Lund University
Department of astronomy and theoretical physics

Demarcating good solutions in system biology computer models using artificial neural networks

Author: André Larsson

Supervisors: Henrik Jönsson
Mattias Ohlsson

June 4, 2010

Abstract

How close a computer model comes to recreating real-world phenomena often depends on the value of its internal parameters, but investigating the outcome of the model for every point in parameter space is in practice an impossible task. Here an artificial neural network is used as a numerical predictor on two different system biology computer models. A semi-implicit solver was also implemented for one of these models in order to speed up simulations in stiff regions of parameter space. The performance of the neural networks were measured using the area under the receiver operating characteristic curve (AUC), and neural networks were used as numerical predictors for three different four-dimensional parameter regions. In the first region a training data set of 500 points were used and an auc of 1.0 was achieved. In the second region a training data set of 1000 points were used and an auc of 0.97 was obtained. In the last region training data sets of 100, 250, 1000 and 3000 points were used and the auc of the neural networks was 0.86, 0.95, 0.97 and 0.97 respectively.

Contents

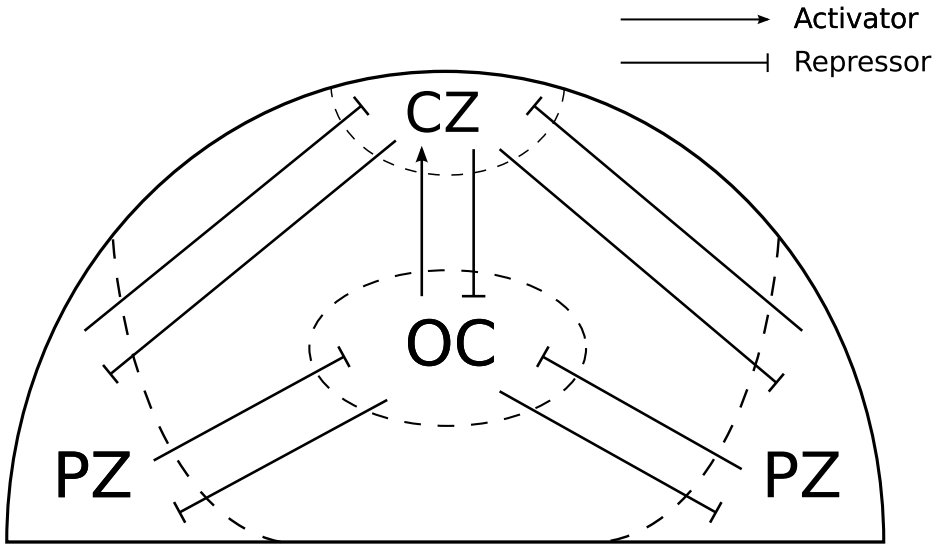

Figure 1: A simplified view of the different regions within the SAM of a plant and how they interact with each other. WUS is expressed in the OC, CLV3 is expressed in the CZ and KAN is expressed in the PZ.

1 Introduction

In science, the use of computer models have proven to be useful when explaining emergent phenomena and as a way of exploring new ideas and suggestions. Specifically, the field of system biology merges the methods of physics and biology when crafting computer models of biological entities, where the goal often is to explain an observed behaviour using, e.g., differential equations representing the interactions and dynamics of the system. Here, the behaviour of two such models will be investigated, although the procedure for doing so is not dependent on their biological nature. The two models considered both simulate a part of the shoot apical meristem (SAM), responsible for the aerial growth of plants. As such, these models are closely related to experimental results.

It has been found that the shoot and root meristems maintain a steady population of stem cells throughout environmental changes and despite the ongoing cell divisions [Sablowski, 2007]. The meristem is subdivided into regions expressing different genes, this structure is in turn maintained by a network of complex interactions, including feedback loops which maintain the organization of the meristem. Specifically, the SAM contains a negative feedback loop between the gene *WUSCHEL* (*WUS*) expressed in the organising center (OC), and the gene *CLAVATA3* (*CLV3*) expressed in the central zone (CZ) [Williams and Fletcher, 2005], preventing an uncontrolled growth of the stem cell population (Figure 1). WUS promote stem cell proliferation and the stem cells in turn promotes CLV3, hence WUS acts as an activator of *CLV3*. CLV3 in turn diffuses outwards from the stem cells and represses WUS, and a self-regulating negative feedback network between WUS and CLV3 is created [Sablowski, 2007].

Since the meristem inherently is a complex system, it can be hard to understand and explain it using only analytical methods. Finding the equilibrium

solutions or discovering emergent behaviour from a set of differential equation with only pen and paper can be a daunting task. For the SAM region it is not always obvious how different hypothesis affect the resulting behaviour of the meristem, see for example [Jönsson et.al., 2005]. Computer models have therefore been used to further explain and increase our understanding of these systems, evaluating the effect of different hypotheses in relation to experimental results.

The structure of the SAM have been modelled in [Jönsson et.al., 2005] and more recently in [Sturk, 2010], whereas in the latter a perhiperal zone PZ expressing the proposed gene *KANADI* (*KAN*) was added to the meristem along with the CLV3 and the WUS regions. The PZ is introduced in order to spatially determine, together with the OC, the region of WUS expression. All of the interactions between KAN, WUS and CLV3 within this model are transmitted via signalling molecules, which may or may not be the proteins themselves. Some of the details of the signalling pathways between the expressed genes are modelled in [Sahlin et.al., 2010], where the interaction between WUS and CLV3 are studied in more detail. A new approach for evaluating the behaviour of these two models is introduced here.

A common problem when modelling a complex system is the number of parameters to be tuned and determined. Each different set of parameters corresponds to a different behaviour of the model, and since each parameter can in theory take on any value it is an impossible task to investigate them all in a finite amount of time; some discretisation and demarcation of parameter space is necessary. Even so, it is often a challenging task to investigate even a small region of the possible sets of parameters. This is related to the *curse of dimensionality*, meaning that the volume of parameter space increases exponentially with the number of parameters. It can quickly become a very time-consuming task to uniformly and adequately sample a region in a high dimensional parameter space.

When searching for solutions in a model it is common to define an energy function quantifying how close the model is to some desired output. The problem of finding good solutions to the model can then be cast into the problem of minimising the error function, hence function minimising algorithms like gradient descent or simulated annealing can be used [Press, 2007:2]. The benefits of, e.g., simulated annealing is that a large volume in parameter space can be sampled when searching for the single points in parameter space corresponding to energy minima. The result of such an optimisation procedure is then a list of scattered points in parameter space considered to be good solutions. But a more general method would be to find good *regions* in parameter space as opposed to good *points*.

It can be of interest to know where the boundary between good and bad sets of parameters are. A real system is typically subject to environmental changes and noise, therefore a model should not be too sensitive to small parameter perturberations, representing noise in the sorrounding environment. Knowing the volume of the regions with good parameters may give us some idea of this sensitivity, this is one of the reasons for shifting focus from a list of scattered valid solutions in parameter space to a search of valid coherent regions.

An artificial neural network (ANN) can be used together with a scan of the

parameter space for discriminating regions of good parameters. However, it is not a good idea to blindly approach the vast parameter space armed only with an ANN. Reference points consisting of good parameters will most likely be needed, as some starting point is needed. The procedure will then be to first search for good points in parameter space, then to scan around these good points to be able to find good regions. By training an ANN on a list of simulated data points it should find the boundaries separating good from bad solutions, and the trained ANN can then be used to provide a numerical prediction in the scanned subregions of parameter space.

It was found in [Sturk, 2010] that some regions in parameter space for the SAM model took much longer time to simulate than other regions, making it impossible to adequately scan parameter space in this neighbourhood. A more stable numerical solver could alleviate this problem, making a larger portion of parameter space accessible to the use of an ANN. To this end an implicit numerical solver was implemented; its stability properties [Press, 2007:1] should allow for longer steplengths and thus decrease the total simulation time.

Along with the implementation of an implicit numerical solver, subregions in parameter space of the SAM model and the CLV3-WUS signalling model was here scanned and investigated with neural networks of appropriate architectures. The results suggests that good solutions in parameter space does indeed form coherent regions, making it possible for an ANN to demarcate these and provide a numerical prediction for the scanned region. Further, the performance of the implemented implicit numerical solver was carefully evaluated, and it is suggested that although the implicit solver does show some beneficial properties more work is needed on finding a better algorithm for solving sparse linear systems in order for the solver to be useful.

2 Methods and Models

2.1 Model of the SAM region

The SAM in a plant consist of the CZ expressing CLAVATA (CLV) and harboring the stem cells, the PZ where stem cells go to become new organs and the OC expressing WUS [Sablowski, 2007] (Figure 1). The stem cells, whose proliferation is promoted by WUS, induces CLV3 expression which in turn represses WUS, hence a self-regulating negative feedback loop is formed. The OC have been found to be able to fully recover even after it has been removed using laser ablation [Heisler and Jönsson, 2007], suggesting that there is an external activator network maintaining the existence and position of the OC.

