University College London
Engineering Faculty
Department of Computer Science
MSc Machine Learning

Adversarial generation of gene expression data

Ramon Viñas Torné

Supervised by:
Dr. Kevin Bryson, UCL

September 2018

This report is submitted as part requirement for the MSc degree in Machine Learning at University College London. It is substantially the result of my own work except where explicitly indicated in the text.

A la Neula, t'estimem.

The code for this project can be found at:
https://github.com/rvinas/
adversarial-generation-of-gene-expression-data

Abstract

The problem of reverse engineering gene regulatory networks from high-throughput expression data is one of the biggest challenges in bioinformatics. In order to benchmark network inference algorithms, simulators of well-characterized expression datasets are often required. However, existing simulators have been criticized because they fail to emulate key properties of gene expression data (Maier et al., 2013).

In this thesis we address two problems. First, we study and propose mechanisms to faithfully assess the realism of a synthetic expression dataset. Second, we design an adversarial simulator of expression data, gGAN, based on a generative adversarial network (Goodfellow et al., 2014). We show that our model outperforms existing simulators by a large margin in terms of the realism of the generated data. More importantly, our results show that gGAN is, to our best knowledge, the first simulator that passes the Turing test for gene expression data proposed by Maier et al. (2013).

Acknowledgements

First, I would like to express my most sincere gratitude to *el Patronat de l'Escola Politècnica Superior de la Universitat de Girona* for providing me with the fellowship *Josep Maria Ginés i Pous*. I could not have studied Machine Learning at *University College London* without this fellowship.

Second, I am deeply grateful to Dr. Kevin Bryson for his brilliant supervision on my project and his unfailing support throughout the process of my research. I have learnt a lot from his feedback and it has been a great pleasure to work under his guidance.

Last but not least, I would like to thank my family and friends for providing me with continuous encouragement during my years of study.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Objectives 2

1.3 Contribution 2

1.4 Outline 3

2 Related work **4**

2.1 SynTReN 4

2.1.1 Network topology 4

2.1.2 Transition functions 5

2.1.3 Generating data 5

2.2 GeneNetWeaver 6

2.2.1 Network topology 6

2.2.2 Dynamical model 7

3 Methods **8**

3.1 Evaluating artificial gene expression data 8

3.1.1 Clustering gene expression patterns 8

3.1.2 Maier's histograms of statistical properties of expression data 12

3.1.3 Critique to Maier's histograms 15

3.1.4 Our quality assessment metrics 16

3.2 Generative adversarial networks 21

3.2.1 Cost functions 22

3.2.2 Optimal discriminator 23

3.2.3 Global optimality 23

3.2.4 Connection to maximum likelihood estimation 24

3.2.5 Training a GAN 27

4 Experiments **28**
4.1 Dataset . . . 28
4.1.1 *E. coli* gene expression M^{3D} data . . . 28
4.1.2 *E. coli* gene regulatory interactions and RegulonDB . . . 29
4.1.3 Data preparation . . . 30
4.2 Model design . . . 33
4.2.1 Generator's architecture . . . 33
4.2.2 Discriminator's architecture . . . 34
4.3 Adversarial training . . . 36
4.3.1 Constraining the weights of the noise model . . . 36
4.3.2 Regularization . . . 36
4.3.3 Training algorithm . . . 37
4.3.4 Hyperparameter tuning . . . 38
4.4 Software and hardware references . . . 39
4.5 Results . . . 41
4.5.1 Evaluating gGAN gene expression data . . . 41
4.5.2 Comparison with other methods . . . 47

5 Conclusions **54**
5.1 Summary of thesis achievements . . . 54
5.2 Future work . . . 55

Bibliography **56**

A Additional figures **61**
A.1 Graphical model of our GAN . . . 61
A.2 Evolution of evaluation metrics along epochs . . . 62

B Random simulator **65**
B.1 Simulator of random expression data . . . 65
B.1.1 Evaluating expression data from random simulator . . . 65

C Other exploratory investigations within the project **69**
C.1 Matching the individual gene expression histograms from the *E. coli* M^{3D} dataset . . . 69

List of Abbreviations

Abbreviations

Adam adaptive moment estimation

cAMP cyclic adenosine monophosphate

CRP cAMP receptor protein

DCGAN deep convolutional generative adversarial network

DNA deoxyribonucleic acid

DREAM Dialogue for Reverse Engineering Assessments and Methods

E. coli *Escherichia coli*

GAN generative adversarial network

gGAN gene expression GAN

GNW GeneNetWeaver

GRI gene regulatory interaction

GRN gene regulatory network

JSD Jensen-Shannon divergence

KDE kernel density estimation

KL Kullback-Leibler divergence

M^{3D} Many Microbe Microarrays Database

MIT Massachusetts Institute of Technology

mRNA messenger RNA

ODE ordinary differential equation

PDF probability density function

ReLU rectified linear unit

RMA robust multi-array average

RNA ribonucleic acid

SDE stochastic differential equation

SynTReN SYNthetic Transcriptional REgulatory Networks

TF transcription factor

TG target gene

List of Figures

2.1 SynTReN activations 6

3.1 Clustering gene expression data from the *E. coli* M^{3D} dataset 9
3.2 Intensity and gene ranges histograms 13
3.3 Histograms of TF-TG and TG-TG correlations 14
3.4 Example in which $\gamma(\mathbf{D}^X, \mathbf{D}^Z)$ is not correlated with $\gamma(\mathbf{T}^X, \mathbf{T}^Z)$ . . . 18
3.5 Intuition of the γ-based quality checks 18
3.6 Generative Adversarial Network framework 21
3.7 Bayesian network of traditional GANs 22

4.1 RegulonDB core gene regulatory network of *E. coli* 30
4.2 Architecture of the GAN 35
4.3 Background distributions for different settings of ξ 40
4.4 Distribution of intensities (all genes) 41
4.5 Range of gene expressions (all genes) 42
4.6 Background distribution of pairwise gene correlation coefficients (all genes) 42
4.7 Histogram of TF-TG interactions (all genes) 43
4.8 Histogram of TG-TG interactions (all genes) 43
4.9 Histograms of the TF activity (all genes) 44
4.10 Overview of gGAN gene distributions 45
4.11 Distribution of intensities (CRP hierarchy) 48
4.12 Range of gene expressions (CRP hierarchy) 49
4.13 Background distribution of the correlation coefficients between all pair of genes (CRP hierarchy) 49
4.14 Histogram of TF-TG interactions (CRP hierarchy) 50
4.15 Histogram of TG-TG interactions (CRP hierarchy) 50
4.16 Histograms of the TF activity (CRP hierarchy) 51

A.1 Bayesian network of our GAN 61

A.2 Evolution of $\gamma(\mathbf{D}^X, \mathbf{D}^Z)$ for different settings of ξ 62
A.3 Evolution of $\gamma(\mathbf{D}^Z, \mathbf{T}^Z)$ for different settings of ξ 62
A.4 Evolution of $\gamma(\mathbf{T}^X, \mathbf{T}^Z)$ for different settings of ξ 63
A.5 Evolution of $\psi(\mathbf{D}^X, \mathbf{D}^Z)$ for different settings of ξ 63
A.6 Evolution of $\phi(\mathbf{D}^X, \mathbf{D}^Z)$ for different settings of ξ 64

B.1 Distribution of intensities for random simulator (CRP hierarchy) . . . 65
B.2 Range of gene expressions for random simulator (CRP hierarchy). . . 66
B.3 Background distribution of correlation coefficients between all pairs of genes for random simulator (CRP hierarchy) 66
B.4 Histogram of TF-TG interactions for random simulator (CRP hierarchy) 67
B.5 Histogram of TG-TG interactions for random simulator (CRP hierarchy) 67
B.6 Histogram of the TF activity for random simulator (CRP hierarchy). 68

List of Tables

4.1 Quantitative assessment of the generated data (all genes) 44
4.2 Quantitative assessment of the generated data (CRP hierarchy) . . . 51

List of Algorithms

1 Agglomerative hierarchical clustering 11
2 Training a GAN 27
3 Selecting a hierarchy of genes 31
4 Training our GAN 37
5 Matching the individual distributions of a set of genes 70

Chapter 1

Introduction

1.1 Motivation

Over the last two and a half decades, the emergence of technologies such as spotted microarrays (Schena et al., 1995) and RNA-seq (Mortazavi et al., 2008) has enabled to accurately measure the expression levels of thousands of genes. In parallel, the interest of the research community for analyzing these profiles has increased over years, and several bioinformatics tools have been developed with the aim of finding recurring patterns in expression data. In particular, an important effort has been put into reverse engineering gene regulatory networks (GRNs) for a better understanding of the complex interactions that occur among genes. These networks are usually represented as a directed graph, in which edges indicate an enhancing or inhibitory regulatory effect of the rate of gene transcription from a transcription factor (TF) to a target gene (TG). As a result of these studies, several network inference methods (Yu et al., 2004; Margolin et al., 2006; Yip et al., 2010; among others) have been developed.

However, relatively little effort has been devoted to benchmarking the performance of the bioinformatics methods that infer the structure of GRNs, along with their dynamical properties, from high-throughput experimental data. Evaluating these algorithms is usually a challenging task, because we often lack well-understood biological networks to use as gold standards. When such gold standards are not available, assessing the performance of these network inference algorithms requires repeatedly testing them on large, high-quality datasets with many experimental conditions derived from well-characterized networks (Van den Bulcke et al., 2006). Unfortunately, datasets of an appropriate size are usually unavailable. Under this scenario, there is a clear need for simulators of well-characterized expression datasets.

1.2 Objectives

Generating realistic expression data is a challenging task for two reasons. First, the number of possible gene regulatory networks grows exponentially with the number of genes. This poses a major challenge in order to model the complex regulatory interactions that occur among genes, as these networks are usually unknown. Second, it is difficult to determine to which extent the expression data generated by a simulator is realistic. Unlike in other domains such as image generation, in which one can empirically assess whether an image is realistic, evaluating the realism of a gene expression dataset is difficult because we often lack reliable gold standards and we do not have an intuitive understanding of high-dimensional expression data.

The goal of this thesis is to address these two challenges. First, we aim to study and propose mechanisms to faithfully assess the quality of the simulated expression data. More specifically, we aim to check whether the synthetic expression data matches several relevant statistical properties of the real data. Second, our main goal is to implement an adversarial simulator based on a generative adversarial network (GAN; Goodfellow et al., 2014). This framework describes a method for estimating a generative model by playing a two-player game, in which the first player learns to generate samples from a particular distribution, and the second tries to discriminate them from the samples coming from the true data distribution. This novel deep learning framework has shown promising results for tasks such as image or audio generation, and to our best knowledge GANs have not yet been applied to build a simulator of gene expression data.

1.3 Contribution

As a result of studying the problem of generating realistic gene expression data, we present two main contributions.

First, we develop our own novel quality assessment scores to measure whether several statistical properties of real expression data are preserved in synthetically generated datasets. These metrics aim to cover a wide range of properties, and we leverage them a posteriori to optimize and evaluate the performance of our model.

Second, we design a simulator of *E. coli* gene expression data (∼4,400 genes) based on a generative adversarial network (Goodfellow et al., 2014). To our best knowledge, GANs have not yet been applied for this purpose. We show that our approach outperforms existing simulators (SynTReN; Van den Bulcke et al., 2006,

and GeneNetWeaver; Schaffter et al., 2011) by a large margin in terms of the realism of the generated data, and that our model passes the Turing test for gene expression data proposed by Maier et al. (2013).

1.4 Outline

This document is structured into multiple chapters that provide a guided itinerary to the problem of generating realistic gene expression data.

First, chapter 2 presents two existing simulators of expression data: SynTReN (section 2.1) and GeneNetWeaver (GNW; section 2.2). These simulators use ordinary and stochastic differential equations to model gene interactions, and they both have been widely used for the purpose of benchmarking network inference algorithms.

Second, chapter 3 describes the methods that we propose to solve the problem of generating realistic gene expression data. In section 3.1, we review the work by Maier et al. (2013) to characterize several gene expression properties, and we define our own novel evaluation metrics. On the other hand, section 3.2 describes the framework of generative adversarial networks, and reviews several theoretical properties of GANs as well as their relationship with maximum likelihood estimation.

Third, in chapter 4 we present the experimental details of our approach along with the obtained results. Section 4.1 describes the *E. coli* M^{3D} dataset and the RegulonDB database of gene regulatory interactions. Subsequently, we discuss the strategy that we follow to prepare the data and define the train-test split. In section 4.2 we provide the details about the architecture of our adversarial model. Next, section 4.3 describes the training algorithm as well as how we tune the hyperparameters. We enumerate our list of software and hardware references in section 4.4. Finally, we describe and discuss the results of our approach in section 4.5, where we also provide a comparison with existing methods.

Lastly, chapter 5 concludes the thesis. Concretely, section 5.1 summarizes the achievements of this thesis, and section 5.2 points out some experiments and ideas that are worth investigating in the future.

Chapter 2

Related work

2.1 SynTReN

SynTReN (Van den Bulcke et al., 2006) is a simulator that creates synthetic transcriptional regulatory networks and produces artificial gene expression data that approximates experimental data. These synthetic networks are characterized by a topology of genes and a model based on Michaelis-Menten and Hill kinetics for each gene interaction. The reader may refer to Chapter 1 of the MIT Systems Biology course (Oudenaarden, 2004) for a detailed explanation of these equations.

2.1.1 Network topology

To generate the network topology, SynTReN uses the known topology of the organism's biological network as the base graph. Then, the paper vaguely describes two different algorithms that select a subgraph from the initial graph. The first algorithm, *neighbor addition*, selects a starting node arbitrarily, and iteratively adds randomly selected nodes. The algorithm only retains selected nodes that are connected to the current graph. The second algorithm, *cluster addition*, starts by selecting a node and all of its neighbours as the initial graph. At each iteration, a node is selected randomly, and then it is added to the graph along with all of its neighbors. Likewise, only nodes that are connected to the current graph are retained.

In addition to the selected subnetworks, SynTReN adds a network of background genes to the current topology in order to imitate pathways that are not influenced by the simulated experimental conditions. SynTReN assumes that these genes are not elicited by the simulated experimental conditions, although their expression values influence the dynamics of the system. The choice of the initial network topology,

the number of nodes in the network, and the number of background nodes are user-definable hyperparameters.

2.1.2 Transition functions

A transition function defines a model that describes how the expression of a gene depends on the expression of a transcription factor. SynTReN uses a linear system of ODEs based on Michaelis-Menten and Hill kinetic equations in order to model these gene regulatory interactions. The biological noise is modelled based on a lognormal distribution superposed on the kinetic equations. The level of noise is a hyperparameter.

SynTReN models both inductive and inhibitory regulatory interactions. In addition, it attempts to model complex interactions in which several regulators exhibit different types of interactions (including synergistic, cooperative and competitive interactions) on a common target gene. A different kinetic equation is selected for each of these interactions, depending on the number of activators and repressors as well as on the hyperparameter that indicates the fraction of complex interactions.

The parameters of the resulting Michaelis-Menten and Hill kinetic equations are chosen from a distribution that allows a comprehensive range of interaction kinetics. Figure 2.1 shows some examples of activation interactions. The paper does not give low-level details about how these equations are used in SynTReN, nor how the resulting ordinary differential equations are used to generate synthetic expression data. It is also not clear from the paper whether the data produced by SynTReN, which is normalized to be between 0 and 1, is the raw expression data or the log expression data.

2.1.3 Generating data

In order to sample gene expression data from the model, SynTReN assumes that the network is triggered by some external conditions. To simulate this behavior, the expression values of a subset of genes without regulatory inputs are set to a different value in each experiment. Because the expression value of certain input genes may be correlated in realistic data, SynTReN provides hyperparameters to control the number of correlated inputs and their correlation strength.

The expression values of the remaining genes are iteratively computed using their transition functions, starting from the input genes. When loops are present within the network, the inputs of a transition function from a certain gene might be unknown. In

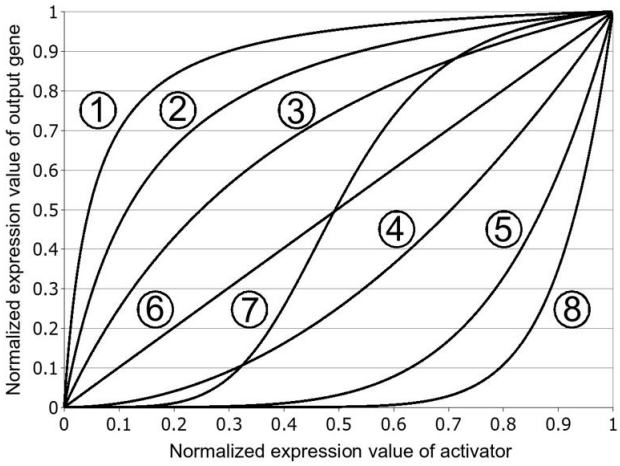


Figure 2.1: Types of interactions for one activator modelled by SynTReN. The normalized expression value of a target gene (ordinate axis) depends on the normalized expression value of its activator (abscissa axis) and two kinetic parameters that control the shape of the activation function. Source: Van den Bulcke et al. (2006)

this case, the expression values of such inputs are arbitrarily set in each experiment. The number of experiments and samples for each condition is user-definable.

2.2 GeneNetWeaver

GeneNetWeaver (GNW; Schaffter et al., 2011) generates dynamical models of GRNs with the purpose of constructing appropriate benchmarks for algorithms that reverse engineer GRNs from experimental expression data. To generate network topologies, GNW extracts modules from well-studied biological networks such as the one of *E. coli* (Gama-Castro et al., 2016). Then, dynamical models are used to model gene interactions both at transcription and translation levels.

2.2.1 Network topology

The network topology is generated by extracting modules from well-studied biological networks. Module extraction starts by selecting a random gene from the source network, and a subnetwork is expanded by iteratively adding neighboring genes until the desired module size is reached. The neighbor that leads to the highest modularity is chosen at each iteration. The modularity is defined as the difference between the number of edges within the subnetwork and the expected number of edges in a randomized graph of the same size (Marbach et al., 2009). These extracted subnetworks aim to find functionally related genes in the source network.

2.2.2 Dynamical model

The topologies resulting from the module extraction are endowed with dynamical models that describe how regulatory interactions occur amongst genes both at transcription and translation levels. In particular, for each gene i of a network, the rates F_i^{RNA} and F_i^{Prot} of change of mRNA and protein concentration are defined as (Schaffter et al., 2011):

$$F_i^{RNA}(\mathbf{x}, \mathbf{y}) = \frac{\mathrm{d}x_i}{\mathrm{d}t} = m_i \cdot f_i(\mathbf{y}) - \lambda_i^{RNA} \cdot x_i \tag{2.1}$$

$$F_i^{Prot}(\mathbf{x}, \mathbf{y}) = \frac{\mathrm{d}y_i}{\mathrm{d}t} = r_i \cdot x_i - \lambda_i^{Prot} \cdot y_i \tag{2.2}$$

where m_i is the maximum transcription rate, r_i the translation rate, λ_i^{RNA} and λ_i^{Prot} the mRNA and protein degradation rates, and $\mathbf{x}$ and $\mathbf{y}$ the vectors of mRNA and protein concentration levels (Schaffter et al., 2011), respectively. The activation function $f_i(\cdot) \in [0, 1]$ calculates the relative activation of a gene given the concentration levels of its associated TFs.

Random fluctuations that influence the mRNA and protein concentration levels are modelled using independent Gaussian white-noise processes (Gillespie, 2000). In addition, GNW models experimental noise using Gaussian and log-normal noise models.

Chapter 3

Methods

3.1 Evaluating artificial gene expression data

Assessing to which extent simulators are able to generate realistic datasets is a difficult task, since we often lack reliable gold standards and we do not have an intuitive understanding of high-dimensional expression data. Designing a metric that perfectly quantifies the degree of realism of simulated data would necessarily require having a perfect model of the gene regulatory interactions along with other layers of regulation (such as protein-protein interactions or cellular states). As a consequence, other methods are required.

In this section we examine several mechanisms to evaluate the quality of synthetic data. First, we introduce a preliminary method (hierarchical clustering) that we leverage a posteriori. Then, we review the measures proposed by Maier et al. (2013) to assess several key properties of synthetic data. Lastly, we develop our own novel quality assessment measures, which evaluate some statistical properties more accurately and capture idiosyncrasies not covered by Maier et al. (2013).

3.1.1 Clustering gene expression patterns

Clustering is a natural approach for grouping genes that show similar expression patterns. In particular, hierarchical clustering allows to visualize relationships between closely related groups of genes by forming a gene dendrogram. The branch lengths of the resulting tree reflect the degree of similarity between pairs of gene clusters according to a predefined similarity measure. The dendrogram is often used to rearrange genes in the original expression matrix in order to enforce related genes to be adjacent to each other. Figure 3.1 shows an example of hierarchical clustering on the *E. coli* M^{3D} dataset.

These tree-like structures are familiar to biologists, since they are traditionally employed in phylogenetic and evolutionary sequence analyses. Even though they do not necessarily carry evolutionary information about the clustered genes, they are useful to represent varying degrees of similarity and they require few assumptions about the nature of the data (Eisen et al., 1998). In this study we use hierarchical clustering to test whether genes that tend to cluster together in real data also show this behavior in synthetically generated datasets.

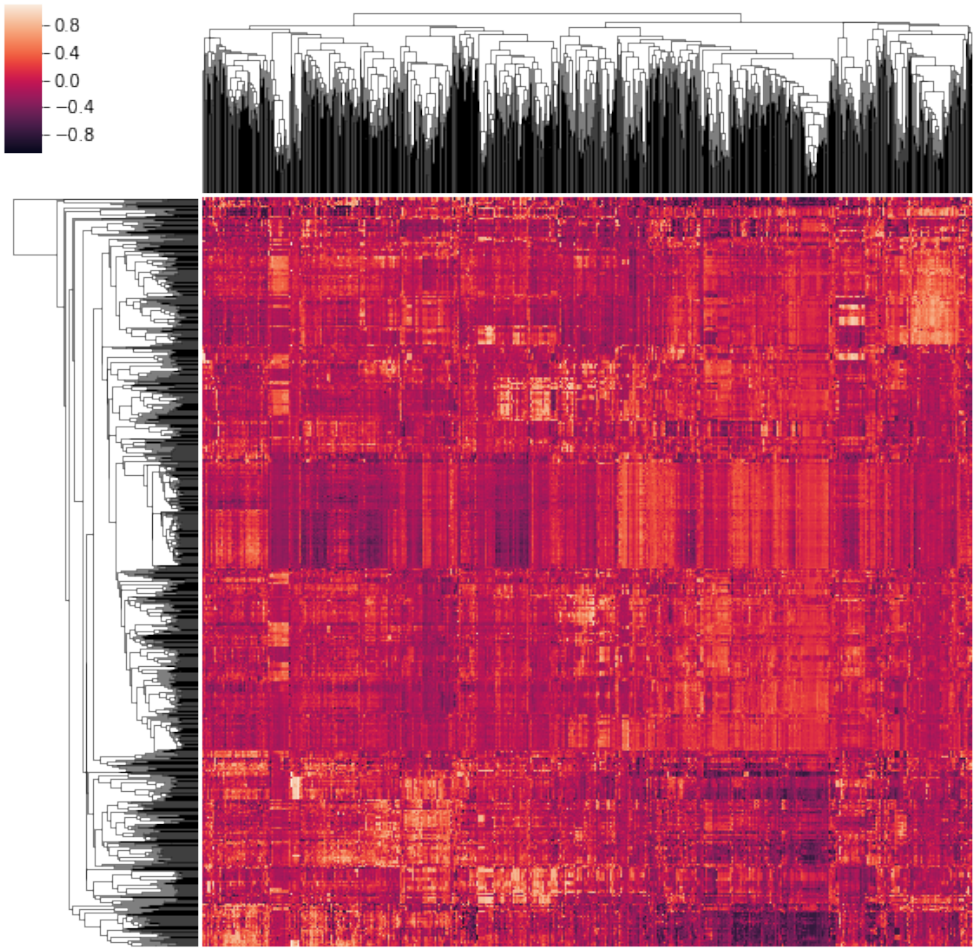


Figure 3.1: Clustering gene expression data from the *E. coli* M^{3D} dataset (rows: samples, columns: genes). We use normalized expression values for the *E. coli* CRP hierarchy of genes (1076 genes; see section 4.1.3.1). We rearrange the expression matrix to enforce related genes to be adjacent to each other. Samples are also reordered according to the resulting dendrogram of samples.

