Hiscock *BMC Bioinformatics* (2019) 20:214
https://doi.org/10.1186/s12859-019-2788-3

BMC Bioinformatics

METHODOLOGY ARTICLE Open Access

# Adapting machine-learning algorithms to design gene circuits

Tom W. Hiscock[1,2]

## Abstract

**Background:** Gene circuits are important in many aspects of biology, and perform a wide variety of different functions. For example, some circuits oscillate (e.g. the cell cycle), some are bistable (e.g. as cells differentiate), some respond sharply to environmental signals (e.g. ultrasensitivity), and some pattern multicellular tissues (e.g. Turing's model). Often, one starts from a given circuit, and using simulations, asks what functions it can perform. Here we want to do the opposite: starting from a prescribed function, can we find a circuit that executes this function? Whilst simple in principle, this task is challenging from a computational perspective, since gene circuit models are complex systems with many parameters. In this work, we adapted machine-learning algorithms to significantly accelerate gene circuit discovery.

**Results:** We use gradient-descent optimization algorithms from machine learning to rapidly screen and design gene circuits. With this approach, we found that we could rapidly design circuits capable of executing a range of different functions, including those that: (1) recapitulate important in vivo phenomena, such as oscillators, and (2) perform complex tasks for synthetic biology, such as counting noisy biological events.

**Conclusions:** Our computational pipeline will facilitate the systematic study of natural circuits in a range of contexts, and allow the automatic design of circuits for synthetic biology. Our method can be readily applied to biological networks of any type and size, and is provided as an open-source and easy-to-use python module, GeneNet.

**Keywords:** Gene circuits, Machine learning, Numerical screens

## Background

Biological networks – sets of carefully regulated and interacting components – are essential for the proper functioning of biological systems [1, 2]. Networks coordinate many different processes within a cell, facilitating a vast array of complex cell behaviors that are robust to noise yet highly sensitive to environmental cues. For example, transcription factor networks program the differentiation of cells into different cell types [3–5], orchestrate the patterning of intricate structures during development [6, 7], and allow cells to respond to dynamic and combinatorial inputs from their external environment [8, 9]. In addition to transcriptional regulation, many other processes form biological networks, including protein-protein interactions [10], post-translational modifications [11], phosphorylation [12] and metabolism [13, 14].

Understanding how these networks execute biological functions is central to many areas of modern biology, including cell biology, development and physiology. Whilst the network components differ between these disciplines, the principles of network function are often remarkably similar. This manifests itself as the recurrence of common network designs ("network motifs") in transcriptional, metabolic, neuronal and even social networks [15–19]. For example, negative feedback is a network design that achieves homeostasis and noise resilience, whether that be in the regulation of glucose levels, body temperature [20], stem cell number [21], or gene expression levels [22].

A major challenge to understand and ultimately engineer biological networks is that they are complex dynamical systems, and therefore difficult to predict and highly non-intuitive [23]. Consequently, for anything other than

Correspondence: twh27@cam.ac.uk
[1]Cancer Research UK, Cambridge Institute, Li Ka Shing Centre, Robinson Way, Cambridge CB2 0RE, UK
[2]Wellcome Trust/Cancer Research UK Gurdon Institute, University of Cambridge, Cambridge, UK

the simplest networks, verbal descriptions are insufficient, and we rely on computational models, combined with quantitative data to make progress. For example, after decades of genetics, biochemistry, quantitative microscopy and mathematical modeling, a fairly complete, and predictive, description of *Drosophila* anterior-posterior patterning is emerging [24–29].

Quantitative approaches have also proven useful in the rational design of circuits for synthetic biology [30, 31]. If we are to successfully engineer biological processes (e.g. for the production of biomaterials, for use as biosensors in synthetic biology, or for regenerative medicine [31, 32]), then we need clear design principles to construct networks. This requires formal rules that determine which components to use and how they should interact. Mathematical models, combined with an expanding molecular biology toolkit, have enabled the construction of gene circuits that oscillate [33], have stable memory [34], or form intricate spatial patterns [35].

One approach to analyze and design gene circuits is to propose a network and, through computational analysis, ask whether it (i) fits some observed data, or (ii) performs some desired function; and if not, modify parameters until it can. For example, Elowitz and Leibler [33] proposed the repressilator circuit, used simulations to show it should oscillate, and demonstrated its successful operation in *E. coli*. This approach critically relies on starting with a "good" network, i.e. one that is likely to succeed. How do you choose a "good" network? In the study of natural networks, this can be guided by what is known mechanistically about the system (e.g. from chromatin immunoprecipitation sequencing data). However, often the complete network is either unknown or too complicated to model and therefore researchers must make an educated guess for which parts of the network are relevant. For synthetic circuits, one can emulate natural designs and/or use intuition and mathematical modeling to guide network choice. In both cases, these approaches start from a single network – either based on some understanding of the mechanism, or on some intuition of the researcher, or both – and then ask what function this network performs. (Note, throughout we use the term "function" to refer to two things: (1) some real biological function, e.g. patterning the *Drosophila* embryo, or (2) some engineered function in synthetic biology, e.g. an oscillator circuit.)

Here we aim to do the exact opposite, namely to ask: given a prescribed function, what network(s) can perform this function? Equivalently, this means considering the most general network architecture possible (i.e. all genes can activate/repress all other genes), and then determining for what parameters (i.e. what strengths of activation/repression) the network executes the desired function. Such numerical screens have discovered a wide range of interesting gene circuits, including: fold change detectors [36], robust oscillators [37], stripe-forming motifs [38], polarization generators [39], robust morphogen patterning [40], networks that can adapt [41], gradients that scale [42] and biochemical timers [43].

These studies demonstrate that unbiased and comprehensive in silico screens of gene circuits can generate novel and useful insights into circuit function. However, the drawback of such an approach is that it is computationally expensive, and becomes prohibitively slow as the network size is increased, due to the high dimensional parameter spaces involved. For example, consider a gene circuit consisting of $N$ genes, where each gene can activate or repress any other gene. There are then $N^2$ interactions in this network, i.e. at least $N^2$ parameters. It is therefore challenging to scan through this high dimensional parameter space to find parameter regimes where the network performs well.

This motivates more efficient algorithms to search through parameter space. One example is Monte Carlo methods and their extensions, which randomly change parameters and then enrich for changes that improve network performance [44, 45]. Another approach that has had great success is evolutionary algorithms [46]. Here, populations of gene circuits are 'evolved' in an process that mimics natural selection in silico, whereby at each step of the algorithm, there is (1) selection of the 'fittest' networks (those that best perform the desired function), followed by (2) mutation / random changes to the circuit parameters. Evolutionary algorithms have been successfully used to design circuits that exhibit oscillations, bistability, biochemical adaptation and even form developmental patterns [47–51].

Here we designed an alternative approach inspired by gradient-descent algorithms, which underpin many of the advances in modern machine learning. We find that such approaches can significantly accelerate the computational screening of gene circuits, allowing for the design of larger circuits that can perform more complex functions. In machine learning, complex models (typically 'neural networks') with a large number of parameters (typically millions) are fit to data to perform some prescribed function [52, 53]. For example, in computer vision, this function could be to detect a human face in a complex natural scene [54]. Many of the successes in machine learning have been underpinned by advances in the algorithms to fit parameters to data in high dimensions. Central to these algorithms is the principle of "gradient descent", where instead of exhaustively screening parameter space, or randomly moving within it, parameters are changed in the direction that most improves the model performance [55]. An analogy for gradient descent is to imagine you are walking on a mountain range in the fog and wish to descend quickly.

An effective strategy is to walk in the direction of steepest downhill, continuously changing direction as the terrain varies, until you reach the base. Analogously, gradient descent works by iteratively changing parameters in the "steepest" direction with respect to improving model performance.