The SAM was modelled using a reaction-diffusion system [Jönsson et.al., 2005] where the network of signalling molecules are represented by a set of differential equations. The OC is maintained using a Brusselator network [Prigogine and Lefever, 1968] tuned to act as the activator network mentioned above. In [Sturk, 2010] the PZ was introduced into the model, using KAN as a marker, and it is this model that here have been investigated. The full model equations can be found in the appendix.

It has been noted that parameters leading to a good behaviour seems to be sparse for this model. Only three good sets of parameters have been found at the time of writing. To avoid fumbling around too much in parameter space, these points are used as starting points when scanning parameter space in search for regions of good solutions. Qualitatively there are important differences in behaviour for the model between these sets, for our purposes the most important is the time it takes to simulate the model to equilibrium at the given point in the parameter space. It was found that one of these sets of parameters were troublesome, simulation times were here about forty times longer than for the other two sets of parameters. If this large increase in computational time could be remedied using another kind of numerical solver, e.g., an implicit solver, it would make life easier when scanning parameter space for the ANN. For this, and other reasons (see section 2.5 Numerical Solvers), an implicit solver was implemented and applied to this model.

Since a smaller system is both easier to understand and comes with a parameter space of a lower dimension, making it easier to adequately scan it, a subnetwork of the full organism model was investigated. The concentrations of CLV3 and WUS was kept constant, while the KAN concentrations in the PZ was determined by the parameters in the equation governing the KAN concentrations. The CLV3 and WUS concentrations were taken from a template derived from experimental results. By locking the WUS and CLV3 concentrations in the system, the number of parameters decreases to six, which can be further reduced to five when noting the equivalence of some solutions.

This subnetwork was simulated until a stable state (equilibrium) was found, after which an error was calculated by taking the difference between the substance concentrations of the resulting simulation and a template determined from *in vivo* experiments.

2.2 Model of the signalling between CLAVATA3 and WUSCHEL

For the plant *Arabidopsis thaliana*, it has been suggested [Müller et.al., 2008] that CLAVATA1 (CLV1) together with CORYNE (CRN) form two independent signalling pathways with which CLV3 represses WUS. From experiments with CLV1 loss-of-function mutants it has been shown that the null mutant *clv1-11* have a stronger phenotype than the non-null mutant *clv1-1*, corresponding to an increased *WUS* expression. If there would be only one path to CLV3-WUS repression – a path involving a CLV1/CLV3 complex – the null mutant should remove all CLV3 repression of WUS and hence show the strongest possible phenotype. But when comparing the non-null mutant to the null mutant, an even stronger phenotype is observed in the non-null mutant. This fact can be explained if another pathway for repressing WUS is introduced, the pathway involving CRN. The non-null mutant *clv1-1* is then said to impair the CRN pathway additionally to cancelling the CLV1 pathway, while the null mutant *clv1-11* only cancels the CLV1 path.

Two models of these two pathways was investigated in [Sahlin et.al., 2010]. Here, only the loss-of-signal model is considered, which assumes that, for the *clv1-1* mutant, CLV1 binds to CLV3 as usual but reduces the actual signal sent

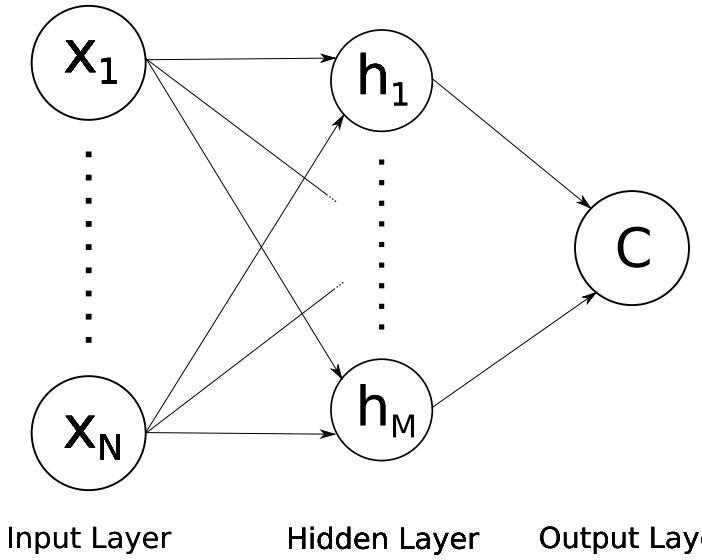

Figure 2: A neural network with N inputs, M hidden nodes and one output node.

to the cell. Further, to explain the stronger phenotype observed in *clv1-1* mutants, this mutant also have to impair the CRN pathway. This is achieved when *clv1-1* sequesters CLV3 by forming a weak (loss-of-signal) complex with it, making CLV3 unavailable for the CRN path. It can be noted that this model is a simplification of the pathway and it does not take into account factors like, e.g., metabolic costs; effects which could be important for the outcome of the simulations [Sahlin et.al., 2010]. The full CLV3-WUS signalling model equations can be found in the Appendix.

A part of the parameter space of the loss-of-signal model was investigated by scanning a subregion in the neighbourhood of a randomly chosen known good solution. An energy function is used to compare an experimental value of the strength of the WUS expression (inferred from the number of carpels) of the mutants to the resulting WUS expression from the simulation. In contrast to the model of the full SAM region, about 25'000 sets of parameter sets were initially accepted, these parameter sets were then tested in a validation step after which 118 parameter sets remained (for the loss-of-signal model). Here the validation step is ignored when scanning parameter space; one of the initial 25'000 parameter sets with a low energy was used as a starting point, and sets were classified as good or bad depending on their assigned energy.

2.3 Evaluating parameter space with an artificial neural network

2.3.1 The nature of an artificial neural network and how to estimate its performance

Many different architectures and algorithms are grouped together under the umbrella term 'artificial neural networks' and what they all have in common is the analogy to real, biological, neural networks. Real neural networks, in a very sim-

plified sense, consists of neurons recieving and sending signals to each other via synapses. The ANN's used here involves nodes organized in different layers, each node receiving a weighted sum of the signals sent by the previous layer. Further, all networks consists of one input layer feeding signals to a hidden layer which in turn feeds its signals into the output layer consisting of one output node (Figure 2).

The signal sent from a node is determined by an activation function, taking a weighted sum of the incoming signals as the argument. The activation function for the hidden nodes is here the hyperbolic tangent function, meaning that the output of hidden node h_l can be written

$$h_l = \tanh\left(\sum_{i=0}^{N} w_{li} x_i\right)$$

where w_{lk} is the weight between input node x_k and output node h_l. Node x_0 (not shown in Figure 2) is defined as $x_0 = +1$ such that the weight w_{l0} becomes the bias applied to the network [Haykin, 2009]. The bias is used to shift the argument of the activation function with a constant. In a similar way, the output node C receives the weighted sum of the previous layer – in this case the hidden layer – and uses this sum as the argument v in the activation function $\varphi(v)$,

$$C = \varphi\left(\sum_{j=0}^{M} w_{Cj} h_j\right)$$

As above, $h_0 = +1$ and w_{C0} is defined the bias applied to C. The activation function is the sigmoidal function

$$\varphi(v) = \frac{1}{1 + e^{-av}}$$

which is a monotonically increasing function of v, bounded by $y = 0$ and $y = 1$. The goal is to create a network correctly mapping a set of parameters to either a one (good solution) or a zero (bad solution) by adjusting its weights.

A hidden node can be interpreted as a surface dividing input space into two regions/classes [Haykin, 2009]. A neural network with an arbitrary number of hidden nodes can hence be interpreted as demarcating good solution in parameter space with the surfaces corresponding to the hidden nodes of the network. Figure 3 is an illustration of how an ANN could impose a decision boundary on a two-dimensional input space in order to separate two classes.

To find the surfaces separating good from bad solutions,the ANN is *trained* with data consisting of points in parameter space together with their complementary one or zero. By training, the parameters of the ANN, its weights, are changed in order to improve the number of correctly classified points. This can be visualised as the ANN changing its decision boundary until all, or most of, the supplied training data have been correctly delimited. The success of an ANN thus depends on how easy it is to demarcate the two different classes in parameter space with surfaces.

