Bioinformatics, 2019, 1–9
doi: 10.1093/bioinformatics/bty945
Advance Access Publication Date: 19 November 2018
Original Paper

Gene expression

Genetic Neural Networks: an artificial neural network architecture for capturing gene expression relationships

Ameen Eetemadi [1,2] **and Ilias Tagkopoulos**[1,2,*]

[1]Department of Computer Science and [2]Genome Center, University of California, Davis, CA 95616, USA

*To whom correspondence should be addressed.

Associate Editor: Bonnie Berger

Received on June 11, 2018; revised on October 27, 2018; editorial decision on November 13, 2018; accepted on November 16, 2018

Abstract

Motivation: Gene expression prediction is one of the grand challenges in computational biology. The availability of transcriptomics data combined with recent advances in artificial neural networks provide an unprecedented opportunity to create predictive models of gene expression with far reaching applications.

Results: We present the Genetic Neural Network (GNN), an artificial neural network for predicting genome-wide gene expression given gene knockouts and master regulator perturbations. In its core, the GNN maps existing gene regulatory information in its architecture and it uses cell nodes that have been specifically designed to capture the dependencies and non-linear dynamics that exist in gene networks. These two key features make the GNN architecture capable to capture complex relationships without the need of large training datasets. As a result, GNNs were 40% more accurate on average than competing architectures (MLP, RNN, BiRNN) when compared on hundreds of curated and inferred transcription modules. Our results argue that GNNs can become the architecture of choice when building predictors of gene expression from exponentially growing corpus of genome-wide transcriptomics data.

Availability and implementation: https://github.com/IBPA/GNN

Contact: iliast@ucdavis.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Prediction of cellular state in novel environments presents a need and an opportunity in systems biology (Carrera *et al.*, 2014; Kim *et al.*, 2016; O'Brien *et al.*, 2015). Surge in the availability of data, advances in computational techniques and exponential increase of computing power have led to the adoption of omics analysis and predictive modeling in a variety of fields, including food safety, drug discovery, biofuel development and precision medicine (Abhyankar *et al.*, 2017; Aucoin *et al.*, 2016; Gonzalez de Castro *et al.*, 2013; Wishart, 2016). The key role of gene expression (GE) in cellular machinery (Carrera *et al.*, 2014) and the cost-effective nature of high-throughput transcriptomics have renewed interest in predictors of gene expression as a proxy of the cellular state (Ay and Arnosti, 2011; Tachibana, 2015). If successful, an accurate predictive GE model can be useful in basic research on understanding how gene expression changes based on environmental stimuli, and in industrial biotechnology by guiding wetlab experimentation to those settings that are more likely to produce the desired results, from recombinant protein expression (Dragosits *et al.*, 2012; Mahalik *et al.*, 2014) to strain engineering (Riglar and Silver, 2018) and drug production (Milne *et al.*, 2009).

When it comes to prediction, artificial neural networks (ANN) outperform other methods in areas such as computer vision and machine translation (LeCun *et al.*, 2015). Despite their success in complex prediction tasks (LeCun *et al.*, 2015; Miotto *et al.*, 2018), their application for steady state GE prediction has been quite limited (Ching *et al.*, 2018). Instead most researchers rely on methods based on linear models, molecular thermodynamics, differential equations, logical circuits and Bayesian networks (Ay and Arnosti, 2011; Kim *et al.*, 2009). The idea of using ANNs for GE prediction is not novel (Vohradsk, 2001) but its adoption has remained stale due to lack of data and limited predictive power of the algorithms so far. Recently, groundbreaking ideas around deep neural networks (DNN) accompanied with the availability of vast transcriptomics repositories have created an unprecedented opportunity to create accurate predictors for genome-wide expression (Ma *et al.*, 2018). For example, a recurrent neural network (RNN) was employed as part of a genome-scale model trained on twenty million data points for steady state GE prediction in novel conditions for bacterium *Escherichia coli* (Kim *et al.*, 2016). In another study, A 3-layer feedforward neural network (FNN) was used for GE prediction when the expression of landmark genes were given (Chen *et al.*, 2016). More recently, a convolutional neural network (CNN) called DeepChrome was used to predict GE from histone modifications (Singh *et al.*, 2016) and a similar tool, DeepPep was developed for predicting protein occurrence in proteomics samples (Kim *et al.*, 2017).

There are several technical challenges to overcome when building an ANN GE model. First, one has to optimize the ANN hyperparameters that determine the underlying ANN architecture. Although general architectures can be trained, architectures that are tailored to incorporate the key properties of a given problem tend to be substantially more accurate. For example CNNs are designed to excel in image tasks (Krizhevsky *et al.*, 2012) based on the idea that high level features in an image can be identified by hierarchical combination of local features (e.g. for a face to be identified, two eyes, nose and mouth would be identified nearby each other). Recently, a new type of ANN called visual interaction network is developed to capture dynamics of physical objects (e.g. billiard balls) from video frames in order to predict object's physical trajectory (Watters *et al.*, 2017). Schema networks take this idea further by capturing causalities between object dynamics as well to enhance prediction in transfer learning (Kansky *et al.*, 2017). Currently, there is no ANN architecture that maps well to the gene regulatory dynamics and the complex expression signatures they produce within cells. To address this challenge, we developed a novel feed-forward architecture, coined *Genetic Neural Network*(GNN), that is founded on the observation that gene expression in prokaryotic systems is influenced, at least partially, by the expression level of its transcription factors (TF) (Bonneau *et al.*, 2006; Fang *et al.*, 2017). The fundamental building block of the GNN is the *GNN layer or cell*, a type of node that has been designed to capture the dynamics that govern gene regulation.

A second challenge in training ANNs is to produce sufficient data to avoid over-fitting. DNNs are notorious for their need of large datasets: for instance, ImageNet that is used to train computer vision DNNs contains 14 million images (Deng *et al.*, 2009). In contrast, for the most widely studied bacterium, *E.coli*, there are only 4389 GE profiles, each with the expression of its approximately 4500 genes, across 649 conditions (Kim *et al.*, 2016). One of the common approaches for mitigating the data gap in ML is constrained optimization. In the GNN architecture that we introduce here, we achieve that by constraining the connectivity of the GNN layers based on the transcriptional regulatory network of the organism, which is (partially) known from public databases. These two features, namely the introduction of a new node type and an architecture that have been designed to mimic gene regulatory and expression dynamics, are the key innovations behind the superior performance of GNNs and the main contributions of this paper.

In this article, we focus on steady state GE prediction for small to medium size transcriptional network modules (between 2 and 1000 genes) and with the assumption that the expression of *master regulator (MR)* genes are known. Since MR genes sit on top of the regulatory hierarchy, they play a key role in transcriptional regulation. Given the causal role of MR genes on the GE profile, models that accurately predict the impact of their perturbation are important. In Section 2, we describe how we define, construct and train the GNN model. In Section 3, we introduce competing methods that we compare against and in Section 4 we describe the results of these performance review. The overall methodology is summarized in Figure 1.