A major challenge is to efficiently compute these directions in high dimensions. This relies on being able to differentiate the outputs of a complex model with respect to its many parameters. A key advance in this regard has been to perform differentiation *automatically* using software packages such as Theano [56] and Tensorflow [57]. Here, gradients are not calculated using pen and paper, but instead algorithmically, and therefore can be computed for models of arbitrary complexity.

We realized that training neural networks is in many ways similar to designing biological circuits. Specifically, we start with some prescribed function (or data), and we then must fit a model with a large number of parameters to perform the function (fit the data). We thus reasoned that we could use exactly the same tools as in machine learning to design gene circuits, namely advanced gradient-descent, Adam [58], to fit parameters, and automatic differentiation with Theano/Tensorflow to calculate gradients. We found that such an approach could effectively and rapidly generate circuits that perform a range of different functions, using a fairly simple python module, "GeneNet", which we make freely available.

## Results

### Algorithm overview

We seek an algorithm that can robustly fit parameters of complex gene circuits models. We start by considering a simple, but generic model of a transcriptional gene circuit that has been well-studied across a range of biological contexts [26, 38, 59]. (Later, we will show that our algorithm works just as well for different models). The model comprises $N$ transcription factors, whose concentrations are represented by the $N$-component vector, $\boldsymbol{y}$. We assume that all interactions between genes are possible; this is parameterized by a $N$ x $N$ matrix, $\mathbf{W}$. Thus each element $W_{ij}$ specifies how the transcription of gene $i$ is affected by gene $j$ – if $W_{ij}$ is positive, then $j$ activates $i$; if $W_{ij}$ is negative, then $j$ inhibits $i$. We further assume that each gene is degraded with rate $k_i$. Together this specifies an ordinary differential equation (ODE) model of the network:

$$\frac{dy_i}{dt} = \phi\left(\sum\nolimits_j W_{ij} y_j\right) + I_i - k_i y_i \qquad (1)$$

Here, $\phi(x)$ is a nonlinear function, ensuring that transcription rates are always positive, and saturate at high levels of the input. The task of network design is thus to find the parameters $\mathbf{W}$ and $\mathbf{k}$ such that the network operates as desired, translating the input $\mathbf{I}$ to a specified output.

To fit Eq. 1, we start with an analogy to neural networks. Neural networks are highly flexible models with large numbers of parameters that are capable of performing functions of arbitrary complexity [60], whose parameters must be fit to ensure the network performs some designed function. In fact, this correspondence is more than an analogy if one considers recurrent neural networks (RNNs) [61]. RNNs differ from canonical feedforward neural networks, in that connections between the nodes form a directed cycle, allowing the representation of dynamic behavior (e.g. the leaky-integrate-and-fire RNN model which is similar to Eq. 1). Therefore, we wondered whether the algorithms used to fit RNNs could be straightforwardly adapted to fit gene circuit parameters.

To do this, we start with the simplest example where we wish the network to compute some input-output function, $y = f(x)$. In this case, we allow one of the genes $y_1 \equiv x$, to respond to external input, and examine the output of another gene, $y_N \equiv y$. We then define a "cost", $C$, which tracks how closely the actual output of the network, $y$, matches the desired output of the network, $\hat{y}$. First, this involves specifying what the desired output is; as our first example, we consider the case where we want the network output to respond in an ultrasensitive, switch-like manner to the level of some input, $x$, i.e. $y = 0$ for $x < x_*$ and $y = 1$ for $x > x_*$, as in Fig. 1a. Then, we choose the mean squared error as the form of our cost, i.e. $C = \sum_x (y(x) - \hat{y}(x))^2$.

The goal is then to find the parameters that minimize this cost and therefore specify the network that best gives the desired output. To do this rapidly and efficiently in high dimensional parameter spaces, we use gradient descent. In gradient descent, parameters, $p_i$ are updated in the direction that maximally reduces the cost, i.e. $\delta p_i = -l_r \partial_{p_i} C$, where $l_r$ is the learning rate. Intuitively, for a two-dimensional parameter set, this corresponds to moving directly "downhill" on the cost surface (Fig. 1a).

For classic gradient descent, the learning rate $l_r$ is set to a constant value. However, this can present problems when optimizing a highly complex cost function in high dimensions. Intuitively, and as shown in Fig. 2, we would like the learning rate to adapt as optimization proceeds (and also to be different for each parameter). A more sophisticated version of gradient descent, Adaptive moment estimation, or Adam [58], has been established to overcome these difficulties and is being widely used to fit complex neural network models [62]. For this reason, we choose Adam as our default optimization algorithm,

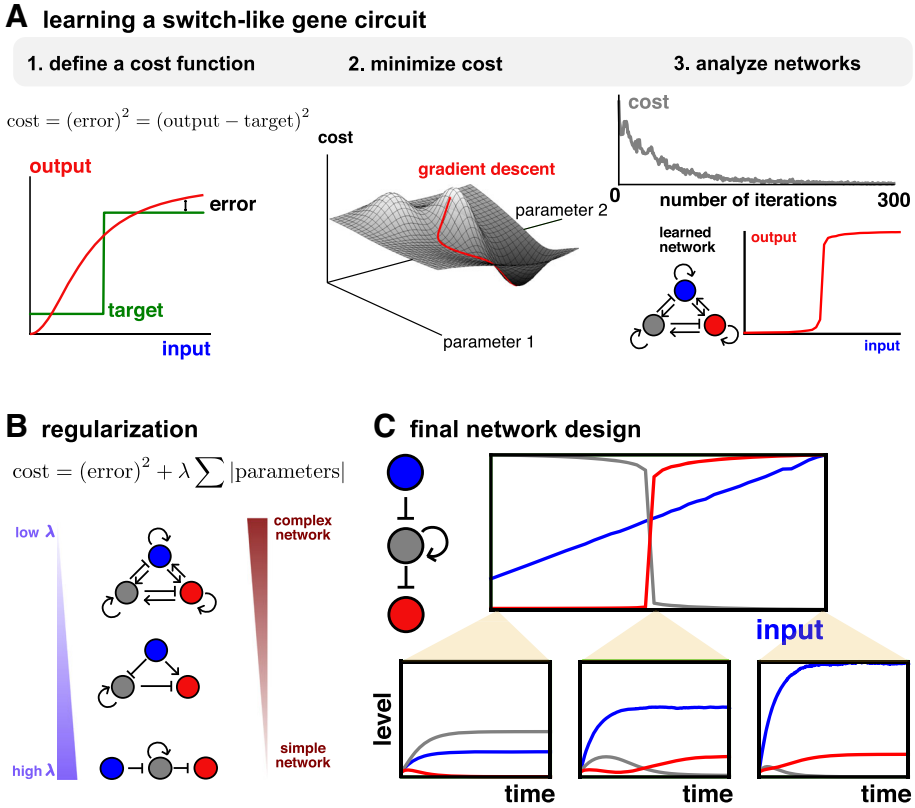


**Fig. 1** Overview of GeneNet. **a** The optimization algorithm consists of three parts: defining a cost function (left), updating parameters to minimize the cost via gradient descent (middle), and analyzing the learned networks (right). **b** Regularization selects networks with varying degrees of complexity. **c** Final design of an ultrasensitive switch. Upper: the final values of each of the three genes as a function of different levels of input. Lower: time traces illustrating network dynamics for three representative values for the input level

which we will later show to be effective for the examples considered in this manuscript.

Minimizing the cost is, in principle, exactly the same whether training neural networks or screening gene circuits. The difference arises when computing the gradient of the cost function with respect to the parameters. For the algebraic equations in feedforward neural networks, the computation is fairly straightforward and can be written down explicitly. For a gene circuit ODE model, however, this is much more difficult. One approach is to estimate gradients using a finite difference procedure [63, 64], in which you compare the model output when each of the system parameters is changed by a small amount. Alternatively, forward sensitivity analysis specifies the time-evolution of the gradients as a set of coupled ODEs. However, both these approaches scale poorly as the number of parameters increases [63].