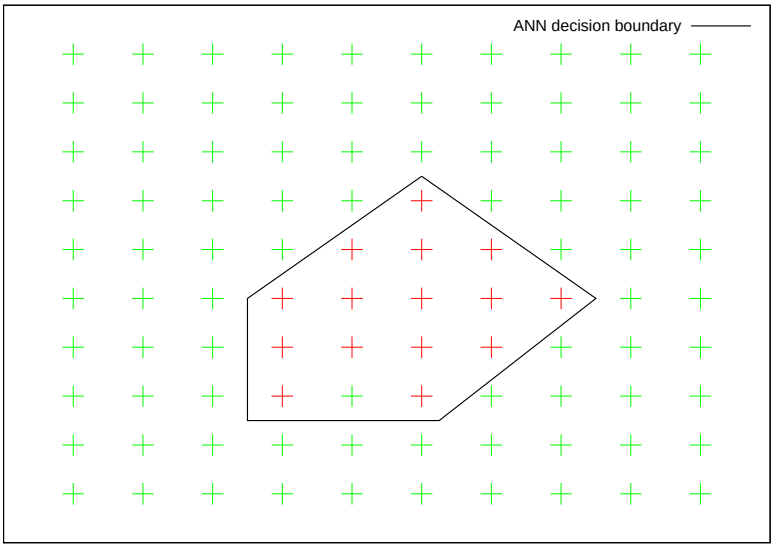

Figure 3: A grid of class [red] points and class [green] points together with a suggestions of how the boundary used by an ANN to separate these different classes could look like. If this boundary would have been used one point would be missclassified.

To avoid overfitting (Figure 4), the training data is split into training and validations sets, where the validation set is used as an early estimator of how well the network will generalize to new data.

When estimating the performance of an ANN, the metrics sensitivity, specificity and BAC was considered. The output C of the networks used here is a number between zero and one, and the predicted class is determined by setting a cut on this output. Input points giving an output above the cut is considered to be of class one, while points giving an output below the cut is considered to be of class zero. At the extremes, using a cut of zero would classify everything as ones and vice versa. A data point of class one, correctly classified as a one is called a True Positive, and a data point of class zero correctly classified as a zero is called a True Negative. The number of True Positives and True negatives can be used to define

$$sensitivity = \frac{\text{number of True Positives}}{\text{number of data points of class one}}$$

$$specificity = \frac{\text{number of True Negatives}}{\text{number of data points of class zero}}$$

The $sensitivity$ and $specificity$ can be used to measure the performance of the network. The BAC is defined as the average of the sensitivity and specificity, here taken using a cut of 0.5. Using a cut of 0.5 seems like the natural choice, and should be so if the number of ones and zeros in the data are about equal.

Another measure of an ANN's performance is the ROC curve and the related scalar AUC. The ROC curve is a plot of $(1 - specificity)$ versus $sensitivity$. $sensitivity$ is by definition a fraction of True Positives, while $(1 - specificity)$ is

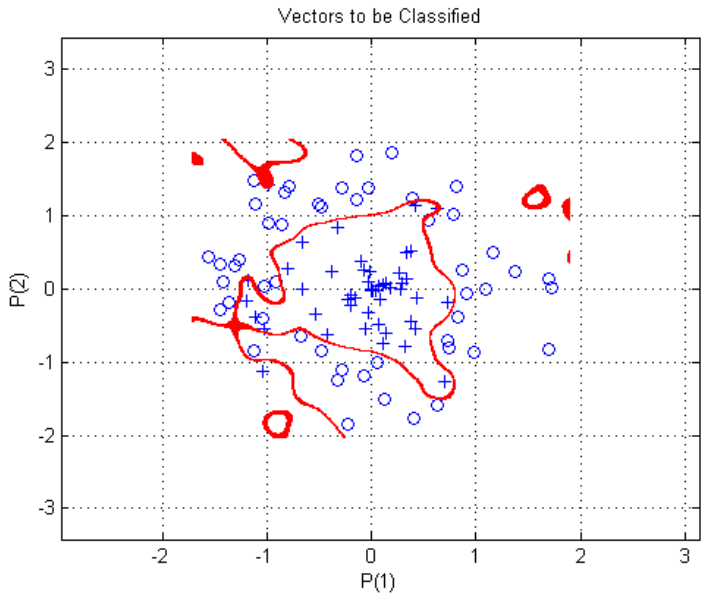

Figure 4: A grid of class [cross] points and class [circle] points together with the boundary used by an 118 hidden node ANN to separate these different classes. The decision boundary does separate the two classes, although it is unlikely to generalize well.

a fraction of False Positives:

$$1 - specificity = \frac{\text{number of False Positives}}{\text{number of data points of class zero}}$$

By plotting the pairs of $(1 - specificity)$ and $sensitivity$ for many different cuts a curve in ROC space is obtained. In general we want many True Positives (a high $sensitivity$) and few False Positives (a low $(1 - specificity)$), meaning that a good classifier will have a ROC curve weighted towards the upper left corner in the ROC plot. Thus a better classifier would have a larger area under the ROC, meaning that the area under the curve (AUC), can be used as a measure of the performance of a classifier. The AUC is cut and class skew independent, also, an ideal classifier would have an AUC of 1, and a dummy classifier guessing the class would have an AUC of 0.5.

As mentioned, overfitting to the data should be avoided and it is therefore of importance to consider the complexity of the ANN. For a very complex network with many hidden nodes, the number of weights can be so large that the network is overtrained on the data, imposing strange and cumbersome decision boundaries not likely to be true. The generalization performance of an overtrained network, i.e., its ability to classify new data drawn from the same distribution as the training data, is most likely to be degraded. Figure 4 is an example of such a behaviour, showing an extreme case with a very complex network. This network classifies every point in the training data correctly, meaning that data points that are real zeros are mapped to a zero (or a number very close to zero) by the network, and

vice versa. This would correspond to having an high AUC on the training phase. But this network is not likely to perform well on new data, meaning that one would see a significant decrease of the AUC when the network is evaluated on a validation set or a independent test set. By having a too complex network, overfitting similar to Figure 4 is possible, especially when you only have a small set of data, meaning that there is a clear benefit for using a more *parsimonious* network if given the chance.

The ANN training/validation algorithm used here comes with a built-in sensitivity analysis, which provides a simple measure of how much an input parameter influence the performance, and hence the output, of the network. For each of the input parameters, the training data is presented to a trained and ready network, but now with one of the input parameters replaced by its mean. By measuring how much the error increases (or decreases) when doing this, it is possible to get an idea of how important a parameter is when predicting the class of a point.

For all of the models, ten different networks were trained on the same data and their averaged output was used when giving their prediction on the test set. Further, the input data is always normalised to have unit variance and zero mean.

2.3.2 Rationale for using an ANN

Using an ANN to find the boundaries between regions of good and bad solutions is also a way to solve the problem of providing a value for points inbetween simulated data. After training the ANN with the known data and their classes, new points can be presented to the network, and a prediction of its class will be obtained. For the prediction to be accurate, the data must be somewhat well-behaved, such that good solutions can be well demarcated from bad ones.

This is intuitively plausible; it can be argued that the good parameters should exist as coherent islands surrounded by bad solutions to the model, as follows. When moving slightly away from a known good solution we hope to find another good, or at least okay, solution, unless the system behaves in a very chaotic way. Given that a real system should be stable against environmental changes and noise we can translate that to our models being stable against small parameter perturberations, giving us a reason for why coherent regions of good solutions should exist.

An ANN is inherently non-linear [Haykin, 2009] which makes it suitable for finding coherent regions of arbitrary shapes. Note that a very complex shape still would be hard for an ANN to find, and that the input data sometimes need to be preprocessed in some way, making use of things we already know to make it easier for the network. A simple example would be to convert data which are known to include exponentially increasing parameters to log-space, which in some situations could make improve the performance of an ANN.

Ideally a predictor for the behaviour of a model could be obtained by simply providing the ANN with some training data, then a training algorithm handles the fit to this data, and an ANN capable of classifying parameter space is hence obtained. Should this work, this could be an easy way to get a new tool for further evaluating the behaviour of a model.

Also, a trained network provides a fast prediction when presenting it to new

data, which in effect can shorten the total time spent on simulating points. The time saved naturally depends of the nature of the model, for very complex models which takes some considerable time to simulate, using a trained network instead could save considerable amounts of time. There is however a trade-off, for a model with longer simulation times it is harder to collect the training data needed for training the network, making it important to know the minimum number of points needed for getting an useful ANN.

2.4 Data sets and the classification of data points into good or bad solutions

The ideal result of an ANN would be a function correctly mapping all sets of parameters to their corresponding class (good or bad solution). However, a definition of what a good and a bad solution is must first be made.