2 Materials and methods

2.1 GNN architecture

The **input layer** of a GNN consists of the expression level of MR genes and gene knockout information (referred to as '*a*' and '*KO*' in Fig. 2). The **output layer** of GNN consists of the predicted gene expression levels. Each **intermediate layer** in GNN predicts expression of a single gene, for instance, L_1, L_2 and L_3 predict the expression of gene a, b and c, respectively (Fig. 2). This architecture is built under the assumption that expression of a gene with d regulators, can be estimated using the activation function $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ where $\boldsymbol{x} \in \mathbb{R}^d_{\geq 0}$ represents the expression level of regulator genes. Therefore inputs of the activation function $f_{\boldsymbol{\theta}}$ for each gene, are made available by ensuring a topological order. For example, in Figure 2**A**, the expression of gene *c* is regulated by gene *b*. Therefore designated layer of *b* (i.e. the L_1 cell) comes before designated layer of *c* (i.e. the L_2 cell) as in Figure 2**B**. Note that topological order is not unique for cyclic graphs. Therefore, when a cycle is detected (here by using depth-first-search), we remove the feedback edges before generating the topological gene ordering.

The **activation function** $f_{\boldsymbol{\theta}}$ is based on the generalized logistic function that recapitulates the non-linear dynamics that govern gene expression, usually modeled through the Hill function (Rosenfeld *et al.*, 2005). More specifically, $f_{\boldsymbol{\theta}}$ is given by:

$$f_\theta(\boldsymbol{x}) = \frac{t_0 + \sum_{k=1}^{d} t_k e^{p_k x_k}}{1 + \sum_{k=1}^{d} b_k e^{p_k x_k}} \tag{1}$$

where θ is the set of function parameters including the input weight vector $\boldsymbol{p} \in \mathbb{R}^d$, numerator weight vector $\boldsymbol{t} \in \mathbb{R}^d_{\geq 0}$, denominator weight vector $\boldsymbol{b} \in \mathbb{R}^d_{\geq 0}$ and bias $t_0 \in \mathbb{R}_{\geq 0}$. Assuming θ is known for all layers, one forward pass results in predicted expression levels $\hat{y}$ for all genes. The final predictions are clamped into a valid range $[y_{min}, y_{max}]$. This will be $[0, 1]$ if data is normalized to this range. Otherwise y_{min} and y_{max} are the minimum and maximum values observed for each gene in the whole dataset. Layer-wise GNN training for θ estimation is explained in the next section.

2.2 Layer-wise training

A separate regression problem is defined in each layer (i.e. for each gene). For the layer corresponding to a particular gene, a corresponding dataset $C = \{X, \mathbf{y}\}$ that consists of regulator gene expression levels X and expression levels of the current gene $\mathbf{y}$ is created:

$$X = \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{x}^{(2)} \\ \vdots \\ \boldsymbol{x}^{(m)} \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \tag{2}$$

To predict $\hat{y}^{(i)} = f_\theta(\boldsymbol{x}^{(i)}),\ \forall i = 1 \ldots m$, we devise a loss function:

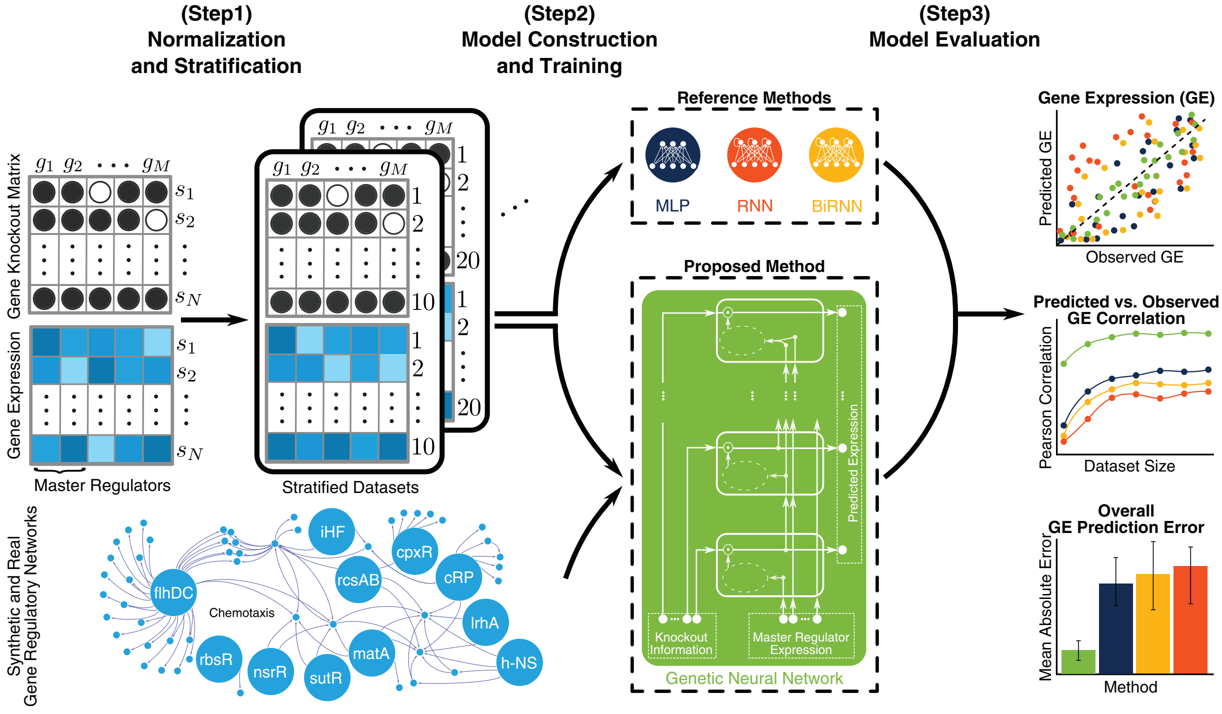

Fig. 1. The proposed Genetic Neural Network (GNN) architecture, is evaluated against various ANN architectures and ANN types (MLPs, RNNs, BiRNNs) in its ability to predict gene expression levels given master regulator expression and knockout information. In Step 1, a compendium of normalized expression levels over a wide spectrum of conditions is created, together with the contextual gene regulatory network information [Chemotaxis pathway here, retrieved from Gama-Castro *et al.* (2016)]. Stratified datasets of various sizes are generated after normalization to drive Step 2, where ANN models are constructed and trained. When applicable, the model architecture is informed by known regulatory relationships. In Step 3, the methods are evaluated through 5-fold cross validation on their predictive performance on gene expression

