Fuzzy Logic for Biological Networks as ML Regression: Scaling to Single-Cell Datasets With Autograd

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present the BioFuzzNet module, a fuzzy logic tool to model signal transduction in biological networks. By equating the optimisation of the fuzzy logic transfer functions to a regression problem, we show that gradient descent is a suitable optimisation method for fuzzy logic modelling. The speed of this approach allows us to scale fuzzy logic modelling to single-cell datasets and leverage available transcriptomics data. Furthermore, the flexibility of gradient descent optimisation allows us to perform arbitrary computations, thereby enabling us to model feedback loops and fit them in simple cases. Promising results also suggest that BioFuzzNet can generate insights in the signalling network topology by identifying logical gates and spurious connections.

1 Introduction

Bio-molecular networks, such as gene regulatory networks or protein signalling network, summarise large bodies of prior knowledge about the interactions of molecular species in biological systems such as cells. The graphs indicate which species interact together and therefore encode how the cellular machinery operates and reacts to external stimuli. Boolean network modelling is a successful approach to identify which chains of interactions (pathways) connect stimuli to cellular behaviour. For this, measured and predicted values are required to be binarised as "high" or "low", describing whether the species is over-expressed or (in)activated compared to baseline levels. The species are connected through logical gates. For example, the formation of a complex of two molecules requires both components to be present. Thus, this can be described as an AND gate whose input are the two molecules and whose output describes the complex.

While Boolean logic models provide insights into relevant pathways, they lack any kind of quantitative information. This is problematic as biological data is predominantly measured on a continuum. This limitation led to the development of fuzzy logic models of bio-molecular networks, which extend the logical framework to continuous values. Fuzzy logic networks are optimized by minimising the discrepancy between the measured data and their predictions. Here, the predictions are the simulated node values given the network topology and parameters, both of which are tuned during optimization. Existing implementations, such as CNORfuzzy [MSRC+11], rely on optimization algorithms for discrete data such as genetic algorithms. Despite successes, this approach has two main drawbacks. First, the genetic algorithms as implemented choose parameters from a discrete set, which means that the explored parameter space is rather limited. Second, genetic algorithms are fairly inefficient as they explore the parameter space randomly.

Modern automatic differentiation tools such as Pytorch [PGM+19] or Jax [BFH+18], make obtaining the gradient of an arbitrary computation very simple. These tools build a computational graph dynamically, which is then used to evaluate the exact gradient of the result with respect to any input parameter. Gradient descent offers two advantages during optimization. It allows the whole

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

parameter space to be accessible and requires a single evaluation to guide updates, so it limits the total number of simulations that need to be run. Since these automatic differentiation frameworks can automatically parallelise their operations on hardware accelerators such as GPUs, they also provide the potential to improve performance by leveraging modern computing resources.

Biological networks often contain feedback loops, which can in principle be included in either Boolean or fuzzy logic networks. Nevertheless, existing approaches avoid feedback behaviour as these complicate the optimization process. However, gradient tracking can work smoothly through multiple node updates, limited only by the memory of the computing device storing the computational graph.

Wanting to improve on current fuzzy logic models, we developed `BioFuzzNet`, a fuzzy logic simulation package that exploits automatic differentiation to permit gradient-based optimization. We show with *in silico* experiments using simulated data that that we can accurately predict network behaviour. Additionally we show that our implementation can, to some extent, handle feedback loops, identify incorrect interactions in the prior knowledge and determine the type of logical gate best representing an interaction.

2 Methods

2.1 Modelling biological networks with fuzzy logic

For fuzzy logic models to handle continuous values, a Hill function is associated with each edge in the network and used to map each node value on the interval $[0, 1]$.

$$\text{Hill}_{K,n}(x) = \frac{x^n}{K^n + x^n}, \tag{1}$$

where $K > 0$, the EC50 parameter, encodes when saturation occurs and n, the cooperativity coefficient, encodes how fast saturation is reached. Note that as inputs are constrained in the $[0, 1]$ interval, the output is constrained to $[0, 1/(1 + K^n)]$, meaning that K^n can be used to completely shut down an edge.

