Supervised Inference of Gene Regulatory Networks

Malabika Ashit Sen

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

T. M. Murali, Chair
Lenwood S. Heath
Chandan Reddy

August 13, 2021
Blacksburg, Virginia

Keywords: Gene Regulatory Networks, Network Inference, Link Prediction, Graph Convolutional Networks, Graph Machine Learning

Supervised Inference of Gene Regulatory Networks

Malabika Ashit Sen

(ABSTRACT)

A gene regulatory network (GRN) records the interactions among transcription factors and their target genes. GRNs are useful to study how transcription factors (TFs) control gene expression as cells transition between states during differentiation and development. Scientists usually construct GRNs by careful examination and study of the literature. This process is slow and painstaking and does not scale to large networks. In this thesis, we study the problem of inferring GRNs automatically from gene expression data. Recent data-driven approaches to infer GRNs increasingly rely on single-cell level RNA-sequencing (scRNA-seq) data. Most of these methods rely on unsupervised or association based strategies, which cannot leverage known regulatory interactions by design. To facilitate supervised learning, we propose a novel graph convolutional neural network (GCN) based autoencoder to infer new regulatory edges from a known GRN and scRNA-seq data. As the name suggests, a GCN-based autoencoder consists of an encoder that learns a low-dimensional embedding of the nodes (genes) in the input graph (the GRN) through a series of graph convolution operations and a decoder that aims to reconstruct the original graph as accurately as possible. We investigate several GCN-based architectures to determine the ideal encoder-decoder combination for GRN reconstruction. We systematically study the performance of these and other supervised learning methods on different mouse and human scRNA-seq datasets for two types of evaluation. We demonstrate that our GCN-based approach substantially outperforms traditional machine learning approaches.

Supervised Inference of Gene Regulatory Networks

Malabika Ashit Sen

(GENERAL AUDIENCE ABSTRACT)

In multi-cellular living organisms, stem cells differentiate into multiple cell types. Proteins called transcription factors (TFs) control the activity of genes to effect these transitions. It is possible to represent these interactions abstractly using a gene regulatory network (GRN). In a GRN, each node is a TF or a gene and each edge connects a TF to a gene or TF that it controls. New high-throughput technologies that can measure gene expression (activity) in individual cells provide rich data that can be used to construct GRNs. In this thesis, we take advantage of recent advances in the field of machine learning to develop a new computational method for computationally constructing GRNs. The distinguishing property of our technique is that it is supervised, i.e., it uses experimentally-known interactions to infer new regulatory connections. We investigate several variations of this approach to reconstruct a GRN as close to the original network as possible. We analyze and provide a rationale for the decisions made in designing, evaluating, and choosing the characteristics of our predictor. We show that our predictor has a reconstruction accuracy that is superior to other supervised-learning approaches.

Dedication

I dedicate this thesis to my constant source of happiness and support - my mummy Mitali Sen, my baba Ashit Sen and my better half - my sister Anamika Sen.

Acknowledgments

First and foremost, I would like to thank my advisor, Dr. T. M. Murali. My Masters journey would not be complete and fulfilling without his guidance and wise counsel. He made me a better researcher than I came here as. I would have not seen my thesis reach the end without his support: be it his prompt responses on Slack even when I posted questions at ungodly hours (Those words of encouragement worked wonders for me. Thank you!) or his critiquing the plots that I made (cue: missing axes labels or bad plot names)—this thesis would not be possible without him.

Next, I would like to thank Aditya Pratapa for letting me build on his research. I was able to make progress in a limited time only because of his early work in this topic. I would like to whole-heartedly thank Aditya for taking the time to provide me with initial setup and knowledge.

I would also like to thank the group members of Computational Systems Biology at Virginia Tech. The group meetings with them where we dissected every research paper have been a huge factor in my learning.

My two years in Blacksburg would not be possible without the support from friends around. Thank you Saurav, Kaushik and Anish.