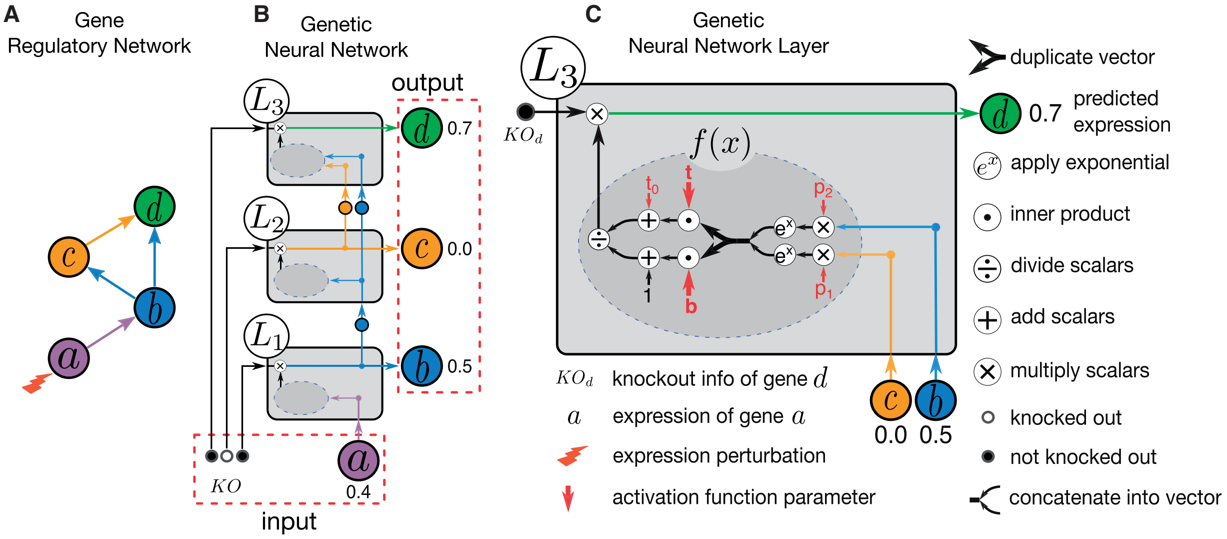

Fig. 2. Genetic Neural Network architecture schematic for a regulatory network example. The aim is to predict gene expression levels given the expression of master regulator[s] (MR) and knockout information (*KO*) for other genes. (**A**) An example gene regulatory network, consisting of a single MR '*a*' and three other genes *b*, *c* and *d* in topological order. Each arrow indicates a direct regulatory relationship. (**B**) The Genetic Neural Network topology that would map the regulatory relationships of the example gene regulatory network. The input consists of the MR expression level *a* and the knockout vector *KO*. Each layer corresponds to a single gene (i.e. L_1, L_2, L_3 correspond to *b*, *c*, *d*, respectively). Prediction of non-MR gene expression is achieved by a forward-pass, from first layer (L_1) to last (L_3). This order ensures that for each layer the expression levels of the regulator[s] are available for the layer's forward pass. (**C**) A dissection of a layer (i.e. GNN node). It consists of the MR gene expression levels *x*, the activation function *f*, knockout information (e.g. KO_d) and finally the output vector by appending the predicted expression (e.g. $\hat{d}$) to the initial inputs of current layer when needed by subsequent layer[s]. Although only L_3 is illustrated in detail, the general form of layers are the same while the inputs and weights vary depending on regulators and training data

$$loss(C, \boldsymbol{\theta}) = \sum_{\mathrm{i}=1}^{\mathrm{m}} [f_{\theta}(\boldsymbol{x}^{(\mathrm{i})}) - y^{(\mathrm{i})}]^2 \tag{3}$$

Hence optimal $\boldsymbol{\theta}$ can be determined by minimizing $loss(C, \boldsymbol{\theta})$:

$$\boldsymbol{\theta}^* = obj_1(C) = \underset{\boldsymbol{\theta}}{ArgMin}\ loss(C, \boldsymbol{\theta}) \tag{4}$$

In order to solve (4) first we show that for a given parameter vector $\boldsymbol{p}$, the parameters $\boldsymbol{w}^* = [t_0^*|\boldsymbol{t}^*|\boldsymbol{b}^*]$ can be uniquely determined using a linear program. To see this, let us assume $\boldsymbol{w}^*$ is determined. Hence we have:

$$\boldsymbol{\theta}^+ = \{t_0^*, \boldsymbol{t}^*, \boldsymbol{b}^*, \boldsymbol{p}\} \tag{5}$$

where $\boldsymbol{\theta}^+$, is the set of parameters for f where $t_0^*, \boldsymbol{t}^*, \boldsymbol{b}^*$ minimize the $loss$ for a given $\boldsymbol{p}$. Therefore predicted expression of each gene $\hat{y}^{(i)}$ can be calculated given the corresponding TF expressions $\boldsymbol{x}^{(i)}$ for $i = 1 \ldots m$:

$$f_{\theta^+}(\boldsymbol{x}^{(i)}) = \frac{t_0^* + \sum_{k=1}^{d} t_k^* h_k^{(i)}}{1 + \sum_{k=1}^{d} b_k^* h_k^{(i)}} = \hat{y}^{(i)},\ h_k^{(i)} = e^{p_k x_k^{(i)}} \tag{6}$$

Assuming $\boldsymbol{b}^* \geqslant 0$, the denominator above will be non-zero hence we can rewrite as:

$$t_0^* + \sum_{k=1}^{d} t_k^* h_k^{(i)} - \hat{y}^{(i)} \sum_{k=1}^{d} b_k^* h_k^{(i)} = \hat{y}^{(i)} \tag{7}$$

Considering $\hat{y}^{(i)} \approx y^{(i)}$, we have:

$$t_0^* + \sum_{k=1}^{d} t_k^* h_k^{(i)} - y^{(i)} \sum_{k=1}^{d} b_k^* h_k^{(i)} \approx y^{(i)} \tag{8}$$

To convert this into matrix form, we define vector $\boldsymbol{w}^*$ and matrices H, Y_e and A.

$$\boldsymbol{w}^* = [t_0^*, t_1^*, t_2^*, \ldots, t_d^*, b_1^*, b_2^*, \ldots, b_d^*]^T \tag{9}$$

Matrix H consists of $h_k^{(i)}$ values (constant for given $\boldsymbol{p}$ and X):

$$H = \begin{bmatrix} h_1^{(1)} & h_2^{(1)} & \ldots & h_d^{(1)} \\ h_1^{(2)} & h_2^{(2)} & \ldots & h_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ h_1^{(m)} & h_2^{(m)} & \ldots & h_d^{(m)} \end{bmatrix}_{m \times d} \tag{10}$$

Matrix Y_e consists of expression levels $y^{(i)}$ repeated d times in columns (constant for given $\boldsymbol{y}$):

$$Y_e = \begin{bmatrix} y^{(1)} & y^{(1)} & \ldots & y^{(1)} \\ y^{(2)} & y^{(2)} & \ldots & y^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ y^{(m)} & y^{(m)} & \ldots & y^{(m)} \end{bmatrix}_{m \times d} \tag{11}$$

Matrix A is calculated using H and Y_e where '$\odot$' represents entry-wise multiplication and '|' represents column-wise matrix concatenation:

$$A = [\mathbf{1}|H|(-Y_e \odot H)]_{m \times (2d+1)} \tag{12}$$

Therefore Equation (8) can be represented in matrix form:

$$A.\boldsymbol{w}^* \approx \boldsymbol{y} \tag{13}$$

To see this, note that for each i: (I) the inner product of $\boldsymbol{w}$ with ith row of A corresponds to the terms on the left side of Equation (8) and (II) the ith element of $\boldsymbol{y}$ corresponds to the term on right side of Equation (8).