Fuzzy logic operators describe how the Hill-transformed activations interact. These operators are analytic continuations of the Boolean logic operators.

$$\texttt{NOT}(x) = 1 - x, \qquad \texttt{AND}(x, y) = x \times y, \qquad \texttt{OR}(x, y) = x + y - x \times y. \tag{2}$$

Signal transduction is simulated via the computation of the logical function for each biological node in the network. Gradient descent constrains us to use a sequential update scheme detailed on algorithm 1 in which each node is updated if and only if all its parent nodes have been updated. For cyclic networks, this update rule is modified as described in algorithm 2 to allow continuous evolution until a steady state is reached.

Optimization occurs by repeatedly simulating signal transduction through the `BioFuzzNet`, computing the mean squared error between the predicted and observed node activities, and updating the transfer function parameters using stochastic gradient descent (see algorithm 3).

2.2 Simulating and fitting loops

When modelling loops we operate under the assumptions that there is a wide range of initialisations leading to the same steady state, and that steady states including orbits can be decently approximated by constant states.

This motivates the simple approach we have taken to optimizing models with feedback loops: we simulate their evolution over a long-enough time that a constant state or an orbit is reached. To account for orbits, we use the average of the last k steps as a prediction, where k should be suffiently large to converge. We then fit the average to the measured value using the MSE loss (see algorithm 2 and algorithm 3). For validation we use topologies shown in fig. S2. We compare fitting a converging negative feedback loop with 50 different random transfer function parameters, a negative feedback loop with oscillatory behaviour and a positive feedback loop with bistable behaviour. As a control

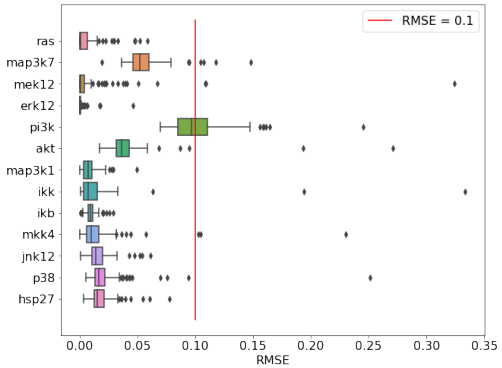


Figure 1: Boxplot of the RMSE on the validation set for 97 replicates when fitting a `BioFuzzNet` to bulk data simulated using `CNORfuzzy`

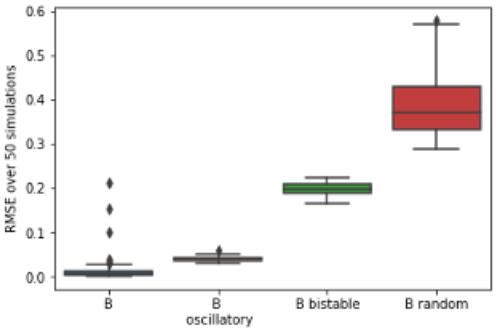


Figure 2: Boxplots of RMSE after 50 replicates of fitting a `BioFuzzNet` to a random negative loop, an oscillatory loop, and a bistable loop

we also compare a random vector to data generated with a negative feedback loop with 50 different random transfer function parameters (see appendix A.2 for details).

2.3 Data-driven logical gate determination

In contrast to existing fuzzy logic software, the `BioFuzzNet` module is not explicitly built for topology inference. Nonetheless, we provide a logical operator which interpolates `AND` and `OR` gates:

$$\texttt{AND_OR}_{\alpha}(x, y) = \alpha\texttt{AND} + (1-\alpha)\texttt{OR}, \tag{3}$$

where $\alpha \in [0, 1]$ is a tunable weight that is jointly optimized with the Hill function parameters to minimize the MSE loss function. We encourage α to be either close to 0 or 1 by adding the penalty $\alpha(1-\alpha)$ to the loss function (see Equation 5). An α close to 1 would indicate an `AND` gate and an α close to 0 an `OR` gate. We tested the performance of this gate identification by simulating synthetic data on randomly-initialized fuzzy networks with definite logical gates and then fitting this data to a network with interpolating gates. We then verified whether the optimization process correctly identified the interpolating gates as nearly-pure `AND` or `OR` gates (see appendix A.2 for details).

2.4 Detecting errors in prior knowledge networks

We simulated data using a `BioFuzzNet` with a given topology (see Figure S4), and fitted an incorrect prior knowledge network to the data we obtained. The incorrect topologies contained spurious edges added to the original topology using a `AND` or an `OR` gate (see appendix A.2 for details).