Last but not the least, I cannot thank enough my family who have been with me in spirit if not here together. My mother, father and sister—thank you for always being there.

Contents

List of Figures

Chapter 1

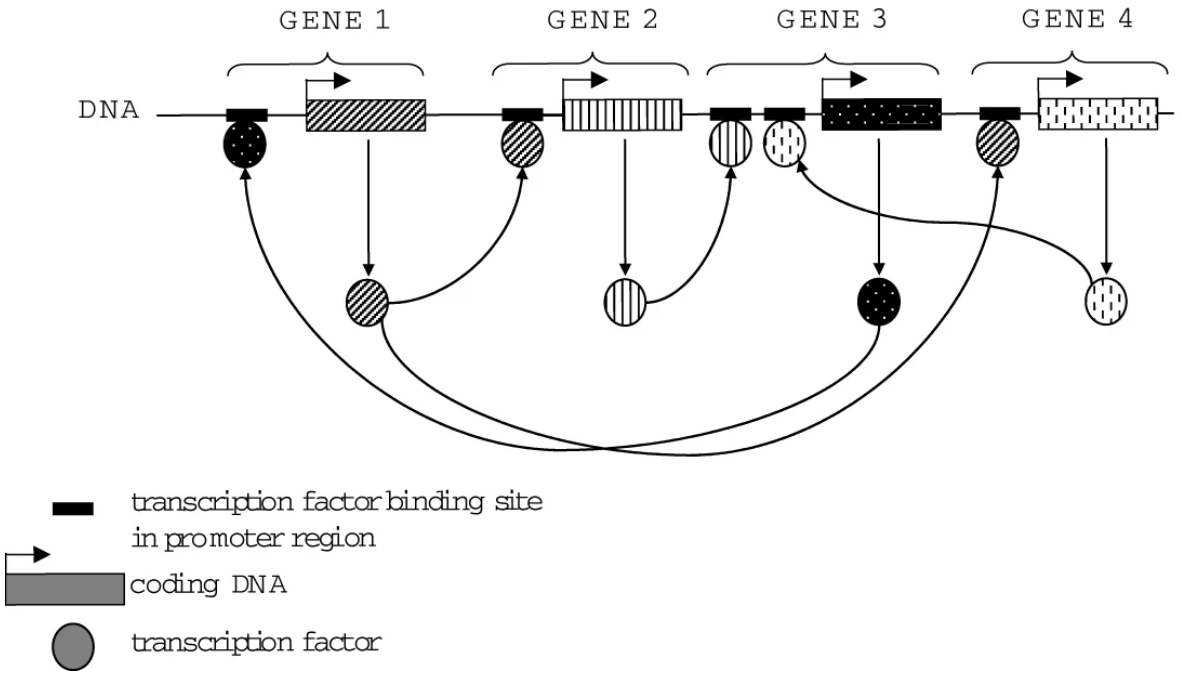
Introduction

This chapter presents the motivation for this thesis by introducing the concept of gene regulatory networks (GRNs) and describing their importance in understanding cellular growth and development. We further explain the use of studying GRNs, which will build up our motivation to infer GRNs experimentally. We describe the types of data used to infer GRNs computationally, stressing improvements in recent measurement technologies, which enable us to get rich data at a single-cell level. We then identify the drawbacks of current methods to infer GRNs. We also describe the nature of the proposed computational method which will leverage single-cell RNA sequencing data. Following this, we give a formal problem definition and conclude by stating the research questions that this thesis aims to solve.

1.1 Gene Regulatory Networks

We refer to Figure 1.1 to explain a GRN, the constituents of a GRN, and the underlying processes and regulatory interactions that it represents. Genes are segments of deoxyribonucleic acid (DNA) consisting of instruction codes for synthesizing the different proteins that function in cells in the body. This process of synthesis consists of two steps: transcription of a gene into messenger RNA (also called "gene expression") and translation of messenger RNA into protein [17]. Proteins called transcription factors (TF) activate or repress the expression of their target genes [37]. A TF interacts with a binding site in the promoter regions [19] of a target gene, as seen in Figure 1.1; the promoter is the region of DNA before the gene

Figure 1.1: A simple representation of a fictional gene regulatory network. Figure taken from Schilttet *al.* [53].



sequence to which the RNA polymerase and TFs bind in order to start transcription.