To convert the approximation in (13) to equality, we add $\boldsymbol{\epsilon_l}, \boldsymbol{\epsilon_r} \in \mathbb{R}^m_{\geq 0}$ to both sides:

$$A.\boldsymbol{w}^* + \boldsymbol{\epsilon_l} = \boldsymbol{y} + \boldsymbol{\epsilon_r} \tag{14}$$

Therefore, the desirable $\boldsymbol{w}^*$ should minimize approximation error $\sum_{i=1}^{m} \boldsymbol{\epsilon_l}^{(i)} + \boldsymbol{\epsilon_r}^{(i)}$. To find $\boldsymbol{w}^*$, we devised obj_2:

$$\begin{aligned} \boldsymbol{w}^* = obj_2(C, p) = \underset{w}{ArgMin} \quad & 1^T\boldsymbol{\epsilon_l} + 1^T\boldsymbol{\epsilon_r} \\ \text{subject to} \quad & A.\boldsymbol{w}^* + \boldsymbol{\epsilon_l} - \boldsymbol{\epsilon_r} = \boldsymbol{y} \\ & \boldsymbol{\epsilon_l} \geqslant 0, \boldsymbol{\epsilon_r} \geqslant 0, \boldsymbol{b} \geqslant 0, \boldsymbol{t} \geqslant 0 \end{aligned} \tag{15}$$

This can be transformed into standard linear programming (LP) form:

$$\begin{aligned} \boldsymbol{w}^* = & \underset{z}{ArgMin}\ \boldsymbol{a}^T.\boldsymbol{z} \\ & \text{subject to } G.\boldsymbol{z} = \boldsymbol{y} \\ & \boldsymbol{z} \geq \boldsymbol{l} \end{aligned} \tag{16}$$

where

$$\boldsymbol{z} = \begin{bmatrix} t_0 \\ \boldsymbol{t} \\ \boldsymbol{b} \\ \boldsymbol{\epsilon_l} \\ \boldsymbol{\epsilon_r} \end{bmatrix}, \boldsymbol{a} = \begin{bmatrix} 0 \\ 0_{d\times1} \\ 0_{d\times1} \\ \mathbf{1}_{m\times1} \\ \mathbf{1}_{m\times1} \end{bmatrix}, \boldsymbol{l} = \begin{bmatrix} 0 \\ 0_{d\times1} \\ 0_{d\times1} \\ 0_{m\times1} \\ 0_{m\times1} \end{bmatrix}, \quad G = [A_{m\times(2d+1)}|I_{m\times m}| - I_{m\times m}] \tag{17}$$

Therefore, for a given input coefficient $\boldsymbol{p}$ and gene expression values $C = \{X, \boldsymbol{y}\}$, the optimal $\boldsymbol{w}^*$ can be estimated by solving $obj_2(C, \boldsymbol{p})$ using the linear program in Equation (16). With this insight, we can solve $obj_1(C)$ (Equation (4)) using an iterative algorithm starting from an initial $\boldsymbol{p}$ vector. In each iteration, first $\boldsymbol{w}^*$ is estimated using $obj_2(C, \boldsymbol{p})$. Then $loss(C, [\boldsymbol{w}^*|\boldsymbol{p}])$ and its gradient w.r.t $\boldsymbol{p}$ are calculated. Finally a new $\boldsymbol{p}$ is generated using the calculated $loss$ and its gradient. Although various gradient based optimization methods can be used for this iterative procedure, we used the conjugate gradient method (Møller, 1993). This is described in Algorithm 1. Line 6 refers to the first step. Line 7 and 8 refer to second step. The last step is done by the *ConjugateGradient* function in each iteration.

In practice we run the algorithm 10 times with different initial random values for $\boldsymbol{p}$ and use the one which gives the best fit (i.e. lowest value for $loss$ in Equation (3)). The complexity of layer-wise training algorithm is $O(m^{5.5})$ as derived in Supplementary Section S1.

Note that, there are two choices for matrix X values. First is to use the actual GE values from the training dataset. Second is to replace actual GE values with corresponding predicted ones when calculated from a previous layer. In our experiments the second choice provided slightly better predictive power (hence used for the presented results of Section 4).

3 Competing methods

We compare the GNN method against LASSO, a linear model with ℓ_1 regularization (Tibshirani, 1996), a Multi-layered Perceptron (MLP) (Hornik *et al.*, 1989), a recurrent neural network (RNN) (Pineda, 1987), a bi-directional neural network (BiRNN) (Schuster and Paliwal, 1997) and a linear version of our GNN network (LinGNN). Recall that in our prediction task, the input vector consists of the expression level of the master regulator (MR) genes and

Algorithm 1: Layer-wise training algorithm to estimate activation function parameter vector $\boldsymbol{\theta} = [\boldsymbol{w}|\boldsymbol{p}]$ where $\boldsymbol{p}$ and $\boldsymbol{w}$ consist of input and exponential coefficients, respectively (see Equation (1)). The gene expression dataset C related to a gene, consists of X and $\boldsymbol{y}$. The matrix X contains observed TF expression levels (i.e. inputs to the activation function). The vector $\boldsymbol{y}$ contains corresponding observed GE values for this gene (i.e. activation function outputs).

```
Input: gene expression dataset $C = \{X, y\}$
Output: activation function estimated parameter vector $\theta^* = [w|p]$
$w \leftarrow 0$ (Note: $w$ is a global variable)
$p \leftarrow$ random initial vector
$p \leftarrow$ ConjugateGradient (GetLossP, $p$, C)
return $[w|p]$
Function GetLossP ($p$, C):
    $w \leftarrow obj_2(C, p)$ // solve LP from equation (16)
    $loss\_p \leftarrow loss(C, [w|p])$ //equation (3)
    $loss\_p\_grad \leftarrow \frac{\nabla loss(C,[w|p])}{\nabla p}$
    return $loss\_p$, $loss\_p\_grad$
Function ConjugateGradient (f, x, C):
    /* f: cost function takes variables x and C
       as input and returns the cost and the
       gradient with respect to x.          */
    /* *** Conjugate Gradient Implementation
       ***                                  */
    return $x^*$
```

the knockout information vector. Here, we use the vector $\boldsymbol{v}$ as the concatenation of all inputs, the vector $\boldsymbol{y}$ as the expression level of non-MR genes, and $\hat{\boldsymbol{y}}$ referring to their predicted values. The various ANN architectures are illustrated in Supplementary Figure S2. Unlike GNN, the common ANN architectures have hyper-parameters that need to be first optimized. For our comparison, we use the hyper-parameters that correspond to the best-performing architecture (i.e. the architecture with minimum MAE) by using a traditional search method (Bergstra and Bengio, 2012).

Multi-layer perceptron (MLP): MLP is used by Chen *et al.* (2016) for GE prediction when expression of landmark genes are known. An MLP instance here takes an *input* vector $\boldsymbol{v}$ and calculates the *output* vector $\hat{\boldsymbol{y}}$. To identify the *hyper-parameters*, we examine architectures with 0–3 hidden layers, 5–50 hidden nodes per layer with ℓ_2 regularization coefficient between 0.0 and 0.5.