3.1.1.1 Agglomerative hierarchical clustering

Hierarchical clustering is an algorithm that builds a hierarchy of clusters. Concretely, agglomerative hierarchical clustering (Algorithm 1) starts by assigning a cluster to each gene and iteratively merges the closest clusters until only one cluster is remaining (D'haeseleer, 2005). In order to decide which clusters get merged, we consider several distance metrics and linkage functions that define the distance between the expression measurements of two genes and the distance between two clusters, respectively.

To measure the distance between two m-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ representing the gene expression measurements, Eisen et al. (1998) suggest the Pearson correlation coefficient:

$$d(\mathbf{x}, \mathbf{y}) = 1 - \frac{1}{m} \sum_{i=1}^{m} \left(\frac{\mathbf{x}_i - \mu_{\mathbf{x}}}{\phi_{\mathbf{x}}} \right) \left(\frac{\mathbf{y}_i - \mu_{\mathbf{y}}}{\phi_{\mathbf{y}}} \right) \tag{3.1}$$

where:

$$\mu_{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{g}_i \qquad \phi_{\mathbf{g}} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (\mathbf{g}_i - \mu_{\mathbf{g}})^2} \tag{3.2}$$

They argue that this coefficient captures the gene similarity without taking into account the magnitude of the measured vectors $\mathbf{x}$ and $\mathbf{y}$, and thus this measure conforms well with the biological notion of two coexpressed genes. On the other hand, D'haeseleer (2005) recommends the Euclidean distance $d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_i (\mathbf{x}_i - \mathbf{y}_i)^2}$ for log-ratio data, and the Pearson correlation for absolute-valued data.

Several options can be considered for the linkage function. Let $\mathbf{u}$ and $\mathbf{v}$ be two different clusters. The inter-cluster distance may be defined as (among other options):

- Single linkage. Shortest distance between any member $\mathbf{u}_i$ in cluster $\mathbf{u}$ to any member $\mathbf{v}_j$ in cluster $\mathbf{v}$:

$$D(\mathbf{u}, \mathbf{v}) = \min_{i,j} d(\mathbf{u}_i, \mathbf{v}_j) \tag{3.3}$$

- Complete linkage. Largest distance between any member $\mathbf{u}_i$ in cluster $\mathbf{u}$ to any member $\mathbf{v}_j$ in cluster $\mathbf{v}$:

$$D(\mathbf{u}, \mathbf{v}) = \max_{i,j} d(\mathbf{u}_i, \mathbf{v}_j) \tag{3.4}$$

- Average linkage. Average distance between members of cluster $\mathbf{u}$ and members of cluster $\mathbf{v}$:

$$D(\mathbf{u}, \mathbf{v}) = \frac{1}{|\mathbf{u}| \cdot |\mathbf{v}|} \sum_{i,j} d(\mathbf{u}_i, \mathbf{v}_j) \tag{3.5}$$

- Centroid linkage. Distance between the expression profiles (centroids) $\mu_{\mathbf{u}}$ and $\mu_{\mathbf{v}}$ of clusters $\mathbf{u}$ and $\mathbf{v}$:

$$D(\mathbf{u}, \mathbf{v}) = d(\mu_{\mathbf{u}}, \mu_{\mathbf{v}}) \tag{3.6}$$

D'haeseleer (2005) emphasizes that single linkage generally performs poorly on real datasets and recommends using complete linkage over average linkage. Conversely, the preferred choice of Eisen et al. (1998) is the centroid linkage function.

Algorithm 1: Agglomerative hierarchical clustering. Returns a tree of clusters

Data: We are given a $m \times n$ matrix $\mathbf{X}$ containing the expressions of n genes over m observations, a distance metric d and a linkage function D.

for *i in 1 .. n* **do**
    Initialize cluster:
        $\mathrm{c}_i = \{i\}$
C = $\{c_1, ..., c_n\}$
for *k in 1 .. n-1* **do**
    Find two closest clusters:
        $\mathrm{c}_i, c_j = \underset{c_i \in C, c_j \in C}{\operatorname{argmin}} D(c_i, c_j; d, \mathbf{X})$
    Remove selected clusters from C:
        C = C – $\{\mathrm{c}_i, c_j\}$
    Form new cluster and add it to C:
        $\mathrm{c}_{n+k} = c_i \cup c_j$
        C = C $\cup \{c_{n+k}\}$
return C

3.1.1.2 Silhouette score

Maier et al. (2013) use Silhouette coefficients to measure the separation and tightness of the gene clusters resulting from hierarchical clustering. The Silhouette score (Rousseeuw, 1987) evaluates the quality of the clustering by measuring the separation and the tightness of the clusters. Let $\mathcal{D}(i, \mathcal{C})$ be the average dissimilarity of datapoint i to all objects in cluster $\mathcal{C}$. The Silhouette score is computed based on the average intra-cluster distance and the average nearest-cluster distance for each datapoint i:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{3.7}$$

where $a(i) = \mathcal{D}(i, \mathcal{A})$ with $\mathcal{A}$ being the cluster associated to datapoint i, and $b(i)$ is the minimum $\mathcal{D}(i, \mathcal{C})$ for all $\mathcal{C} \neq \mathcal{A}$. Because $a(i)$ and $b(i)$ are dissimilarities (i.e. $0 \leq a(i), b(i) \leq 2$), the Silhouette coefficient $s(i)$ is bounded as $-1 \leq s(i) \leq 1$. Note

also that this formula assumes that the number of clusters is more than one and that $s(i)$ is set to 0 when $\mathcal{A}$ has only one element.

3.1.2 Maier's histograms of statistical properties of expression data

Maier et al. (2013) present a method that evaluates simulated expression data by comparing several statistical properties of artificial datasets with those of real datasets. Concretely, the study proposes using histograms to visualize the distributions of these specific properties, with the purpose of checking whether they are preserved in synthetically generated datasets. Overlap scores are used to quantify the discrepancies between pairs of histograms. The overlap score $O(\mathbf{a}, \mathbf{b})$ between two histograms $\mathbf{a}$ and $\mathbf{b}$ is computed by dividing the sum of minimum densities over n bins by the sum of maximum densities over n bins:

$$O(\mathbf{a}, \mathbf{b}) = \frac{\sum_{i=1}^{n} \min(\mathbf{a}_i, \mathbf{b}_i)}{\sum_{i=1}^{n} \max(\mathbf{a}_i, \mathbf{b}_i)} \tag{3.8}$$

This score provides a measure that allows to quantify the gap between real and simulated datasets with regard to some chosen statistical properties. Next, we describe the properties considered by Maier et al. (2013).

3.1.2.1 Replicate noise

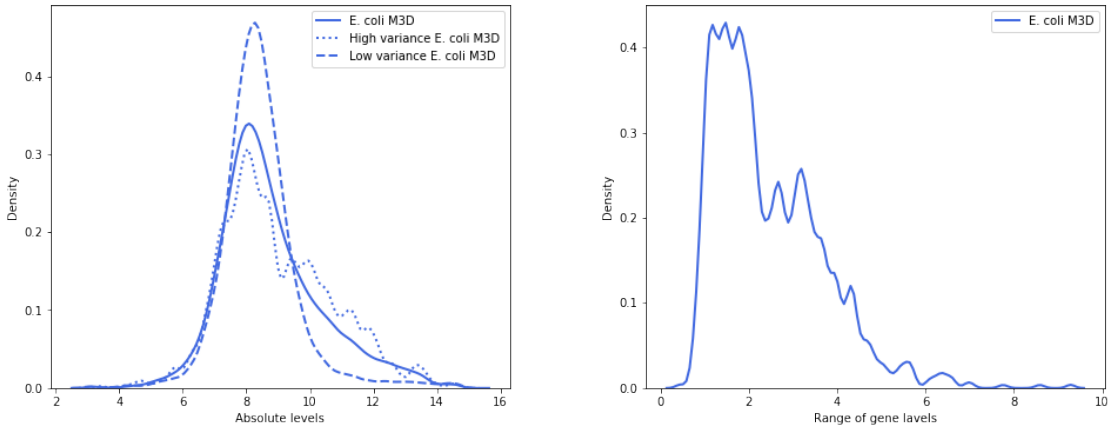
The Pearson's correlation coefficient ρ is computed between all pairs of replicate samples, yielding $k(k-1)/2$ values for each set of k replicates. A histogram is then formed by binning all the coefficients ρ in multiple intervals. The motivation behind this histogram is to capture the likelihood of outliers and determine the reproducibility of measurements.

3.1.2.2 Intensity

This histogram combines all the intensity measurements from a dataset, and it is useful in order to evaluate the effectiveness of data normalization. Usually, the distribution of intensity values is approximately normally distributed in real data. Figure 3.2 (a) shows the intensity histogram for the *E. coli* M^{3D} dataset (Faith et al., 2008).

3.1.2.3 Range of gene expression

This property measures the overall difference between the minimum expression (5% intensity quantile) and the maximum expression (95% intensity quantile). These



(a) Intensity histogram

(b) Range of gene expression

Figure 3.2: Histograms of statistical properties extracted from the *E. coli* M^{3D} dataset (Faith et al., 2008). (a) Histogram of intensities. The dashed and dotted lines show the histogram of the microarrays in the 5% and 95% quantiles of variance. (b) Intensity ranges for individual genes.

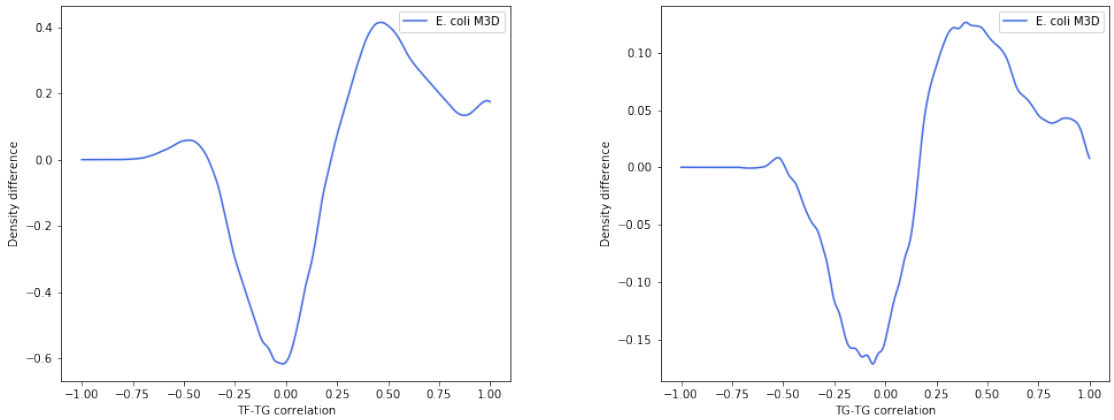
values are computed for each gene, and then a histogram of gene ranges is constructed. Since the gene range of the artificially generated data is rather arbitrary, Maier et al. (2013) suggest multiplying the artificial ranges by a factor to make the median of the real and artificial gene range histograms match. Figure 3.2 (b) shows the gene ranges histogram for the *E. coli* M^{3D} dataset (Faith et al., 2008).

3.1.2.4 Silhouette coefficient

Agglomerative hierarchical clustering (Eisen et al., 1998) is used to cluster gene and samples separately. Then, Silhouette coefficients (Rousseeuw, 1987) are computed at each node in the resulting dendrograms, providing a measure of the separation and tightness of the clusters. Two histograms are formed by binning these coefficients separately for the gene and the sample clusters.

3.1.2.5 Correlation between TFs and TGs

The Pearson's correlation coefficient ρ is computed between the expressions of each TF and the expressions of its TGs, according to the gene regulatory interactions (GRI) gold standards. The resulting distribution is compared with a background distribution of the correlation coefficients between all pairs of genes, and a difference histogram that shows the enrichment of TF-TG interactions is generated. Figure 3.3 (a) shows the TF-TG histogram for the *E. coli* M^{3D} dataset (Faith et al., 2008).



(a) Histogram of TF-TG correlations (b) Histogram of TG-TG correlations

Figure 3.3: Histograms of statistical properties extracted from a subset of genes in the *E. coli* M^{3D} dataset (Faith et al., 2008), using the regulatory interactions from RegulonDB (Gama-Castro et al., 2016). Positive and negative value indicates an increased or decreased chance to detect correlated TF-TG and TG-TG relationships for the given interval, respectively. (a) Histogram of TF-TG interactions. (b) Histogram of TG-TG interactions.

3.1.2.6 Correlation between TGs regulated by the same TF

The Pearson's correlation coefficient ρ is computed between all pairs of TGs regulated by the same TF, according to the GRI gold standards. The resulting distribution is compared with a background distribution of the correlation coefficients between all pairs of genes, and a difference histogram that shows the enrichment of TG-TG relationships is generated. This histogram provides an overview of the degree of co-regulation induced by common TFs. Figure 3.3 (b) shows the TG-TG histogram for the *E. coli* M^{3D} dataset (Faith et al., 2008).

3.1.2.7 Activity of TFs

This property measures the activity of TFs based on the expression of the TGs that they regulate, according to the GRI gold standards. First, the expression of each gene is normalized by subtracting the mean and dividing by the standard deviation of that gene across all samples. Then, a *Mann–Whitney U test* (Mann and Whitney, 1947) is employed to test whether the TGs regulated by a certain TF exhibit rank differences compared to other non-target genes.

This non-parametric statistical test is used to determine whether two sets of independent samples are likely to derive from the same population (null hypothesis).

Unlike *t-test*, *Mann–Whitney U test* does not require the assumption that the populations being compared follow a normal distribution. Let m_1 and m_2 be the sizes of two groups (i.e. TGs and non-TGs for a given TF in a particular chip), and let r_1 and r_2 be the sum of overall ranks from these two groups combined (in case of a tie, the ranks that would have been assigned to all the tied values are averaged). The statistic U is computed as:

$$U = min(U_1, U_2) \tag{3.9}$$

where:

$$U_x = m_1 m_2 + \frac{m_x(m_x + 1)}{2} - r_x \tag{3.10}$$

For large group sizes, the distribution of $U - \frac{1}{2}(m_1 m_2 + 1)$ differs only a negligible amount from the normal distribution (Mann and Whitney, 1947), and thus we can compute p-values for each chip by performing a Z-test. As a result of performing this test for each chip, we obtain several p-values for each TF. These p-values are corrected via the Benjamini-Hochberg's procedure (Benjamini and Hochberg, 1995) to account for multiple testing. Finally, an histogram is formed by computing the fraction of chips, for each TF, in which the null hypothesis is rejected.

3.1.3 Critique to Maier's histograms

Maier et al. (2013) provide a way of characterizing several statistical properties of gene expression datasets. While this method allows to visually compare several distributions, the overlap score is sensitive to symmetries and thus it might be an inaccurate discrepancy metric. For example, suppose that the following vectors measure the fraction of chips in which each TF is found active for two distinct datasets:

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0.5 \\ 0.2 \\ 0.5 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 0.5 \\ 0 \\ 0.5 \\ 0.2 \end{bmatrix} \tag{3.11}$$

Note that the histograms for these two vectors are identical and thus the overlap score is 1. However, the fraction of chips is substantially different for each TF, and therefore the overlap score fails to quantify the low-level discrepancies. Other histograms such as the one constructed with Silhouette scores are also very sensitive to these symmetries, since completely unrelated clusters may have the same Silhouette score. As a consequence, even though Maier's histograms are informative and easy to interpret, we believe it is sensible to design finer-grained metrics to accurately compare these statistical properties.

3.1.4 Our quality assessment metrics

The measures given in section 3.1.2 have already been used in the study by Maier et al. (2013). In this section we develop our own novel quality assessment measures. The main motivations behind these measures are: to evaluate some statistical properties more accurately; and to capture additional properties not covered by Maier et al. (2013).

Ideally, the artificial data should preserve the statistical properties that describe how genes interact with each other. In particular, a good evaluation metric should be able to assess whether the real relationships between groups of genes are also maintained in the generated artificial data. Here we introduce several quality measures based on the correlation coefficient between gene distance matrices.

Let $\mathbf{B}$ be a $n \times n$ symmetric matrix holding the pairwise distances between all genes. In order to measure how faithfully this matrix preserves the pairwise distances with respect to another $n \times n$ distance matrix $\mathbf{A}$, we define the following coefficient:

$$\gamma(\mathbf{A}, \mathbf{B}) = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Big(\frac{\mathbf{A}_{i,j} - \mu_{\mathbf{A}}}{\phi_{\mathbf{A}}} \Big) \Big(\frac{\mathbf{B}_{i,j} - \mu_{\mathbf{B}}}{\phi_{\mathbf{B}}} \Big) \tag{3.12}$$

where:

$$\mu_{\mathbf{J}} = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{J}_{i,j} \tag{3.13}$$

$$\phi_{\mathbf{J}} = \sqrt{\frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (\mathbf{J}_{i,j} - \mu_{\mathbf{J}})^2} \tag{3.14}$$

Intuitively, this coefficient computes the Pearson's correlation between the elements in the upper-diagonal of matrices $\mathbf{A}$ and $\mathbf{B}$. Therefore, $-1 \leq \gamma(\mathbf{A}, \mathbf{B}) \leq 1$, and the pairwise distances of $\mathbf{A}$ are faithfully preserved by $\mathbf{B}$ when $\gamma(\mathbf{A}, \mathbf{B})$ is close to 1. Now, let $\mathbf{X}$ and $\mathbf{Z}$ be two matrices containing m_1 and m_2 n-dimensional observations that are sampled independently from the real p_{real} and synthetic p_g distributions, respectively. Next, we consider several quality measures based on correlation coefficients.

3.1.4.1 Distance between *real* and *artificial* distance matrices

For a given distance function d, we define the distance matrices $\mathbf{D}^X$ and $\mathbf{D}^Z$ as:

$$\mathbf{D}^X_{i,j} = d(\mathbf{X}_{:,i}, \mathbf{X}_{:,j}) \quad \mathbf{D}^Z_{i,j} = d(\mathbf{Z}_{:,i}, \mathbf{Z}_{:,j}) \tag{3.15}$$

Then, the coefficient $\gamma(\mathbf{D}^X, \mathbf{D}^Z)$ measures whether the pairwise distances between genes from the real data are correlated with those from the synthetic data. The coefficient $\gamma(\mathbf{D}^X, \mathbf{D}^Z)$ is close to 1 when the distribution p_g approximates p_{real} well.

3.1.4.2 Distance between *real* and *artificial* dendrograms

Let $C : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}$ be a function that performs agglomerative hierachical clustering according to a given linkage function, taking a $n \times n$ distance matrix as input and returning the $n \times n$ distance matrix of the resulting dendrogram. We define the *real* and *artificial* dendrogrammatic distance matrices $\mathbf{T}^X$ and $\mathbf{T}^Z$ as:

$$\mathbf{T}^X = C(\mathbf{D}^X) \quad \mathbf{T}^Z = C(\mathbf{D}^Z) \tag{3.16}$$

The coefficient $\gamma(\mathbf{T}^X, \mathbf{T}^Z)$ measures the structural similarity between the dendrograms, giving a score close to 1 when the *real* and *artificial* dendrograms have a similar structure. Consequently, this metric encourages the distribution p_g to preserve the relationships among groups of genes that are found in p_{real}.

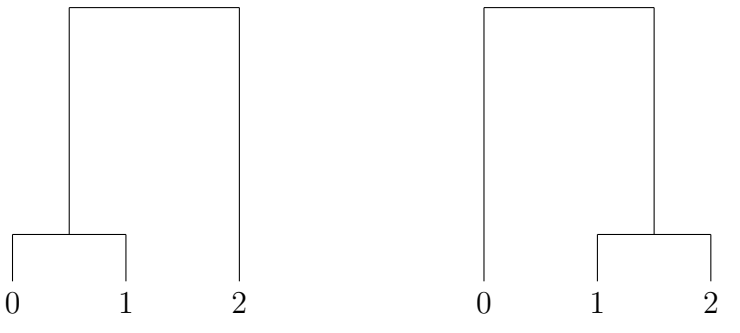
Note also that $\gamma(\mathbf{T}^X, \mathbf{T}^Z)$ does not necessarily correlate well with $\gamma(\mathbf{D}^X, \mathbf{D}^Z)$. Consider for example the distance matrices:

$$\mathbf{D}^X = \begin{bmatrix} 0 & 2 & 10 \\ 2 & 0 & 3 \\ 10 & 3 & 0 \end{bmatrix} \quad \mathbf{D}^Z = \begin{bmatrix} 0 & 3 & 10 \\ 3 & 0 & 2 \\ 10 & 2 & 0 \end{bmatrix} \tag{3.17}$$

The dendrogrammatic distance matrices $\mathbf{T}^X$ and $\mathbf{T}^Z$ resulting from agglomerative hierarchical clustering with complete linkage are:

$$\mathbf{T}^X = \begin{bmatrix} 0 & 2 & 10 \\ 2 & 0 & 10 \\ 10 & 10 & 0 \end{bmatrix} \quad \mathbf{T}^Z = \begin{bmatrix} 0 & 10 & 10 \\ 10 & 0 & 2 \\ 10 & 2 & 0 \end{bmatrix} \tag{3.18}$$

And the coefficients $\gamma(\mathbf{D}^X, \mathbf{D}^Z) = 0.97$ and $\gamma(\mathbf{T}^X, \mathbf{T}^Z) = -0.5$ are substantially different. Figure 3.4 illustrates these dendrograms.



(a) Dendrogram for $\mathbf{T}^X$ (b) Dendrogram for $\mathbf{T}^Z$

Figure 3.4: Dendrograms resulting from agglomerative hierachical clustering with complete linkage for the distance matrices $\mathbf{D}^X$ and $\mathbf{D}^Z$ defined in equation 3.17. Note that $\gamma(\mathbf{D}^X, \mathbf{D}^Z) = 0.97$, but $\gamma(\mathbf{T}^X, \mathbf{T}^Z) = -0.5$ because the dendrograms' structures are substantially different.