Figure 1.1 implicitly represents a GRN. In this graph, each node is a TF or a gene. Each edge is directed from a TF to a gene whose expression it controls. Such an interaction may be annotated with additional information, e.g., whether the TF activates or represses expression of a target gene. A gene may be controlled by more than one TF. In turn, a TF may regulate the expression of many genes. A GRN does not record many aspects of gene regulation, e.g., the sequences of DNA to which a TF binds preferentially. A GRN does not record the location of these DNA sequences in a gene's promoter or the affinity with which a TF binds to them. More importantly, it does not usually represent the dynamic nature of how a single gene's expression over time is controlled by multiple TFs acting in combination.

Despite these simplifications, GRNs have been very useful in shedding light on the mechanisms and processes by which cells differentiate during development in multicel-

lular organsisms [37]. They have also been of use in other life science domains such as applied biomedical research and drug and medicine discovery. One such use is revealing the mechanisms of diseases that occur when cellular processes are dysregulated [40].

1.2 Computational Inference of GRNs

GRNs were first proposed in the 1960s [20]. A common method to construct a GRN for a specific cellular process is to study the relevant literature and determine the critical regulatory interactions that govern that process. While this approach has the advantage of being based on accurate and precise experimental data, it is painstaking and slow. Moreover, it does not scale to GRNs with hundreds or thousands of nodes. These limitations lead to the central question of this thesis: **how can we construct GRNs computationally?**

1.2.1 Single-Cell RNA Sequencing

Since GRNs represent regulation of gene transcription, in principle we could use gene expression data to construct them. In fact, with the advent of high-throughput gene expression measurements in the early 2000s, many computational methods have been developed to automatically infer GRNs [17]. Early methods used DNA microarray data, which was invented in 1981 [26]. Later methods exploited the development of RNA sequencing (RNA-seq) data. RNA-seq is a method in which we can sequence and measure the quantities of RNA molecules in a sample. Traditionally, bulk RNA-seq data has been used to gather gene expression data. Bulk RNA-sequencing measures the average expression level for each gene across a large population of input cells. Therefore, it may mask cellular heterogeneity and combine gene expression levels across different types of cells that are present in a sample. Consequently, GRNs constructed from these data may not reflect the differences between individual cells, which can have profound functional consequences, in both unicellular and

multicellular organisms [23]

The advent of single-cell RNA-sequencing (scRNA-seq) has dramatically changed this landscape by enabling us to characterise gene expression in individual cells without the need for purification. A single-cell digital gene expression profiling assay was developed in 2009 by Tang*et al.* [56], but it did not gain popularity until 2014 when new protocols and lower sequencing costs made it more accessible [45]. This next-generation sequencing technology is a powerful and cost-efficient tool for ultra-high-throughput transcriptome analysis. Sequencing an mRNA from a single cell works by first capturing individual cells and then amplifying the miniscule amounts of mRNA from each. Although this is a major challenge in sequencing scRNA data, the end result is identification of subpopulations at a cellular level, distinguished from heterogeneous populations of cells [23]. Single-cell transcriptomic analysis of transitions between different cellular states such as development or differentiation have also been found to reveal new insights into regulatory mechanisms [50].

In this thesis, we aim to use gene expression data derived from scRNA-seq technology to infer GRNs automatically. We believe that using a data source with higher resolution and better throughput will help us understand with greater certainty how individual cell lines are affected by the different genes and hence get an accurate representation of the GRN at a cellular level.