We realized that machine learning libraries, such as Theano and Tensorflow, could provide a much faster and more direct way of computing gradients, since they permit automatic (or "algorithmic") differentiation of computer programs. Specifically, by implementing a simple differential equation solver in Theano/Tensorflow, we can "differentiate" the solver in a single line of code and thereby compute gradients rapidly, even in high dimensions. Moreover, whilst Eq. 1 resembles a neural network model, this is not at all necessary. Rather, *any* dynamics that can be specified algorithmically (in code) can be accommodated. The general procedure is to write an ODE solver/simulator for the model, in Theano/Tensorflow code. Since the solver algorithm consists of many elementary operations (addition, multiplication) combined, each of which can be differentiated, then the entire solver can be differentiated by automatically combining these using the product and chain rule. Automatic differentiation in Theano/Tensorflow is analogous to calculating the adjoint state in adjoint sensitivity analysis [63], but with the advantage that it can be algorithmically calculated for any model.

Together, the gene network model, the cost function, and the gradient descent algorithm define a procedure to design gene circuits (see Methods). We first tested our pipeline by asking it to generate an ultrasensitive switch, a circuit that is seen in vivo [65], and has also been rationally engineered [66]. Indeed, we find that as we step through repeated

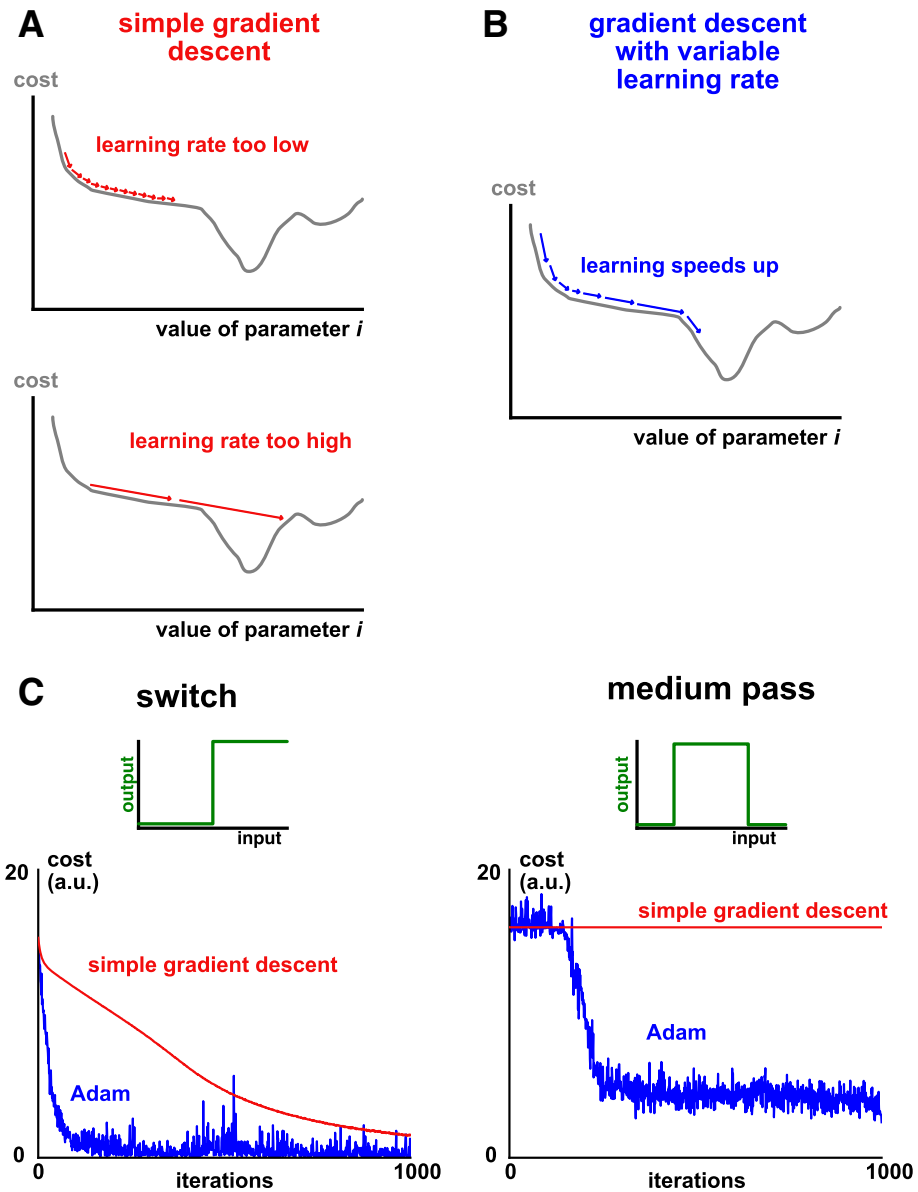


**Fig. 2** Adam is an effective gradient descent algorithm for ODEs. **a** Using a constant learning rate in gradient descent creates difficulties in the optimization process. If the learning rate is too low (upper schematic), the algorithm 'gets stuck' in plateau regions with a shallow gradient (saddle points in high dimensions). If instead the learning rate is too high (lower schematic), important features are missed and/or the learning algorithm won't converge. **b** An adaptive learning rate substantially improves optimization (schematic). Intuitively, learning speeds up when traversing a shallow, but consistent, gradient. **c** Cost minimization plotted against iteration number, comparing classic gradient descent (red) with the Adam algorithm (blue). Left: an ultrasensitive switch. Right: a medium pass filter

iterations of the gradient descent, we efficiently minimize the cost function, and so generate parameters of a gene network model that responds sensitively to its input (Fig. 1a).

To train this circuit, we have used a sophisticated version of gradient descent, Adam. Could a simpler algorithm – namely classic gradient descent with constant learning rate – also work? As shown in Fig. 2, we find that we can generate ultrasensitive switches using classic gradient descent, albeit more slowly and after some fine tuning of the learning rate parameter. We emphasize that, in contrast, Adam works well with default parameters. Furthermore, we find that as we consider functions of increasing complexity, classic gradient descent fails to learn, whilst Adam still can. These results echo outputs from the machine learning community, where Adam (and other related algorithms) significantly outperforms gradient descent [58].

Whilst the network in Fig. 1a performs well, its main drawback is that it is complicated and has many parameters. Firstly, this makes it difficult to interpret exactly what the network is doing, since it is not obvious which interactions are critical for forming the switch. Secondly, it would make engineering such a network more complicated. Therefore, we modified the cost in an attempt to simplify gene networks, inspired by the techniques of "regularization" to simplify neural networks [61]. Specifically, we find that if we add the L1 norm of the parameter sets to the cost, i.e. $C = \sum_x (y(x) - \hat{y}(x))^2 + \lambda \sum_{p_i} |p_i|$, we can simplify networks without significantly compromising their performance. Intuitively, the extra term penalizes models that have many non-zero parameters, i.e. more complex models [67]. By varying the strength of the regularization, $\lambda$, we find networks of varying degrees of complexity (Fig. 1b).

The final output of our algorithm is a simplified gene network that defines a dynamical system whose response is a switch-like function of its inputs (Fig. 1c). Therefore, we have demonstrated that machine-learning algorithms can successfully train gene networks to perform a certain task. In the remainder of this work, we show the utility of our pipeline by designing more realistic and complex biological circuits.

## Applications

First, we consider three design objectives for which there already exist known networks, so that we can be sure our algorithm is working well. We find that we can rapidly and efficiently design gene circuits for each of the three objectives by modifying just a few lines of code that specify the objective, and without changing any details or parameters of the learning algorithm. Further, we can screen for functional circuits within several minutes of compute time on a laptop. In each case, the learned network is broadly similar to the networks described previously, lending support to our algorithm.