3 Results

3.1 SGD optimized fuzzy logic models accurately predict simulated data

We first wanted to evaluate `BioFuzzNet` on bulk data. Therefore, we simulated bulk data with the aforementioned `CNORfuzzy` tool using Boolean inputs. `BioFuzzNet` showed a good performance on predicting the node levels. As shown in Figure 1, the median root mean squared error (RMSE) at each node is lower than 0.1. `PI3K` stands out due to its high RMSE, which is most probably due to its high in-degree, with seven incoming edges (see Figure S1).

3.2 Simulating and fitting cyclic topologies

We can model different types of behaviours typical of cyclic topologies, including converging, oscillatory and bistable behaviours (fig. 2, fig. S2 and appendix A.2). As the current optimization method supports only a single steady-state value for each cell, our method can fit converging behaviours accurately but has difficulties fitting oscillatory and bistable behaviours.

3.3 Determining logical gates by tuning `AND-OR` interpolators: preliminary results

We implemented a heuristic method for learning the optimal choice between `AND` and `OR` gates (see section 2.3). The preliminary tests are performed by simulating synthetic data on fuzzy networks with definite logical gates and then fitting this data to a network with only interpolating gates. We then verified whether the optimization process correctly identified the interpolating gates as `AND` or `OR` gates. For all interpolating gates, the parameter determining its gate type was either 0.9991 or 0.0009, indicating an `AND` and an `OR` gate respectively. We ran 10 replicates, each time predicting all 9 gates of the chosen topology (see fig. S3). An average of 1.1 gates were incorrectly identified; with a maximum of two incorrectly identified gates in one replicate. In two out of the ten replicates, all logical gates were correctly identified. Concurrently, the node predictions remained accurate: the maximum observed RMSE at a node over all ten replicates was $6.26 \cdot 10^{-3}$.

The ability to correctly predict the logical gate type is independent of how far downstream the gate is from in the inputs. Rather, `OR` gates are more often incorrectly predicted than `AND` gates. Over all replicates, 10 out of the 11 incorrectly identified gates were `OR` gates incorrectly predicted as `AND` gates.

3.4 Optimizing topologies by eliminating spurious edges: preliminary results

We show some very preliminary results that suggest incorrect prior knowledge edges can be detected and suppressed. We added an incorrect edge to the topology used for simulation and connected it to its target node with an `OR` or an `AND` gate (see section 2.4). In the first case the edge was correctly turned off: only the signal of the true incoming edge passes through the `OR` gate (see eq. (2)). The fitted parameters of the Hill function at the incorrect edge were $n = 6.72$ and $K = 3.16$, thus the maximum value that the transfer function can take on the range $x \in [0, 1]$ is $4 \cdot 10^{-4}$. In the case of the `AND` gate, the edge was correctly turned ON: since this gate is modelled with a multiplication we want the incorrect edge to be close to 1 in order for the correct edge to pass its signal (see eq. (2)). The parameters of the associated Hill function are $n = 2.49$ and $K = 1.77 \cdot 10^{-3}$. In both cases the RMSE on the predictions of the target node of the incorrect edge remained low, $2.44 \cdot 10^{-2}$ in the `OR` case and $2.78 \cdot 10^{-2}$ in the `AND` case. Other preliminary experiments run on a larger network also yielded encouraging results. All edges were correctly turned ON or OFF as expected. The maximum RMSE at the target node was on the order of 10^{-2} as well. This suggests that the `BioFuzzNet` module could be able to correct for errors in the prior knowledge, though a systematic study would be necessary to draw robust conclusions.

4 Discussion

Toward efficient fuzzy network training The optimization of fuzzy logic networks using gradient descent is expected to significantly improve the convergence and performance of these models. Gradient-based optimization typically requires fewer model evaluations than genetic algorithms, and our `PyTorch`-based implementation can leverage the power of modern hardware accelerators. While a systematic comparison to existing software is necessary, our anecdotal experience has shown that `BioFuzzNet` scales to large-scale data such as single-cell omics datasets much more manageably than alternatives.

Modelling and fitting feedback Our preliminary results on synthetic data from a fixed network give encouraging results that `BioFuzzNet` can fit a wide array of cyclic graph topologies and parameters. We plan on testing our model on both a wide array of random networks and real data to confirm the solidity of our results.