The energy functions used in the investigated models compare the result of the simulation to some desired output, which are taken from real experiments, making it possible to say whether the model provides a good solution or not. By knowing the measuring errors for the real experiments it is possible to get an idea of how large errors should be tolerated. However, since biological considerations are second to computational considerations regarding the use of an ANN for parameter space evalutation, a more pragmatic view of the error/energy function was taken. A cut on the energy was used to determine if the solution was good or bad. The cut was tuned such that thirty to fourty per cent of the data was classified as good solutions, for the model simulating the SAM region, while making sure that the energy levels for the good solutions still were kept within reasonable bounds. For the model of the CLV3-WUS signalling, only three percent of the data was classified as ones, in order to still have a sane energy of the good points.

The summed-square energy function used in the SAM model was

$$E = \frac{1}{3N} \sum_i \Bigg(\Big([\mathrm{WUS}]_i - [\mathrm{WUS}]_i^* \Big)^2 + \Big([\mathrm{CLV3}]_i - [\mathrm{CLV3}]_i^* \Big)^2 + \Big([\mathrm{KAN}]_i - [\mathrm{KAN}]_i^* \Big)^2 \Bigg)$$

where the sum is over all cells in the SAM, [WUS] is the concentration of WUS taken from the simulation and [WUS]* is the template concentration taken from experiments. The KAN and CLV3 terms are defined in the same way. The difference between simulated and experimental concentrations is thus squared and summed, and divided by 3N, where N is the number of cells of the system.

Data sets of different sizes was used for the different models. When simulating a subregion of parameterspace of the full SAM model, a data set of 10'000 points was obtained and two independent data sets of 500 points each was randomly drawn from the initial data set. These two data sets were then used as a training data set and a test set, respectively. For the subnetwork of the SAM, where the CLV3 and WUS concentrations was constant, training data sets of different sizes was randomly extracted from an initial data set of 10'000 points. The resulting networks were then used as numerical predictors on an independent test set consisting of 1'000 points randomly drawn from the parameter space of the model.

Finally, for the CLV3-WUS signalling model,a data set of 12^4 ≈20'000 points was obtained, from this a training set of 1'000 was randomly drawn and tested on a 10'000 point data set also randomly drawn from the same initial data, but independent of the training set.

2.5 Numerical solvers

A Runge-Kutta (RK) solver with a fifth order correction [Press, 2007:2] was used as the numerical solver in the SAM model. This implementation uses a variable step length; if the error between the fourth and the fifth order calculations is too large the step length is decreased and vice versa. The RK solver is an explicit solver, meaning that the next step is evaluated using the current derivatives, as opposed to an implicit solver where the derivative is evaluated at the next time step [Press, 2007:1].

Because we are only interested in the equilibrium state of the model, an implicit solver could be better than an explicit one. For a linear system an implicit solver converges towards the equilibrium state, even when taking long steps, for the cost of a reduced accuracy of the initial dynamics [Press, 2007:1]. When facing a stiff set of equations, which can happen when a concentration depends on variables of very different magnitudes, an explicit solver typically needs to take very short steps in order to avoid a diverging solution. Since some regions in the parameter space for the SAM model was found to be stiff, the stability properties of an implicit solver should allow for longer steplengths and faster simulation times than the RK solver. Even though the stability of an implicit solver only is guaranteed for linear systems, non-linear sets of equations also benefits from the use of an implicit solver.

When taking a step using an implicit method the derivative of the future state of the model is needed. This derivative depends on the state of the system, which at this point is still unknown, meaning that some numerical approximation of this future derivative have to be made. The solver then becomes a *semi-implicit* solver, and can be seen as an approximation of a true implicit solver, where the approximation consist of the numerical evaluation of the future derivative. Using a semi-implicit over an explicit method should however still be an improvement, the behaviour of a semi-implicit solver approximates the behaviour of a true implicit solver, meaning that it too should be stable as long as we do not take very long time steps.

Two different semi-implicit solvers were implemented, a semi-implicit Euler (SIE) solver and a Rosenbrock stiffly stable (ROSS) solver [Press, 2007:1], differing mainly in how they approximate the future derivatives. The SIE solver approximates the future derivative by taking a simple Euler step to the next state and evaluating the derivative at this future step. By doing a better approximation of this next state, the next derivative gets closer to its real value, and the semi-implicit method closes in on an ideal true implicit solver. The ROSS does exactly this, by predicting the next step and hence the future derivative, using a Runge-Kutta-like scheme with the end result of an higher order solver.

Two different semi-implicit solvers were implemented in the organism model. The solvers used were based on code that can be found in [Press, 2007:1] and both

were written in C++. The first solver tested was a semi-implicit Euler method (SIE). Following [Press, 2007:1], the SIE solver is constructed in the following way. Here $\mathbf{y}_n$ represents a vector with the different substance concentrations in the cells at time step n. The term $\mathbf{f}(\mathbf{y}_{n+1})$ is the vector of derivatives evaluated at step $n+1$ and $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ is the Jacobian matrix containing the partial derivatives of the change in concentrations. The equation for getting the next time step with an implicit solver is then

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(\mathbf{y}_{n+1})$$

The last term can be linearly approximated,

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\left[\mathbf{f}(\mathbf{y}_n) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right|_{\mathbf{y}_n} \cdot \left(\mathbf{y}_{n+1} - \mathbf{y}_n\right)\right]$$

Rearranging this last equation gives

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \cdot \left[\mathbf{1} - h\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right]^{-1} \cdot \mathbf{f}(\mathbf{y}_n) \tag{1}$$

TEST TEST TEST

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \cdot \mathbf{f}(\mathbf{y}_n) \tag{2}$$

TEST TEST TEST which is the equation used for calculating the state at the next time step, for the semi-implicit Euler method.

Although the implicit solvers were thoroughly tried and tested, for all practical purposes the explicit RK solver was used when scanning regions in parameter space. It was found that the implicit solvers would need to be further improved in some aspects in order to gain some speed on the RK solver. For the CLV3-WUS signalling model a numerical equilibrium finder [Sahlin et.al., 2010] was used for scanning parameter space.

3 Results

3.1 Investigating a subset of the parameter space for the SAM model

One of the solutions already known was used as a starting point when scanning a subset of the parameter space for the full SAM model, see parameter set 3 in [Sturk, 2010]:Table 3. Four parameters were scanned starting from half their value going up to double their original value, using ten logarthimic steps per parameter.

It was expected that the solution should be invariant to parameter $K_{CLV3,2}$, related to the activation signal from WUS to CLV3. This was also found to be the case, by plotting the data and using simple sensitivity analysis with the help of the ANN. The remaining parameter, parameter $K_{CLV3,2}$, was held at the value closest to its value in the original solution, though its exact value should not matter since the outcome of the model was invariant to this parameter.

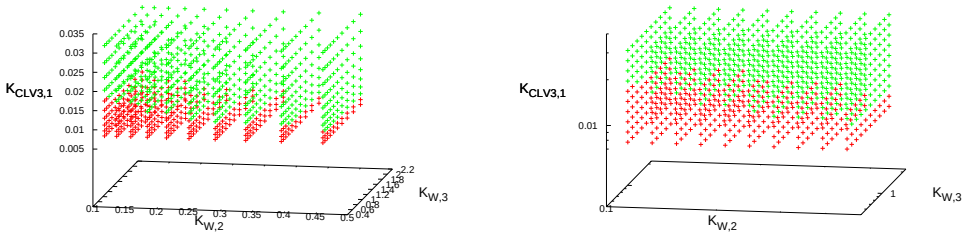

Figure 5: Solution space for three of the parameters of the organism model, consisting of good [red] and bad [green] solutions, plotted in **(A)** linear space and **(B)** log space. The good solutions looks like they could be easily demarcated.

Logarithmised data	AUC	BAC
No	0.997	0.968
Yes	1.000	0.990

Table 1: Performance of an ANN trained on a subset of parameters of the organism model, measured on a test set.

It can be seen that the good solutions form a coherent region, and it looks like it could be well demarcated by chosing an proper surface for the job (Figure 5). Looking at the data suggests that a simple plane could to the job, therefore a simple ANN with no hidden nodes, in effect dividing space into two regions using a plane, could do the job. The data looks like an easy classification problem both in linear and log-space, cf. Figure 5:A and 5:B.

It was found that the ANN could handle the task of classifying the data into good or bad solutions without any problems. Also, by logarithmising the data before presenting it to the ANN, the performance of the network was found to be just as high. The end result can be found in Table 1. It can be noted that the network trained on the logarithmised data did slightly better, but the difference is too small to draw any conclusions from this.

This region of good solutions are perhaps too trivial a task to complete, as indicated by the good performance of the simple ANN used here. However it shows that the network behaves as expected, and that a simple problem can be solved very well using the simplest possible ANN.