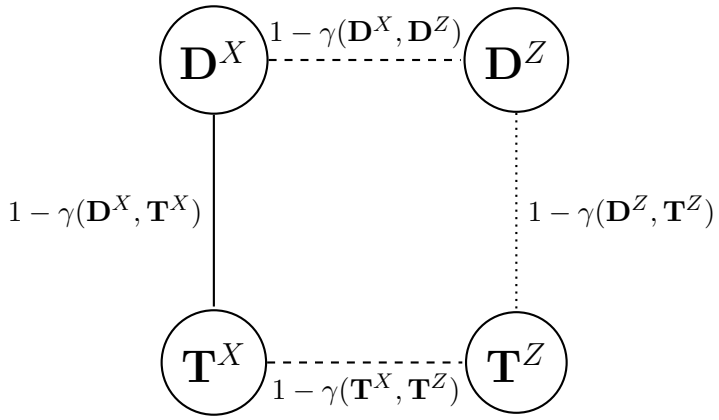


Figure 3.5: Intuition of the γ-based quality checks. When the synthetic data resembles the real data, the distance matrices $\mathbf{D}^X$ and $\mathbf{D}^Z$ are similar and the distance $1 - \gamma(\mathbf{D}^X, \mathbf{D}^Z)$ is close to 0. Likewise, $1 - \gamma(\mathbf{T}^X, \mathbf{T}^Z)$ is close to 0 when the synthetic and real dendrogram structures are similar. Even though we do not have control over $1 - \gamma(\mathbf{D}^X, \mathbf{T}^X)$, the distance $1 - \gamma(\mathbf{D}^X, \mathbf{T}^Z)$ should not deviate a lot from it when the synthetic data is realistic. Ideally, for a perfect distribution p_g, the nodes $\mathbf{D}^X$ and $\mathbf{D}^Z$; $\mathbf{T}^X$ and $\mathbf{T}^Z$ would overlap.

3.1.4.3 Squared difference between cophenetic correlation coefficients

The coefficient $\gamma\big(\mathbf{D}^J, C(\mathbf{D}^J)\big)$ is known as the cophenetic correlation coefficient (Sokal and Rohlf, 1962), and it measures how faithfully a dendrogram preserves the original distance matrix. Concretely, this coefficient quantifies the amount of information that is lost with respect to the original distance matrix when we perform hierarchical clustering on a gene expression dataset.

It is reasonable to expect the cophenetic coefficients from the real and synthetic datasets to be similar when the synthetic data is realistic. In other words, we expect the value $\big(\gamma\big(\mathbf{D}^X, C(\mathbf{D}^X)\big) - \gamma(\mathbf{D}^Z, C(\mathbf{D}^Z))\big)^2$ to be close to 0 when the distribution p_g approximates p_{real} well. Figure 3.5 shows the intuition behind the distance correlation coefficients.

3.1.4.4 Weighted sum of TF-TG similarity coefficients

Let $\mathcal{F}$ be the set of TF indices, $\mathcal{G}$ a function returning the set of indices of the TGs that are regulated by a given TF, and w_f a positive coefficient associated to the importance of transcription factor f. We define the following coefficient:

$$\psi(\mathbf{D}^X, \mathbf{D}^Z) = \frac{1}{\sum_{f\in\mathcal{F}} w_f} \sum_{f\in\mathcal{F}} w_f \cdot \upsilon(\mathbf{d}_f^X, \mathbf{d}_f^Z) \tag{3.19}$$

where the vector $\mathbf{d}_f^J$ is defined as:

$$\mathbf{d}_f^J = \big(\mathbf{D}_{f,g}^J : g \in \mathcal{G}(f)\big)^\top \tag{3.20}$$

and $\upsilon(\mathbf{a}, \mathbf{b})$ is the cosine similarity between vectors $\mathbf{a}$ and $\mathbf{b}$:

$$\upsilon(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^\top \mathbf{b}}{||\mathbf{a}||_2 \cdot ||\mathbf{b}||_2} \tag{3.21}$$

In this case, the cosine similarity is preferred over the Pearson's correlation coefficient ρ because for certain TFs (i.e. the ones regulating a small amount of genes) the vector $\mathbf{d}_f^J$ might potentially have a standard deviation 0, and as a result ρ would be undefined. The cosine similarity is bounded as $-1 \leq \upsilon(\mathbf{a}, \mathbf{b}) \leq 1$, and a value of 1 indicates that $\mathbf{a}$ and $\mathbf{b}$ point towards the same direction.

The coefficient $\psi(\mathbf{D}^X, \mathbf{D}^Z)$ measures whether the TF-TG dependencies in the synthetic data resemble the ones from the real data. It is bounded as $-1 \leq \psi(\mathbf{D}^X, \mathbf{D}^Z) \leq 1$, and a value close to 1 indicates that the subsets of TGs for different TFs have the same general direction in terms of the vectors defined by their gene expression values. The coefficients w_f are arbitrary, but two reasonable choices are $w_f = 1$ (all TFs are equally important) and $w_f = |\mathcal{G}(f)|$ (the importance of a TF is proportional to the number of genes that it regulates).

3.1.4.5 Weighted sum of TG-TG similarity coefficients

Similarly, we define a coefficient ϕ to measure whether the expression of TGs regulated by the same TF in synthetic data conforms well with the analog expressions in real data:

$$\phi(\mathbf{D}^X, \mathbf{D}^Z) = \frac{1}{\sum_{f\in\mathcal{F}} w_f} \sum_{f\in\mathcal{F}} w_f \sum_{g\in\mathcal{G}(f)} \upsilon(\mathbf{q}^X_{f,g}, \mathbf{q}^Z_{f,g}) \tag{3.22}$$

where $\mathbf{q}^J_{f,g}$ is the vector of distances between gene g and all the genes regulated by f (excluding g):

$$\mathbf{q}^J_{f,g} = \left(\mathbf{D}^J_{g,j} : j \in (\mathcal{G}(f) - \{g\})\right)^\top \tag{3.23}$$

3.1.4.6 Weighted correlation of TF activity chip rates

For a given dataset $\mathbf{J}$, let $c_{\mathbf{J},f}$ be the chip rate of transcription factor f for which the null hypothesis from section 3.1.2.7 is rejected (i.e. fraction of chips in which transcription factor f is found active). We define the following weighted Pearson's correlation coefficient:

$$\omega(\mathbf{X}, \mathbf{Z}) = \frac{s(\mathbf{X}, \mathbf{Z})}{\sqrt{s(\mathbf{X})s(\mathbf{Z})}} \tag{3.24}$$

where the weighted covariance is defined as:

$$s(\mathbf{X}, \mathbf{Z}) = \frac{1}{\sum_{f\in\mathcal{F}} w_f} \sum_{f\in\mathcal{F}} w_f \cdot (c_{\mathbf{X},f} - \mu_{\mathbf{X}}) \cdot (c_{\mathbf{Z},f} - \mu_{\mathbf{Z}}) \tag{3.25}$$

the weighted variances as:

$$s(\mathbf{J}) = \frac{1}{\sum_{f\in\mathcal{F}} w_f} \sum_{f\in\mathcal{F}} w_f \cdot (c_{\mathbf{J},f} - \mu_{\mathbf{J}})^2 \tag{3.26}$$

and the weighted mean as:

$$\mu_{\mathbf{J}} = \frac{1}{\sum_{f\in\mathcal{F}} w_f} \sum_{f\in\mathcal{F}} w_f \cdot c_{\mathbf{J},f} \tag{3.27}$$

The coefficient $\omega(\mathbf{X}, \mathbf{Z})$ is a metric that measures whether the transcription factors show similar activity patterns in two distinct datasets. A value close to 1 indicates that each transcription factor is found active in a similar fraction of chips. Once more, the coefficients w_f measure the importance of each TF.

3.2 Generative adversarial networks

Generative adversarial networks (GANs) are a framework for estimating generative models via an adversarial process (Goodfellow et al., 2014). They are often described as a two-player game in which both players are encouraged to improve. One player, the *generator*, creates samples that are intended to be indistinguishable from the ones coming from a given data distribution. The other player, the *discriminator*, learns to determine whether samples come from the *fake* distribution (*fake* samples) or the *real* data distribution (*real* samples). Figure 3.6 shows the basic idea of generative adversarial networks, and figure 3.7 shows the graphical model of traditional GANs.

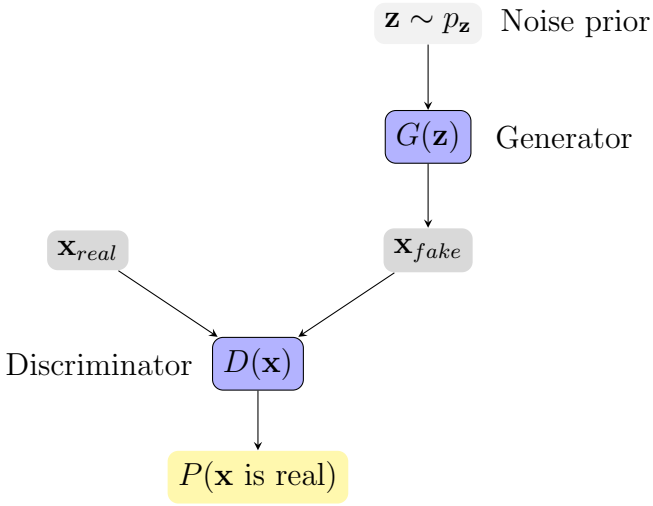


Figure 3.6: Generative Adversarial Network framework. The generator $G(\mathbf{z})$ receives a vector $\mathbf{z}$ sampled from a noise prior distribution $p_{\mathbf{z}}$, and generates a synthetic sample $\mathbf{x}_{fake}$. The discriminator $D(\mathbf{x})$ tries to distinguish *real* samples from *fake* samples, producing the probability of $\mathbf{x}$ coming from the *real* data distribution. The competition between the two players drives the game and makes both players increasingly better.

These two players are represented by functions $D(\mathbf{x})$ and $G(\mathbf{z})$, where $D(\mathbf{x})$ indicates the probability of $\mathbf{x}$ coming from the data distribution as opposed to coming from the model distribution $G(\mathbf{z})$. These functions are differentiable with respect to their parameters θ_D and θ_G, and in the GAN framework they are represented by neural networks. The discriminator attempts to minimize a cost function $J_D(\theta_D, \theta_G)$ with respect to θ_D, while the generator wants to minimize $J_G(\theta_D, \theta_G)$ with respect to θ_G. The solution to this game (θ_D, θ_G) is a local minima of J_D with respect to θ_D and a local minima of J_G with respect to θ_G, corresponding to a Nash equilibrium (Goodfellow, 2017).

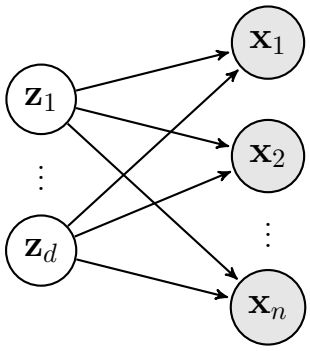


Figure 3.7: Bayesian network of traditional GANs. Every latent variable in $\mathbf{z}$ influences every observable variable in $\mathbf{x}$.

3.2.1 Cost functions

The cost function of the discriminator is defined as (Goodfellow et al., 2014):

$$J_D(\theta_D, \theta_G) = -\frac{1}{2}\langle \log D(\mathbf{x}) \rangle_{\mathbf{x}\sim p_{data}} - \frac{1}{2}\langle \log(1 - D(G(\mathbf{z}))) \rangle_{\mathbf{z}\sim p_z} \tag{3.28}$$

where $\mathbf{z}$ is a latent variable sampled from a given noise distribution p_z, and p_{data} is the data distribution. This cost function combines the cross-entropy losses for both the *real* and the *fake* data. Intuitively, the first term punishes labelling *real* data as *fake*, while the second term penalizes classifying *fake* data as *real*.

There are several options for the generator's objective function. The first option involves flipping the sign of the discriminator's cost function:

$$J_G(\theta_D, \theta_G) = -J_D(\theta_D, \theta_G) \tag{3.29}$$

Although this approach is theoretically sound, in practise it has some problems with gradient-based methods, because when the discriminator successfully rejects *fake* samples the term $\log(1 - D(G(\mathbf{z})))$ in equation 3.28 saturates and the generator's gradients become too weak (Goodfellow et al., 2014). For this reason, it is more common to define the generator's cost as:

$$J_G(\theta_D, \theta_G) = -\frac{1}{2}\langle \log D(G(\mathbf{z})) \rangle_{\mathbf{z}\sim p_z} \tag{3.30}$$

For this cost function, the generator's gradients are strong when the discriminator is not fooled by the generator's samples.

3.2.2 Optimal discriminator

Goodfellow et al. (2014) showed that, assuming that p_{data} and p_g are nonzero everywhere, the optimal discriminator D^*_G for a given generator G is:

$$D^*_G(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})} \tag{3.31}$$

where p_g is the data distribution modelled by generator G. To arrive to this result, we wish to find the discriminator D^*_G that minimizes the following equation:

$$J_D(\theta_D, \theta_G) = -\frac{1}{2}\langle \log D_G(\mathbf{x}) \rangle_{\mathbf{x}\sim p_{data}} - \frac{1}{2}\langle \log(1 - D_G(\mathbf{x})) \rangle_{\mathbf{x}\sim p_g} \tag{3.32}$$

Note that this is equivalent to the cost function from equation 3.28. Next, we find $\frac{\partial J_D}{\partial D_G(\mathbf{x})}$ and equate it to 0 in order to find the minima:

$$\frac{\partial J_D}{\partial D_G(\mathbf{x})} = 0 \tag{3.33}$$

$$\int_{\mathbf{x}} p_{data}(\mathbf{x}) \frac{\partial}{\partial D_G(\mathbf{x})} \log D_G(\mathbf{x}) \mathrm{d}x + \int_{\mathbf{x}} p_g(\mathbf{x}) \frac{\partial}{\partial D_G(\mathbf{x})} \log(1 - D_G(\mathbf{x})) \mathrm{d}x = 0 \tag{3.34}$$

$$\int_{\mathbf{x}} \Big(p_{data}(\mathbf{x}) \frac{1}{D_G(\mathbf{x})} - p_g(\mathbf{x}) \frac{1}{1 - D_G(\mathbf{x})} \Big) \mathrm{d}x = 0 \tag{3.35}$$

and the optimal discriminator follows from solving for $D_G(\mathbf{x})$ in equation 3.35. Note also from equation 3.31 that, if the generator G approximates p_{data} perfectly ($p_g = p_{data}$), then $D(\mathbf{x}) = \frac{1}{2}$ as the discriminator can not differentiate between these distributions.

3.2.3 Global optimality

Here we show that the global minimum of the GAN two-player game is achieved when the generator is optimal (i.e. when $p_g = p_{data}$). But before this, let us define two widely used distributional divergences.

The Kullback-Leibler divergence between two distributions is defined as:

$$\mathrm{KL}(q(x) \mid\mid p(x)) = \Big\langle \log \frac{q(x)}{p(x)} \Big\rangle_{q(x)} \tag{3.36}$$

$$= \langle \log q(x) \rangle_{q(x)} - \langle \log p(x) \rangle_{q(x)} \geq 0 \tag{3.37}$$

The Jensen-Shannon divergence between two distributions is defined as:

$$\mathrm{JSD}(q(x) \mid\mid p(x)) = \frac{\mathrm{KL}(q(x) \mid\mid p(x))}{2} + \frac{\mathrm{KL}(p(x) \mid\mid q(x))}{2} \geq 0 \tag{3.38}$$

Now, let us consider the version in which the two players play a zero-sum game (equation 3.29). In this case, the game can be summarized with a single number given by the value function:

$$V(G, D) = \langle \log D(\mathbf{x}) \rangle_{\mathbf{x} \sim p_{data}} + \langle \log(1 - D(\mathbf{x})) \rangle_{\mathbf{x} \sim p_g} \tag{3.39}$$

Goodfellow et al. (2014) showed that, for any given generator G and an optimal discriminator D^*_G, the global minimum of $V(G, D^*_G)$ is achieved if and only if $p_g = p_{data}$, with a value of $-\log 4$. To see this, we first write $V(G, D^*_G)$ using equation 3.31:

$$V(G, D^*_G) = \Big\langle \log \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})} \Big\rangle_{\mathbf{x} \sim p_{data}} + \Big\langle \log \frac{p_g(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})} \Big\rangle_{\mathbf{x} \sim p_g} \tag{3.40}$$

Now, suppose we have an optimal generator G^* such that $p_g = p_{data}$. In this case, $D^*_{G^*}(\mathbf{x}) = \frac{1}{2}$ as the discriminator can not distinguish *fake* samples from *real* samples. As a consequence, we note from equation 3.40 that $V(G^*, D^*_{G^*}) = -\log 4$.

This value is a minimum and it can be only reached when $p_g = p_{data}$ (Goodfellow et al., 2014). Observe that, when we calculate $S(G) = V(G, D^*_G) - V(G^*, D^*_{G^*})$ we obtain:

$$\begin{aligned}
S(G) &= V(G, D^*_G) + 2\log 2 && (3.41)\\
&= \Big\langle \log \frac{2p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})} \Big\rangle_{\mathbf{x} \sim p_{data}} + \Big\langle \log \frac{2p_g(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})} \Big\rangle_{\mathbf{x} \sim p_g} && (3.42)\\
&= \mathrm{KL}\Big(p_{data}(\mathbf{x}) \;\Big|\Big|\; \frac{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}{2}\Big) + \mathrm{KL}\Big(p_g(\mathbf{x}) \;\Big|\Big|\; \frac{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}{2}\Big) && (3.43)\\
&= 2 \cdot \mathrm{JSD}\big(p_{data}(\mathbf{x}) \;||\; p_g(\mathbf{x})\big) && (3.44)
\end{aligned}$$

and we have that $S(G) \geq 0$ and $V(G, D^*_G) \geq V(G^*, D^*_{G^*})$ because the Jensen-Shannon divergence is always non-negative and zero only when the distributions are the same. As a result, the minimum is reached only when $p_g = p_{data}$.

3.2.4 Connection to maximum likelihood estimation

By default, generative adversarial networks are not an instance of the family of models that learn via maximum likelihood estimation. However, we can relate them to the principle of maximum likelihood if we tweak the generator's cost function (equation 3.30).

The principle of maximum likelihood estimates the parameters of a model by finding those which maximize the likelihood of observing the training data. For a

dataset with m observations, the maximum likelihood estimate of p_g is given by:

$$\theta_G^* = \underset{\theta_G}{\operatorname{argmax}} \prod_{i=1}^{m} p_g(\mathbf{x}_i|\theta_G) \tag{3.45}$$

where $\mathbf{x}_i$ is the i-th training sample. This equation is usually rewritten as:

$$\theta_G^* = \underset{\theta_G}{\operatorname{argmax}} \sum_{i=1}^{m} \log p_g(\mathbf{x}_i|\theta_G) \tag{3.46}$$

since the logarithm increases monotonically and thus it does not change the maximum. Now, let $q(\mathbf{x})$ be the empirical distribution:

$$q(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^{m} \mathcal{I}[\mathbf{x} = \mathbf{x}_i] \tag{3.47}$$

where $\mathcal{I}[\cdot]$ is the indicator function. We usually do not have access to $p_{data}(\mathbf{x})$, so we often use the empirical distribution $q(\mathbf{x})$ as a proxy.

Note that finding the maximum likelihood estimation is equivalent to minimizing the KL-divergence between the empirical distribution and p_g:

$$\begin{aligned}
\theta_G^* &= \underset{\theta_G}{\operatorname{argmin}} \, \mathrm{KL}(q(\mathbf{x}) \;||\; p_g(\mathbf{x}|\theta_G)) && (3.48)\\
&= \underset{\theta_G}{\operatorname{argmin}} \; -\langle \log p_g(\mathbf{x}|\theta_G) \rangle_{q(\mathbf{x})} && (3.49)\\
&= \underset{\theta_G}{\operatorname{argmax}} \; \sum_{i=1}^{m} q(\mathbf{x}_i) \log p_g(\mathbf{x}|\theta_G) && (3.50)\\
&= \underset{\theta_G}{\operatorname{argmax}} \; \sum_{i=1}^{m} \log p_g(\mathbf{x}_i|\theta_G) && (3.51)
\end{aligned}$$

The generator's cost functions that we considered previously (equations 3.29 and 3.30) do not perform maximum likelihood estimation. However, if we assume that $p(\mathbf{x}|\theta_G)$ is nonzero everywhere, we can relate GANs with the principle of maximum likelihood by finding a function f such that the expected gradient of the following generator's cost (Goodfellow, 2017)

$$J_G = \langle f(\mathbf{x}) \rangle_{\mathbf{x} \sim p_g} \tag{3.52}$$

is equivalent to the expected gradient of $\mathrm{KL}(p_{data}(\mathbf{x}) \;||\; p_g(\mathbf{x}|\theta_G))$:

$$\frac{\partial}{\partial \theta_G} \mathrm{KL}(p_{data}(\mathbf{x}) \;||\; p_g(\mathbf{x}|\theta_G)) = -\Big\langle \frac{\partial}{\partial \theta_G} \log p_g(\mathbf{x}|\theta_G) \Big\rangle_{\mathbf{x} \sim p_{data}} \tag{3.53}$$

First, observe that the expected gradient of J_G (equation 3.52) is:

$$\frac{\partial}{\partial \theta_G} J_G = \frac{\partial}{\partial \theta_G} \int_{\mathbf{x}} p_g(\mathbf{x}|\theta_G) f(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{3.54}$$

If we assume that both $f(\mathbf{x})$ and $\frac{\partial f(\mathbf{x})}{\partial \theta_G}$ are continuous, and that $f(\mathbf{x})$ vanishes for infinite values of $\mathbf{x}$, we can push the differentiation into the integral (Goodfellow, 2017):

$$\frac{\partial}{\partial \theta_G} J_G = \int_{\mathbf{x}} \frac{\partial}{\partial \theta_G} p_g(\mathbf{x}|\theta_G) f(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{3.55}$$

$$= \int_{\mathbf{x}} f(\mathbf{x}) \Big[\frac{\partial}{\partial \theta_G} p_g(\mathbf{x}|\theta_G) \Big] \mathrm{d}\mathbf{x} \tag{3.56}$$

And since $p_g(\mathbf{x}|\theta_G) \geq 0 \ \forall \mathbf{x}$, we can use the identity $p_g(\mathbf{x}|\theta_G) = \exp(\log(p_g(\mathbf{x}|\theta_G)))$:

$$\frac{\partial}{\partial \theta_G} J_G = \int_{\mathbf{x}} f(\mathbf{x}) \Big[\frac{\partial}{\partial \theta_G} \exp\left(\log p_g(\mathbf{x}|\theta_G)\right) \Big] \mathrm{d}\mathbf{x} \tag{3.57}$$

$$= \int_{\mathbf{x}} f(\mathbf{x}) p_g(\mathbf{x}|\theta_G) \Big[\frac{\partial}{\partial \theta_G} \log p_g(\mathbf{x}|\theta_G) \Big] \mathrm{d}\mathbf{x} \tag{3.58}$$

$$= \Big\langle f(\mathbf{x}) \frac{\partial}{\partial \theta_G} \log p_g(\mathbf{x}|\theta_G) \Big\rangle_{\mathbf{x} \sim p_g} \tag{3.59}$$

The resulting expression looks similar to the gradient of the KL-divergence (equation 3.53), but the expectation is over $p_g(\mathbf{x}|\theta_G)$ instead of $p_{data}(\mathbf{x})$. To overcome this problem, we can use a trick from importance sampling (Goodfellow, 2017). If we set $f(\mathbf{x}) = -\frac{p_{data}(\mathbf{x})}{p_g(\mathbf{x}|\theta_G)}$, we are able to compensate for the fact that the samples are being drawn from the generator instead of the data distribution. Note that we must copy θ_G into $f(\mathbf{x})$ in order to have $\frac{\partial f(\mathbf{x})}{\partial \theta_G} = \mathbf{0}$, but this happens naturally when we obtain the value of $\frac{p_{data}(\mathbf{x})}{p_g(\mathbf{x}|\theta_G)}$ (Goodfellow, 2017). This value is a density ratio that compares the hypothesis of $\mathbf{x}$ coming from the real distribution as opposed to being generated by generator G.