Continuous alternatives to discrete optimization Genetic algorithms have the clear advantage over gradient-based optimization of having much weaker constraints. Nevertheless, our preliminary results show that discrete data such as graph topology and gate choice can be made smooth and successfully learned, at least in simple cases such as the examples we tested. We cannot yet claim that this approach is successful in a wide array of situations, but hope to confirm this rigorously in upcoming work.

References

[BFH+18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[KB14] Diederick P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[MSRC+11] Melody K. Morris, Julio Saez-Rodriguez, David C. Clarke, Peter K. Sorger, and Douglas A. Lauffenburger. Training Signaling Pathway Maps to Biochemical Data with Constrained Fuzzy Logic: Quantitative Analysis of Liver Cell Responses to Inflammatory Stimuli. *PLoS Computational Biology*, 7(3), 2011.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[Sae22] Saez-Rodriguez group. CellNOptR: Models and Documentation, 2022.

[SMO+03] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, 2003.

A Appendix

A.1 Optimization of a `BioFuzzNet`

Optimization occurs through a stochastic minibatch gradient descent conducted using the ADAM optimizer [KB14]. The loss function used in the core optimization script is:

$$\text{Loss}_{\texttt{BioFuzzNet}}(B, I, n, b) = \frac{1}{b}\sum_{k=1}^{b}\left(\frac{1}{n}\sum_{i=1}^{n}(B(I)_i - y_i)^2\right), \tag{4}$$

with B the network topology, I the input combinations for multiple cells, n the number of cells simulated and b the batch size. $B(I)_i$ represents the predicted values for a given batch at observed node i and y_i the measured values at this node for the batch.

A modified loss function with an added regularisation term is used when fitting a `BioFuzzNet` with interpolated gates:

$$\text{Loss}_{\texttt{BioMixNet}}(B, I, n, b, m) = \text{Loss}_{\texttt{BioFuzzNet}}(B, I, n, b) + \sum_{j=1}^{m}\left(\alpha_j \cdot (1 - \alpha_j)\right), \tag{5}$$

where m is the number of interpolated gates in the network and all other notations are identical to those in eq. (4). The function of the regularisation term is to constrain the interpolated gates to be either `AND` or `OR` gates after optimization.

A.2 Simulation parameters

Data simulated with `CNORfuzzy` The data and prior knowledge network from the `LiverDREAM` dataset [MSRC+11] (accessible at the `CellNOpt` website [Sae22]) was used as input to the `CNORfuzzy` software to obtain a network topology. The default `CNORfuzzy` parameters were used, except for the `sizeFac` parameter which was set to 0.1. We chose one of the obtained topologies with 4 input nodes (see Figure S1). We then simulated data using `CNORfuzzy` with this topology and different Boolean inputs, covering all combinations with a single activated input node, all combinations with two activated input nodes, all combinations with three activated input nodes, as well as the case where no input node is activated, for a total of 15 different input combinations.

Predicting simulated data To evaluate the accuracy on simulated data, we ran a Monte-Carlo cross-validation and fit 100 `BioFuzzNet` networks to the data simulated using `CNORfuzzy`. At each replicate, 5 out of the 15 input combinations were randomly chosen and excluded from the training set, to be used as a validation set. 3 out of the 100 folds failed due to a numerical error and the resulting folds sampled 79 different training sets. Batch size was set to 3, learning rate to $2 \cdot 10^{-3}$, and the optimization was run for 700 epochs.

Simulating and fitting loops fig. S2 shows the topologies and behaviour of loops used for evaluating our method when fitting to loops. The negative feedback loop can lead to oscillatory behaviours and the positive feedback loop to bistable behaviours.
The transfer function parameters used to simulate an oscillating behaviour in the experiment shown in fig. 2 are (after exponential transformation and rounding to the second decimal) :

$$n_{A,\texttt{AND}} = 0.60 \text{ and } K_{A,\texttt{AND}} = 0.40$$
$$n_{B,C} = 4.15 \text{ and } K_{B,C} = 0.51$$
$$n_{C,\texttt{NOT}} = 1.81 \text{ and } K_{C,\texttt{NOT}} = 0.77.$$

Those used to simulate a bistable behaviour in the same experiment are:

$$n_{A,\texttt{OR}} = 4.88 \text{ and } K_{A,\texttt{OR}} = 1.22$$
$$n_{B,C} = 1.84 \text{ and } K_{B,C} = 0.35$$
$$n_{C,\texttt{OR}} = 1.65 \text{ and } K_{C,\texttt{OR}} = 0.64.$$