### French-flag circuit

The first design objective is motivated from the French-Flag model of patterning in developmental biology [68]. Here, a stripe of gene expression must be positioned at some location within a tissue or embryo, in response to the level of some input. This input is typically a secreted molecule, or "morphogen", which is produced at one location, and forms a gradient across the tissue. In order to form a stripe, cells must then respond to intermediate levels of the input. To identify gene circuits capable of forming stripes, we ran our algorithm using the desired final state as shown in Fig. 3a. The algorithm converges on a fairly simple network, where the input directly represses, and indirectly activates, the output, thus responding at intermediate levels of the input (Fig. 3a). Exactly the same network was described in a large-scale screen of stripe-forming motifs [38], and has been observed in early *Drosophila* patterning [69], suggesting that our learned design may be a common strategy.

### Pulse detection

In our second example, we consider a more complicated type of input, namely pulses of varying duration. In many cases, cells respond not just to the level of some input, but also to the duration [70, 71]. We sought a circuit design to measure duration, such that once an input exceeding a critical duration is received, the output is irreversibly activated (Fig. 3b). As before, by changing a few lines of code, and within a few minutes of laptop compute time, we can efficiently design such a circuit (Fig. 3b). This circuit shares features (such as double inhibition and positive feedback) with networks identified in a comprehensive screen of duration detection motifs [43].

### Oscillator

Our third example takes a somewhat different flavor, where instead of training a network to perform a specific input/output function, we train a network to self-generate a certain dynamical pattern – oscillations [72, 73]. In this case, the cost is not just dependent on the final state, but on the entire dynamics, and is implemented by the equation $C = \sum_t (y(t) - A\cos(\omega t))^2$, where $A$ is the amplitude and $\omega$ the frequency of the oscillator. Minimizing this cost yields a network that gives sustained oscillations, and is reminiscent of the repressilator network motif that first demonstrated synthetic oscillations [33]. Interestingly, when plotting how the cost changed in the minimization algorithm, we saw a precipitous drop at a certain point (Additional file 1: Figure S1), demonstrating visually the transition through a bifurcation to produce oscillations.

## Extensions

### Networks of increased size

To illustrate the scalability of GeneNet, we considered larger networks (up to 9 nodes, with 81 parameters) and asked whether they could also be successfully screened. As shown in Fig. 4a, we find that the median number of iterations required to train a French Flag circuit is largely insensitive to the network size, demonstrating that GeneNet is scalable to models of increased complexity.

### More complex / realistic ODE models

Whilst Equation 1 is a good description for transcriptional gene circuits [38], we asked whether different

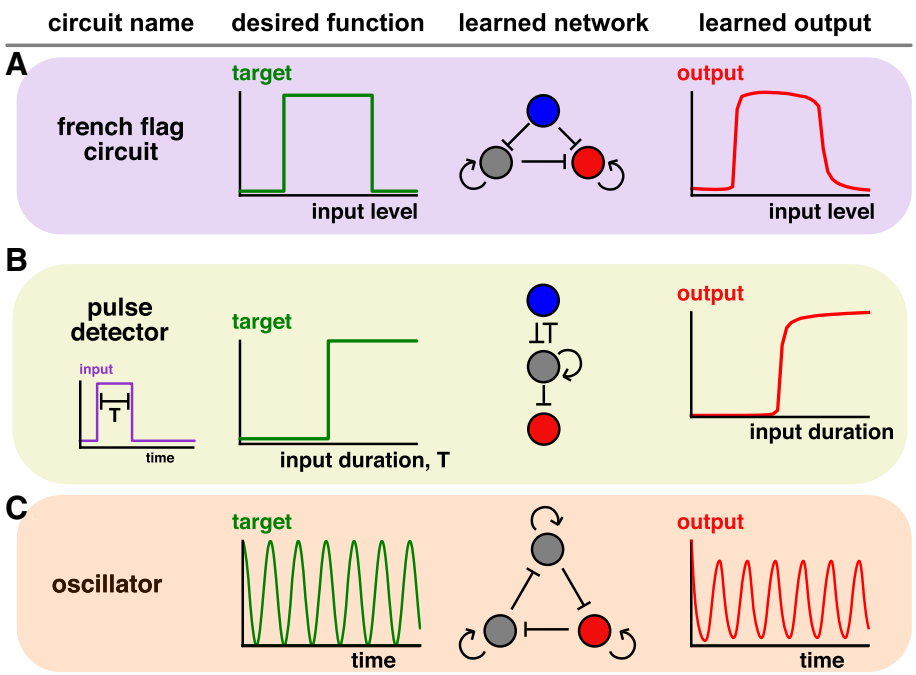


**Fig. 3** Using GeneNet to learn gene circuits. **a** A French-Flag circuit responds to intermediate levels of input to generate a stripe. The red node corresponds to the output gene, and the blue node the input. **b** A "duration-detector" which is irreversibly activated when stimulated by pulses exceeding a certain critical duration. As above, the red node corresponds to the output gene, and the blue node the input. **c** An oscillator

molecular interactions and model complexities could be incorporated into our pipeline. One simple extension is to allow pairs of molecules to bind, facilitating their degradation; this has been shown to be useful when evolving oscillator circuits in silico [48]. To include dimerization-based decay, we modified Eq. 1 to the following:

$$\frac{dy_i}{dt} = \phi\left(\sum_j W_{ij} y_j\right) - k_i y_i - \Gamma_{ij} y_i y_j \quad (2)$$

Here, $\Gamma_{ij}$ is a symmetric matrix that represents the degradation rates of each dimer pair possible. We find that, without modifying the optimization algorithm, we can train this more complicated gene circuit model; an example output for training an oscillator is shown in Fig. 4b.

Another possibility is that the additive input model specified in Eq. 1 must be extended, since gene circuits often rely on the coincidence of multiple independent events. Therefore, we considered a rather different type of circuit, built of multiple, independent repressors. In this case, circuit dynamics are described by a product of Hill functions:

$$\frac{dy_i}{dt} = \prod_j \frac{1}{1 + \left(W_{ij} y_j\right)^n} + I_i - k_i y_i \quad (3)$$

where we set the Hill coefficient, n, to be 3. Again, without modifying neither the structure nor the parameters of the learning algorithm, GeneNet is capable of designing a co-operative, repressor-only medium pass (French Flag) circuit (Fig. 4c).

Together, these results suggest that GeneNet can be straightforwardly adapted to work on a range of different biological models.

### Alternative cost functions

The final extension we consider is the form of the cost function. We have so far focused on the mean squared error as our measure of the model's goodness-of-fit. However, there are other options, and work from the field of evolutionary algorithms suggests that the choice of cost function can have an impact on the efficacy of circuit design [46]. To determine if GeneNet can be modified to work with different cost functions, we trained a switch-like circuit using the negative cross-entropy cost function, commonly used in image classification problems. Again, without changing the optimization algorithm, we can efficiently learn circuit parameters (Fig. 4d).

## Comparison to other algorithms

As outlined in the introduction, there are several other methods for in silico circuit design, including: (1) comprehensive enumeration of circuit parameters / topologies; and (2) evolutionary algorithms. How does our method compare?