Recurrent neural network (RNN): RNN is used by Kim *et al.* (2017) for GE prediction when genetic and environmental perturbations are characterized as input. Similar to an RNN used by Kim *et al.* (2017), a fully connected RNN instance here, takes a sequence of *input vectors* (Williams and Zipser, 1989). The same vector $\boldsymbol{v}$ is repeated multiple times (depending on the depth *hyper-parameter* t) as input. The *output* vector of RNN $\hat{\boldsymbol{y}}$ corresponds to the output of the last rollout of the RNN (only). For *hyper-parameters*, we examine architectures with depth t between 1 and 20 and ℓ_2 regularization coefficients between 0.0 and 0.5.

Bidirectional recurrent neural network (BiRNN): Our BiRNN instances are set-up exactly same as our simple RNN ones except that they are bidirectional.

LASSO: Linear regression with ℓ_1 regularization (i.e. LASSO) is a widely used regression method that improves the generalization power of the linear model by reducing the number of features through an ℓ_1 penalty in the objective function (Tibshirani, 1996). In our setting, it is equivalent to an MLP with no hidden layers, identity activation function and ℓ_1 regularization. For *hyper-parameter* optimization, we examined regularization coefficient from 0.0 to 5.0.

For training competing ANN architectures and Lasso, we used RMSProp (Tieleman and Hinton, 2012). The loss function used in RMSProp here is the mean squared error (MSE) plus regularization of model weights w as in Equation 18. We run RMSProp with learning rate of 0.001 until convergence. The training is stopped whenever MSE has less than 0.0001 improvement in the last 100 epochs.

$$loss_w = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 + \lambda_1||\boldsymbol{w}|| + \lambda_2||\boldsymbol{w}||_2^2 \qquad (18)$$

Linear GNN (LinGNN): Regulatory network connections can be incorporated into a linear model for GE prediction, given TF expression level as it has demonstrated in previous models (Carrera *et al.*, 2009; Galagan *et al.*, 2013). To evaluate the performance of a linear model with the proposed architecture, we developed LinGNN, which has the same GNN framework, but a linear function in Equation (19) is used for node activation, instead of the nonlinear activation in Equation (1). Here $b \in \mathbb{R}$ is the bias term, $a \in \mathbb{R}^d$ vector consists of additive coefficients and $M \in \mathbb{R}^{d\times d}$ consists of multiplicative coefficients.

$$f_{b,a,M}(\boldsymbol{x}) = b + \sum_{i=1}^{d}(a_i x_i + \sum_{j=1}^{d} M_{i,j} x_i x_j) \qquad (19)$$

For training the LinGNN, the same layer-wise training strategy is used. However given the linear function, we used the OLS for parameter fitting to solve Equation (4) (instead of Algorithm 1).

3.1 Evaluation metrics

To evaluate the model performance given observed GE values $\boldsymbol{y}$ and corresponding predicted GE values $\hat{\boldsymbol{y}}$, we use the Pearson Correlation Coefficient (PCC) and Mean Absolute Error (MAE) (Kvålseth, 1985).

4 Empirical results

We designed two sets of experiments to assess the impact of network complexity and data availability (aforementioned challenges) on predictive power.

4.1 Network complexity impact

First, we constructed the full TRN of *E.coli* curated by RegulonDB v9.4 (Gama-Castro *et al.*, 2016). Second, from this full TRN, we extracted 33 network modules each containing between 10 and 1000 genes. This was done using the greedy module extraction method in GeneNetWeaver software (Schaffter *et al.*, 2011) also used in number of DREAM challenges (Marbach *et al.*, 2010, 2012). Third, for each extracted TRN module, we identified the MR genes by selecting genes that are not regulated by any other gene within that network. Fourth, for each TRN module, we performed thousands of steady state thermodynamic simulation experiments using GeneNetWeaver with added microarray noise. Each simulation run requires kinetic parameters for each gene which are randomly initialized by the simulation software (and unknown to GNN and other predictive models). These *in silico* experiments consist of

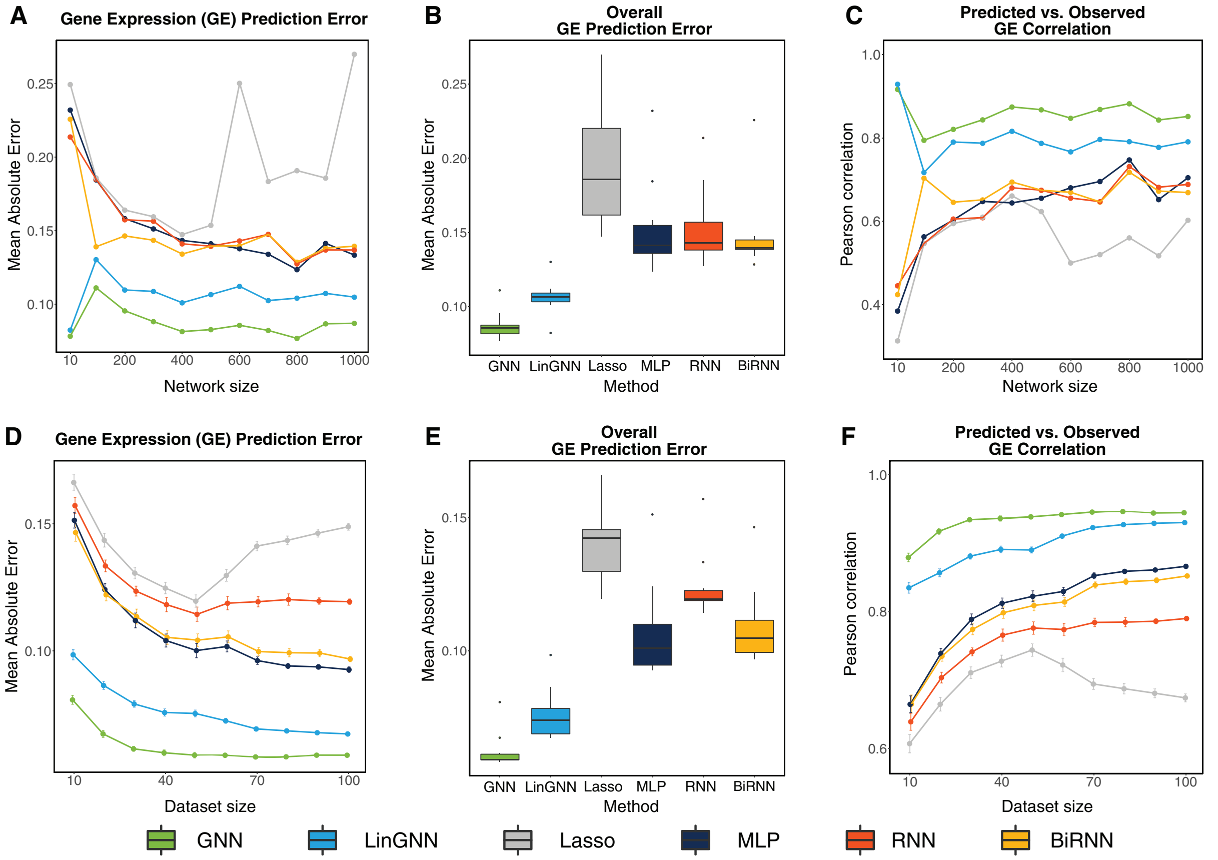

Fig. 3. 5-fold cross validation performance evaluation using data acquired through thermodynamic simulation. (**A**), (**B**) and **C** show performance of all methods given randomly selected network modules for *E.coli* transcription network. Better performance of GNN can be seen in all cases. The BiRNN architecture outperforms other conventional ANN architectures in some cases. (**D**), (**E**) and (**F**) show the prediction performance for TRN module of chemotaxis (61 genes) in *E.coli*. Results suggest that GNN performs in call cases particularly on smaller datasets