Data-driven logical gate determination The topology of the network used in this experiment is shown in Figure S3. To generate a synthetic dataset, we created 10 different `BioFuzzNet` models by randomly assigning the logical gate type AND or OR to each of the 9 logical gates in the network. For each model, we simulated signal transduction from continuous inputs to obtain 3600 "single cells". Transfer function parameters were identical between all networks. A `BioFuzzNet` with the same topology but 9 interpolated gates was fit to each of those 10 datasets. Optimization was conducted on a training set of 3000 datapoints, and the remaining 600 points were used as a validation set. Batch size was set to 200, the learning rate was set to $5 \cdot 10^{-3}$ and the optimization occurred over 300 epochs.

Detecting errors in prior knowledge networks A `BioFuzzNet` with the topology shown in fig. S4a was used to simulate 2000 "single cells" from continuous inputs. Then we adapted the topology to include a spurious edge and tried to fit these adapted topologies to the generated dataset. One adaptation assumed that $F \equiv A \vee E \vee D$, (see Figure S4b) and a second that $F \equiv (A \vee E) \wedge D$. The train and validation set used during optimization and for testing had an equal size of 1000 datapoints. The optimization lasted for 300 epochs, the learning rate was $5 \cdot 10^{-3}$ and the batch size was 50.

A.3 Algorithms

Algorithm 1 Sequential update of an acyclic `BioFuzzNet`

Input: B a `BioFuzzNet` without any cycle and D_{input} the measured value at its root/input nodes
Output: Updated B such that B.PREDICTIONS contains the predicted values at all nodes for input D_{input} and the current transfer function parameters.

```
B.PREDICTIONS[INPUT NODES] ← D_input
NON UPDATED ← B.NODES
TO UPDATE ← INPUT NODES
while NON UPDATED ≠ [ ] do
    node ← TO UPDATE.pop()
    if node ∈ NON UPDATED then
        NON UPDATED PARENTS ← B.parents(node) ∩ NON UPDATED
        if NON UPDATED PARENTS = [ ] then
            B.PREDICTIONS[node] ← update(B.PREDICTIONS[B.parents(node)])  ▷The value of a node depends only on the value of its parent nodes
            NON UPDATED.remove(node)
            TO UPDATE.remove(node)                                        ▷Remove possible duplicates
            TO UPDATE.append(B.children(node))
        else
            for p in NON UPDATED PARENTS do
                TO UPDATE.append(p)        ▷The parent nodes are added before the child nodes
            end for
            TO UPDATE.append(node)        ▷Add the visited node to the queue again if it had non updated parents
        end if
    end if
end while
```

Algorithm 2 Sequential update of a cyclic `BioFuzzNet`

Input: B a `BioFuzzNet` with at least one cycle and D_{input} the measured value at its root/input nodes.

Output: Updated B such that B.PREDICTIONS contains the predicted values at all nodes for input D_{input} and the current transfer function parameters.

```
B.PREDICTIONS[INPUT NODES] ← D_input
L = length(largest cycle in the network)
let k ∈ N                    ▷k is currently hardcoded but should be adjusted to the network topology.
SAVED STATES ← {}            ▷A dictionary to save the last k · L states
for i ∈ (1, ..., 2 · k · L − 1) do
    NON UPDATED ← B.NODES
    TO UPDATE ← INPUT NODES
    while NON UPDATED ≠ [ ] do
        node ← TO UPDATE.pop()
        if node ∈ NON UPDATED then
            NON UPDATED PARENTS ← B.parents(node) ∩ NON UPDATED
            can update ← False
            if NON UPDATED PARENTS = [ ] then
                can update ← True
            else ▷Nodes that are part of a feedback loop can be updated even if all of their parents
are not updated yet.
                PROBLEMATIC PARENTS ← [ ]
                for p ∈ NON UPDATED PARENTS do
                    if ¬ ( ∃ loop ∈ B such that p ∈ loop and node ∈ loop) then
                        PROBLEMATIC PARENTS.append(p)
                    end if
                end for
                if PROBLEMATIC PARENTS = [ ] then
                    can update ← True
                end if
                if can update then
                    B.PREDICTIONS[node] ← update(B.PREDICTIONS[B.parents(node)])
                    NON UPDATED.remove(node)
                    TO UPDATE.remove(node)                ▷Remove possible duplicates
                    TO UPDATE.append(B.children(node))
                else
                    for p in PROBLEMATIC PARENTS do
                        TO UPDATE.append(p) ▷The parent nodes are added before the child nodes
                    end for
                    TO UPDATE.append(node)  ▷Add the visited node to the queue again if it had
non updated parents
                end if
            end if

            if i ≥ k · L then                             ▷Save the last k · L predictions
                SAVED STATES[i] ← B.PREDICTIONS
            end if
            B.PREDICTIONS ← average([SAVED STATES[i] for i ∈ [k · L, 2 · k · L − 1]]) ▷Average
the saved states
```