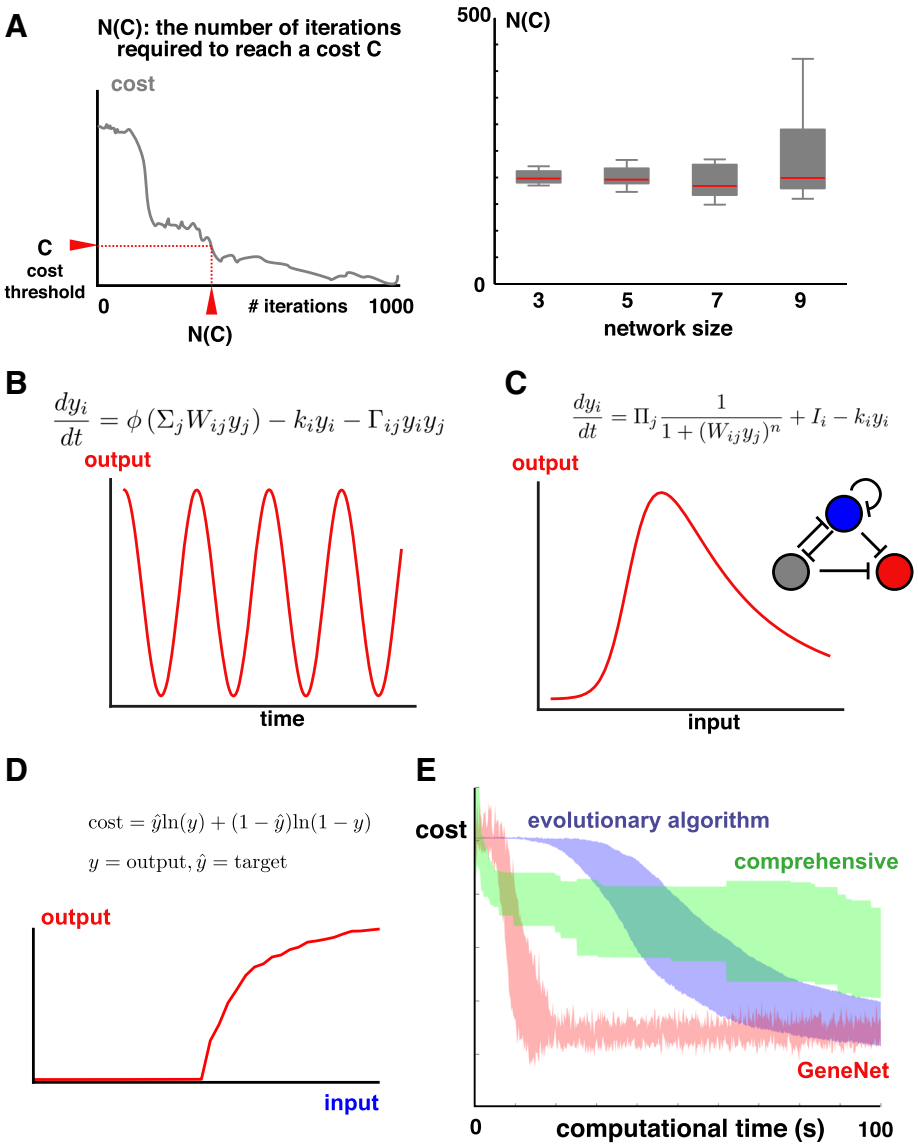


**Fig. 4** Performance and generality of GeneNet. **a** Scalability of GeneNet. Left: schematic of N(C) – the number of iterations required to reach a given network performance. Right: For the same desired circuit function as in Figure 3A, we train networks of varying sizes and provide a boxplot of the number of iterations required to achieve a cost, $C = 0.4$. The median is roughly constant. **b** Example output after training a 2-node network oscillator using the equation provided. **c** Example output, and circuit, after training a 3-node French Flag circuit using the independent repressor circuit. **d** A switch-like network is learned by minimizing the negative cross-entropy function. **e** Comparing computational efficiencies of different circuit design algorithms: GeneNet, evolutionary algorithms and comprehensive enumeration. Each algorithm is run 10 times; the shaded area corresponds to the mean ± standard deviation of the cost value. We see that the cost is rapidly, and reproducibly minimized by GeneNet

Firstly, it is worth noting that there are key shared features. For each approach, one defines: (1) an ODE-based model of a gene circuit, and (2) a cost function that selects networks to execute a specific function. Therefore, these methods can be used somewhat interchangeably. However, differences arise in exactly how the algorithms select the networks with high performance.

Comprehensive screens consider all possible parameters and then identify those that (globally) minimize the cost. The advantage of an exhaustive screen is that one can be certain to find the global optimal solution; however, this comes with the drawback of being computationally expensive, and prohibitively slow for larger gene circuits.

In contrast, evolutionary algorithms iteratively improve circuit performance by randomly mutating parameters across a population of circuits, and retaining circuits with the highest cost. This approach can generate functioning circuits with fewer computations than a comprehensive screen, and has the added advantage of

providing some insight into how gene circuits might evolve during natural selection.

GeneNet is neither exhaustive, nor does it provide an insight into the evolvability of gene circuits. However, its key advantage is speed; in particular, its excellent scalability to larger networks (Fig. 4a) and thus a capacity to train circuits to execute more complicated functions. We performed a side-by-side comparison in training a French Flag circuit for each of the three approaches (comprehensive screens, evolutionary algorithms and GeneNet) and, consistent with our expectation, see that GeneNet significantly outperforms the other two in terms of speed (Fig. 4e). To demonstrate the real utility of our approach, we end by considering a more complicated design objective.

## A more complex circuit: a robust biological counter

In our final example, we attempt a more ambitious design– a biological counter – to demonstrate that GeneNet can also design circuits to perform more complex computations and functions. Whilst counters are found in some biological systems (such as in telomere length regulation [74]), we focus our aim on designing a counter for synthetic biology. There are numerous applications for such counters, two of which are: (1) a safety mechanism programming cell death after a specified number of cell cycles, (2) biosensors that non-invasively count the frequency of certain stimuli, particularly low frequency events [75].

We consider an analog counter, where we wish some input set of pulses to result in an output equal (or proportional) to the number of pulses (Fig. 5a). For example, the "input" here could be the level of a cell cycle related protein to count divisions [76]. As is shown in Fig. 5b, the simplest way to count would be simply to integrate over time the levels of the input. One implementation of this used an analog memory device driven by CRISPR mutagenesis [77], i.e. when the stimulus is present, Cas9 is active and mutations arise. However, a major shortcoming of such a circuit is that it is unreliable and sensitive to variations in pulse amplitude and duration that are often present (Fig. 5b).

Therefore, we sought to systematically design a novel gene circuit to count pulses that would be robust to their amplitude and duration. To do this, we provided a complex ensemble of input stimuli, each containing a different number of pulses, of varying amplitudes and durations. For each input we then defined the desired output to be equal to the number of pulses present, and trained the network to minimize the mean squared error cost, as before.

Strikingly, this procedure uncovers a network that is highly robust in counting pulse number (Fig. 5c).

Looking more deeply into the network, we see that it has learned a very interesting way to count, with two key ingredients. Firstly, it acts as an "off-detector". Specifically, examining the dynamic time traces reveals that the network responds *after* the pulse has occurred, as it turns off. Mechanistically, when the input increases, this allows build up of the purple node and repression of the orange node. However, activation of the downstream green node is only possible once the input pulse has ended, and the repression from the purple node has been alleviated. In this way, the circuit responds to the termination of the pulse, and is thus robust to its duration.

Secondly, the network uses "digital encoding" to be robust to the level of the input. This is achieved by having the green node undergo an "excitable pulse" of stereotyped amplitude and duration, which is then integrated over time by the red node to complete the counter. By using "digital" pulses of activity with an excitable system, the circuit is therefore insensitive to the precise levels of the input. Together, this forms a circuit that reliably counts despite large variations in input stimulus.

We emphasize that these behaviors have not been hard-coded by rational design, but rather have emerged when training the network to perform a complex task. This example therefore shows that a more challenging design objective can be straightforwardly accommodated into our gene network framework, and that it is possible to learn rather unexpected and complex designs.

## Discussion

By combining ODE-based models of gene networks with the optimization methods of machine learning, we present an approach to efficiently design gene circuits. Whilst we have focused on gene networks and transcriptional regulation as a proof of principle, our algorithm is rather general and could easily be extended to learn other networks, such as phosphorylation, protein-protein interaction and metabolic networks, so long as they are described by ordinary differential equations. Further, whilst the networks we have focused on are relatively small and have been trained on a personal laptop, our Theano/Tensorflow pipelines can be easily adapted to run much faster on GPUs, and therefore we expect that large networks could also be trained effectively [56].