This density ratio can be estimated using the discriminator. To see this, we first introduce a dummy variable y indicating whether a sample comes from the real or the synthetic distribution, and we write the probabilities $p_{data}(\mathbf{x})$ and $p_g(\mathbf{x}|\theta_G)$ in conditional form:

$$p_{data}(\mathbf{x}) = \rho(\mathbf{x}|y=1) \tag{3.60}$$

$$p_g(\mathbf{x}|\theta_G) = \rho(\mathbf{x}|y=0) \tag{3.61}$$

Next, we rewrite $-f(\mathbf{x})$ using the Bayes' theorem:

$$\frac{p_{data}(\mathbf{x})}{p_g(\mathbf{x}|\theta_G)} = \frac{\rho(\mathbf{x}|y=1)}{\rho(\mathbf{x}|y=0)} \tag{3.62}$$

$$= \frac{\rho(y=1|\mathbf{x})\rho(\mathbf{x})}{\rho(y=1)} \Big/ \frac{\rho(y=0|\mathbf{x})\rho(\mathbf{x})}{\rho(y=0)} \tag{3.63}$$

Since $p_g(\mathbf{x}|\theta_G)$ is nonzero everywhere, and if we assume a uniform prior $\rho(y=0) = \rho(y=1) = \frac{1}{2}$, the expression above simplifies to:

$$\frac{p_{data}(\mathbf{x})}{p_g(\mathbf{x}|\theta_G)} = \frac{\rho(y=1|\mathbf{x})}{\rho(y=0|\mathbf{x})} \tag{3.64}$$

which can be estimated with the discriminator as:

$$\frac{p_{data}(\mathbf{x})}{p_g(\mathbf{x}|\theta_G)} = \frac{D(\mathbf{x})}{1 - D(\mathbf{x})} \tag{3.65}$$

3.2.5 Training a GAN

We optimize for θ_D and θ_G using a gradient-based optimization algorithm and iterating until convergence. To compute the gradients, we use backpropagation (Rumelhart et al., 1988). The algorithm proposed by Goodfellow et al. (2014) for training a GAN is shown in algorithm 2.

Algorithm 2: Training a GAN with a gradient-based optimization method (Goodfellow et al., 2014). k controls the number of discriminator updates per iteration, and GradientUpdate is a function that updates the parameters. The other notation is inherited from this chapter.

Data: We are given a data distribution p_{data} and a noise distribution p_z.

while *not convergence criteria is reached* **do**
  for k *steps* **do**
    Sample mini-batch $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from p_z
    Sample mini-batch $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ from p_{data}
    Update discriminator:
      $\mathbf{g} \leftarrow \nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} \Big[\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)}))) \Big]$
      $\theta_D \leftarrow \text{GradientUpdate}(\theta_D, \mathbf{g})$
  Sample mini-batch $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from p_z
  Update generator:
    $\mathbf{g} \leftarrow \nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} \Big[\log(D(G(\mathbf{z}^{(i)}))) \Big]$
    $\theta_G \leftarrow \text{GradientUpdate}(\theta_G, \mathbf{g})$

Chapter 4

Experiments

4.1 Dataset

In this study, we leverage data from *Escherichia coli* (*E. coli*) to design a generative model of its expression data. This bacterium has been especially useful to molecular biologists because of its relatively simplicity and the ease with which it can be studied in the laboratory (Cooper, 2000). *E. coli* grows very fast (it divides every 20 to 60 minutes), and experimental results can be collected after a few hours of preparing the cultures. In addition, it is convenient to work with *E. coli* expression data because it has a relatively small and simple genome (∼4,400 genes) and its gene expression mechanisms are relatively well understood (Salgado et al., 2006).

4.1.1 *E. coli* gene expression M^{3D} data

Many Microbe Microarrays Database (M^{3D}; Faith et al., 2008) is a database that contains gene expression data for microbes such as *Escherichia coli*, *Saccharomyces cerevisiae* and *Shewanella oneidensis*. The data from M^{3D} comes from single-channel Affymetrix microarray experiments that were carried out by different laboratories (Faith et al., 2008). The data was uniformly normalized using log-scale robust multi-array average (RMA; Irizarry, 2003) to reduce batch effects and make the samples comparable across conditions.

The *E. coli* M^{3D} database consists of 907 chips. There are 466 unique conditions, hence some of these chips measure expressions from replicate experiments. The database provides both the 907 original measurements and the 466 expressions averaged across replicate experiments. In this study we use the data from the 907 original measurements, and we leverage the information about replicate experiments to sensibly split the data into a train and a test set. This train-test partition strategy is

further discussed in section 4.1.3.

Each of the 907 chips consists of 7459 probes. These probes include the expression values of genes, intergenic regions and control probes. In this study we are specifically interested in gene expression values, so we exclude intergenic regions and control probes. The resulting dataset consists of 907 samples and 4297 features corresponding to the expression values of *E. coli* genes. The log-scaled RMA-normalized gene expression values range from 2.94 to 15.19, with an overall mean value of 8.77.

4.1.2 *E. coli* gene regulatory interactions and RegulonDB

The gene regulatory network of *E. coli* is one of the most well-characterized transcriptional networks of a single cell. These networks dictate how the expressions of certain genes (transcription factors; TFs) regulate the expression of other genes (target genes; TGs) by activating and/or repressing their rate of gene transcription. The gene regulatory network of *E. coli* has been widely studied for several decades and the amount of experimentally validated knowledge is currently the largest available for any organism (Salgado et al., 2006).

RegulonDB (Gama-Castro et al., 2016) is a database that integrates biological knowledge about the transcriptional regulatory mechanisms of *E. coli*. The database gathers information from multiple biological studies to reconstruct the structure of the *E. coli* GRN. For each regulatory interaction, RegulonDB provides the name of the TF; the gene regulated by the TF; the regulatory effect (activation and/or repression); the experimental evidences that support the existence of the regulatory interaction; and a measure of the evidence strength (weak, strong or confirmed). Figure 4.1 shows the core gene regulatory network of *E. coli* according to RegulonDB.

This study leverages information from RegulonDB for two purposes. The first one is motivated by the fact that the number of samples (907) of the *E. coli* M^{3D} dataset is rather scarce in comparison to the number of genes (4297). In this work, as one of our experiments, we use RegulonDB to select a meaningful subset of *E. coli* genes by choosing a hierarchy of genes whose transcription is directly or indirectly regulated by one of the *E. coli* master regulators. The details about our selection of genes are provided in section 4.1.3. Secondly, we use RegulonDB to characterize several statistical properties of the simulated datasets, with the aim of comparing them with those of the real dataset. Concretely, we use the *E. coli* gene regulatory network from RegulonDB to check whether the following properties hold in the synthetic dataset: the correlation between TFs and TGs; the correlation between TGs regulated by the

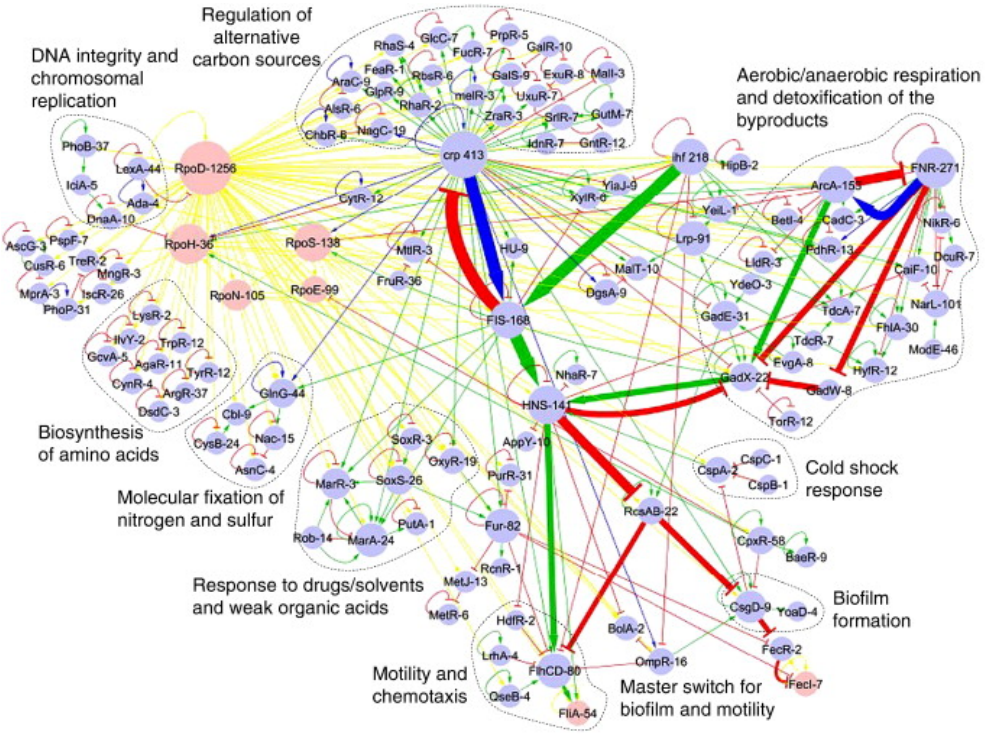


Figure 4.1: Core gene regulatory network of *E. coli*, obtained from RegulonDB. Blue and pink nodes are TFs and sigma factors, respectively. Each label is accompanied by a number showing the number of regulatory targets for each node. Green, red, blue and yellow edges represent the following effects, respectively: activation, repression, dual interaction and sigma transcription. Specific subnetworks are delineated with dashed lines. Source: Martinez-Antonio et al. (2008)

same TF; the activity of TFs based on statistical tests. These properties are further described in section 3.1.

4.1.3 Data preparation

In this section we discuss how we preprocess the *E. coli* M^{3D} expression dataset. The goal of this step is to facilitate the learning process and to enable the model to find complex expression patterns more easily. First, we present a reasonable way to relax the problem (i.e. reduce the number of genes) by selecting a meaningful hierarchy of genes. Second, we explain how we standardize the expression data to keep it within a reasonable range, and how we recover the original gene ranges after producing the artificial expressions. Lastly, we discuss the strategy that we follow to split the data into a train and a test set in a sensible manner.

4.1.3.1 Selecting the CRP hierarchy

As one of the experiments in our study, we select a meaningful subset of *E. coli* genes whose expression is directly or indirectly regulated by the master regulator cAMP receptor protein (CRP). CRP regulates global patterns of transcription in response to carbon availability, and it is one of the best characterized global transcriptional regulators in *E. coli*. This receptor increases the levels of promoter occupancy by binding to the DNA sites in the promoters of its target genes, and by interacting with RNA polymerase to activate their transcription (Grainger and Busby, 2008).

Information from the *E. coli* gene regulatory network is used to extract the regulatory hierarchy in which CRP is the root node. Concretely, we use algorithm 3 with the RegulonDB network of regulatory interactions to select the CRP hierarchy from the *E. coli* M^{3D} dataset. Note that, when we break loops, the resulting structure is a directed acyclic graph, as edges going from children to ancestors are ignored.

Algorithm 3: Selecting a hierarchy of genes. Returns the sets of nodes and edges in the hierarchy.

Data: We are given a set of genes $\mathcal{G}$, a root gene $g_r \in \mathcal{G}$, a network $\mathcal{S}$ of gene regulatory interactions, and a boolean b indicating whether to break loops.

Initialize sets of edges, previous and current included nodes, and ancestors of each node:
$\mathcal{E} \leftarrow \{\}$
$\mathcal{V}' \leftarrow \{\}$
$\mathcal{V} \leftarrow \{g_r\}$
$\mathcal{A}(g) \leftarrow \{g\} \quad \forall g \in \mathcal{G}$
while $\mathcal{V} \neq \mathcal{V}'$ **do**
  Update previous nodes:
    $\mathcal{V}' \leftarrow \mathcal{V}$
  Select nodes to be added:
    $\mathcal{C} \leftarrow \{g_1 \in \mathcal{G} \mid \exists g_2 \in \mathcal{V}, (g_2 \rightarrow g_1) \in \mathcal{S}, \neg b \vee \neg g_1 \in \mathcal{A}(g_2)\}$
  Add them to set of nodes:
    $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{C}$
  for *each* $g_1 \in \mathcal{C}$ **do**
    Find parents of g_1:
      $\mathcal{P} \leftarrow \{g_2 \in \mathcal{V}' \mid (g_2 \rightarrow g_1) \in \mathcal{S}, \neg b \vee \neg g_1 \in \mathcal{A}(g_2)\}$
    Add edges coming from parents:
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{g_2 \rightarrow g_1 \mid g_2 \in \mathcal{P}\}$
    Update ancestors:
      $\mathcal{A}(g_1) \leftarrow \mathcal{A}(g_1) \cup \mathcal{P} \cup \bigcup_{p \in \mathcal{P}} \mathcal{A}(p)$
return $\mathcal{V}$, $\mathcal{E}$

4.1.3.2 Data standardization

We standardize the expression values to keep them within a reasonable range and to make the learning process easier. Let $\mathbf{X}$ be an $m \times n$ matrix of expression values, where m is the number of samples and n is the number of genes. We standardize the data as follows:

$$\mathbf{Z}_{i,j} = \frac{\mathbf{X}_{i,j} - \mu_{\mathbf{X},j}}{\kappa \cdot \phi_{\mathbf{X},j}} \tag{4.1}$$

where κ is user-definable and:

$$\mu_{\mathbf{X},j} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{X}_{i,j} \tag{4.2}$$

$$\phi_{\mathbf{X},j} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (\mathbf{X}_{i,j} - \mu_{\mathbf{X},j})^2} \tag{4.3}$$

The expression values of each gene in the resulting standardized expression matrix $\mathbf{Z}$ have mean 0 and standard deviation κ^{-1}.

Now, let $\widehat{\mathbf{Z}}$ be an $m \times n$ matrix produced by a given generative model. To recover the original gene expression ranges, we first standardize the produced expression values to make them have mean 0 and standard deviation 1 for each gene:

$$\widetilde{\mathbf{Z}}_{i,j} = \frac{\widehat{\mathbf{Z}}_{i,j} - \mu_{\widehat{\mathbf{Z}},j}}{\phi_{\widehat{\mathbf{Z}},j}} \tag{4.4}$$

Finally, we apply the following transformation to recover the original ranges:

$$\widetilde{\mathbf{X}}_{i,j} = \mu_{\mathbf{X},j} + \phi_{\mathbf{X},j} \cdot \widetilde{\mathbf{Z}}_{i,j} \tag{4.5}$$

Each gene j has now mean $\mu_{\mathbf{X},j}$ and standard deviation $\phi_{\mathbf{X},j}$.

4.1.3.3 Train-test split

Unlike in other domains where the model's target or the evaluation metrics are clearly defined for each sample, the need for a test set in this unsupervised problem is questionable. One might argue that reporting the generalization results on a separate test set is not necessary, as we are trying to approximate the real data distribution and the train/test data are both sampled from this distribution.

However, we opt for having a test set for two reasons. First, the number of samples of the *E. coli* M^{3D} dataset is rather scarce, and thus they might not represent the real distribution of *E. coli* gene expressions well enough. Therefore, comparing the

properties of our generated data against those of an unseen test set might provide us with a more realistic view of our generalization capability. Second, keeping a separate test set allows to compare its properties with those of the training set, and this gives an upper bound for our evaluation metrics.

We split the *E. coli* M^{3D} data into train (680 samples) and test (227 samples) sets. We partition the data in a random manner, constraining replicate samples to lie within the same set. This prevents the leakage of information between sets. We use the train set to train our generative model and tune its hyperparameters, whereas the test set is kept apart and used only to report the final results.

4.2 Model design

We use a Generative Adversarial Network as our adversarial model. The GAN is composed by a generator that tries to produce realistic expression samples, and a discriminator that attempts to distinguish them from real samples. These two players compete with each other and, in an ideal scenario, the functions that characterize them should converge to the global optimum described in section 3.2.3. Figure 4.2 shows an overview of our GAN, whereas figure A.1 depicts its corresponding graphical model. Next, we describe the architectures of the generator and the discriminator in detail.

4.2.1 Generator's architecture

The generator should be able to produce a diverse, biologically-plausible set of expression arrays. In other words, we wish to generate batches of samples similar to those sampled directly from the real distribution $P(\mathbf{x}_{real})$. To this end, the generator must account for the underlying biological processes that underpin gene expression. In addition to emulating gene regulatory interactions, the generator is expected to integrate a noise model that simulates random fluctuations due to biological processes such as protein synthesis, which might influence considerably the way in which genes are expressed.

The generator $G_1 : \mathbb{R}^d \to \mathbb{R}^n$ is a parametric function that takes some noise $\mathbf{z} \in \mathbb{R}^d$ as input and produces a vector $\mathbf{x}_{fake} \in \mathbb{R}^n$ of synthetic gene expressions as output. We model the G_1 as a feedforward neural network that takes random noise $\mathbf{z}$ sampled from an isotropic Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$ as input, and produces $\mathbf{x}_{fake}$ by learning inner representations that depend on the input noise. The dimensionality d of the noise vector $\mathbf{z}$ is an hyperparameter, and a higher value for d enables a richer set of inner

representations. By using a large enough random input $\mathbf{z}$, we aim to provide the generator with a series of latent variables that define the state of several biological processes related to gene expression.

In addition to these biological processes, the expression data is subject to stochastic fluctuations (Raser and O'shea, 2005) that might influence each gene in a different manner. The noisy input $\mathbf{z}$ allows G_1 to model regulatory interactions as well as biological noise, but this source of randomness affects the expression of several genes in a joint manner, and therefore the generator might be prone to obviate gene-independent sources of randomness. To account for these random fluctuations, we design a generator $G_2 : \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}^n$ that integrates an additive white Gaussian noise model as follows:

$$G_2(\mathbf{z}, \mathbf{s}) = G_1(\mathbf{z}) + Q(\mathbf{s}) \tag{4.6}$$

with $Q(\mathbf{s}) = \sigma \odot \mathbf{s}$, where σ is a vector of n learnable parameters that model the standard deviation of the additive noise for each gene. This simple noise model adds σ-scaled noise $\mathbf{s}$ sampled from an isotropic Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$. We do not introduce a parameter to model the mean of the Gaussian, as the systematic error for each gene is already captured by the bias term in the output layer of G_1. We provide experimental details about the generator network in section 4.3.4.

4.2.2 Discriminator's architecture

The goal of the discriminator is to distinguish *real* expression arrays, sampled from the real distribution $P(\mathbf{x}_{real})$, from the *fake* samples produced by the generator. In other words, the discriminator attempts to find patterns in *real* samples that are not present in *fake* samples, and vice versa. Thus, the discriminator is guiding the generator to produce more realistic samples by providing gradients of the generator's parameters that point towards the minima in which the *real* distribution of expression data is best approximated.

The discriminator $D : \mathbb{R}^n \to \mathbb{R}^1$ is a parametric function that models the conditional probability distribution $P(\mathbf{x} \text{ is } \textit{real} \mid \mathbf{x})$. We model D as a fully-connected neural network with a sigmoid $f(x) = \frac{1}{1+e^{-x}}$ output activation, which is the most natural choice for binary classification tasks. The concrete characteristics of the discriminator network (such as the number of hidden layers, activation functions, and regularization mechanisms, among others) are determined experimentally. We describe them in section 4.3.4.

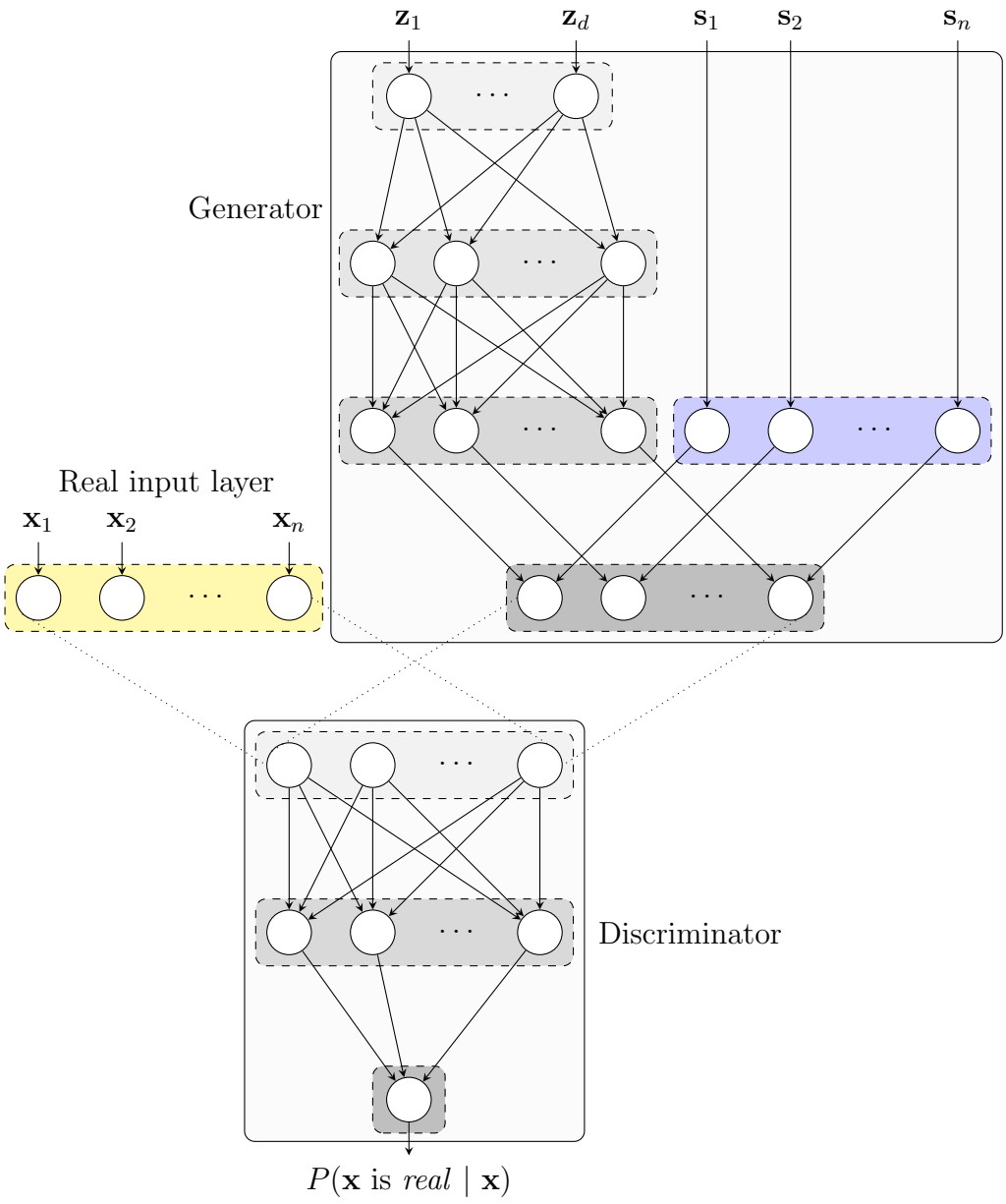


Figure 4.2: Architecture of the GAN. The generator receives noise from two sources as input. The input $\mathbf{z} \in \mathbb{R}^d$ aims to provide the generator with a series of latent variables that define the state of several biological processes, whereas the input $\mathbf{s} \in \mathbb{R}^n$ is specifically designed for the additive white Gaussian noise model with learnable variances. The discriminator takes expression data from two different sources as input: an input layer fed with gene expression samples from the *real* distribution; and the output layer of the generator that produces *fake* samples. Both networks are trained with a gradient-based algorithm, and the gradients are computed via backpropagation (Rumelhart et al., 1988). Note also that the generator's gradients are backpropagated through the discriminator.