3.2 Investigating a subnetwork of the full SAM model

The parameters directly determining the concentrations of KAN in the model was varied while treating the other aspects of the model, except the concentration of KAN itself, as being constant. The differential equation governing the change in the KAN concentrations are shown in equation (3). Terms inside brackets represents the concentrations, here wS_i is the inhibitory signal coming from WUS,

Parameter	Min value	Max value
V_{max}	0.070	11.68
K_{WUS}	0.001	7.30
n_{WUS}	0.500	4.00
K_Y	0.001	7.30
n_Y	0.500	4.00
d_{KAN}	0.050	0.05

Table 2: Region of parameter space investigated for the organism model.

and Y is the inhibitory signal coming from CLV3, while K is the expression of KAN. The parameters in equation (3) was scanned in order to provide data for the ANN to train on.

$$\frac{d[\mathrm{K}]}{dt} = - d_{KAN} \cdot [\mathrm{KAN}] + V_{max} \cdot \frac{K_{WUS}^{nWUS}}{K_{WUS}^{nWUS} + [\mathrm{wS}]_i^{nWUS}} \cdot \frac{K_Y^{nY}}{K_Y^{nY} + [\mathrm{Y}]^n} \tag{3}$$

The last term in the above equation is built from two Hill functions. The K parameters here are thus acting as Michaelis-Menten constants, determining the center of their Hill function. Starting at a low K should make the factor saturated, meaning that changes in corresponding concentrations should not make any practical difference. For example, by starting at $K_{WUS} = 0.001$ a very low concentration of WUS suffices to have a large inhibitory effect on the KAN concentration. Increasing the concentration of WUS further from this will not make any practical difference, as the Hill function is bounded from above. Also, by having K_{WUS} at a high value (here 11.68), a large concentration of WUS is needed in order to significantly affect the KAN concentration. The K-parameters was hence varied in a way that should cover many different behaviours of the model.

By changing the degradation term d_{KAN}, it was found that the overall error was shifted considerably. But systems with different degradation terms can be seen as equivalent, as increasing the degradation term gives rise to the same equilibrium state as a corresponding decrease of the activation parameters would give. This can be seen by setting equation (3) to zero, which is the case after finding an equibilibrium, the dimension of this equation can easily be reduced by dividing it with the degradation term. Since we are mainly interested in the good solutions it is best to choose the value of the degradation term which provides the best solutions i.e., the lowest overall error. To this end the degradation term was set to the value 0.05, and the resulting region that was investigated is specified in Table 2.

It was found that by logarithmising the data before presenting it to the ANN, classification performance increased considerably. Since an ANN is non-linear this is somewhat suprising, as it could be argued that the ANN should be able to accommodate the exponential differences in the input data. However, by translating the data into logspace, the region with good parameters may become more easy to discern for the network, making it easier to find surfaces separating this region. As before, the data is also normalized to have zero mean and unit variance.

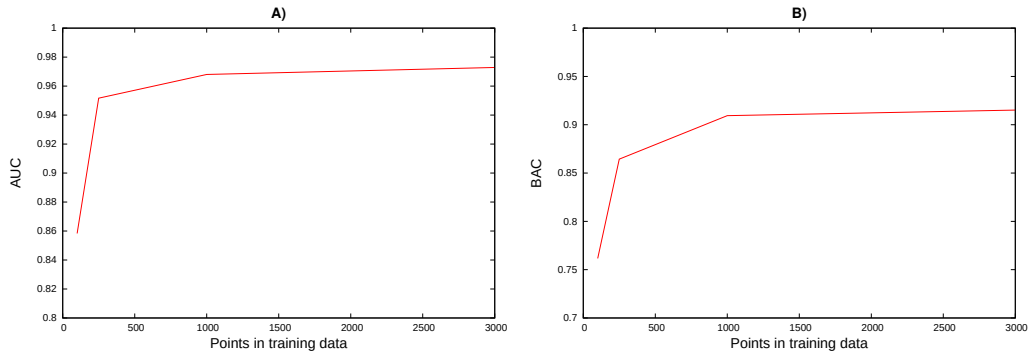

Figure 6: Performance of the ANN on a prospective test of new randomly drawn data points, showing **(A)** the AUC and **(B)** the BAC on a test set as a function of the number of training data, comparing the ANN's prediction with actual results from simulations.

Number of points	Points per parameters
3000	5.0
1000	4.0
250	3.0
100	2.5

Table 3: How the total number of points relate to the sampling frequency

The network was trained on sets of randomly drawn data of different sizes, in order to estimate how many points that are needed to provide enough information for the network. It was noted that smaller sets of data lead to a better training result, showing that the fewer points gives the network the possibility of matching the given training data very well, although the generalization performance should decrease. The network found to provide a good combination of performance and parsimony had four hidden nodes and five input nodes, one for each input parameter, excluding the degradation term. This gives the network about twenty adjustable weights, meaning that, as a rule of thumb, at least 200 data points should be used to avoid overtraining. Ten networks of the same architecture and training on the same data was used as an ensemble, whose averaged output was used as the predicton.

When increasing the number of data given to the network to train on, the performance is increased, although the increase seems to flatten out for large numbers of training data. For estimating this generalization performance, a test set of 1000 points independent of the training data was used. Training sets consisting of 100, 250, 1000 and 3000 points was tested. An ANN training on a data set with 100 points manages to find some structure of the data, although by increasing the training set size to 250, there is a significant increase in performance (Figure 6). The AUC jumps from 0.86 to 0.95 when increasing the number of points from

Parameter	Min value	Max value
K3	0.355	35.5
K6	0.359	35.9
K	0.145	14.5
n	0.166	16.6

Table 4: Investigated region in CLA-WUS signal model.

100 to 250, which is a significant boost in performance. Increasing the number of training points further do increase the AUC, but it looks like the AUC is bounded by the value 0.98 (approximately). When training on 3000 points, this corresponds to the same number of points as a sampling of parameter space of five points per parameter would give. In this way the sampling frequency can be related to the performance of the network, see Table 3 for an idea of how the size of the training data relates to the sampling frequency. For this data, 3 points per parameter seems to be enough to get a high AUC (almost 0.96) classifier. It should however be noted that it is equally or even more important to choose points at the right places, i.e., where the shift from good to bad solutions take place.

After a network is trained, it passes its judgement on the data faster than the time it would take to do a full simulation, this is one of the main reasons for using an ANN. Here, the simulation time for the subnetwork was relatively fast compared to that of the real model. Even so, the processing time for the ensemble of the ten networks used for the above results was 0.018 of the time it took to obtain these values with the computer simulation.

3.3 Evaluating parameter space for the CLV-WUS signal model

In the model investigating the CLV-WUS signalling pathways, an energy minima was chosen at random and a region around this point was scanned. The range of the scanned parameters can be found in Table 4 and the full parameter set used as the starting point can be found in the appendix (Table 8).

Plots of snapshots of the scanned four-dimensional parameter space shows how the distribution of good points crawls when parameter K is shifted (Figure 7). For K values less than 1.78 this distribution qualitatively looked the same for all K's, suggesting that this corresponds to the transition to a K small enough to make the correspoding Hill function saturated for all possible concentrations of X (the signal molecule repressing WUS via the Hill function). The class distribution was skewed towards bad solutions; the ratio between good and bad solutions was about 1:30 in this data set (Figure 7). As such, only the ROC curve and AUC are used here as measures of classifier performance as they are independent of class skew.

Although it is suggested that the class of a parameter set depends on all of its parameters when plotting parameter space (Figure 7), it was found that an ANN using three of the parameters as input, disregarding the K parameter, behaved in roughly the same way as an ANN using all four parameters for prediction. Com-

Number of input parameters	AUC (training set)	AUC (validation set)
3	0.83	0.79
4	0.83	0.77

Table 5: Networks with four hidden nodes trained with undersampling. The AUC is derived from the average output of ten networks.