Our approach could also be extended to incorporate other types of differential equation, such as partial differential equations. This would allow us to understand how networks operate in a spatial, multicellular context, throughout an entire tissue, and thus provide useful insights into how different structures and patterns are formed during development [32]. Other extensions would be to use real data as inputs to the learning algorithm, in

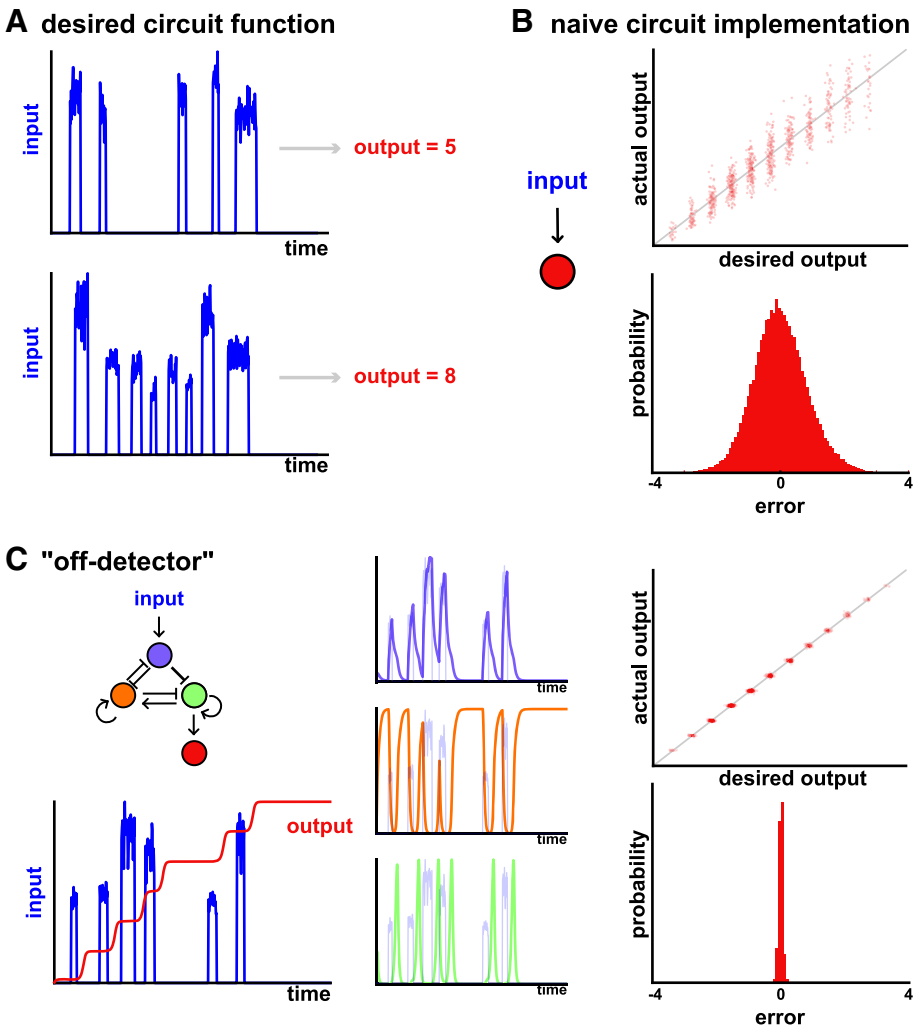


**Fig. 5** Designing a robust biological counter using GeneNet. **a** Desired input/output function of a robust biological counter. **b** Simply integrating the input over time yields an analog counter with significant error, as shown by the spread in the input/output function (upper) and the relative errors between the two (lower). **c** GeneNet learns an "off-detector" network (left), with substantially reduced error rates (right). Inspection of gene dynamics (middle) shows that an excitable "digital" pulse of green activity is initiated at the end of the input pulse (shaded blue line)

which case more sophisticated algorithms would be required to deal with parameter uncertainty [78, 79].

One drawback of our approach is that it selects only a single gene circuit out of many, and thus may ignore alternative circuits that may also be useful or relevant. A natural extension would therefore be to combine the speed of GeneNet's parameter optimization with a comprehensive enumeration of different network topologies, thus generating a complete 'atlas' of gene circuits [38].

Finally, one concern with machine learning methods is that the intuition behind the models is hidden within a "black box" and opaque to researchers, i.e. the machine, not the researcher, learns. We would like to offer a slightly different perspective. Instead of replacing the researcher, our algorithm acts as a highly efficient way to screen models. In this sense, one shouldn't view it as a tool to solve problems, but rather as an efficient way to generate new hypotheses. The role of the scientist is then to: (1) cleverly design the screen (i.e. the cost) such that the algorithm can effectively learn the desired function, and (2) to carefully analyze the learned circuits and the extent to which they recapitulate natural phenomena. The distinct advantage of our approach over neural networks is that we learn real biological models – gene circuits – which are both directly interpretable as mechanism, and provide specific assembly instructions for synthetic circuits.

## Conclusions

In this work, we have developed an algorithm to learn gene circuits that perform complex tasks (e.g. count pulses), compute arbitrary functions (e.g. detect pulses

of a certain duration) or resemble some real biological phenomenon (e.g. a French-flag circuit). We have demonstrated that these networks can be trained efficiently on a personal laptop, and require neither fine-tuning of algorithm parameters nor extensive coding to adapt to different network designs. This ease-of-use means that researchers addressing questions in basic biology can quickly generate models and hypotheses to explain their data, without investing a lot of time carefully building and simulating specific models. Equally, our approach should also allow synthetic biologists to rapidly generate circuit designs for a variety of purposes.

## Methods

### Gene network model

We considered the following set of coupled ODEs as our gene circuit model, motivated by [26, 27, 38]:

$$\frac{dy_i}{dt} = k_i\left(\phi\left(\sum\nolimits_j W_{ij}y_j\right) + I_i - y_i\right)$$

Here, $y_i$ denotes the concentration of gene $i$, where $i = 1 \ldots N$ for an N-node network. $W_{ij}$ is a matrix correspond to network weights: $W_{ij} > 0$ means that gene $i$ activates gene $j$, and $W_{ij} < 0$ means that gene $i$ represses gene $j$. $k_i$ is a vector that represents the (assumed linear) degradation of gene $i$. $I_i$ is the prescribed input to the system, which we assume directly causes transcription of gene $i$. The function $\phi$ is a nonlinearity chosen such that the gene network saturates; we choose $\phi(x) = 1/(\exp(x) - 1)$, as in [38]. $y_i^0$ represent the initial conditions of the system. Note, this is a non-dimensionalized version of Eq. 1, whereby gene concentrations are normalized to their maximal level. Note, also, that we add further terms for Fig. 4b and c as discussed in the text.

### Algorithm details

We coded the algorithm using python v2.7.5, and the machine-learning library, Theano v0.8.2, which performs static optimizations for speed and permits automatic differentiation. (We also provide an implementation on Tensorflow).

The key steps in the algorithm are:

1. Define an ODE solver

We choose a simple Euler integration method, whereby the differential equation $\dot{y} = f(y, t)$ is solved iteratively: $y_{n+1} = y_n + f(y_n, t_n)\delta t$. We set $\delta t = 0.01$. We implement the solver using the theano.scan feature, which optimizes the computation of loops. Note, that it is straightforward to implement other solvers, e.g. stiff solvers or adaptive sizes.

2. Define the desired function of the network

This consists of two items: (1) a collection of different inputs to train the network on, and (2) for each input, a desired output. For example, in Fig. 3b, we must include inputs of varying durations, and a function that computes whether the pulse exceeds some critical duration. The input can be static in time (as in Fig. 2a), dynamic in time (as in Fig. 3b) or zero (as in Fig. 3c). The desired output can be the final state of the network (as in Fig. 3b), or the complete time dynamics of the network (Fig. 3c). See Additional file 2: Table S1 for further details.