4.3 Adversarial training

We train the generator and the discriminator in an adversarial manner. In this section we provide the details about our training algorithm, which introduces some nuances with respect to algorithm 2 (Goodfellow et al., 2014). At each iteration, we sample two minibatches: one drawn from the real dataset and the other from the model's prior over the latent variables (an isotropic Gaussian). Then, we use these minibatches to estimate the gradients of the generator's and discriminator's loss functions, and we perform a gradient-based update. We choose to use Adam optimization algorithm (Kingma and Ba, 2014), which has been successfully employed to train remarkable GANs such as DCGANs (Radford et al., 2015) and CycleGANs (Zhu et al., 2017).

4.3.1 Constraining the weights of the noise model

We constrain the norm of the standard deviation parameters from the noise model $Q(\mathbf{s})$. This decision is motivated by the fact that the generator has no incentive to add the gene-wise Gaussian noise to the simulated data. Assuming that the distribution of the additive noise is indeed a σ-scaled isotropic Gaussian, the discriminator might average it out and learn that noiseless samples (in which the real signal is preserved) are more likely to occur than those with noise.

In this case, the generator is encouraged to produce samples without the additive white Gaussian noise. As a consequence, the weights σ of the additive noise model $Q(\mathbf{s})$ are pushed towards zero. To overcome this problem, we introduce a constraint to control the norm of σ. Concretely, we propose to renormalize σ after each training iteration in order to enforce $|\sigma|_1 = \xi$, where ξ is an hyperparameter that controls the overall amount of noise being added.

4.3.2 Regularization

We use dropout in the fully-connected hidden layers of both the discriminator and the generator. This technique randomly drops units along with their connections (Srivastava et al., 2014) with probability p during training, and scales the output of the remaining neurons by $\frac{1}{1-p}$ to maintain the expected output. This makes the network less reliant on specific neurons, and thus it reduces overfitting. Applying dropout can also be seen as creating a large ensemble of neural networks in a computationally efficient manner.

In addition, we clip the weights of the discriminator after each update, constraining them to be within a user-definable range. This constitutes a form of regularization,

and contributes to prevent the discriminator from remembering the few M^{3D} samples by limiting its complexity. Other GAN versions such as Wasserstein GANs (Arjovsky et al., 2017) also clip the weights of the discriminator.

On the other hand, we apply one-sided label smoothing (Salimans et al., 2016) to prevent the discriminator from being overconfident, which often implies overfitting. Similarly to the proof in section 3.2.2, one can show that, when we replace positive targets with α and negative targets with β, the optimal discriminator is:

$$D_G^*(\mathbf{x}) = \frac{\alpha p_{data}(\mathbf{x}) + \beta p_g(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})} \tag{4.7}$$

This is problematic because when $p_{data}(\mathbf{x})$ is close to zero and $p_g(\mathbf{x})$ is large, $p_g(\mathbf{x})$ is not encouraged to move close to $p_{data}(\mathbf{x})$ in those areas, and therefore the global optimum (see section 3.2.3) changes. Hence, we only smooth positive labels. More specifically, we set α to a random value between 0.7 and 1, and β to 0.

4.3.3 Training algorithm

Algorithm 4 summarizes our training algorithm.

Algorithm 4: Training our GAN. AdamUpdate is a function that updates the parameters according to the Adam optimization algorithm (Kingma and Ba, 2014). ClipWeights is a function that clips parameters to make them be within a given range. The vector $\sigma \in \theta_G$ contains the parameters from the gene-wise noise model, while ξ is a hyperparameter that controls the norm of σ. The hyperparameter c controls the range of the discriminator's weights θ_D.

Data: We are given a data distribution p_{data} and a noise distribution p_z.

while *not convergence criteria is reached* **do**
- Sample mini-batch $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from p_z
- Sample mini-batch $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ from p_{data}
- Update discriminator:
  - Sample noisy labels $\{\alpha^{(1)}, ..., \alpha^{(m)}\}$ uniformly from $[0.7, 1]$
  - $\mathbf{g} \leftarrow \nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} \left[\alpha^{(i)} \log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)}))) \right]$
  - $\theta_D \leftarrow \text{AdamUpdate}(\theta_D, \mathbf{g})$
  - $\theta_D \leftarrow \text{ClipWeights}(\theta_D, -c, c)$
- Sample mini-batch $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from p_z
- Update generator:
  - $\mathbf{g} \leftarrow \nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} \left[\log(D(G(\mathbf{z}^{(i)}))) \right]$
  - $\theta_G \leftarrow \text{AdamUpdate}(\theta_G, \mathbf{g})$
  - $\sigma \leftarrow \frac{\xi \cdot \sigma}{\sum_{j=1}^{n} |\sigma_j|}$

4.3.4 Hyperparameter tuning

We use the train set to tune the hyperparameters of our GAN. Unlike in other tasks where the model's target is clearly defined, we consider that using a validation set is unnecessary for this problem. We believe that it is hard for our generator to overfit to the training set for two reasons. First, because our evaluation metrics are not defined for a single example, but rather for the full generated dataset. Second, because the generator is never shown the training data. Instead, the generator updates its parameters according to the feedback that it receives from the discriminator. Moreover, Wu et al. (2016) argue that GANs are typically less affected by overfitting.

We optimize the hyperparameters of two GANs trained on a different set of genes. The first is trained using the group of genes from the CRP hierarchy (1076 genes; see section 4.1.3.1), whereas the second uses the full set of *E. coli* M^{3D} genes (4297). In both cases, we find the Adam optimizer with the configuration recommended by Radford et al. (2015) (learning rate of 0.0002, first momentum coefficient of 0.9 and second momentum coefficient of 0.999) to work reasonably well both for the generator and the discriminator. In all experiments, we clip the discriminator's weights between -0.1 and 0.1 after each iteration, in order to limit its complexity. We initialize all the weights using Xavier uniform initialization (Glorot and Bengio, 2010), and we initially set the bias terms to 0. In addition, we use a batch size of 32, and we train the GANs for 4000 epochs, selecting the model in which the coefficient $\gamma(\mathbf{D}^X, \mathbf{D}^Z)$ (see section 3.1) is higher.

In terms of the architecture, we use one hidden layer both for the discriminator and the generator. For the latter network, we append an extra layer with local connections that adds the noise produced by the noise model $Q(\mathbf{s})$, as depicted in figure 4.2. In both experiments, we find that adding more hidden layers does not yield significant improvements in our evaluation scores. We use a leaky ReLU activation $f(x) = max(0.3x, x)$ in all the hidden layers. Radford et al. (2015) recommend using a ReLU activation $f(x) = max(0, x)$ for the generator's hidden layers, but in this problem the forementioned leaky ReLU works better. We use a sigmoid activation function for the discriminator, which is the most natural choice for binary classification. Besides, we use a linear activation for the generator, as restricting the range of the output expression values would conflict with the standardization procedure described in section 4.1.3.2.

As a result of our experimentation, for the first GAN (1076 genes), we end up with an input layer of $d = 20$ units and a hidden layer of $\sim$1000 neurons both for the discriminator and the generator. For the second GAN (4297 genes), an input layer

with $d = 100$ and a hidden layer of $\sim$2000 units yield good results. All these hidden neurons are dropped out with a probability of 0.5. This value provides the highest level of regularization (Baldi and Sadowski, 2013) and boosts our evaluation scores notably. In contrast, our models do not benefit from using batch normalization (Ioffe and Szegedy, 2015), which is often recommended to stabilize the learning process of the GAN (Radford et al., 2015).

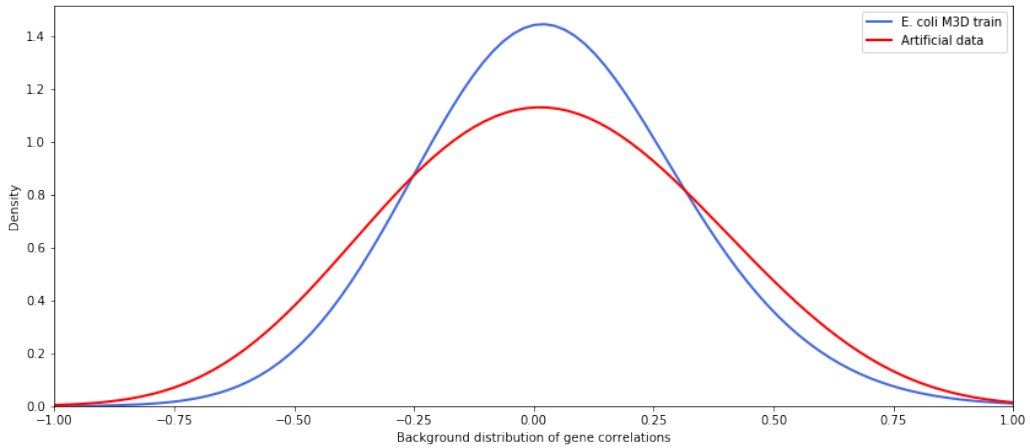
Finally, one of the most critical hyperparameters of our GAN is ξ, because it controls the overall amount of noise being added to the output data. In order to optimize its value, we attempt to match the synthetic and real background distributions of correlation coefficients between all pairs of genes. On the one hand, when ξ is too small, the amount of added noise is small and genes are exaggeratedly correlated among each other. On the other hand, when ξ is too large, genes become less correlated among each other and the background distribution has more density around 0. Figure 4.3 shows these background distributions for different settings of ξ. Additionally, in section A.2 we show that an appropriate value for ξ improves our evaluation scores.

4.4 Software and hardware references

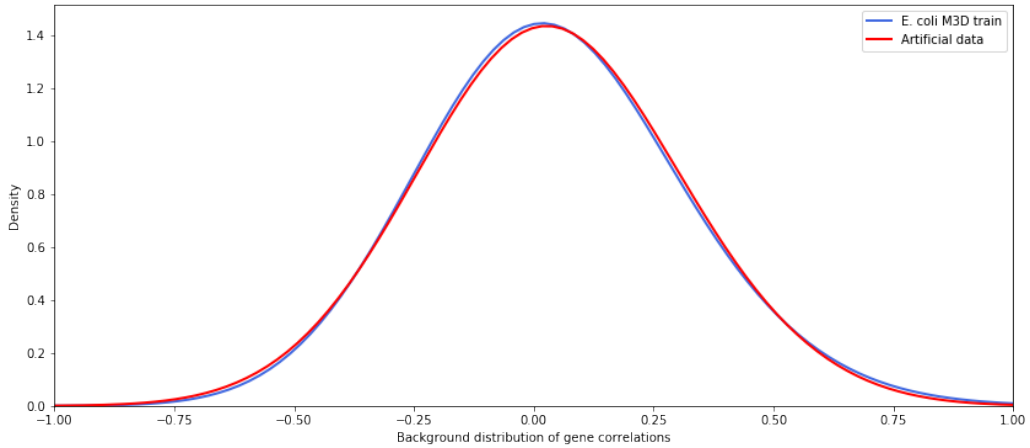
Our implementation uses Python 3.5 (Rossum, 1995). To implement the generative adversarial network, we use Keras (Chollet et al., 2015) and TensorFlow (Abadi et al., 2015). We also use the following numerical computing and statistical libraries: SciPy (Jones et al., 2001), NumPy (Walt et al., 2011), pandas (McKinney, 2010), scikit-learn (Pedregosa et al., 2011) and statsmodels (Seabold and Perktold, 2010). In addition, we use matplotlib (Hunter, 2007), seaborn (Waskom et al., 2017) and Jupyter (Kluyver et al., 2016) to visualize our results.

We train and tune our models on a AWS *p2.xlarge* instance[1] with a NVIDIA Tesla K80 GPU[2]. It takes between 1 and 10 minutes to train the GAN for the full set of *E. coli* M^{3D} genes, depending on the chosen architecture and the hyperparameters. For a generator with more than 4,000,000 parameters, generating an expression dataset of 680 samples takes about 10 seconds on a i7 machine (2,2 GHz) with 16 GB of RAM. Evaluating the synthetic data and visualizing the results for the full set of genes (i.e. computing gene correlations, running hypothesis tests, performing KDE to plot smooth histograms, ...) takes about 15 minutes on the same machine.

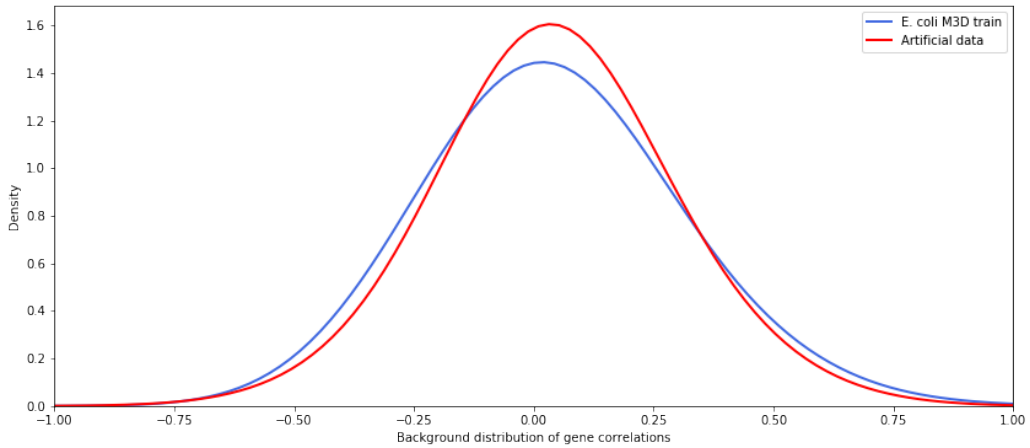
[1] `https://aws.amazon.com/es/ec2/instance-types/p2/`
[2] `https://www.nvidia.com/en-us/data-center/tesla-k80/`



(a) Background distribution with $\kappa = 2$ and $\frac{\xi}{n} = 0$



(b) Background distribution with $\kappa = 2$ and $\frac{\xi}{n} = 0.25$



(c) Background distribution with $\kappa = 2$ and $\frac{\xi}{n} = 0.5$

Figure 4.3: Background distributions for different settings of ξ on the CRP hierarchy dataset. Let n be the number of genes. We observe that the background distributions approximately match when $\frac{\xi}{n} = 0.25$ (for the full set of genes, $\frac{\xi}{n} = 0.18$). These experiments were performed for $\kappa = 2$ (see section 4.1.3.2 for details about κ).

4.5 Results

In this section we examine the properties of the synthetic expression data produced by our Generative Adversarial Network. We call gGAN (*g*ene expression *GAN*) to our generative model. To sample data from gGAN, we feed the generator with random noise sampled from the model's prior over the latent variables $\mathbf{z}$ (an isotropic Gaussian distribution) and we compute the forward pass. Then, we rescale the output as described in section 4.1.3.2 (equations 4.4 and 4.5). We also ensure that all the expression values lie between 2.96 and 15.19 (minimum and maximum expression values in the training set, respectively) by clipping the few outliers that are induced by the randomness of the white Gaussian noise model. By repeatedly executing this process, we generate a dataset of 680 samples, equivalent to the training set in size.

4.5.1 Evaluating gGAN gene expression data

Here we evaluate the quality of the synthetic data generated by gGAN for the full set of *E. coli* M^{3D} (4297 genes). We examine the properties described in section 3.1, using the test samples as a representation of the true data distribution. Additionally, we contrast the forementioned properties between the training and the test sets. Since both sets are sampled from the real distribution, this comparison should provide us with an overview of the irreducible error and an approximate upper bound on our evaluation metrics. We discuss these results in section 4.5.1.9.

4.5.1.1 Gene intensities

Figure 4.4 depicts the overall histograms of gene intensities.

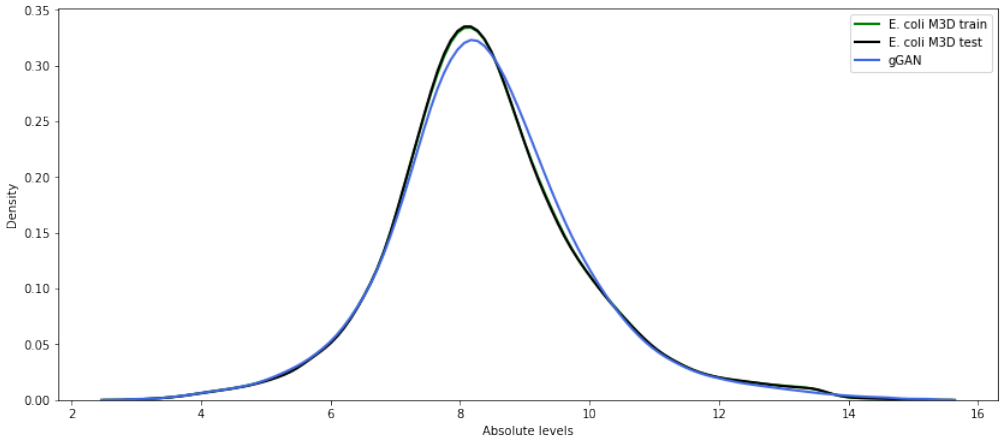


Figure 4.4: Distribution of gene intensities.

4.5.1.2 Gene ranges

Figure 4.5 shows the histogram of gene ranges. To form this histogram, we compute the differences between the minimum expression (5% intensity quantile) and the maximum expression (95% intensity quantile) for each gene.

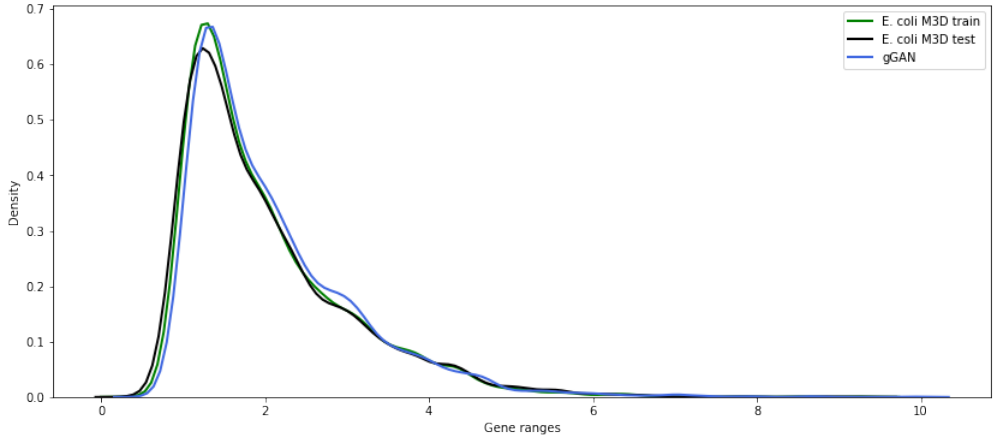


Figure 4.5: Range of gene expressions.

4.5.1.3 Background distribution of correlation coefficients

Figure 4.6 illustrates the background distribution of correlation coefficients between all pairs of genes. A higher amount of noise results in a more peaked distribution.

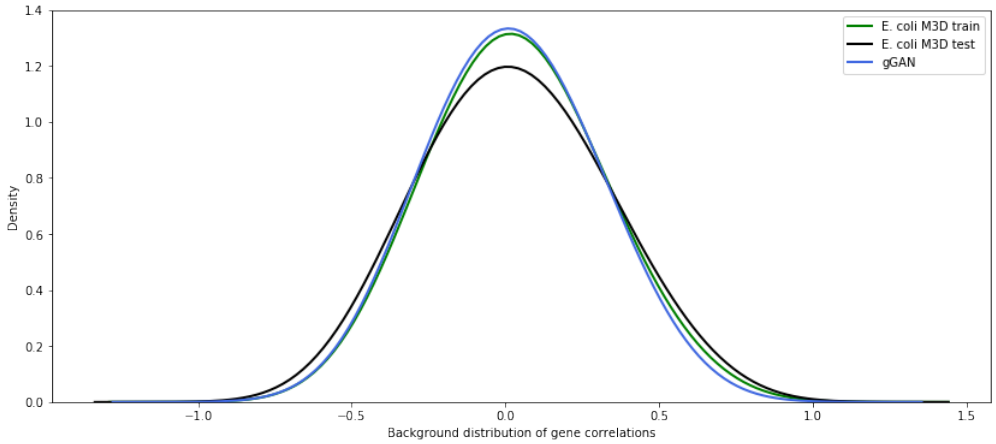


Figure 4.6: Background distribution of the Pearson's correlation coefficients between all pairs of genes.

4.5.1.4 TF-TG histogram

Figure 4.7 shows the density difference between the distribution of TF-TG interactions and the background distribution (section B.1.1.3). To construct the former distribution, we compute the Pearson's correlation between each TF and its TGs.