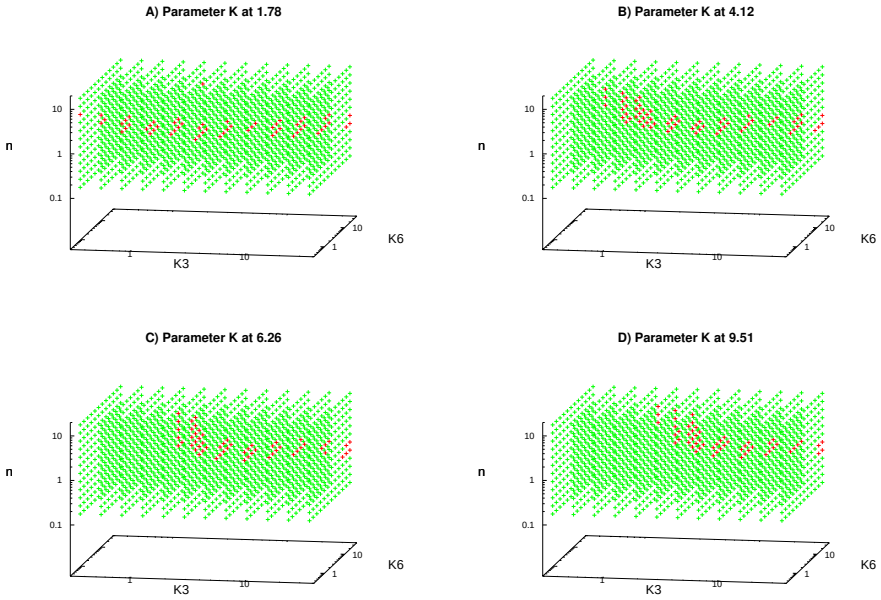

Figure 7: Logplots of snapshots of parameter space for the CLA-WUS signal model, a [red] point is a good solutions, a [green] point is a bad solution. The good solutions move away from low K6 and K3 values when K is increased. Note the stray red point in **(A)**, such isolated red points were found for all K's less than 1.78.

paring the ROC-curves in Figure 8 between the classifiers using three respectively all four scanned parameters as input, it is hard to notice any difference, in fact the ROC areas can be seen to be the same to the third decimal. It was only possible to discard the K parameter, taking away any of the other parameters led to a considerable decrease in performance.

Since there was a skew towards bad solutions (zeros) in this data set, training the ANN with an undersampling of zeros were tested. Both an ANN with all four inputs and an ANN using only the three input parameters K3, K6 and n were trained by undersampling the zeros in the data, but no improvement of the performance was seen. In fact, both networks performed worse already at the training phase (Table 5), meaning that training without undersampling seems to find a better decision boundary. Note that, as for the above result, both the three input network and the four input network trained with undersampling behaved in a roughly equivalent way.

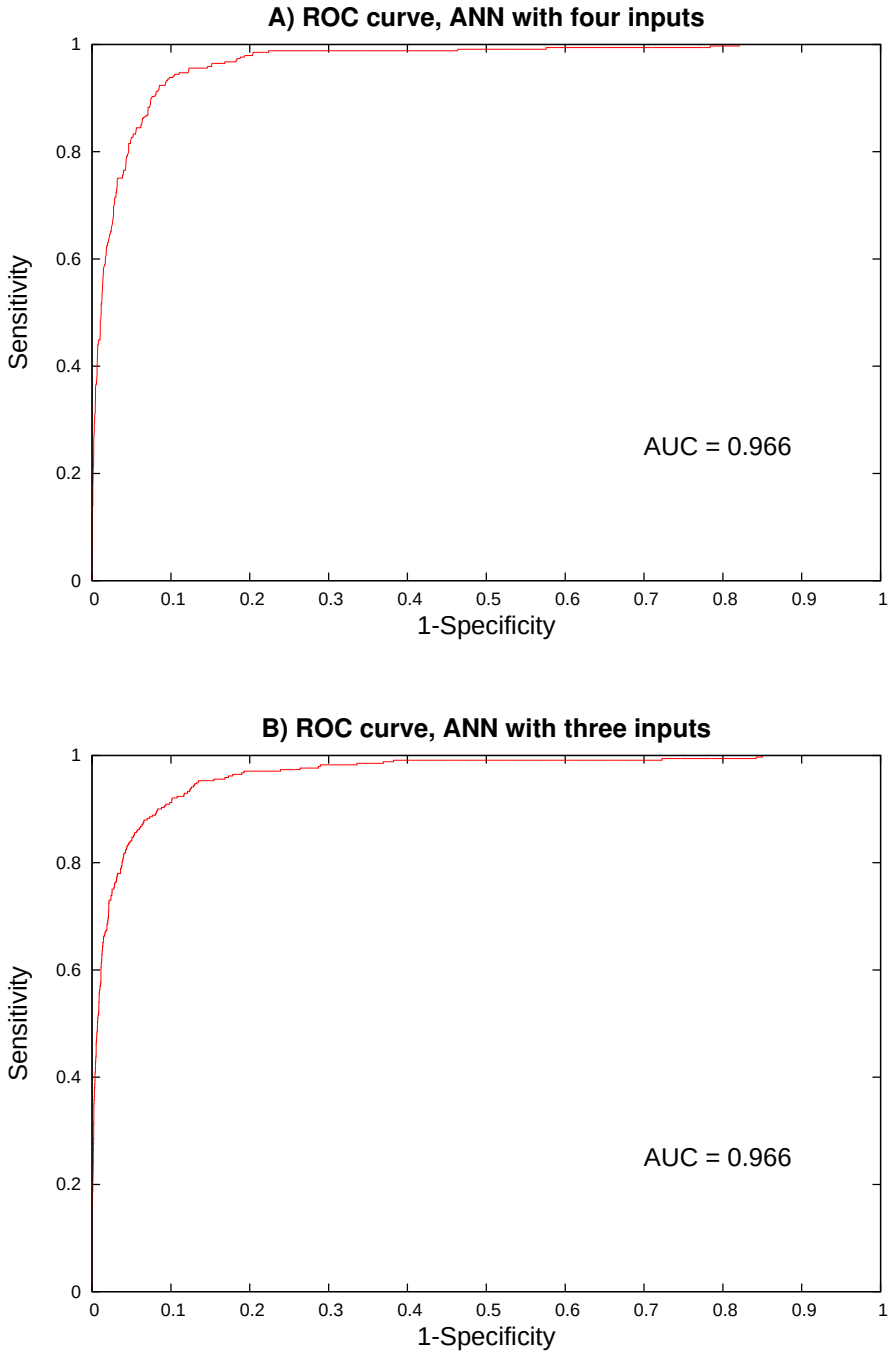

Figure 8: ROC curves for the networks predicting the class on a test set using **(A)** only three of the input parameters and **(B)** using all four input parameters.

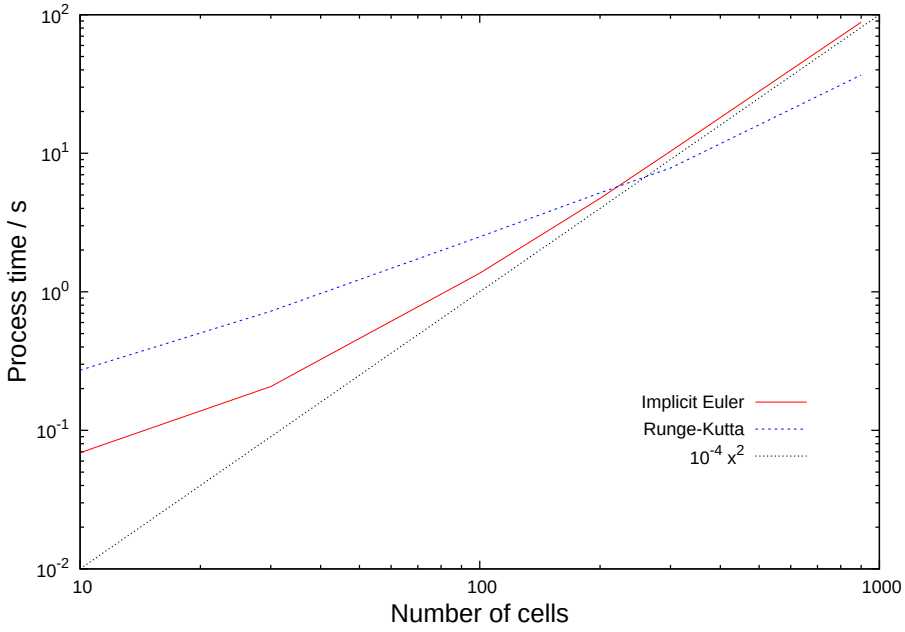

Figure 9: Process time for a model simulating diffusion between cells.

3.4 Performance of the semi-implicit solvers

The evaluation of the inverse matrix in equation (1) was found to be problematic, since the Jacobian matrix of the system typically is very large. The vector **y** representing the state of the model contains concentrations of the substances for all of the cells. With 189 cells and 9 different species the size of the system is close to 2000. Each of these variables can in theory affect any other of the variables, making for a Jacobian of magnitude 10^6 and thus a large matrix that needs to be inverted. Though in practice the cells generally only affects themselves and their neighbours, implying that the Jacobian should be sparse i.e., mostly consisting of zeros. For this reason the sparse system solver [Davis T A, 2004] was used when inverting this matrix, in hope of exploiting the sparseness for a decreased simulation time.

However, even using a sparse matrix solver, the computational time for the SIE method seems to scale as N^2, where N is the system size (Figure 9). The system here consisted of a one-dimensional line of cells starting out with an unevenly distributed substance that was allowed to diffuse into neighbouring cells. It can be seen that the explicit Runge-Kutta solver scaled better with the size of this system, showing a linear increase in simulation time. Of note is the large stepsize used by the SIE method in this system, a step size of 100 was succesfully (no divergent concentrations) used, although in a more complex system steps of this length are unlikely to work.