3. Define the cost to be minimized

As discussed in the main text, we use the mean squared error as the cost. Since we are often not concerned with absolute levels of any gene, but rather the relative levels, we modify the cost such that the network output can be rescaled by a factor $A$, which is a learnable parameter, i.e. $y \to Ay$. For regularization, we add a term $\lambda\Sigma_{i,\ j}|W_{ij}|$ to the cost, with the aim of simplifying the network matrix $W_{ij}$.

4. Define the parameters that are to be fit

For all simulations, we fit the network weights, $W_{ij}$ to the data. For Figs. 1, 3a, b, we do not allow $k_i$ to change, and instead set $k_i = 1$. For Figs. 3c and 4c, we use the $k_i$ as learnable parameters. For Fig. 3c, we must allow the initial conditions of the network to be learned, such that the oscillator has the correct phase.

5. Define the optimization algorithm

We use the Adam optimizer [58] with parameters $l_r = 0.1$, $b_1 = 0.02$, $b_2 = 0.001$, $e = 10^{-8}$ in all cases.

6. Initialize the model parameters

We set $k_i$ to be one and the network weights to be a normally distributed random number with mean zero and standard deviation 0.1. Initial conditions (except for Fig 3c) are set as $y_i^0 = 0.1$.

7. Train the network

We iteratively perform the optimization procedure to update parameters, initially setting $\lambda = 0$. At each step, we train the network using a subset of total input/output data, using a "batch" of $B$ input/output pairs. The idea is that batching the data in this way adds stochasticity to the optimization and therefore avoids local minima.

8. Regularize the network

Once a network has been trained, we now regularize it by increasing $\lambda$ and re-implementing the optimization procedure. A range of different $\lambda$ values is used until a network of desired complexity is achieved.

9. "Prune" the network

In the final step, we retrain a simplified network. Specifically, starting from the regularized network, we set any small network weights to be exactly zero, i.e. $W_{ij}^{(\text{prune})} = 0 \quad \forall i,j : W_{ij} < \epsilon$, and then optimize over the remaining weights, using $\lambda = 0$.

10. Save parameters

See Additional file 2: Table S1 for the parameter values for the networks learned.

### Implementation details

Networks were designed was performed on a Macbook air, 1.3GHz Intel Core i5, with 8GB 1600 MHz DDR3 RAM. We repeated the learning algorithm for each of the designs in the paper several times, with different regularization levels $\lambda$, and found similar, or often identical, network topologies to be learned in each case. In the figures we report a representative network, where $\lambda$ has been chosen manually to give a minimal network that still performs the function well.

Additional file 2: Table S2 gives details of the algorithm implementation specific to the networks learned.

For the comparative and extension studies in Fig. 4, we developed a TensorFlow implementation of an evolutionary algorithm and compared its speed to the Tensorflow implementation of GeneNet, using the same cost function. We repeated this for a Tensorflow implementation of a comprehensive screen, for which we randomly sample parameters and retain the (global) minimum cost value. These were performed on a MacBook Pro, 2.7 GHz Intel Core i5, with 8 GB 1867 MHz DDR3 RAM.

## Additional files

**Additional file 1: Figure S1.** Cost minimization. Example traces of the cost minimization during the optimization procedure. Note, in all cases (and particularly in the oscillator), there are sharp drop-offs in cost, which are likely reflecting bifurcation points in the dynamics. (PDF 177 kb)

**Additional file 2: Table S1.** Parameter values for networks learned in the main text. **Table S2.** Algorithm implementation parameters. **Table S3.** Speed tests. (DOCX 70 kb)

### Abbreviations

Adam: Adaptive moment estimation; ODE: Ordinary differential equation; RNN: Recurrent neural network

### Acknowledgements

I thank John Ingraham for inspiring this project over breakfast and dinnertime conversations, and for his enthusiasm and generosity in teaching me machine learning. I thank Sean Megason for advice and mentoring.

### Funding

This work was supported by an EMBO Long-term fellowship, ALTF 606–2018. This funding source did not play any role in study design, data collection/analysis, or manuscript preparation.

### Availability of data and materials

GeneNet (Theano and Tensorflow versions) is available on github: "https://github.com/twhiscock/GeneNet-".

### Author's contributions

TWH designed and performed the research, and wrote the paper. The author read and approved the final manuscript.

### Ethics approval and consent to participate

Not applicable.

### Consent for publication

Not applicable.

### Competing interests

The author declares that he has no competing interests.

## Publisher's Note



Received: 18 March 2019 Accepted: 2 April 2019
Published online: 27 April 2019

### References

1. Barabasi AL, Oltvai ZN. Network biology: understanding the cell's functional organization. Nat Rev Genet. 2004;5(2):101–13.
2. Zhu X, Gerstein M, Snyder M. Getting connected: analysis and principles of biological networks. Genes Dev. 2007;21(9):1010–24.
3. Goode DK, et al. Dynamic gene regulatory networks drive hematopoietic specification and differentiation. Dev Cell. 2016;36(5):572–87.
4. Plath K, Lowry WE. Progress in understanding reprogramming to the induced pluripotent state. Nat Rev Genet. 2011;12(4):253–65.
5. Cahan P, et al. CellNet: network biology applied to stem cell engineering. Cell. 2014;158(4):903–15.
6. Davidson EH. Emerging properties of animal gene regulatory networks. Nature. 2010;468(7326):911–20.
7. Rhee DY, et al. Transcription factor networks in Drosophila melanogaster. Cell Rep. 2014;8(6):2031–43.
8. Lopez-Maury L, Marguerat S, Bahler J. Tuning gene expression to changing environments: from rapid responses to evolutionary adaptation. Nat Rev Genet. 2008;9(8):583–93.
9. Mangan S, Alon U. Structure and function of the feed-forward loop network motif. Proc Natl Acad Sci U S A. 2003;100(21):11980–5.
10. Stelzl U, et al. A human protein-protein interaction network: a resource for annotating the proteome. Cell. 2005;122(6):957–68.
11. Minguez P, et al. Deciphering a global network of functionally associated post-translational modifications. Mol Syst Biol. 2012;8:599.
12. Linding R, et al. Systematic discovery of in vivo phosphorylation networks. Cell. 2007;129(7):1415–26.
13. Jeong H, et al. The large-scale organization of metabolic networks. Nature. 2000;407(6804):651–4.
14. Fiehn O. Combining genomics, metabolome analysis, and biochemical modelling to understand metabolic networks. Comp Funct Genomics. 2001;2(3):155–68.
15. Alon U. Network motifs: theory and experimental approaches. Nat Rev Genet. 2007;8(6):450–61.
16. Alon U. Biological networks: the tinkerer as an engineer. Science. 2003;301(5641):1866–7.
17. Shen-Orr SS, et al. Network motifs in the transcriptional regulation network of Escherichia coli. Nat Genet. 2002;31(1):64–8.