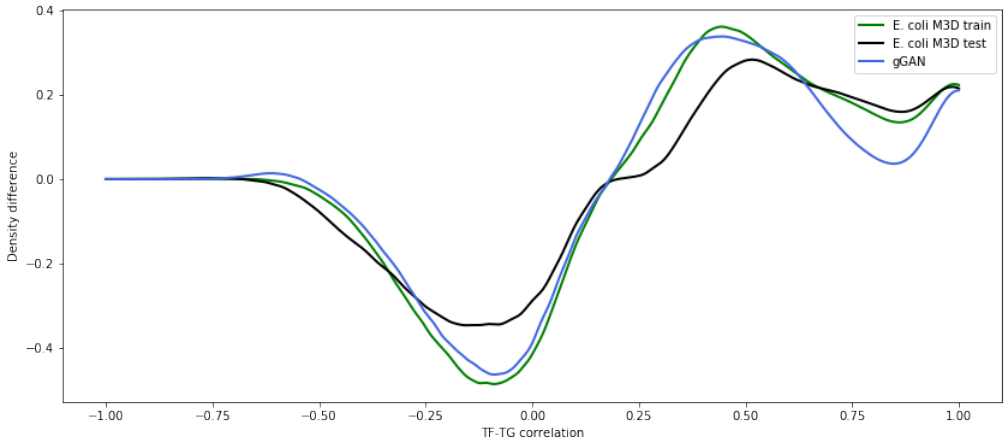


Figure 4.7: Histogram of TF-TG interactions. It shows to what extent TF-TG pairs are enriched (> 0) or depleted (< 0) with respect to the background distribution.

4.5.1.5 TG-TG histogram

Figure 4.8 depicts the density difference between the distribution of TG-TG interactions and the background distribution (section B.1.1.3). To construct the former distribution, we compute the Pearson's correlation coefficient between TGs regulated by the same TF.

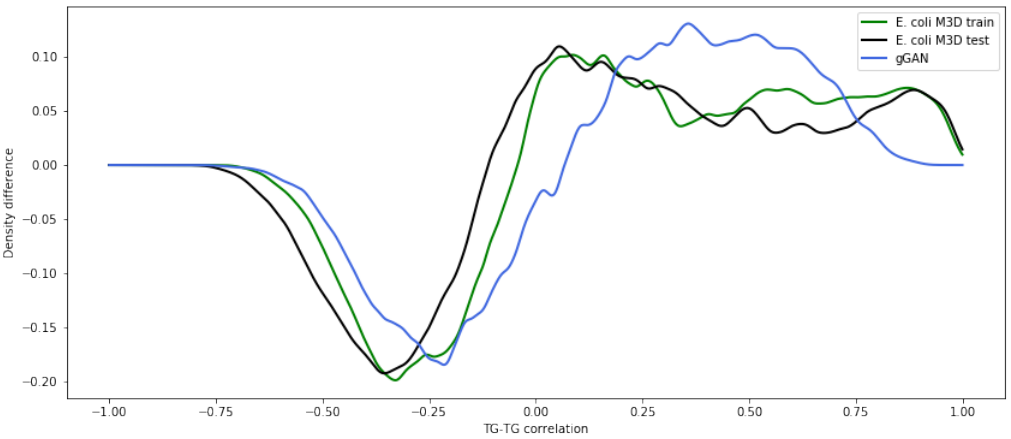


Figure 4.8: Histogram of TG-TG interactions. It shows to what extent TG-TG pairs are enriched (> 0) or depleted (< 0) with respect to the background distribution.

4.5.1.6 TF activity

Figure 4.9 illustrates the histograms of the TF activity (see section 3.1.2.7). These histograms are formed by computing the fraction of samples in which TF targets exhibit rank differences with respect to other non TF targets, according to a two-sided Mann-Whitney rank test. These tests are corrected with the Benjamini-Hochberg's procedure in order to account for multiple testing and reduce the false discovery rate.

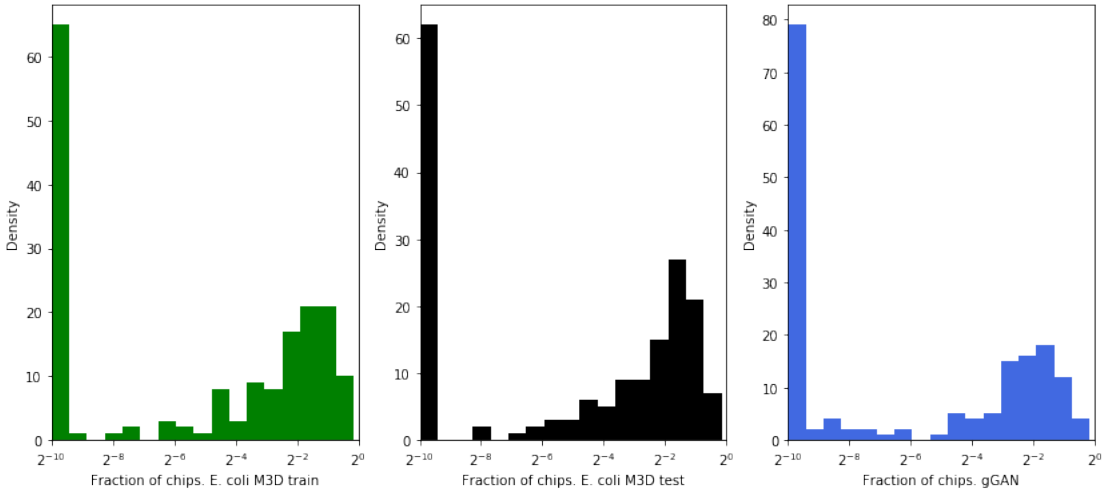


Figure 4.9: Histograms of the TF activity.

4.5.1.7 Quantitative comparison

We show our evaluation scores (section 3.1.4) in table 4.1. We compare the performance of gGAN with the approximate lower and upper bounds given by the random simulator (appendix B) and the samples from the *E. coli* M^{3D} train set, respectively.

Table 4.1: Quantitative assessment of the generated data. Let $\mathbf{X}$ be the test set, and $\mathbf{Z}$ the matrix of observations sampled from any given simulator. [**a**] $S_{dist} = \gamma(\mathbf{D}^X, \mathbf{D}^Z)$. [**b**] $S_{dend} = \gamma(\mathbf{T}^X, \mathbf{T}^Z)$. [**c**] $S_{sdcc} = (\gamma(\mathbf{D}^X, \mathbf{T}^X) - \gamma(\mathbf{D}^Z, \mathbf{T}^Z))^2$. [**d**] $S_{tftg} = \psi(\mathbf{D}^X, \mathbf{D}^Z)$. [**e**] $S_{tgtg} = \phi(\mathbf{D}^X, \mathbf{D}^Z)$. [**f**] $S_{tfac} = \omega(\mathbf{X}, \mathbf{Z})$. To form the dendrograms, we use agglomerative hierarchical clustering (section 3.1.1.1) with Pearson distance (equation 3.1) and complete linkage (equation 3.4). For the three latter scores, the importance of each TF is set to be proportional to its number of TGs.

Simulator	S_{dist} [**a**]	S_{dend} [**b**]	S_{sdcc} [**c**]	S_{tftg} [**d**]	S_{tgtg} [**e**]	S_{tfac} [**f**]
Random	0.0002	0.0003	0.2426	0.2711	-0.0026	0.0613
gGAN	0.8021	0.2986	0.0002	0.8179	0.8621	0.9301
Real	0.9252	0.3565	0.0008	0.9228	0.9515	0.9799

4.5.1.8 Overview of gene distributions

Figure 4.10 shows an overview of the gene distributions of data produced by gGAN.

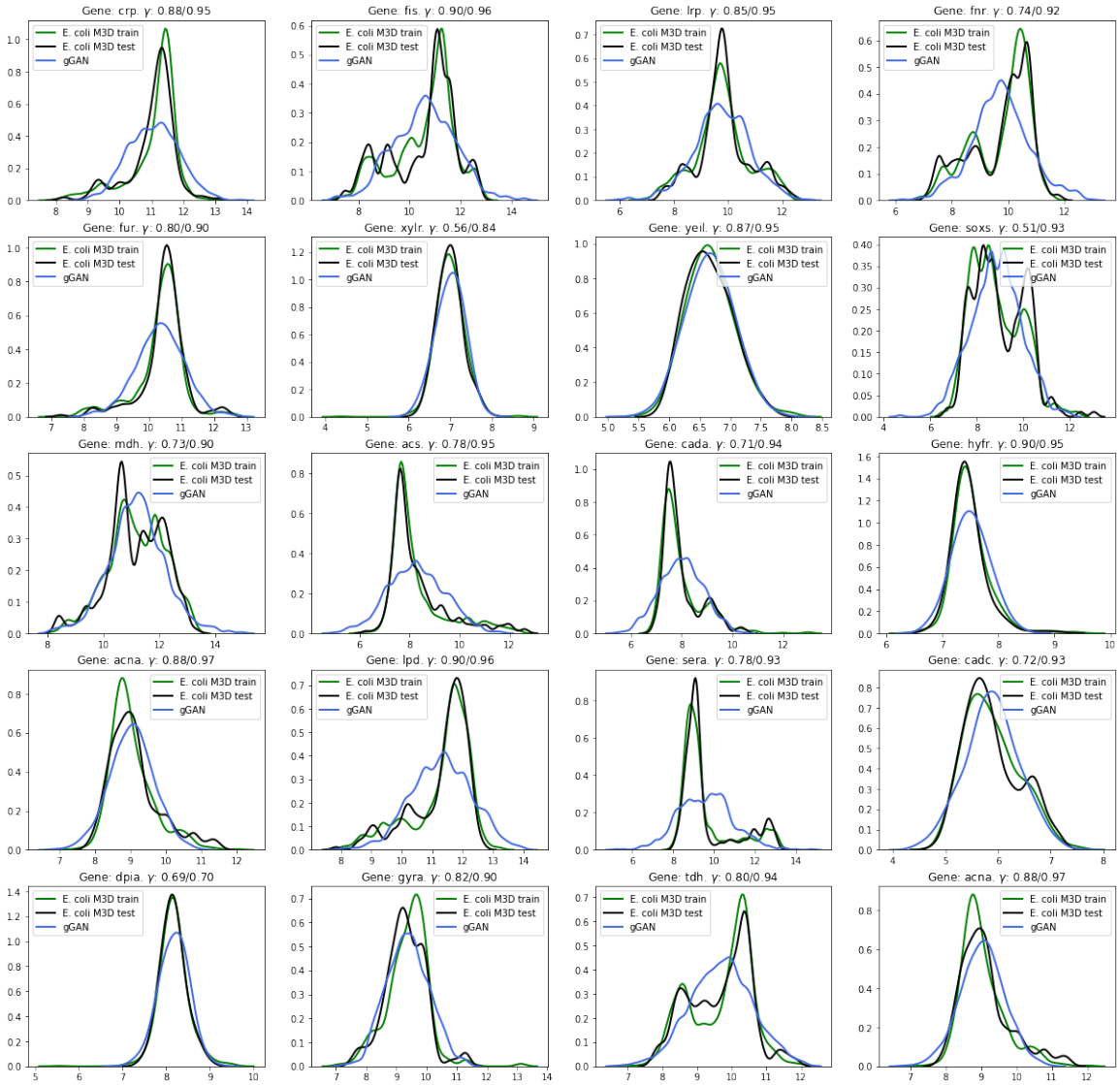


Figure 4.10: Overview of the gene distributions from a dataset (680 samples) produced by gGAN (x-axis: log expression. y-axis: density). The first row corresponds to the univariate distributions of master regulator genes (Gama-Castro et al., 2016). Genes in the next rows are directly regulated by each master regulator in the given column (among others), and are chosen randomly among the 4297 genes. Let $\mathbf{D}^X$ and $\mathbf{D}^Z$ be two 4297×4297 matrices corresponding to the gene distance matrices (as defined in section 3.1.4.1) of the *E. coli* M^{3D} test set and the gGAN dataset, respectively. Then, coefficient γ measures the Pearson's correlation between $\mathbf{D}^X_{i,:}$ and $\mathbf{D}^Z_{i,:}$ for any given gene i. The second value indicates the approximate upper bound for γ, obtained when $\mathbf{D}^Z$ is set to be the distance matrix from the *E. coli* M^{3D} train set. Appendix C describes an unfruitful approach to make these histograms match.

4.5.1.9 Discussion

So far we have shown several results that characterize some properties of the data generated by gGAN. These results do not guarantee that the data is realistic, but they constitute an exhaustive series of checks that every realistic dataset should necessarily pass. Here we review the performance of gGAN on these checks, and we analyze some properties of this simulator with regards to them. Note that we exclude from our analysis the histograms of replicate noise (as we do not generate replicates) and Silhouette coefficients (since, as discussed in section 3.1.3, this histogram is very sensitive to symmetries, and the metric S_{dend} measures this property more accurately).

The gene intensities and gene ranges distributions illustrated in figures 4.4 and 4.5 show that gGAN is able to closely match the overall gene expression values and ranges of the real *E. coli* M^{3D} data. In parallel, in figure 4.10 we observe that the true mean and standard deviation of each gene are preserved. This is mostly due to the rescaling procedure (section 4.1.3) applied at the output of the generator, as the GAN is not directly trained to match the univariate gene distributions (which are specific to the M^{3D} dataset, but not necessarily to the *E. coli* real distribution of gene expressions). Concretely, the generator's loss penalizes a single sample for not being realistic rather than punishing a batch of samples for not matching the M^{3D} univariate distributions. In other words, the GAN is specifically optimized to preserve the way in which genes interact with each other. Additionally, the fact that the real *E. coli* univariate distributions are generally more peaked than those of gGAN explains the small discrepancy between the overall gene intensities around the mean.

The background distribution depicted in figure 4.6 reflects the overall histogram of correlation coefficients among genes. This distribution shows that the majority of genes are uncorrelated or slightly correlated, whereas values around the tails correspond to pairs of genes that exhibit a strong negative (left tail) or positive (right tail) correlation. By increasing the amount of noise being added to the generated data, the correlation among pairs of genes is reduced and thus the background distribution becomes more peaked. For the gGAN simulator, in order to closely match the background distribution of the *E. coli* M^{3D} train set, we tuned hyperparameter ξ (see section 4.3.4) to regulate the amount of noise being added to the generated data.

Note that, in the background distribution, a high correlation between two genes does not necessarily imply a TF-TG regulatory interaction between them, as they could both be regulated by a common ancestor. Instead, we summarize to what extent TF-TG pairs are enriched or depleted in figure 4.7. A positive or a negative difference indicates that the probability of detecting a true TF-TG interaction in the

given interval is increased or decreased, respectively, with respect to the background distribution. Similarly, figure 4.8 summarizes the same property for pairs of TGs regulated by the same TF. In both cases, gGAN is able to match these density differences (using the synthetic background distribution as reference) reasonably well.

In figure 4.9 we examine whether the overall activity of TFs in the samples generated by gGAN resembles the levels of TF activity in the M^{3D} data, according to the statistical tests described in section 3.1.2.7. Since the activity of a TF depends on the circumstances under which each sample was measured, this property is specific to the M^{3D} dataset and thus it does not necessarily represent the true distribution p_{data}. Nonetheless, we observe a common trend regarding the TF activity between the M^{3D} and the gGAN datasets: a large amount of TFs are found inactive in every chip, whereas a considerable amount of TFs are active in a significant portion of chips.

Finally, our most meaningful results are shown in table 4.1. These measures provide a quantitative summary on several gene expression properties along with an approximate upper bound on them. First, S_{dist} indicates to which extent the pairwise gene distances are preserved in the data generated by gGAN. In figure 4.10 we show a finer-grained version of this coefficient for each gene. S_{dend} measures whether the gene clusters formed in the gGAN data are similar to those from the real data. S_{sdcc} shows the squared difference between the cophenetic coefficients of the *E. coli* M^{3D} and gGAN expression matrices (lower is better). Lastly, S_{tftg}, S_{tgtg} and S_{tfac} summarize whether the RegulonDB (Gama-Castro et al., 2016) *E. coli* TF-TG, TG-TG and TF activity correlations are preserved in the gGAN dataset, respectively.

4.5.2 Comparison with other methods

Here we compare our approach with other existing methods (SynTReN, GNW). SynTReN (section 2.1) and GNW (section 2.2) are widely used methods for generating synthetic gene expression data, and they both use a GRN as input, including the regulatory effect (activation, inhibition, dual or unknown) of each interaction. In particular, SynTReN does not support input networks with loops. This is an important downside, because GRNs are hardly ever directed acyclic graphs.

To overcome this issue, one could potentially use the full set of *E. coli* M^{3D} genes and break the loops in a random manner. However, this approach would come at the risk of ignoring some critical regulatory interactions. Instead, in order to provide a fairer comparison, we opt for testing the performance of these methods on the CRP hierarchy (see section 4.1.3.1). In this way, for SynTReN, outgoing edges from genes

in top levels of the hierarchy have priority over edges coming from bottom levels. We do not remove any loop for GNW (i.e. all edges from the subset are kept).

We generate a gene expression dataset of 680 samples both for SynTReN and GNW. For both methods, we create a network with 1076 nodes (and without background nodes), and we connect them according to the forementioned CRP hierarchy. For SynTReN, we use the default parameters (probability for complex 2-regulator interactions: 0.3; biological noise: 0.1 out of 1; experimental noise: 0.1 out of 1; noise on correlated inputs: 0.1 out of 1) and CRP as the only external node. For GNW, we choose to model gene interactions with stochastic differential equations (coefficient of noise term: 0.05), and we produce multifactorial experiments using the default settings for the DREAM4 network inference challenge[3].

SynTReN and GNW produce normalized log expression values ranging from 0 to 1. Neither SynTReN nor GNW specify how to rescale the resulting data. The analysis conducted by Maier et al. (2013) does not indicate how the data resulting from these simulators is treated either, although they suggest multiplying the artificial ranges by a factor to make the medians of the real and artificial gene range histograms match. In both cases, we opt for using the rescaling procedure described in section 4.1.3.2 (equations 4.4 and 4.5) to adjust the mean and the standard deviation of each gene. In addition, we also clip a few outliers to ensure that all the expression values lie between the overall minimum and maximum expression values from the training set.

4.5.2.1 Gene intensities

Figure 4.11 depicts the overall histograms of gene intensities.

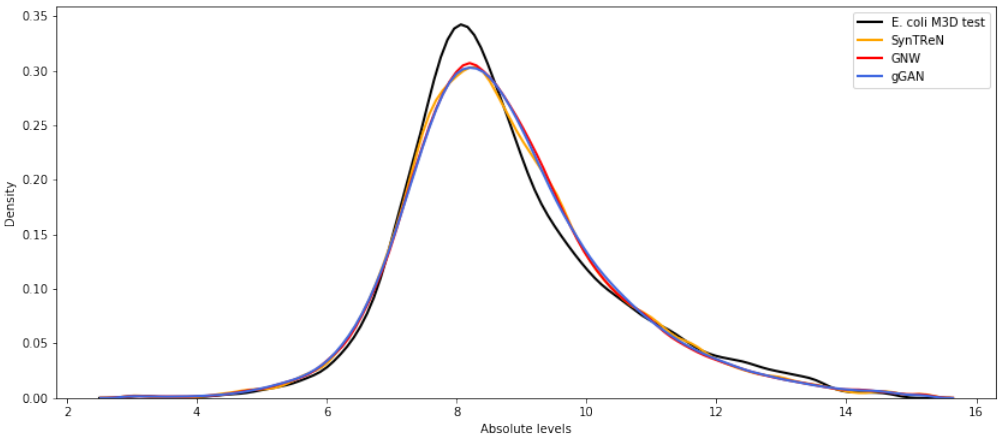


Figure 4.11: Distribution of gene intensities.

[3] `http://gnw.sourceforge.net/dreamchallenge.html`

4.5.2.2 Gene ranges

Figure 4.12 shows the histogram of gene ranges. To form this histogram, we compute the differences between the minimum expression (5% intensity quantile) and the maximum expression (95% intensity quantile) for each gene.

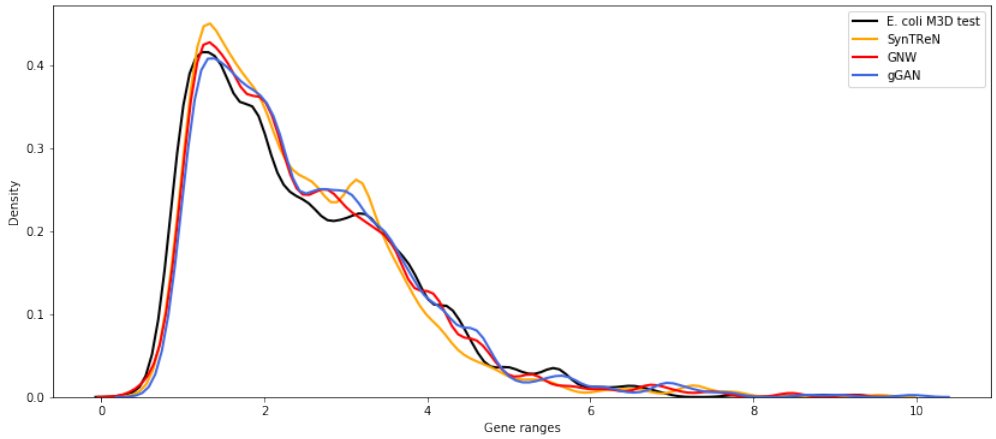


Figure 4.12: Range of gene expressions.

4.5.2.3 Background distribution of correlation coefficients

Figure 4.13 illustrates the background distribution of correlation coefficients between all pairs of genes. A higher amount of noise results in a more peaked distribution.

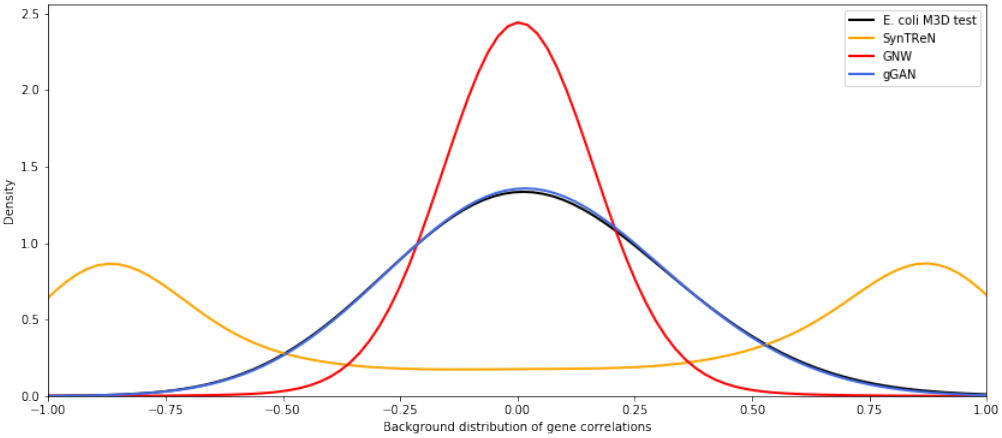


Figure 4.13: Background distribution of the Pearson's correlation coefficients between all pairs of genes.

4.5.2.4 TF-TG histogram

Figure 4.14 shows the density difference between the distribution of TF-TG interactions and the background distribution (section B.1.1.3). To construct the former distribution, we compute the Pearson's correlation between each TF and its TGs.