random multi-factorial perturbations for MR genes and single gene knockouts for other genes giving rise to GE dataset with thousands GE profiles. Fifth, for each GE dataset we identified 10 dissimilar GE profiles. To identify these dissimilar GE profiles in a dataset, we performed hierarchical clustering (Day and Edelsbrunner, 1984) with cluster size of 10 and selected one GE profile from each cluster randomly. Finally, we performed 5-fold cross-validation (CR) for the task of GE prediction given MR expression levels and knockout information in each dataset. Results in Figure 3A and C, show that GNN method outperforms other methods on datasets with network sizes ranging between 10 and 1000 when compared based on MAE and PCC metrics in 5-fold CR setting. Our results show that the GNN has a smaller error (average MAE 0.09 ± 0.01; PCC 0.86 ± 0.03) than LinGNN (average MAE 0.11 ± 0.01; PCC 0.80 ± 0.05), Lasso (average MAE 0.19 ± 0.04; PCC 0.55 ± 0.09), MLP (average MAE 0.15 ± 0.03; PCC 0.63 ± 0.10), RNN (average MAE 0.15 ± 0.03; PCC 0.63 ± 0.08) and BiRNN (average MAE 0.15 ± 0.03; PCC 0.65 ± 0.08). Figure 3B depicts the overall performance on all datasets. Schematic of the synthetic data generation pipeline is presented in Supplementary Figure S1.

4.2 Effect of data availability

In order to assess predictive power of various architectures based on data availability, we picked the transcription network of chemotaxis since it is one of the most well-studied signaling pathways (Long *et al.*, 2017).

First, we constructed the TRN of chemotaxis. To do so, we used KEGG (Kanehisa *et al.*, 2017) to get list of 57 genes involved in chemotaxis signal transduction pathway of bacterium *E.coli*. We then added to this list, the genes directly involved in transcription of chemotaxis genes. We then removed genes that are not involved in any transcriptional regulation (i.e. genes that have no reported TF listed) hence 61 genes with 84 TF-Gene relationships remained. For TF-Gene relationships we used all regulatory relationships that are based on experimental evidence curated in RegulonDB v9.4. Second, we identify MR genes in the chemotaxis TRN as the list of genes with no TF within the chemotaxis TRN. Third, we performed thermodynamic simulations (with same method mentioned in 4.1) 100 times. Each time we used different set of network parameters (TF binding affinities, degradation rates, etc.) for the chemotaxis network using GeneNetWeaver. This gave us 100 different datasets each with thousands of GE profiles and their corresponding knockout information. Fourth, from each GE dataset we extract 10 stratified datasets with varying sizes (10–100 GE profiles each). To generate a stratified dataset of size *K*, we perform hierarchical clustering (Day and Edelsbrunner, 1984) with number of clusters set to *K* and randomly pick one profile from each cluster. Finally, to evaluate performance of GNN and competing methods we perform 5-fold CR validation for the task of GE prediction given expression level of MR genes and knockout information. Results in Figure 3D and F, show that GNN method outperforms other methods on stratified datasets with sizes ranging between 10 and 100 profiles each when compared based on MAE and PCC metrics in 5-fold CR setting. Figure 3E shows the overall performance on all datasets. Note that the gap between GNN and other methods is larger on smaller dataset sizes. Supplementary Figures S3 and S4 show predictive performances in higher resolution for a randomly selected chemotaxis dataset related to 10 distinct experiments.

4.3 In vivo experiments

For in vivo evaluation, we used Affymetrix gene expression dataset of bacterium *E.coli* [compiled and made available by Marbach *et al.* (2012) also known as DREAM5 challenge data]. The dataset had been normalized already using Robust Multichip Averaging (RMA) (Bolstad *et al.*, 2003). The compendium's GE data corresponds to genetic and environmental perturbation experiments on various strains. We only used profiles from the wild type strain (MG1655) for our evaluations. There are 427 wild type GE profiles. For replicates, we use the mean GE value resulting in 227 GE profiles corresponding to unique experimental settings.

4.3.1 Transcription network

To identify transcription network, we used GENIE3 (Irrthum *et al.*, 2010) which performed best for transcription network inference in DREAM5 challenge (Marbach *et al.*, 2012). The network inference method GENIE3 takes GE data as input and produces a list of TF-Gene relationships ordered based on confidence level we call *edge_candidates*.

4.3.2 Master regulators

The set of transcription factors on top of the regulatory hierarchy are referred to as master regulators (Chan and Kyba, 2013). To define this more concretely, we use directional graph $G = \{V, E\}$ to represent a TRN where gene x is represented by vertex $v_x \in V$. An edge $(v_y, v_x) \in E$ represents transcriptional regulation of gene x by the product of gene y. The estimated confidence for edges are stored in matrix W where $W_{x,y} \in \mathbb{R}_{\geq 0}$ represents the reported confidence from network inference for edge $(v_y, v_x) \in E$. Note that $W_{x,y} = 0$ if there is no edge between vertices v_x, v_y. Here an MR gene is considered to be a TF gene that is not regulated by any other TF. Additionally in cases where genes inside a regulatory cycle are non-reachable using any MR gene, the gene inside the cycle with maximum *impact* will be selected as MR among them. The *impact*(x) for given gene x is calculated as in Equation (20) where d_x is the number of genes regulated by the product of gene x:

$$impact(x) = \frac{1}{d_x} \sum_{j \in V} W_{x,j} \tag{20}$$

4.3.3 In vivo evaluation pipeline and results

Sub-sampling and network inference: We generate 10 datasets using stratified sampling (each containing 11 GE profiles). For each of the 10 datasets, we perform network inference using the remaining samples by GENIE3. This provides 10 networks, each with a dataset that was not used to infer the network. From each network, we extract 11 TRN modules with number of genes ranging from 10 to 1000 generating 110 TRN modules in total. To extract a module with N number of genes, we start with an empty network $G = \{V, E\}$. First, we add edges to this network starting from highest confidence (from *edge_candidates* list produced by GENIE3), until $|V| = N$. Second we add 20% more edges from *edge_candidates*. Finally we run the greedy module extraction method using GeneNetWeaver (Schaffter *et al.*, 2011) with the desired network size N to extract a TRN network module.

Dataset construction: For each network module, MR genes are identified using method explained in Section 4.3.2. Module's corresponding dataset (which consist of GE profiles not used in inferring the parent network) is then partitioned into input (GE of MR genes) and output (GE of non-MR genes). Stratified sampling and 5-fold cross validation is performed same as explained in Section 4.1.

The role of TRN information: We performed a separate experiment to evaluate the role of TRN information on predictive performance of methods. In this experiment, we randomly shuffle the gene names in the GE output data after the network inference step (this is same shuffling the node names in the network while preserving MR gene names). We then perform same 5-fold cross validation as explained before. This simulates a situation where the network information used by the model is random. Corresponding results are reported as GNN-rnd and LinGNN-rnd in Figure 4 using dashed lines.