It was further noticed that the SIE method did even worse when increasing the dimensions to two for the diffusion system above. Keeping the number of cells constant at 100 but using different topologies, SIE did significantly worse when

Method	Topology	Time/s
SIE	1D	9.8
SIE	2D	13.1
RK	1D	4.3
RK	2D	4.4

Table 6: Computational time when solving two different 100 cell diffusion systems of different topologies.

simulating a two-dimensional 10×10 system (2D) compared to a one-dimensional line of 100 cells (1D) (Table 6). This can be explained by the increased number of neighbours in a two-dimensional versus a one-dimensional grid. For both solvers an increased number of neighbours means more diffusion to calculate for, but it is only SIE that sees a substantial decrease in performance. This suggests that it is the additional step of inverting the matrix which limits the SIE.

By carefully evaluting and running the code, it was confirmed that it really was the sparse matrix inversion that bottle-necked the performance of the SIE solver. The other calculations done by the solver were negligible compared to this process. Thus, to be able to successfully use an implicit method focus should be on finding a better way to invert the sparse matrix. It can be thought that it is beneficial if there is some order to the positions of the non-zero elements, since a sparse matrix with very scattered non-zero elements may be harder to invert. By sorting the input data such that cells that are close together in the meristem are close together in the vector used by the numerical solver, imposing some order in the sparse matrix to be inverted, considerable time savings were actually achieved. Though this sorting gave a large boost (at least a factor ten) in performance the solver still only managed to raise its status from being atrociously slow to just slow, compared to the RK solver.

Aside from the SIE solver, a higher-order Rosenbrock stiffly stable (ROSS) solver was implemented in order to be able to take longer step lengths. The ROSS solver is a generalization of the Runge-Kutta method, where the derivatives are evaluated at several carefully chosen steps between the current and the next time step. In effect this means that there are some more linear equations to be solved, but the problematic issue of inverting the matrix only have to be done once every time step, as for the SIE solver. Since it was found that the other calculations were negligible compared to this process, even after sorting the data, the extra matrix multiplications carried out by the ROSS solver should not make any large difference in processing time. In fact, when simulating the full model the increase in processing time when using ROSS instead of SIE was found to be as small as two percent. Therefore, the two solvers SIE and ROSS can be considered to be equally fast (read slow), differing only in that ROSS is of higher order.

The RK solver had problems in one of the regions, where the time for simulating one timestep was more than a factor 40 larger compared to an easy region (Table 7). In this hard region the RK method needed to take very small time steps in order for the solution to remain accurate and stable, suggesting that this

Point (difficulty)	Time in seconds for simulating one timestep with RK
Parameter 3 (easy)	0.05
Parameter 2 (hard)	2.11

Table 7: Two different sets of parameters, the RK solver having problems working through one of them.

region in parameter space results in a stiff problem. This was one of the initial reasons for implementing an implicit solver, its stability properties should make it accurate and stable even at long step lengths in stiff regions.

Given the current performance of the implicit solvers and the RK solver, it is possible to ask how long the step lengths for the implicit solvers need to be in order to get a better performance. By calculating the time needed for simulating one timestep for each of the solvers and comparing these numbers, it was found that in the easy region a steplength of 112 steps is needed and in the hard region a steplength of 4 steps is needed of the ROSS solver to be faster than the RK solver. Alas the longest possible steplength was found to be too short; using steplengths longer than 0.1 was made the solver unstable when simulating using the ROSS solver. It can be noted that the SIE solver, as opposed to ROSS, was not able to remain stable even for a stepsize of 0.1.

4 Discussion

The process of training a network was straighforward, given that a list of data points and their class, separated into training and test sets, is at hand. Given this list and a training algorithm of choice, it was found that only the number of hidden nodes seemed to make a considerable impact on the performance of the network. Other parameters that were used when training and specifying the network only needed to be tuned and optimised once, after which altering these parameters (within an order of magnitude) did not seem to make a big difference on the performance of the network and training algorithm. This suggests that a default network training algorithm can be created, that optimises performance by just tuning the number of hidden nodes, using default values everywhere else in the training algorithm. However, such an automated network trainer would in practice need some kind of supervisor (preferably human), although it could provide with a quick and easy way of obtaining a numerical predictor from a list of data points. To simplify things further, the pre-processing of the data sets could also be streamlined when only working within one model.

Here, the prediction time of an ANN was only compared with the simulation times for a smaller subnetwork of the full model, when using an ANN on a larger model the savings made in time should be even larger. Adding additional input nodes for each new parameter to the ANN will make the complexity of the network grow linearly, while adding a new term to the model probably increases the model's

complexity beyond linearity. An argument for this is that the full model needs more time steps to reach equilibrium than needed by the subnetwork. Thus, in the full model the number of mathematical operations carried out for each step forward in time should increase approximately linearly with the number of parameters, but on top of that the total number of time steps needed to reach equilibrium are also increased, meaning that the simulation time will increase more than the prediction time of the ANN.

Two parameters were found to be redundant for the prediction of the ANN, in the first case this was to be expected, but the second case was a suprise. In the second case, the K parameter in the CLV3-WUS signalling model, the plots of parameter space looked different for different K. Still, a network which did not take the K value in account did just as well as a network taking all four parameters into account. Given a point with a certain K3, K6 and n value, its class may be different when shifting the K value (Figure 8), meaning that there should not be enough information in only the K3, K6 and n parameters for the network to determine the class, but this is what the ANN was found to do. However, it was observed that for K values less than 1.78 the parameter space looked approximately the same, this implies that about $\frac{7}{12}$ of K3-K6-n-K space is invariant to K.

It could be that the error made by missclassifying some of the points for the five snapshots dependent of K is negliable. If this would be the explanation, such an result may still be useful. By just looking at plots of the data, it seems necessary to take parameter K into account in order to get a good prediction of the class. It is not obvious that this parameter can be discarded for a negliable error. A network only using three inputs are more parsimonious than a network using four inputs, so if these are indistinguable – even if it is due to an error being negliable – it is always better to use the three input network.

The process time of the implicit solver was found to scale as N^2, where N is the system size, while the number of non-zero elements in the Jacobian matrix relevant to the linear system to be solved should scale as N, since cells only directly effect their immediate neighbours. It is plausible that a more efficient linear system solver can be constructed such that it scales linearly with number of non-zero elements in the Jacobian, hence scaling linearly with the system size. If such an algorithm would be implemented, a huge boost in simulation performance could be achieved.

5 Conclusion

Neural networks were able to provide numerical predictions significantly faster than simulating the model to equilibrium. Because of the exponential increase in volume when adding parameters to scan for, the step of collecting enough training data for the network can, for high-dimensional space, be a hard task. However, this is a problem for complex models in general and part of the motivation of using an ANN for numerical prediction. An ANN can be used to better exploit all of the information in a scanned region, in effect increasing the benefit of scanning parameter space by providing a numerical prediction for the complete scanned

region.

It was found that good solutions did exist as coherent regions in the scanned parameter space for the models considered, with the exception of a few scattered points for the CLV3-WUS signalling model. The existence of coherent regions was evident from the high AUC on tests sets of the networks and by plotting parameter space. For higher dimensions, it could be that the boundary between good and bad solutions becomes more complex, but since the networks used here had relatively few nodes, less than or about the same as the number of dimensions of the input data, there should be room for handling more complex surfaces if that would be the case.

The implemented implicit solvers were more stable, i.e., longer timesteps were possible, than the explicit RK solver, but the large linear system to be solved limited the performance of the implicit solvers. It was found that the process time of the implicit solver was proportional to N^2, N being the system size, which for large systems cancels the timesavings made by being able to take longer timesteps. More work is needed on implementing a more efficent solver for (sparse) linear systems.

6 Appendix

6.1 The equations of the SAM model

The equations governing the WUS, CLV3 and KAN concentrations are

$$\begin{aligned}
\frac{d[\mathrm{WUS}]}{dt} &= -\, d_{WUS}[\mathrm{WUS}] + V_{maxW} \cdot \frac{[\mathrm{A}]^n}{K_{W,1}^n + [\mathrm{A}]^n} \cdot \frac{K_{W,2}^n}{K_{W,2}^n + [\mathrm{Y}]^n} \cdot \frac{K_{W,3}^n}{K_{W,3}^n + [\mathrm{kS}]^n} \\
\frac{d[\mathrm{CLV3}]}{dt} &= -\, d_{CLV3}[\mathrm{CLV3}] + V_{maxCLV3} \cdot \frac{K_{CLV3,1}^n}{K_{CLV3,1}^n + [\mathrm{kS}]^n} \cdot \frac{[\mathrm{wS}_a]^n}{K_{CLV3,2}^n + [\mathrm{wS}_a]^n} \\
\frac{d[\mathrm{KAN}]}{dt} &= -\, d_{KAN}[\mathrm{KAN}] + V_{maxKAN} \cdot \frac{K_{KAN,1}^n}{K_{KAN,1}^n + [\mathrm{wS}_i]^n} \cdot \frac{K_{KAN,2}^n}{K_{KAN,2}^n + [\mathrm{Y}]^n}
\end{aligned}$$

Each of the above equations consists of a degradation term and a chain of Hill functions dependent on signal molecules.