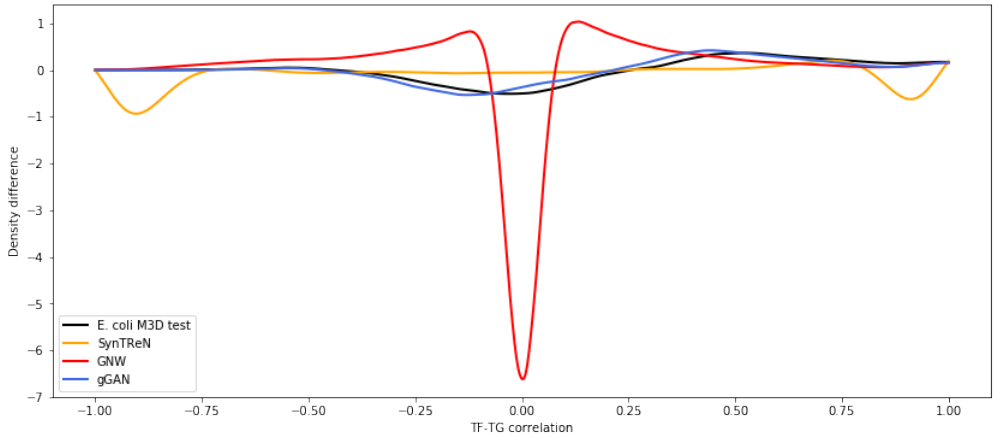


Figure 4.14: Histogram of TF-TG interactions. It shows to what extent highly correlated TF-TG pairs are enriched (> 0) or depleted (< 0).

4.5.2.5 TG-TG histogram

Figure 4.15 depicts the density difference between the distribution of TG-TG interactions and the background distribution (section B.1.1.3). To construct the former distribution, we compute the Pearson's correlation coefficient between TGs regulated by the same TF.

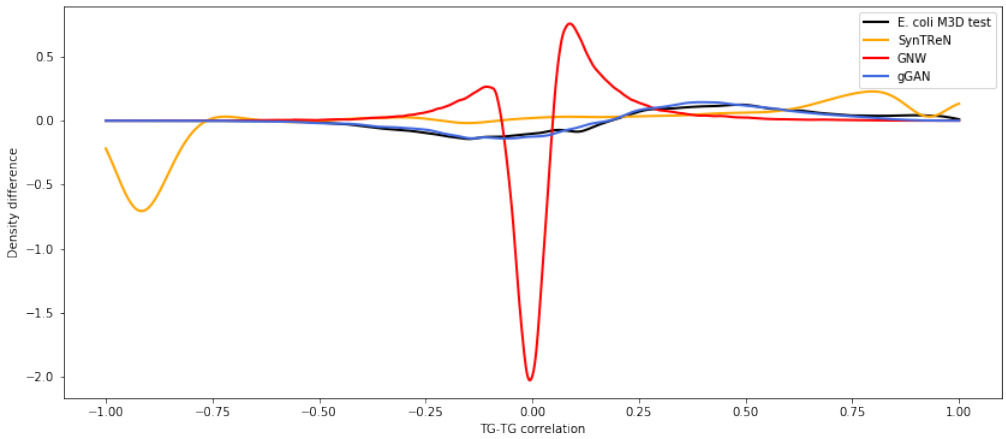


Figure 4.15: Histogram of TG-TG interactions. It shows to what extent highly correlated TG-TG pairs are enriched (> 0) or depleted (< 0).

4.5.2.6 TF activity

Figure 4.16 illustrates the histograms of the TF activity. These histograms are formed by computing the fraction of samples in which TF targets exhibit rank differences with respect to other non TF targets, according to a two-sided Mann-Whitney rank test. These tests are corrected with the Benjamini-Hochberg's procedure in order to account for multiple testing and reduce the false discovery rate.

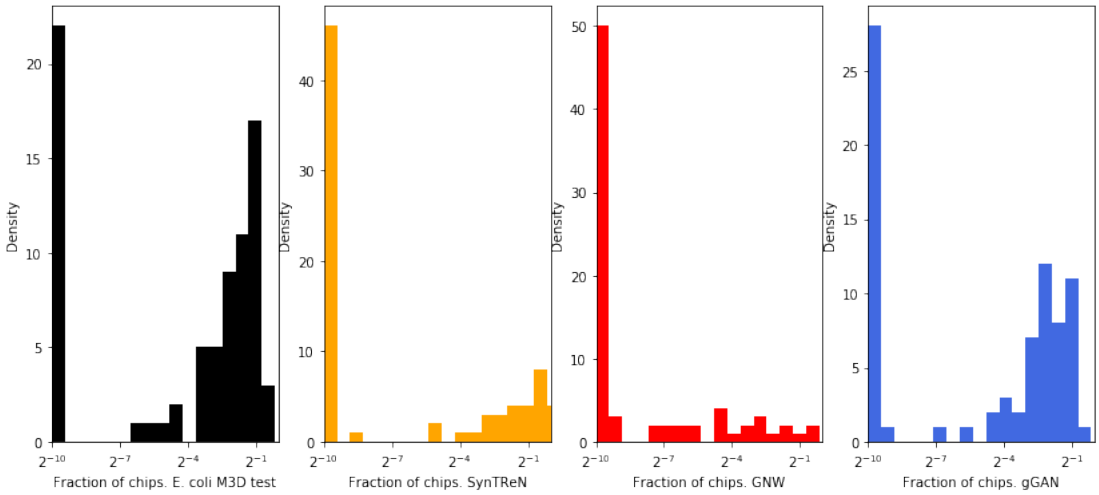


Figure 4.16: Histograms of the TF activity.

4.5.2.7 Quantitative comparison

We show our evaluation scores (see section 3.1.4) in table 4.2.

Table 4.2: Quantitative assessment of the generated data. We include the results for a *random* (appendix B) and a *real* (M^{3D} train) simulators. Let $\mathbf{X}$ be the test set, and $\mathbf{Z}$ the matrix of observations sampled from any given simulator. $\mathbf{gGAN_1}$ is trained on all genes (section 4.5.1). $\mathbf{gGAN_2}$ is trained on the CRP hierarchy. [**a**] $S_{dist} = \gamma(\mathbf{D}^X, \mathbf{D}^Z)$. [**b**] $S_{dend} = \gamma(\mathbf{T}^X, \mathbf{T}^Z)$. [**c**] $S_{sdcc} = (\gamma(\mathbf{D}^X, \mathbf{T}^X) - \gamma(\mathbf{D}^Z, \mathbf{T}^Z))^2$. [**d**] $S_{tftg} = \psi(\mathbf{D}^X, \mathbf{D}^Z)$. [**e**] $S_{tgtg} = \phi(\mathbf{D}^X, \mathbf{D}^Z)$. [**f**] $S_{tfac} = \omega(\mathbf{X}, \mathbf{Z})$. For the three latter measures, the importance of each TF is proportional to its number of TGs.

Simulator	S_{dist} [**a**]	S_{dend} [**b**]	S_{sdcc} [**c**]	S_{tftg} [**d**]	S_{tgtg} [**e**]	S_{tfac} [**f**]
Random	0.0000	-0.0002	0.2363	0.2299	-0.0132	0.0989
SynTReN	0.0234	0.0294	0.1436	0.1915	0.2296	0.6808
GNW	0.0521	0.0386	0.0599	0.1560	0.1775	0.3008
$\mathbf{gGAN_1}$	0.7592	0.4019	0.0014	0.8073	0.8528	0.9175
$\mathbf{gGAN_2}$	0.8206	0.4020	0.0020	0.8599	0.8829	0.9015
Real	0.9109	0.5197	0.0002	0.9143	0.9467	0.9715

4.5.2.8 Discussion

Here we discuss the quality of the data generated by SynTReN, GNW and gGAN for the CRP hierarchy. To provide approximate lower and upper bounds to our evaluation metrics, we also analyze the results for a random simulator (see appendix B) and the simulator based on the real samples from the *E. coli* M^{3D} train set, respectively.

In the GNW analysis by Maier et al. (2013), the absolute gene expression levels of *E. coli* and their gene ranges do not match. We attribute this to the fact that the GNW data that they generate is not properly rescaled a posteriori. Maier et al. (2013) suggest multiplying the artificial ranges by a factor to make the median of the real and artificial gene range histograms match. Here, in contrast, we show in figures 4.11 and 4.12 that the rescaling procedure described in section 4.1.3 allows to accurately match the overall gene expression values and ranges for all three methods. In addition, this procedure does not significantly compromise the performance of the simulators on the upcoming evaluation checks, as they are all based on correlation coefficients and thus they are not sensitive to the gene scales.

We find one of the major differences between the data produced by the three simulators in figure 4.13. The background distributions of the Pearson's correlation coefficients are considerably different. On the one hand, the distribution for SynTReN is bimodal, and genes are either highly negatively correlated or highly positively correlated among each other. On the other hand, the distribution for GNW is significantly more peaked than the real distribution. This could be due to an exaggerated amount of noise being added to the generated data, which reduces the absolute correlation among pairs of genes. Lastly, a proper value for hyperparameter ξ (see section 4.3.4) helps gGAN match the real background distribution accurately. These results notoriously compromise the TF-TG and TG-TG histograms (figures 4.14 and 4.15), from which we can not draw any conclusions.

In figure 4.16 we observe that, for all three simulators, a large amount of TFs are found inactive in every chip. However, for GNW and SynTReN the density of TFs that are active in a significant portion of chips is considerably smaller than for the *E. coli* M^{3D} test set. Even though the TF activity histogram is specific to the M^{3D} dataset (and not necessarily to the real *E. coli* distribution), we find these densities rather small. In contrast, the overall TF activity of M^{3D} in these regions is better preserved in the dataset generated by gGAN.

Table 4.2 shows a quantitative comparison of the three methods, including the lower and upper bounds given by the random and real simulators, respectively. We

observe that gGAN closely approximates the upper bound in every metric, outperforming SynTReN and GNW by a large margin. In fact, SynTReN and GNW are not much better than the random simulator in terms of the realism of the generated data. We attribute this mainly to the fact that SynTReN and GNW rely exclusively on the source gene regulatory network to produce synthetic data. In addition, this shows that the linear system of ODEs/SDEs defined via Michaelis-Menten and Hill equations are not enough for modelling gene dependencies in spite of being theoretically well-grounded.

In contrast, gGAN leverages real expression data to build the generative model. This model is specifically optimized in an unsupervised manner to preserve the way in which genes interact with each other. More specifically, by playing a two-player game, the discriminator acts as a teacher that instructs the generator on how to optimize its parameters in order to produce realistic gene expression data. Furthermore, our results show that the GAN trained on the full set of genes (gGAN_1; section 4.5.1) is also capable of generating realistic expression data for a subnetwork of genes.

On balance, we observe that current simulators of synthetic gene expression (SynTReN, GNW) have a low degree of realism, as pointed out by Maier et al. (2013). Our results introduce additional quantitative measures that further confirm this finding. More importantly, our results show that gGAN is, to our best knowledge, the first gene expression simulator that passes the Turing test for gene expression data employed by Maier et al. (2013). Moreover, the realism scores of this simulator closely approximate the upper bound given by the real data distribution.

Chapter 5

Conclusions

5.1 Summary of thesis achievements

In this thesis we have studied the problem of generating realistic *E. coli* gene expression data. We have divided this problem into two main tasks: assessing the realism of a dataset, and building a generative model for producing gene expression data.

For the first task, we have developed our own novel evaluation scores. These metrics allow to accurately quantify the discrepancies of several statistical properties between the synthetic and real data distributions. Moreover, to provide a qualitative analysis of the simulated expression data, the histograms proposed by Maier et al. (2013) have allowed us to visualize and interpret several meaningful properties.

As a result of our analysis, we have shown that existing simulators fail to emulate key properties of gene expression data. In particular, one of the most surprising results is that SynTReN and GNW poorly preserve the properties derived from gene regulatory networks such as the TF-TG interactions. This is undesirable, as these simulators are specifically designed for the purpose of benchmarking network inference algorithms, and the goal of the linear systems of ODEs/SDEs is precisely to simulate relationships between TFs and TGs.

For the second task, we have implemented a simulator based on a generative adversarial network (Goodfellow et al., 2014). To our best knowledge, GANs have not previously been applied to build a simulator of expression data. To enable learning from a high-dimensional dataset with a scarce number of samples and keep the complexity of our model under control, our training algorithm incorporates several regularization mechanisms such as dropout and one-sided label smoothing.

One of the most characteristic components of our GAN is a layer that adds white Gaussian noise with learnable variances. This noise model seeks to imitate the effect of random processes that occur in nature. By introducing an hyperparameter that

controls the amount of noise being added by this layer, we have managed to match the real background distribution of gene correlations. We have also shown that an appropriate amount of noise has a positive impact on our realism scores.

After carefully optimizing our adversarial model, the resulting simulator generates expression data with a high degree of realism according to our evaluation metrics. We have shown that our approach outperforms existing simulators by a large margin in terms of the realism of the generated data, and the quality scores of our simulator are reasonably close to the upper bound given by the real data. More importantly, our results show that gGAN is, to our best knowledge, the first gene expression simulator that passes the Turing test for gene expression data employed by Maier et al. (2013).

5.2 Future work

There are other ideas and experiments that are worth researching further.

First, our current approach exclusively leverages gene expression data to build the generative model. However, other important layers of regulation take part in the process of gene expression. For example, other biological events such as protein-protein interactions or cellular states contribute to this process. From our point of view, it would be helpful to integrate features about these processes into our model when possible. Currently, our intuition is that some of these biological states are being described by the latent space of the generator.

Second, it would be worth exploring other noise models. In this study, in addition to the noise being modelled by the actual generator, we integrated an additive white Gaussian noise model with learnable variances, with the aim of mimicking random fluctuations of biological processes. We believe that this noise model might be over-simplistic, and our GAN could potentially benefit from other theoretically grounded approaches. For example, Raser and O'shea (2005) suggest that heavier-tailed noise is probably more appropriate.

Finally, it would be interesting to investigate whether our approach can be extended to other organisms. The GAN presented in this study is specifically devised for generating *E. coli* expression data. This bacterium has a small genome size (∼4,400 genes) and its expression mechanisms are relatively well understood. However, other organisms are known to exhibit more complex expression patterns. In addition, information about the transcriptional regulatory mechanisms in other organisms might be more scarce or nonexistent. This presents a major challenge, as the evaluation metrics from this study assume knowledge about the organism's regulatory interactions.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein GAN. *ArXiv e-prints*.

Baldi, P. and Sadowski, P. J. (2013). Understanding dropout. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 2814–2822. Curran Associates, Inc.

Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Cooper, G. (2000). *Cells As Experimental Models. 2nd edition*. Sunderland (MA): Sinauer Associates.

D'haeseleer, P. (2005). How does gene expression cluster work? 23:1499–501.

Eisen, M. B., Spellman, P. T., Brown, P. O., and Botstein, D. (1998). Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868.

Faith, J. J., Driscoll, M. E., Fusaro, V. A., Cosgrove, E. J., Hayete, B., Juhn, F. S., Schneider, S. J., and Gardner, T. S. (2008). Many microbe microarrays database: uniformly normalized affymetrix compendia with structured experimental metadata. *Nucleic Acids Research*, 36(suppl1):D866–D870.

Gama-Castro, S., Salgado, H., Santos-Zavaleta, A., Ledezma-Tejeida, D., Muñiz-Rascado, L., García-Sotelo, J. S., Alquicira-Hernández, K., Martínez-Flores, I., Pannier, L., Castro-Mondragón, J. A., Medina-Rivera, A., Solano-Lira, H., Bonavides-Martínez, C., Pérez-Rueda, E., Alquicira-Hernández, S., Porrón-Sotelo, L., López-Fuentes, A., Hernández-Koutoucheva, A., Moral-Chávez, V. D., Rinaldi, F., and Collado-Vides, J. (2016). RegulonDB version 9.0: high-level integration of gene regulation, coexpression, motif clustering and beyond. *Nucleic Acids Research*, 44(D1):D133–D143.

Gillespie, D. T. (2000). The chemical Langevin equation. *The Journal of Chemical Physics*, 113(1):297–306.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

Goodfellow, I. J. (2017). NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160.

Grainger, D. C. and Busby, S. J. (2008). Chapter 4 - global regulators of transcription in escherichia coli: Mechanisms of action and methods for study. volume 65 of *Advances in Applied Microbiology*, pages 93 – 113. Academic Press.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Irizarry, R. (2003). Summaries of affymetrix genechip probe level data. 31:15e–15.

Jones, E., Oliphant, T., Peterson, P., et al. (2001). SciPy: Open source scientific tools for Python.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides, F. and Schmidt, B., editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press.

Maier, R., Zimmer, R., and Küffner, R. (2013). A Turing test for artificial expression data. *Bioinformatics*, 29(20):2603–2609.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60.

Marbach, D., Schaffter, T., Mattiussi, C., and Floreano, D. (2009). Generating realistic in silico gene networks for performance assessment of reverse engineering methods. *Journal of Computational Biology*, 16(2):229–239. PMID: 19183003.

Margolin, A. A., Nemenman, I., Basso, K., Wiggins, C., Stolovitzky, G., Favera, R. D., and Califano, A. (2006). Aracne: An algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC Bioinformatics*, 7(1):S7.

Martinez-Antonio, A., Janga, S. C., and Thieffry, D. (2008). Functional organization of escherichia coli transcriptional regulatory network. 381:238–47.

McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L., and Wold, B. (2008). Mapping and quantifying mammalian transcriptomes by rna-seq. *Nat Meth*, 5(7):621–628.

Oudenaarden, A. V. (2004). *Michaelis-Menten kinetics*. MIT.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434.

Raser, J. M. and O'shea, E. K. (2005). Noise in gene expression: origins, consequences, and control. *Science*, 309(5743):2010–2013.

Rossum, G. V. (1995). Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.

Salgado, H., Gama-Castro, S., Peralta-Gil, M., Díaz-Peredo, E., Sánchez-Solano, F., Santos-Zavaleta, A., Martínez-Flores, I., Jiménez-Jacinto, V., Bonavides-Martínez, C., Segura-Salazar, J., Martínez-Antonio, A., and Collado-Vides, J. (2006). RegulonDB (version 5.0): Escherichia coli k-12 transcriptional regulatory network, operon organization, and growth conditions. *Nucleic Acids Research*, 34(suppl_1):D394–D397.

Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training GANs. *CoRR*, abs/1606.03498.

Schaffter, T., Marbach, D., and Floreano, D. (2011). GeneNetWeaver: in silico benchmark generation and performance profiling of network inference methods. *Bioinformatics*, 27(16):2263–2270.

Schena, M., Shalon, D., Davis, R. W., and Brown, P. O. (1995). Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–470.

Seabold, S. and Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*.

Silverman, B. W. (2018). *Density estimation for statistics and data analysis*. Routledge.

Sokal, R. R. and Rohlf, F. J. (1962). The comparison of dendrograms by objective methods. taxon 11: 33-40. 11:33–40.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Van den Bulcke, T., Van Leemput, K., Naudts, B., van Remortel, P., Ma, H., Verschoren, A., De Moor, B., and Marchal, K. (2006). Syntren: a generator of synthetic gene expression data for design and analysis of structure learning algorithms. *BMC Bioinformatics*, 7(1):43.

Walt, S. v. d., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science and Engg.*, 13(2):22–30.

Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D. C., Augspurger, T., Halchenko, Y., Cole, J. B., Warmenhoven, J., de Ruiter, J., Pye, C., Hoyer, S., Vanderplas, J., Villalba, S., Kunter, G., Quintero, E., Bachant, P., Martin, M., Meyer, K., Miles, A., Ram, Y., Yarkoni, T., Williams, M. L., Evans, C., Fitzgerald, C., Brian, Fonnesbeck, C., Lee, A., and Qalieh, A. (2017). mwaskom/seaborn: v0.8.1 (september 2017).

Wu, Y., Burda, Y., Salakhutdinov, R., and Grosse, R. B. (2016). On the quantitative analysis of decoder-based generative models. *CoRR*, abs/1611.04273.

Yip, K. Y., Alexander, R. P., Yan, K.-K., and Gerstein, M. (2010). Improved reconstruction of in silico gene regulatory networks by integrating knockout and perturbation data. *PLOS ONE*, 5(1):1–9.

Yu, J., Smith, V. A., Wang, P. P., Hartemink, A. J., and Jarvis, E. D. (2004). Advances to Bayesian network inference for generating causal networks from observational biological data. *Bioinformatics*, 20(18):3594–3603.

Zhu, J., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593.

Appendix A

Additional figures

A.1 Graphical model of our GAN

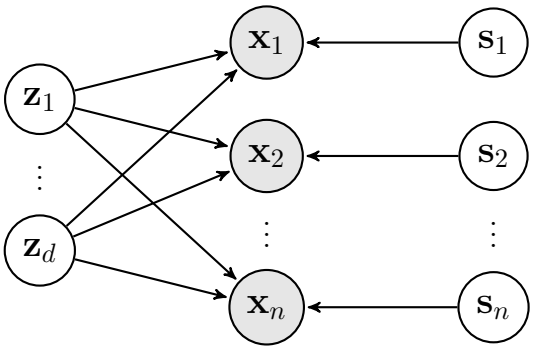


Figure A.1: Bayesian network of our GAN. Every latent variable in $\mathbf{z}$ influences every observable variable in $\mathbf{x}$, whereas every latent variable in $\mathbf{s}$ influcences exclusively its corresponding gene in $\mathbf{x}$.

A.2 Evolution of evaluation metrics along epochs

Distance between real and artificial distance matrices

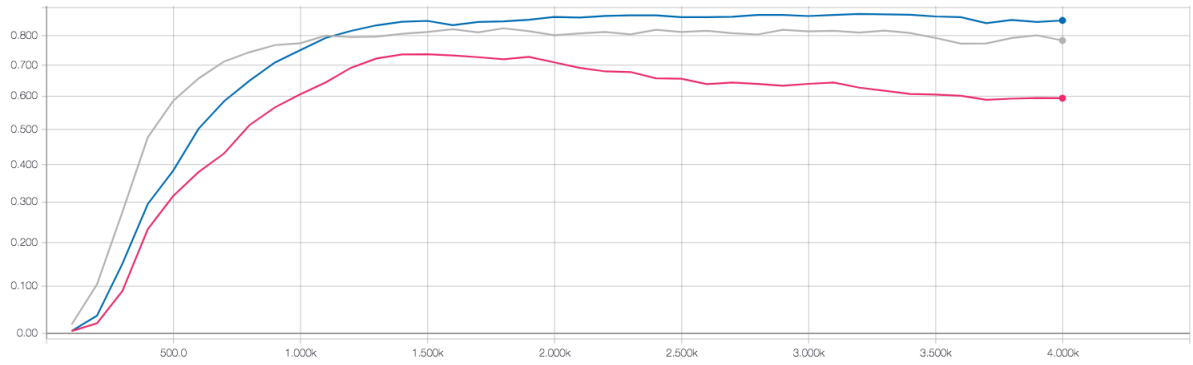


Figure A.2: Evolution of $\gamma(\mathbf{D}^X, \mathbf{D}^Z)$ for different settings of ξ for the CRP hierarchy dataset and $\kappa = 2$ (x-axis: training epochs, y-axis: $\gamma(\mathbf{D}^X, \mathbf{D}^Z)$). Let n be the number of genes. The setting of ξ for each curve is as follows. Grey: $\frac{\xi}{n} = 0$, blue: $\frac{\xi}{n} = 0.25$, pink: $\frac{\xi}{n} = 0.5$.