Figure 4 summarizes the results indicating better overall prediction performance for GNN (average MAE 0.58 ± 0.02; PCC 0.81 ± 0.04) compared to LinGNN (average MAE 0.60 ± 0.01; PCC 0.79 ± 0.02), Lasso (average MAE 0.81 ± 0.04; PCC 0.70 ± 0.02), MLP (average MAE 0.78 ± 0.06; PCC 0.75 ± 0.06), RNN (average MAE 0.93 ± 0.12; PCC 0.68 ± 0.07) and BiRNN (average MAE 0.81 ± 0.03; PCC 0.74 ± 0.04). Note that in vivo GE values range from 3 to 15 while in silico GE values are normalized between 0 and one.

4.4 Runtime comparison

For runtime comparison of methods, we used a dataset with size 10 for a network of 1000 genes and evaluated the training time. As in Table 1, GNN is lacking in terms of runtime compared to other methods. LinGNN performs best due to fast OLS operations on small datasets. Other methods require hyper-parameter optimization adding to their runtime (e.g. BiRNN is slower than when 50 hyper-parameter combinations are used). Note that the training procedure (described in Section 2.2) is inherently parallel. Therefore a parallel implementation can make the training approximately n times faster where n is the number of cores used.

5 Conclusion

We presented GNN, an artificial neural network that incorporates gene regulatory network into it's architecture to predict GE in novel conditions given minimal training data. A trained GNN takes the expression level of MR genes and information about knockout experiments and predicts the expression of the rest of genes in the given transcription network. We compared GNN with three common neural network architectures, linear regression with ℓ_1 regularization and a network based linear model. Our comparison benchmarks include in vivo micro array data and thermodynamic simulation data for real biological network (e.g. chemotaxis). In our evaluations GNN showed considerably higher prediction performance when tested on hundreds of real TRNs extracted from *E.coli*'s full TRN. This was in spite of the fact that GNN did not enjoy the hyper-parameter optimization used for competing methods. The prediction performance gap was particularly higher on smaller datasets. Although this is not the first time ANNs were employed for GE prediction, this is a novel architecture with a custom

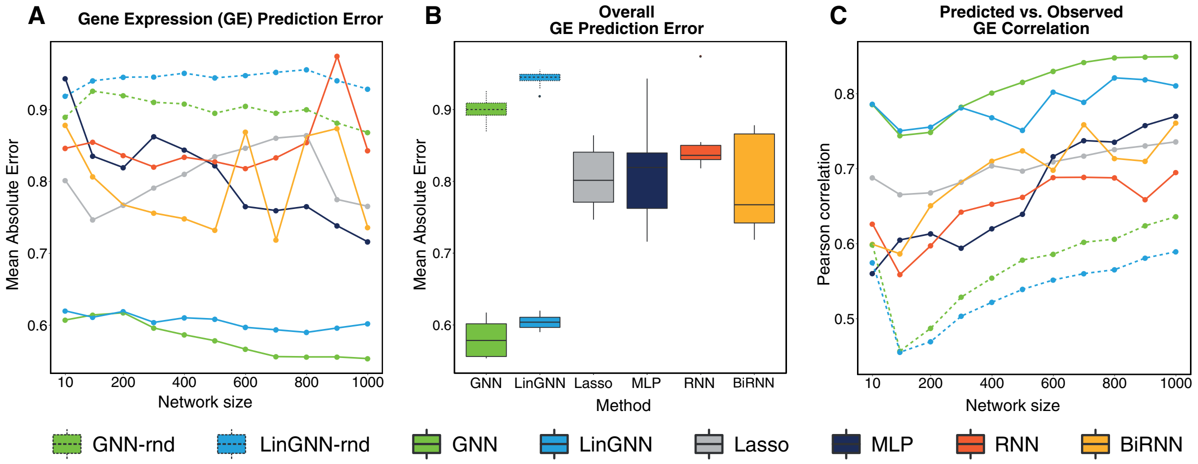

Fig. 4. Fivefold cross validation performance evaluation using in vivo microarray data. (**A**), (**B**) and (**C**) show performance of all methods on 110 randomly selected network modules on inferred transcription network of *E.coli.* GNN shows better overall performance. GNN-rnd and LinGNN-rnd show the performance of TRN based methods when inferred TRN is randomized

Table 1. Training time for dataset with 1000 genes and 10 GE profiles

Architecture	GNN	LinGNN	Lasso	MLP	RNN	BiRNN
Runtime (min: sec)	24: 26	0: 01	0: 12	0: 15	0: 27	1: 16

designed node that is tailored for gene expression prediction. Concomitantly, to best of our knowledge, this is the first time that TRN, expression of MR genes and gene knockout information are used together for this task.

One limitation of the GNN architecture that we described in this paper is that our implementation cannot take into account feedback loops, as it is based on a feed-forward network. Given the prevalence of cycles in biological networks (Brandman and Meyer, 2008), such limitation is expected to negatively impact predictive power. A natural extension would be to apply the GNN cell to recurrent neural networks (RNNs), which have the capacity to connect through time multiple instances of acyclic network maps, by feeding to the hidden layer of the next time slice, the hidden layer output of the previous time slice. Although individual GNN models are acyclic, together they have potential to model dynamics that arise in biological cycles. It would be also good to test the performance of this method in larger networks with tens of thousands of nodes. For that to happen with the non-linear GNNs, we need to take advantage of parallelism for the training algorithm, as training time is a consideration (Table 1). It would also help if the activation function is modified to one that can formulate a convex problem and its optimization in layer-wise training. Given that the linear GNN is performing quite well, despite its simplicity and more than an order of magnitude faster performance, a system that runs the LinGNN for very large networks (>5000 nodes) and non-linear GNN otherwise, would score high in prediction, runtime performance and scalability.

Extensions to this work include the integration of contextual information, such as gene sequence, experimental settings and metabolic pathways. More thorough validation in large compendia (e.g. see Kim *et al.*, 2016) and multiple pathways may further pinpoint the limitations of this approach. Although our focus here is bacterial regulation, this work can be extended to organisms of higher complexity, albeit with modifications that would rectify the large discrepancy in the number of genes [from 182 in bacterium *Carsonella ruddii* (Nakabachi *et al.*, 2006) to more than 20 000 in humans] and more complex modes of regulation.

Acknowledgement

This work used the Bridges cluster from Extreme Science and Engineering Discovery Environment (XSEDE) through allocation TG-BCS180001.

Funding

This work was supported by grants from National Science Foundation (1516695, 1743101 and 1254205) awarded to IT.

Conflict of Interest: none declared.

References

Abhyankar,W. *et al.* (2017) 'omics' for microbial food stability: proteomics for the development of predictive models for bacterial spore stress survival and outgrowth. *Int. J. Food Microbiol.*, **240**, 11–18.

Aucoin,H.R. *et al.* (2016) Omics in chlamydomonas for biofuel production. In: Yuki,N. and Yonghua,L.-B. (eds) *Lipids in Plant and Algae Development*. Springer International Publishing, Switzerland pp. 447–469.