The species CLV3 and KAN have their own signal molecule, Y and kS respectively, while WUS promotes the two signal molecules wS_a and wS_i, since WUS function both as an activator and a repressor. The equations for the signal molecules are

$$\begin{aligned}
\frac{d[\mathrm{kS}]}{dt} &= -\, d_{kS}[\mathrm{kS}] + k_{kS}[\mathrm{KAN}] + D_{kS}\Delta[\mathrm{kS}] \\
\frac{d[\mathrm{wS}_a]}{dt} &= -\, d_{wS_a}[\mathrm{wS}_a] + k_{wS_a}[\mathrm{WUS}] + D_{wS_a}\Delta[\mathrm{wS}_a] \\
\frac{d[\mathrm{wS}_i]}{dt} &= -\, d_{wS_i}[\mathrm{wS}_i] + k_{wS_i}[\mathrm{WUS}] + D_{wS_i}\Delta[\mathrm{wS}_i] \\
\frac{d[\mathrm{Y}]}{dt} &= -\, d_{Y}[\mathrm{Y}] + k_{Y}[\mathrm{CLV3}] + D_{Y}\Delta[\mathrm{Y}]
\end{aligned}$$

Each of the equations governing the signal molecules involves a degradation term, a growth term proportional to the concentration of the corresponding species of the signal molecule and a diffusion term enabling passive transport of the signals between cells. The factor Δ[C] in the diffusion term is the difference in concentrations of C between adjacent cells.

Finally, the activator network is modelled with a Brusselator [Prigogine and Lefever, 1968], tuned such that the concentrations of A has a spike in the center of the SAM in order to spatially determine the OC. The equations are

$$\begin{aligned}\frac{d[\mathrm{A}]}{dt} &= a - (b+\beta)[\mathrm{A}] + c[\mathrm{A}]^2[\mathrm{B}] - d[\mathrm{A}] + D_A\Delta[\mathrm{A}] \\ \frac{d[\mathrm{B}]}{dt} &= b[\mathrm{A}] - c[\mathrm{A}]^2[\mathrm{B}] + D_B\Delta[\mathrm{B}]\end{aligned}$$

6.2 The equations of the CLV3-WUS signalling model

The scanned parameters K, n, k_3 and k_6 are all involved in the promotion of WUS; K is the Michaelis-Menten constant in a Hill function and n is the exponent, the equations directly effected by these parameters are

$$\begin{aligned}\frac{d[\mathrm{WUS}]}{dt} &= k_7\frac{K^n}{K^n + [\mathrm{X}]^n} - d_W[\mathrm{WUS}] \\ \frac{d[\mathrm{X}]}{dt} &= t_4\big(s_4 - [\mathrm{X}]\big) + k_3[\mathrm{CLV1/CLV3}] + k_6[\mathrm{CRN/CLV3}]\end{aligned}$$

k_3 and k_6 set the strength of the CLV1 and CRN pathways acting on WUS – via the signal molecule X – while K and n defines the midpoint and shape of the Hill function representing X's repression of WUS. The equations governing CLV1, CLV3, CRN, and the complexes CLV1/CLV3, CRN/CLV3 are

$$\begin{aligned}\frac{d[\mathrm{CLV1}]}{dt} &= t_1\big(s_1 - [\mathrm{CLV1}]\big) - k_1[\mathrm{CLV1}]\cdot[\mathrm{CLV3}] + k_2[\mathrm{CLV1/CLV3}] \\ \frac{d[\mathrm{CRN}]}{dt} &= t_2\big(s_2 - [\mathrm{CRN}]\big) - k_4[\mathrm{CRN}]\cdot[\mathrm{CLV3}] + k_5[\mathrm{CRN/CLV3}] \\ \frac{d[\mathrm{CLV3}]}{dt} &= t_3\big(s_3 - [\mathrm{CLV3}]\big) + k_W[\mathrm{WUS}] - k_1[\mathrm{CLV1}]\cdot[\mathrm{CLV3}] \\ &\quad + k_2[\mathrm{CLV1/CLV3}] - k_4[\mathrm{CRN}]\cdot[\mathrm{CLV3}] + k_5[\mathrm{CRN/CLV3}] \\ \frac{d[\mathrm{CLV1/CLV3}]}{dt} &= k_1[\mathrm{CLV1}]\cdot[\mathrm{CLV3}] - k_2[\mathrm{CLV1/CLV3}] - t_1[\mathrm{CLV1/CLV3}] \\ \frac{d[\mathrm{CRN/CLV3}]}{dt} &= k_4[\mathrm{CRN}]\cdot[\mathrm{CLV3}] - k_5[\mathrm{CRN/CLV3}] - t_2[\mathrm{CRN/CLV3}]\end{aligned}$$

The above equations govern the wild type plant, the plant mutants are modifications of these equations in some way [Sahlin et.al., 2010]. For example, the *clv1-11* null mutant is modelled by removing all presence of CLV1, and the *clv1-1* mutant is, for the loss-of-signal model considered here, modelled by exchanging the K3 parameter with the K3WEAK parameter.

6.3 Parameter set used for the CLV3-WUS signalling model

Parameter	Value
K1	4.02179
K2	2.1154
K3	3.55378
K4	0.648106
K5	1.85869
K6	3.58846
K7	2.54869
K	1.44599
N	1.6586
T1	0.483809
S1	1.6867
T2	0.151979
S2	4.03733
T3	2.60364
S3	0.716203
T4	0.62934
S4	1.75982
DW	1.12363
KW	2.72028
K3WEAK	0.26239

Table 8: Parameter set used as the starting point when scanning parameters in the CLA-WUS model. K3WEAK replaces K3 for the *clv1-1* mutant in the loss-of-signal model considered here.

7 References

[Sablowski, 2007] Sablowski R, The dynamic plant stem cell niches, Curr Opin Plant Biol, 2007;10(6):639–644

[Williams and Fletcher, 2005] Williams L and Fletcher J C, Curr Opin Plant Biol, 2005;8:582–586

[Sturk, 2010] Sturk C, A study of the shoot apical meristem by means of computational modeling and microarray analysis [master's thesis]. Lund: Computational biology and biological physics division at Lund university; 2010 Jan 16

[Press, 2007:1] Press W, Stiff Sets of Equations. In: Numerical recipes. 3rd ed. Cambridge Univ Pr. Print. 2007 pp.931–940

[Press, 2007:2] Press W, Numerical recipes. 3rd ed. Cambridge Univ Pr. Print. 2007

[Heisler and Jönsson, 2007] Heisler M G and Jönsson H, Modelling meristem development in plants, Curr Opin Plant Biol, 2007;10:82–97

[Prigogine and Lefever, 1968] Prigogine I and Lefever R, Symmetry breaking instabilities in dissaptive systems, J Chem Phys, 1968;48:1695–1700

[Jönsson et.al., 2005] Jönsson H et.al., Modelling the organization of the WUSCHEL expression domain in the shoot apical meristem, Bioinformatics, 2005;21(1):232–240

[Müller et.al., 2008] Müller R et.al., The Receptor Kinase CORYNE of Arabidopsis Transmits the Stem Cell-Limiting Signal CLAVATA3 Independently of CLAVATA1, The Plant Cell, 2008;20:934–946

[Sahlin et.al., 2010] Sahlin P et.al., Models of sequestration and receptor crosstalk for explaining multiple mutants in plant stem cell regulation. In: Modelling the development of phyllotactic patterns at the shoot apical meristem of Arabidopsis thaliana [PhD dissertation]. Lund: Computational biology and biological physics division at Lund university; 2010 April 30 TBC...

[Haykin, 2009] Simon H, Neural networks and learning machines. 3rd ed. Upper Saddle River, N.J.: Prentice Hall; Print. 2009

[Davis T A, 2004] Davis T A, Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method, ACM T Math Software, 2004;30(2):196–199

[Jönsson and Krupinski, 2009] Jönsson H and Krupinski P, Modelling plant growth and pattern formation, Curr Opin Plant Biol, 2009;13(1–7):1–5