18. Milo R, et al. Network motifs: simple building blocks of complex networks. Science. 2002;298(5594):824–7.
19. Rosenfeld N, Elowitz MB, Alon U. Negative autoregulation speeds the response times of transcription networks. J Mol Biol. 2002;323(5):785–93.
20. Simon E, Pierau FK, Taylor DC. Central and peripheral thermal control of effectors in homeothermic temperature regulation. Physiol Rev. 1986;66(2): 235–300.
21. Shraiman BI. Mechanical feedback as a possible regulator of tissue growth. Proc Natl Acad Sci U S A. 2005;102(9):3318–23.
22. Lestas I, Vinnicombe G, Paulsson J. Fundamental limits on the suppression of molecular fluctuations. Nature. 2010;467(7312):174–8.
23. Alon U. An Introduction to Systems Biology: Design Principles of Biological Circuits. Chapman & Hall/CRC; 2006.
24. Fowlkes CC, et al. A quantitative spatiotemporal atlas of gene expression in the Drosophila blastoderm. Cell. 2008;133(2):364–74.
25. Gregor T, et al. Stability and nuclear dynamics of the bicoid morphogen gradient. Cell. 2007;130(1):141–52.
26. Jaeger J, et al. Dynamical analysis of regulatory interactions in the gap gene system of Drosophila melanogaster. Genetics. 2004;167(4):1721–37.
27. Jaeger J, et al. Dynamic control of positional information in the early Drosophila embryo. Nature. 2004;430(6997):368–71.
28. Manu, et al. Canalization of gene expression in the Drosophila blastoderm by gap gene cross regulation. PLoS Biol. 2009;7(3):e1000049.
29. Manu, et al. Canalization of gene expression and domain shifts in the Drosophila blastoderm by dynamical attractors. PLoS Comput Biol. 2009;5(3): e1000303.
30. Mukherji S, van Oudenaarden A. Synthetic biology: understanding biological design from synthetic circuits. Nat Rev Genet. 2009;10(12):859–71.
31. Khalil AS, Collins JJ. Synthetic biology: applications come of age. Nat Rev Genet. 2010;11(5):367–79.
32. Davies J. Using synthetic biology to explore principles of development. Development. 2017;144(7):1146–58.
33. Elowitz MB, Leibler S. A synthetic oscillatory network of transcriptional regulators. Nature. 2000;403(6767):335–8.
34. Gardner TS, Cantor CR, Collins JJ. Construction of a genetic toggle switch in Escherichia coli. Nature. 2000;403(6767):339–42.
35. Liu C, et al. Sequential establishment of stripe patterns in an expanding cell population. Science. 2011;334(6053):238–41.
36. Adler M, et al. Optimal regulatory circuit topologies for fold-change detection. Cell Syst. 2017;4(2):171–181 e8.
37. Li Z, Liu S, Yang Q. Incoherent inputs enhance the robustness of biological oscillators. Cell Syst. 2017;5(1):72–81 e4.
38. Cotterell J, Sharpe J. An atlas of gene regulatory networks reveals multiple three-gene mechanisms for interpreting morphogen gradients. Mol Syst Biol. 2010;6:425.
39. Chau AH, et al. Designing synthetic regulatory networks capable of self-organizing cell polarization. Cell. 2012;151(2):320–32.
40. Eldar A, et al. Robustness of the BMP morphogen gradient in Drosophila embryonic patterning. Nature. 2002;419(6904):304–8.
41. Ma W, et al. Defining network topologies that can achieve biochemical adaptation. Cell. 2009;138(4):760–73.
42. Ben-Zvi D, et al. Scaling of the BMP activation gradient in Xenopus embryos. Nature. 2008;453(7199):1205–11.
43. Gerardin, J. and W.A. Lim, The design principles of biochemical timers: circuits that discriminate between transient and sustained stimulation. biorxiv preprint https://doi.org/10.1101/100651, 2017.
44. Perkins TJ, et al. Reverse engineering the gap gene network of Drosophila melanogaster. PLoS Comput Biol. 2006;2(5):e51.
45. Crombach A, et al. Efficient reverse-engineering of a developmental gene regulatory network. PLoS Comput Biol. 2012;8(7):e1002589.
46. Francois P. Evolving phenotypic networks in silico. Semin Cell Dev Biol. 2014;35:90–7.
47. Francois P, Siggia ED. A case study of evolutionary computation of biochemical adaptation. Phys Biol. 2008;5(2):026009.
48. Francois P, Hakim V. Design of genetic networks with specified functions by evolution in silico. Proc Natl Acad Sci U S A. 2004;101(2):580–5.
49. Francois P, Hakim V, Siggia ED. Deriving structure from evolution: metazoan segmentation. Mol Syst Biol. 2007;3:154.
50. Noman N, et al. Evolving robust gene regulatory networks. PLoS One. 2015; 10(1):e0116258.
51. Smith RW, van Sluijs B, Fleck C. Designing synthetic networks in silico: a generalised evolutionary algorithm approach. BMC Syst Biol. 2017;11(1):118.
52. Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science. 2006;313(5786):504–7.
53. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436–44.
54. Li H, Lin Z, Shen X, Brandt J, Hua G. A convolutional neural network cascade for face detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2015. p. 5325–34.
55. Amari S-i. Backpropagation and stochastic gradient descent method. Neurocomputing. 1993;5(4):185–96.
56. Bergstra J, et al. Theano: a CPU and GPU math compiler in Python. In: Proc. 9th Python in Science Conf. 2010;1:3–10.
57. Abadi, M., et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016.
58. Kingma, D. and J. Ba, Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.
59. Molinelli EJ, et al. Perturbation biology: inferring signaling networks in cellular systems. PLoS Comput Biol. 2013;9(12):e1003290.
60. Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Netw. 1989;2(5):359–66.
61. Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press; 2016. https://www.deeplearningbook.org/.
62. Ruder, S., An overview of gradient descent optimization algorithms. arXiv: 1609.04747, 2016.
63. Frohlich F, et al. Scalable parameter estimation for genome-scale biochemical reaction networks. PLoS Comput Biol. 2017;13(1):e1005331.
64. Uzkudun M, Marcon L, Sharpe J. Data-driven modelling of a gene regulatory network for cell fate decisions in the growing limb bud. Mol Syst Biol. 2015; 11(7):815.
65. Tyson JJ, Chen KC, Novak B. Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell. Curr Opin Cell Biol. 2003; 15(2):221–31.
66. Palani S, Sarkar CA. Synthetic conversion of a graded receptor signal into a tunable, reversible switch. Mol Syst Biol. 2011;7:480.
67. Brunton SL, Proctor JL, Kutz JN. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc Natl Acad Sci U S A. 2016;113(15):3932–7.
68. Wolpert L. Positional information and the spatial pattern of cellular differentiation. J Theor Biol. 1969;25(1):1–47.
69. Clyde DE, et al. A self-organizing system of repressor gradients establishes segmental complexity in Drosophila. Nature. 2003;426(6968):849–53.
70. Hopfield JJ. Kinetic proofreading: a new mechanism for reducing errors in biosynthetic processes requiring high specificity. Proc Natl Acad Sci. 1974; 71(10):4135–9.
71. Mangan S, Zaslaver A, Alon U. The coherent feedforward loop serves as a sign-sensitive delay element in transcription networks. J Mol Biol. 2003; 334(2):197–204.
72. Novak B, Tyson JJ. Design principles of biochemical oscillators. Nat Rev Mol Cell Biol. 2008;9(12):981–91.
73. Stricker J, et al. A fast, robust and tunable synthetic gene oscillator. Nature. 2008;456(7221):516–9.
74. Marcand S, Gilson E, Shore D. A protein-counting mechanism for telomere length regulation in yeast. Science. 1997;275(5302):986–90.
75. Friedland AE, et al. Synthetic gene networks that count. Science. 2009; 324(5931):1199–202.
76. Slomovic S, Pardee K, Collins JJ. Synthetic biology devices for in vitro and in vivo diagnostics. Proc Natl Acad Sci U S A. 2015;112(47):14429–35.
77. Perli SD, Cui CH, Lu TK. Continuous genetic recording with self-targeting CRISPR-Cas in human cells. Science. 2016;353(6304):aag0511.
78. Liepe J, et al. A framework for parameter estimation and model selection from experimental data in systems biology using approximate Bayesian computation. Nat Protoc. 2014;9(2):439–56.
79. Calderhead B, Girolami M, Lawrence ND. Accelerating Bayesian inference over nonlinear differential equations with Gaussian processes. Adv Neural Inf Proces Syst. 2009;21:217–24.