Ay,A. and Arnosti,D.N. (2011) Mathematical modeling of gene expression: a guide for the perplexed biologist. *Crit. Rev. Biochem. Mol. Biol.*, **46**, 137–151.

Bergstra,J. and Bengio,Y. (2012) Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, **13**, 281–305.

Bolstad,B.M. *et al.* (2003) A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, **19**, 185–193.

Bonneau,R. *et al.* (2006) The inferelator: an algorithm for learning parsimonious regulatory networks from systems-biology data sets de novo. *Genome Biol.*, **7**, R36.

Brandman,O. and Meyer,T. (2008) Feedback loops shape cellular signals in space and time. *Science*, **322**, 390–395.

Carrera,J. *et al.* (2009) Model-based redesign of global transcription regulation. *Nucleic Acids Res.*, **37**, e38–e38.

Carrera,J. *et al.* (2014) An integrative, multi-scale, genome-wide model reveals the phenotypic landscape of *Escherichia coli*. *Mol. Syst. Biol.*, **10**, 735.

Chan,S.S.-K. and Kyba,M. (2013) What is a master regulator? *J. Stem Cell Res. Ther.*, **3**, 1–2.

Chen,Y. *et al.* (2016) Gene expression inference with deep learning. *Bioinformatics*, **32**, 1832–1839.

Ching,T. *et al.* (2018) Opportunities and obstacles for deep learning in biology and medicine. *J. Royal Soc. Interface*, **15**, 20170387.

Day,W.H. and Edelsbrunner,H. (1984) Efficient algorithms for agglomerative hierarchical clustering methods. *J. Class.*, **1**, 7–24.

Deng,J. *et al.* (2009) Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition, 2009. CVPR 2009*. IEEE, Miami, FL, pp. 248–255.

Dragosits,M. *et al.* (2012) A synthetic biology approach to self-regulatory recombinant protein production in *Escherichia coli*. *J. Biol. Eng.*, **6**, 2.

Fang,X. *et al.* (2017) Global transcriptional regulatory network for *Escherichia coli* robustly connects gene expression to transcription factor activities. *Proc. Natl. Acad. Sci. USA*, **114**, 10286–10291.

Galagan,J.E. *et al.* (2013) The mycobacterium tuberculosis regulatory network and hypoxia. *Nature*, **499**, 178.

Gama-Castro,S. *et al.* (2016) Regulondb version 9.0: high-level integration of gene regulation, coexpression, motif clustering and beyond. *Nucleic Acids Res.*, **44**, D133–D143.

Gonzalez de Castro,D. *et al.* (2013) Personalized cancer medicine: molecular diagnostics, predictive biomarkers, and drug resistance. *Clin. Pharmacol. Therap.*, **93**, 252–259.

Hornik,K. *et al.* (1989) Multilayer feedforward networks are universal approximators. *Neural Netw.*, **2**, 359–366.

Irrthum,A. *et al.* (2010) Inferring regulatory networks from expression data using tree-based methods. *PLoS One*, **5**, e12776.

Kanehisa,M. *et al.* (2017) Kegg: new perspectives on genomes, pathways, diseases and drugs. *Nucleic Acids Res.*, **45**, D353–D361.

Kansky,K. *et al.* (2017) Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *arXiv preprint arXiv: 1706.04317*.

Kim,H.D. *et al.* (2009) Transcriptional regulatory circuits: predicting numbers from alphabets. *Science*, **325**, 429–432.

Kim,M. *et al.* (2016) Multi-omics integration accurately predicts cellular state in unexplored conditions for *Escherichia coli*. *Nat. Commun.*, **7**, 13090.

Kim,M. *et al.* (2017) Deeppep: deep proteome inference from peptide profiles. *PLoS Comput. Biol.*, **13**, e1005661.

Krizhevsky,A. *et al.* (2012) Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*, Neural Information Processing Systems (NIPS), Lake Tahoe, Nevada, USA, pp. 1097–1105.

Kvålseth,T.O. (1985) Cautionary note about r 2. *Am. Stat.*, **39**, 279–285.

LeCun,Y. *et al.* (2015) Deep learning. *Nature*, **521**, 436.

Long,Z. *et al.* (2017) Cell-cell communication enhances bacterial chemotaxis toward external attractants. *Sci. Rep.*, **7**, 12855.

Ma,J. *et al.* (2018) Using deep learning to model the hierarchical structure and function of a cell. *Nat. Methods*, **15**, 290.

Mahalik,S. *et al.* (2014) Genome engineering for improved recombinant protein expression in *Escherichia coli*. *Microb. Cell Factories*, **13**, 177.

Marbach,D. *et al.* (2010) Revealing strengths and weaknesses of methods for gene network inference. *Proc. Natl. Acad. Sci. USA*, **107**, 6286–6291.

Marbach,D. *et al.* (2012) Wisdom of crowds for robust gene network inference. *Nat. Methods*, **9**, 796.

Milne,C.B. *et al.* (2009) Accomplishments in genome-scale in silico modeling for industrial and medical biotechnology. *Biotechnol. J.*, **4**, 1653–1670.

Miotto,R. *et al.* (2018) Deep learning for healthcare: review, opportunities and challenges. *Brief. Bioinf*, **19**, 1236–1246.

Møller,M.F. (1993) A scaled conjugate gradient algorithm for fast supervised learning. *Neural Netw.*, **6**, 525–533.

Nakabachi,A. *et al.* (2006) The 160-kilobase genome of the bacterial endosymbiont carsonella. *Science*, **314**, 267–267.

O'Brien,E.J. *et al.* (2015) Using genome-scale models to predict biological capabilities. *Cell*, **161**, 971–987.

Pineda,F.J. (1987) Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.*, **59**, 2229.

Riglar,D.T. and Silver,P.A. (2018) Engineering bacteria for diagnostic and therapeutic applications. *Nat. Rev. Microbiol.*, **16**, 214.

Rosenfeld,N. *et al.* (2005) Gene regulation at the single-cell level. *Science*, **307**, 1962–1965.

Schaffter,T. *et al.* (2011) Genenetweaver: in silico benchmark generation and performance profiling of network inference methods. *Bioinformatics*, **27**, 2263–2270.

Schuster,M. and Paliwal,K.K. (1997) Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.*, **45**, 2673–2681.

Singh,R. *et al.* (2016) Deepchrome: deep-learning for predicting gene expression from histone modifications. *Bioinformatics*, **32**, i639–i648.

Tachibana,C. (2015) Transcriptomics today: microarrays, RNA-seq, and more. *Science*, **349**, 544–546.

Tibshirani,R. (1996) Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B (Methodological)*, **58**, 267–288.

Tieleman,T. and Hinton,G. (2012) Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA Neural Netw. Mach. Learn.*, **4**, 26–31.

Vohradsk,J. (2001) Neural network model of gene expression. *FASEB J.*, **15**, 846–854.

Watters,N. *et al.* (2017) Visual interaction networks. *arXiv preprint arXiv: 1706.01433*.

Williams,R.J. and Zipser,D. (1989) A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, **1**, 270–280.

Wishart,D.S. (2016) Emerging applications of metabolomics in drug discovery and precision medicine. *Nat. Rev. Drug Discov.*, **15**, 473.