tang, Quan Zou, and Fei Guo. A comprehensive overview and critical evaluation of gene regulatory network inference technologies. *Briefings in Bioinformatics*, 22(5):bbab009, 2021.

Appendix A

ODE systems used in thesis

Ontogeny of nascent CD8 memory T-cells

$$
\begin{aligned}
\frac{dN(t)}{dt} &= -\mu_N N(t) - \delta_{NE} P(t) N(t) \\
\frac{dE(t)}{dt} &= \delta_{NE} P(t) N(t) + [\rho_e P(t) - \mu_{EE} E(t) - \mu ELL(t) - \mu_E - \delta_{EL}] E(t) \\
\frac{dL(t)}{dt} &= \delta_{EL} E(t) - [\mu_{LL} L(t) + \mu_{LE} E(t) + \mu_L + \delta_{LM}] L(t) \\
\frac{dM(t)}{dt} &= delta_{LM} L(t) - \mu_M M(t) \\
\frac{dP(t)}{dt} &= [\rho_P P(t) - \mu_{PE} E(t) - \mu_{PL} L(t) - \mu_P] P(t)
\end{aligned}
\tag{A.1}
$$

Cell apoptosis

$$
\begin{aligned}
\frac{dx_1(t)}{dt} &= -k_1 x_4 x_1 + k_{d1} x_5 \\
\frac{dx_2(t)}{dt} &= k_{d2} x_5 - k_3 x_2 x_3 + k_{d3} x_6 + k_{d4} x_6 \\
\frac{dx_3(t)}{dt} &= -k_3 x_2 x_3 + k_{d3} x_6 \\
\frac{dx_4(t)}{dt} &= k_{d4} x_6 - k_1 x_4 x_1 + k_{d1} x_5 - k_5 x_7 x_4 + k_{d5} x_8 + k_{d2} x_5 \\
\frac{dx_5(t)}{dt} &= -k_{d2} x_5 + k_1 x_4 x_1 - k_{d1} x_5 \\
\frac{dx_6(t)}{dt} &= -k_{d4} x_6 + k_3 x_2 x_3 - k_{d3} x_6 \\
\frac{dx_7(t)}{dt} &= -k_5 x_7 x_4 + k_{d5} x_8 + k_{d6} x_8 \\
\frac{dx_8(t)}{dt} &= k_5 x_7 x_4 - k_{d5} x_8 - k_{d6} x_8
\end{aligned}
\tag{A.2}
$$