Cophenetic coefficient between distance and dendrogrammatic matrices

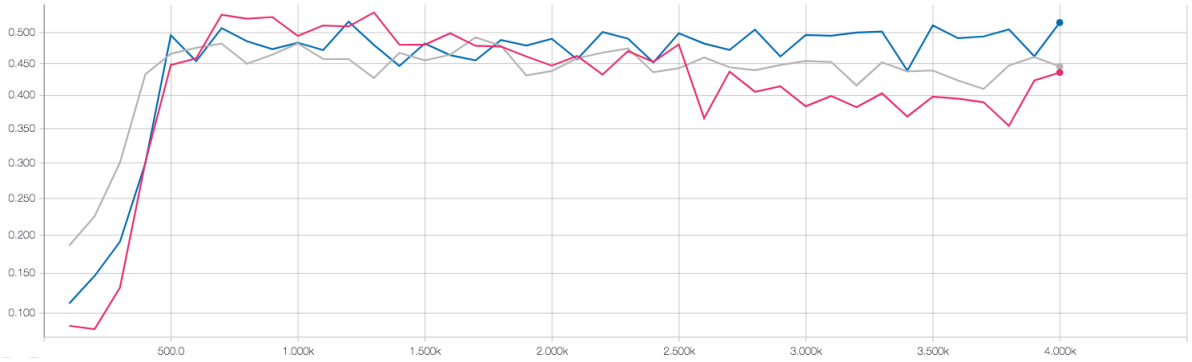


Figure A.3: Evolution of $\gamma(\mathbf{D}^Z, \mathbf{T}^Z)$ for different settings of ξ for the CRP hierarchy dataset and $\kappa = 2$ (x-axis: training epochs, y-axis: $\gamma(\mathbf{D}^Z, \mathbf{T}^Z)$). The closer to $\gamma(\mathbf{D}^X, \mathbf{T}^X) = 0.5308$, the better. Let n be the number of genes. The setting of ξ for each curve is as follows. Grey: $\frac{\xi}{n} = 0$, blue: $\frac{\xi}{n} = 0.25$, pink: $\frac{\xi}{n} = 0.5$.

Distance between real and artificial dendrograms

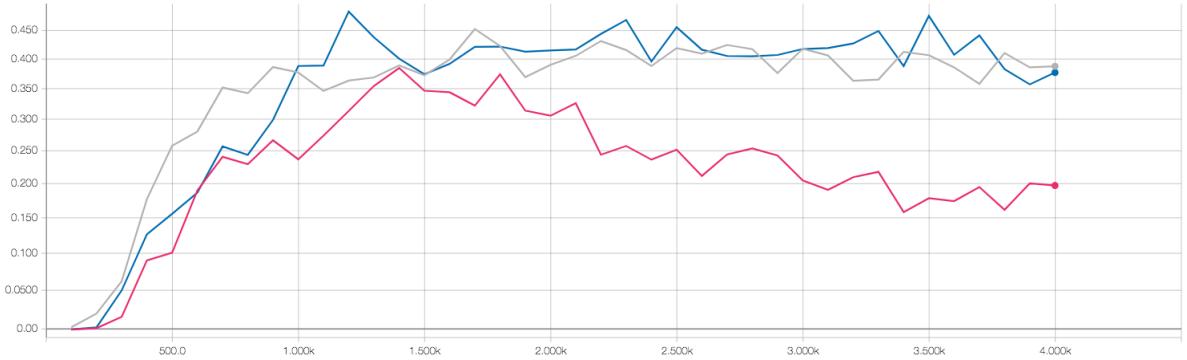


Figure A.4: Evolution of $\gamma(\mathbf{T}^X, \mathbf{T}^Z)$ for different settings of ξ for the CRP hierarchy dataset and $\kappa = 2$ (x-axis: training epochs, y-axis: $\gamma(\mathbf{T}^X, \mathbf{T}^Z)$). Let n be the number of genes. The setting of ξ for each curve is as follows. Grey: $\frac{\xi}{n} = 0$, blue: $\frac{\xi}{n} = 0.25$, pink: $\frac{\xi}{n} = 0.5$.

Weighted sum of TF-TG similarity coefficients

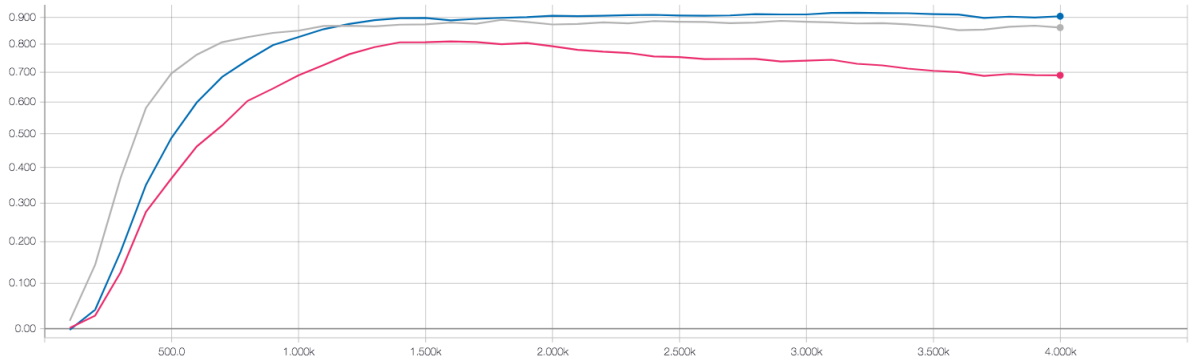


Figure A.5: Evolution of $\psi(\mathbf{D}^X, \mathbf{D}^Z)$ for different settings of ξ for the CRP hierarchy dataset and $\kappa = 2$ (x-axis: training epochs, y-axis: $\psi(\mathbf{D}^X, \mathbf{D}^Z)$). Let n be the number of genes. The setting of ξ for each curve is as follows. Grey: $\frac{\xi}{n} = 0$, blue: $\frac{\xi}{n} = 0.25$, pink: $\frac{\xi}{n} = 0.5$.

Weighted sum of TG-TG similarity coefficients

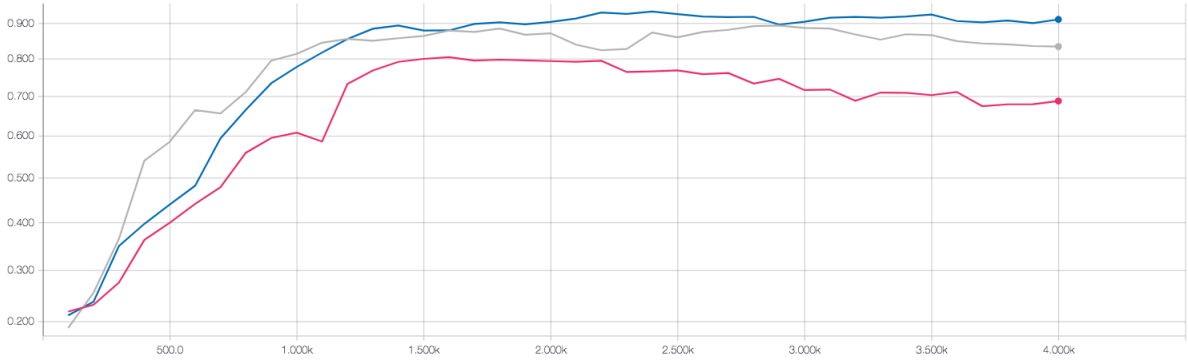


Figure A.6: Evolution of $\phi(\mathbf{D}^X, \mathbf{D}^Z)$ for different settings of ξ for the CRP hierarchy dataset and $\kappa = 2$ (x-axis: training epochs, y-axis: $\phi(\mathbf{D}^X, \mathbf{D}^Z)$). Let n be the number of genes. The setting of ξ for each curve is as follows. Grey: $\frac{\xi}{n} = 0$, blue: $\frac{\xi}{n} = 0.25$, pink: $\frac{\xi}{n} = 0.5$.

Appendix B

Random simulator

B.1 Simulator of random expression data

To find an approximate lower bound on our evaluation metrics, we define a simple random simulator of expression data and generate a $m \times n$ matrix $\mathbf{X}$ of expression values, where m is the number of samples and n the number of genes. We start by randomly sampling a $m \times n$ matrix from a uniform distribution $\mathcal{U}(0, 1)$. Then, we rescale these values with the procedure defined in section 4.1.3.2. Finally, we obtain $\mathbf{X}$ by clipping the values that not within the minimum and maximum expression values from the training set. Next, we show the Maier's histograms for this simulator.

B.1.1 Evaluating expression data from random simulator

B.1.1.1 Gene intensities

Figure 4.4 depicts the overall histograms of gene intensities.

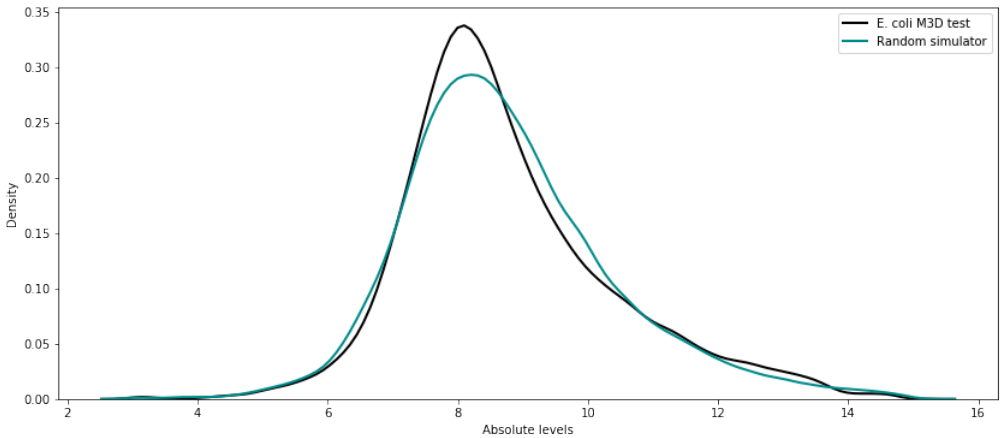


Figure B.1: Distribution of intensities for random simulator (CRP hierarchy).

B.1.1.2 Gene ranges

Figure 4.5 shows the histogram of gene ranges. To form this histogram, we compute the differences between the minimum expression (5% intensity quantile) and the maximum expression (95% intensity quantile) for each gene.

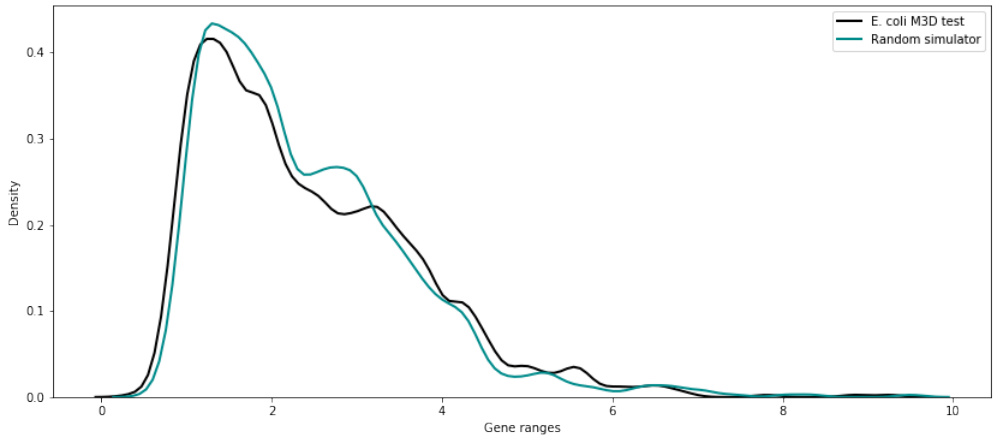


Figure B.2: Range of gene expressions for random simulator (CRP hierarchy).

B.1.1.3 Background distribution of correlation coefficients

Figure 4.6 illustrates the background distribution of correlation coefficients between all pairs of genes. A higher amount of noise results in a more peaked distribution.

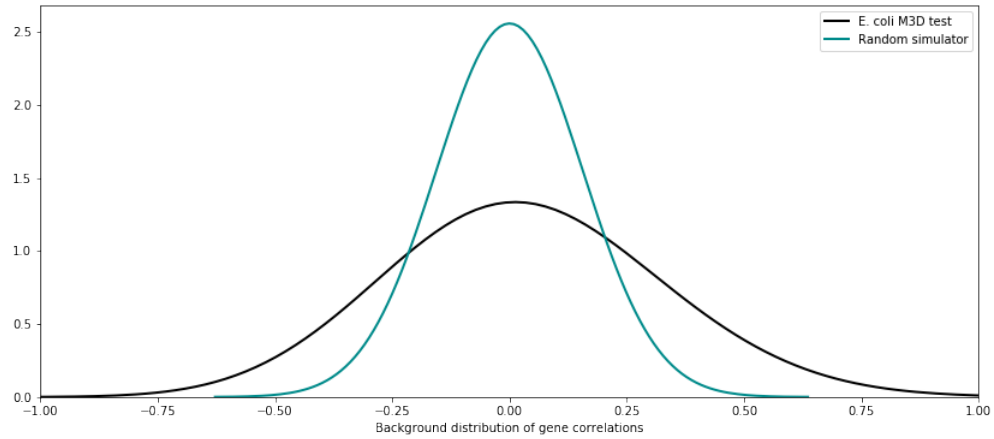


Figure B.3: Background distribution of the Pearson's correlation coefficients between all pairs of genes for random simulator (CRP hierarchy).

B.1.1.4 TF-TG interactions

Figure 4.7 shows the density difference between the distribution of TF-TG interactions and the background distribution (section B.1.1.3). To construct the former distribution, we compute the Pearson's correlation between each TF and its TGs.

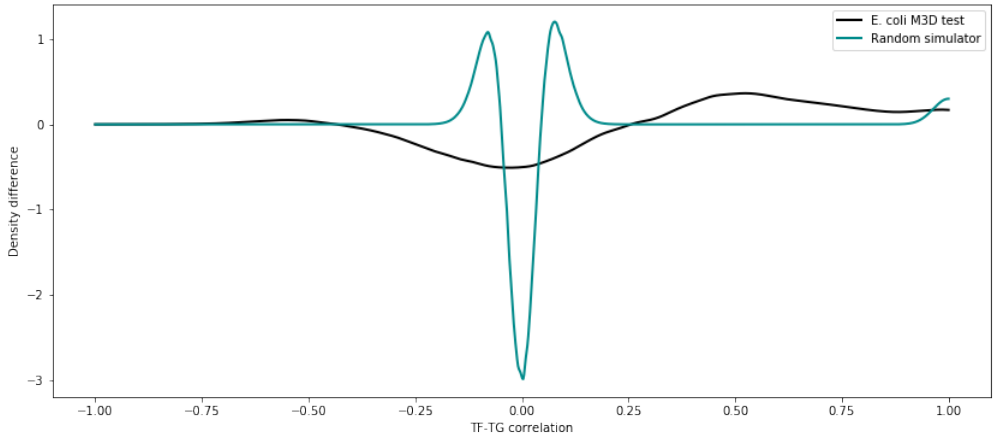


Figure B.4: Histogram of TF-TG interactions. It shows to what extent TF-TG pairs are enriched (> 0) or depleted (< 0) with respect to the background distribution.

B.1.1.5 TG-TG interactions

Figure 4.8 depicts the density difference between the distribution of TG-TG interactions and the background distribution (section B.1.1.3). To construct the former distribution, we compute the Pearson's correlation coefficient between TGs regulated by the same TF.

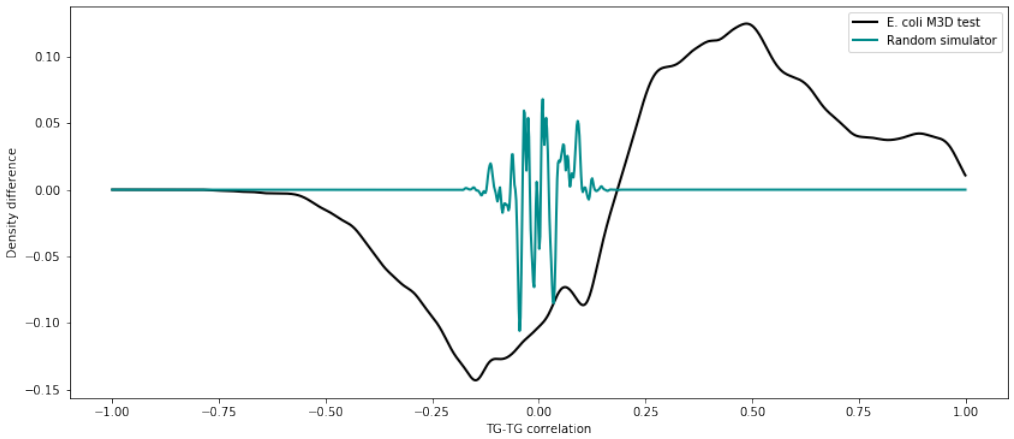


Figure B.5: Histogram of TG-TG interactions. It shows to what extent TG-TG pairs are enriched (> 0) or depleted (< 0) with respect to the background distribution.

B.1.1.6 TF activity

Figure 4.9 illustrates the histograms of the TF activity (see section 3.1.2.7). These histograms are formed by computing the fraction of samples in which TF targets exhibit rank differences with respect to other non TF targets, according to a two-sided Mann-Whitney rank test. These tests are corrected with the Benjamini-Hochberg's procedure in order to account for multiple testing and reduce the false discovery rate.

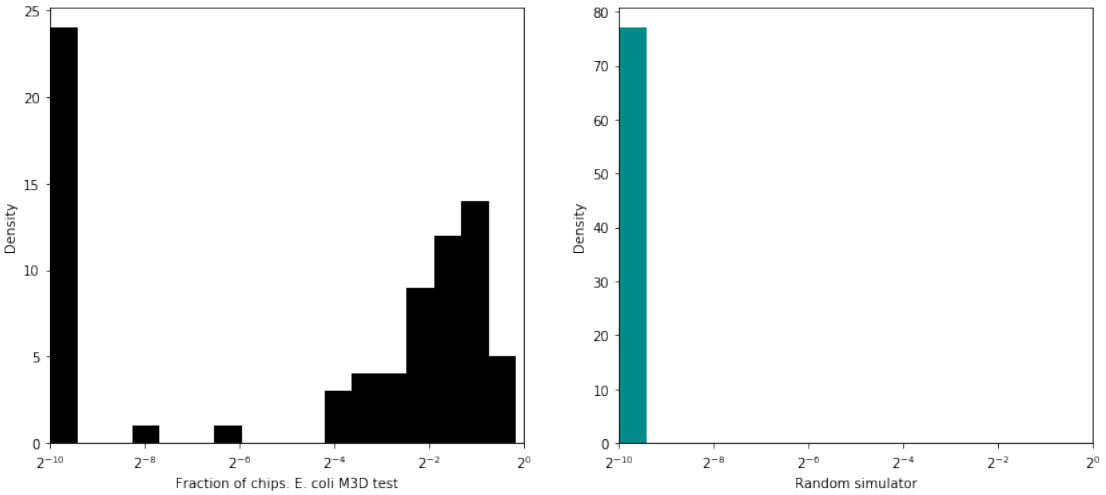


Figure B.6: Histogram of the TF activity for random simulator (CRP hierarchy).

Appendix C

Other exploratory investigations within the project

C.1 Matching the individual gene expression histograms from the *E. coli* M^{3D} dataset

We also attempt to match certain individual gene expression histograms (see section 4.5.1.8) from the *E. coli* M^{3D} dataset via rejection sampling. The idea is to check whether the individual histogram of a gene is preserved once the distributions of its TFs match between the real and the synthetic datasets. In order to do this, we leverage two methods: kernel density estimation (KDE; Silverman, 2018) and rejection sampling. Algorithm 5 summarizes our approach. This computation is tractable when the number of TFs for the target gene is small. However, our algorithm is very inefficient for a large number of TFs (performing KDE and finding an accurate rejection sampling constant is expensive, and the forementioned constant is large). Our first results are unfruitful (the individual histograms obtained for the selected genes are not significantly different from those depicted in figure 4.10), and we believe it is worth investiganting this idea further.

Algorithm 5: Matching the individual distributions of a set of genes. This algorithm performs multidimensional rejection sampling for the set of TFs that regulate the expression of a given gene. KDE is a function that performs Gaussian Kernel Density Estimation (Silverman, 2018). $\mathbf{X}$ is a $m \times n$ matrix of gene expressions sampled from the real distribution, where m is the number of samples and n the number of genes. Let $G : \mathbb{R}^d \to \mathbb{R}^n$ be our generator (we ignore the additive noise input for simplicity). Let $\mathcal{G}$ be a function that takes the index of a TF and returns the set of indices of the TGs that it regulates.

Data: We are given a gene index g for which we want to match the *E. coli* M^{3D} individual distributions of its TFs. We are also given two integers k and t corresponding to the number of synthethic samples that are used to approximate the synthetic PDF and the number of samples that we want to generate, respectively.

Find the set $\mathcal{F}$ of TFs that regulate g:
  $\mathcal{F} \leftarrow \{f \mid g \in \mathcal{G}(f)\}$
Approximate real PDF of expression data for set $\mathcal{F}$:
  $\overline{\mathbf{X}} \leftarrow (\mathbf{X}_{:,f} : f \in \mathcal{F})$
  $\hat{p}_{real} \leftarrow \text{KDE}(\overline{\mathbf{X}})$
Approximate synthetic PDF of expression data for set $\mathcal{F}$:
  Sample batch $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(k)}\}$ from p_z
  $\mathbf{Z} \leftarrow (G(\mathbf{z}^{(i)}) : i \in 1, ..., k)$
  $\overline{\mathbf{Z}} \leftarrow (\mathbf{Z}_{:,f} : f \in \mathcal{F})$
  $\hat{p}_g \leftarrow \text{KDE}(\overline{\mathbf{Z}})$
Find rejection sampling constant:
  $c \leftarrow \max_{\mathbf{x}} \frac{\hat{p}_{real}(\mathbf{x})}{\hat{p}_g(\mathbf{x})}$
Initialize counter and expressions matrix:
  $i \leftarrow 0$
  $\widehat{\mathbf{Z}} \leftarrow$ empty $t \times n$ matrix
while $i < t$ **do**
  Sample $\mathbf{z}$ from $p_{\mathbf{z}}$
  $\mathbf{x} \leftarrow G(\mathbf{z})$
  $\overline{\mathbf{x}} \leftarrow (\mathbf{x}_f : f \in \mathcal{F})^\top$
  $p \leftarrow \frac{\hat{p}_{real}(\overline{\mathbf{x}})}{c \cdot \hat{p}_g(\overline{\mathbf{x}})}$
  with *probability* p **do**
    $i \leftarrow i + 1$
    $\widehat{\mathbf{Z}}_{i,:} \leftarrow \mathbf{x}$
return $\widehat{\mathbf{Z}}$