Algorithm 3 Parameter optimization iteration for one minibatch (X_{batch}, y_{batch}). Multiple iterations are necessary for a complete optimization.

Input: B a `BioFuzzNet` and Θ the set of transfer functions parameters to optimize.

▷*X_{batch} is the measured values/ground truth at input nodes, y_{batch} the measured values/ground truth at all observed nodes for a batch of cells*

for $node \in$ OBSERVED NODES **do**

  B.GROUND TRUTH$[node] \leftarrow y_{batch}[node]$ ▷*Initialise the ground truth*

end for

for $node \in B$.NODES **do** ▷*Initialise the predictions*

  if $node$ is a root/input node **then**

    B.PREDICTION$[node] \leftarrow y_{batch}[node]$ ▷*The measured values at input nodes are provided to the model*

  else

    B.PREDICTION$[node] \leftarrow$ *random value*

  end if

end for

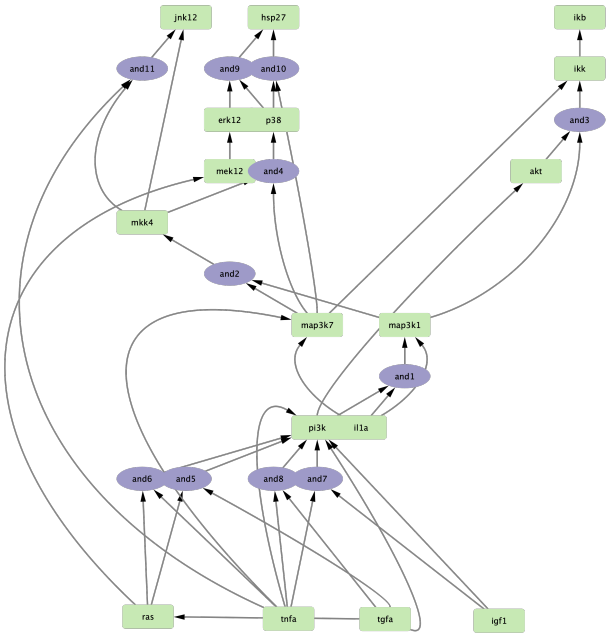
B.PREDICTION $\leftarrow$ **Sequential update**(B) ▷*Simulate the network*

$Loss \leftarrow MSELoss(B$.PREDICTION, B.GROUND TRUTH$)$ ▷*Forward pass*

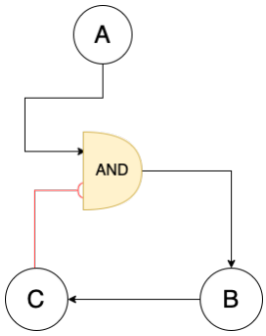
$gradient \leftarrow \nabla_{\Theta} Loss$ ▷*Backward pass using automatic differentiation*

$\Theta \leftarrow$ **ADAM optimizer**$(gradient)$ =0

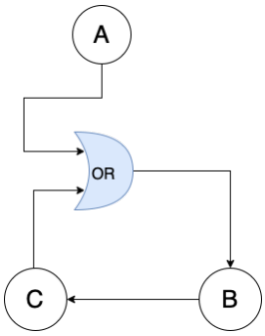
A.4 Supplementary figures



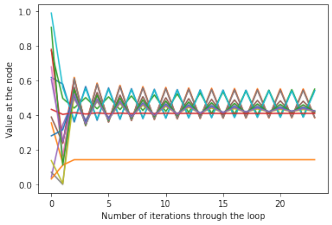
Supplementary Figure S1: Network used to generate data using `CNORfuzzy` [MSRC+11]. Figure generated using Cytoscape version 3.9.1 [SMO+03].



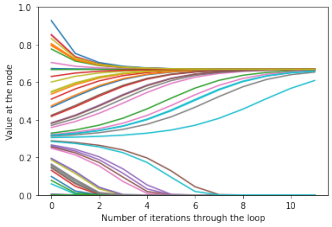
(a) Negative loop topology used for experiments



(b) Positive loop topology used for experiments

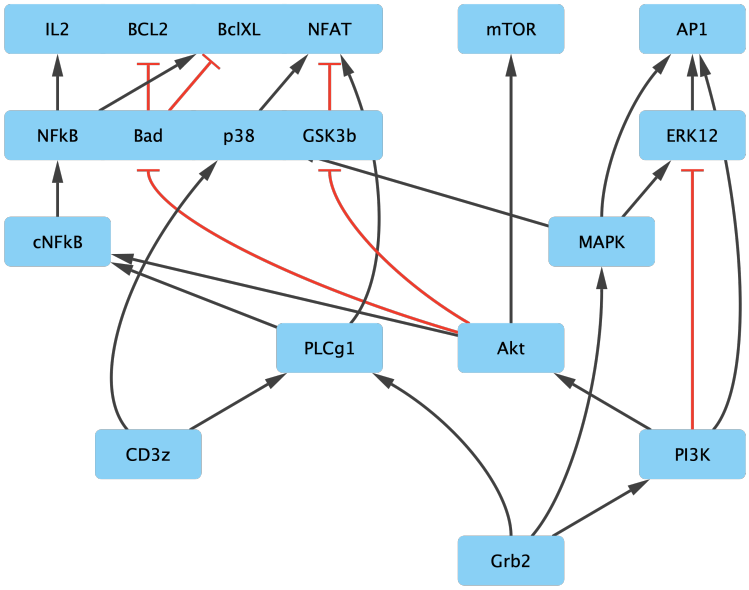


(c) Simulated oscillatory behaviour

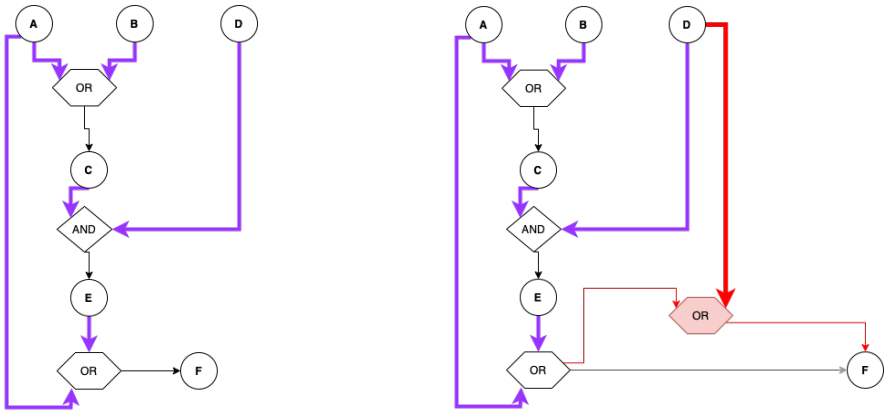


(d) Simulated bistable behaviour

Supplementary Figure S2: Example of loop behaviours simulated using a `BioFuzzNet` and the topology used to generate them. Converging behaviours were also observed with the same topologies but different transfer function parameters.



Supplementary Figure S3: Network topology used for testing the identifiability of logical gates. Logical gates occur every time a node has an in-degree superior to 1. Figure generated using Cytoscape version 3.9.1 [SMO+03].



(a) True topology used to generate the data.

(b) Incorrect prior knowledge network fitted to the data.

Supplementary Figure S4: The prior knowledge topology fitted to the data incorrectly assumed that $F \equiv A \vee E \vee F$. The data was generated with the rule $F \equiv A \vee E \vee F$. Another test was run by fitting an incorrect prior knowledge network assuming that $F \equiv (A \vee E) \wedge F$: its topology is identical to the one shown in Figure S4b except the red logical gate implements the AND logical function. Bold edges indicate edges which comport a Hill transfer function. Red edges and logical gates correspond to incorrect prior knowledge.