1.2.2 Drawbacks of Computational Methods Used to Infer GRNs

We survey existing methods for inferring GRNs from single-cell gene expression data in Chapter 2. Here, we focus on the limitations of existing methods to motivate the approach we are taking in this thesis. Virtually all these methods are unsupervised or association-based techniques [24, 34, 36, 46]. In a recent benchmarking study, Pratapa *et al.* [46] compared the GRNs inferred by twelve representative inference algorithms to a variety of ground truth

datasets. They found that virtually all of these methods had low area under the precision-recall curve (AUPRC) or early precision (i.e., precision at a low value of recall).

These methods were unsupervised, i.e., they did not leverage any experimentally-discovered regulatory interactions between TFs and their targets. While there have been numerous studies to infer GRNs from bulk and scRNA-seq in an unsupervised setting [2, 30, 42], the problem remains unexplored when it comes to scRNA-seq data using supervised methods. We look at the existing supervised methods to infer GRNs in Section 2.5 and explain their drawbacks. These observations have motivated us to explore the use of supervised learning to infer GRNs from scRNA-seq data.

1.2.3 Graph Machine Learning

Graph machine learning has gained popularity in recent times due to the ubiquity of networks [65]. Traditional machine learning approaches that analyze multidimensional feature spaces do not explicitly capture relationships among data points. Graph machine learning uses the concept of *homophily* [39], which states that the properties of a node in a graph are influenced by its neighboring nodes and that similar nodes tend to be connected by edges and hence share similar features. Homophily has been observed in networks from many domains including social science and molecular biology. Machine learning on graphs uses their inherent structure to solve prediction problems [64].

There are various techniques of graph machine learning that have been developed for network data and which are in line with traditional machine learning tasks such as supervised, semi-supervised, and unsupervised learning tasks [4]. The tasks of graph machine learning can be broadly categorized as follows [73]:

1. node classification, where the model learns the labels of nodes after training [22],

2. link prediction [22, 54, 74], where the existence of edge between two nodes in a graph

is to be predicted and,

3. graph classification [67, 71], where a model predicts certain graph-level properties of a single graph given its node and edge features.

Convolutional neural networks (CNNs) [25] are popular in image processing tasks where the data is typically in a two-dimensional (2-D) grid format. Images occur in the form of a regular 2-D grid structure. CNNs use convolutions as filters to focus on each part of the image and to learn properties of different portions of the image. Unlike images, the structure of graphs is not regular since a node may have an arbitrary number of connections. Graph convolutional techniques extend the idea of CNNs to graph structured data. Graph structure is quite different from a regular 2-D structure. Thus, the filter size when performing convolutions on a graph is not fixed. A convolutional layer in graph convolutional networks works by aggregating the features of a nodes with those of its neighbors [65]. Repeated rounds of convolution implicitly increase the neighborhood of nodes that are considered for aggregation.

In bioinformatics, graph machine learning, and especially graph convolutional techniques, have been used for a number of prediction tasks including protein function [70], microRNA-drug association [15], protein interaction [57], protein interface [7], cell growth simulation [35], and gene regulatory network reconstruction [48]. For a review of some techniques that use graph machine learning for tasks in bioinformatics, the reader can refer to Li *et al.* [28].

While there do exist unsupervised machine learning techniques to infer GRNs, graph convolution based techniques and their potential for this task has not yet been fully explored. We explicitly discuss graph machine learning and graph convolution based methods to infer GRNs, and other graph convolution based methods that are directly relevant to this thesis in Section 2.5.

1.3 Contributions of This Thesis

In this chapter, we have introduced GRNs and discussed the importance of inferring GRNs from experimental data pertaining to gene expression levels. We have also described the nature of the experimental data and the cellular resolution that scRNA-seq data can provide. We have described computational methods in Section 1.2, and highlighted a major drawback of many computational approaches for inferring GRNs (low AUPRC or early precision scores).

In Section 1.2.3 we have described the emerging field of graph machine learning, which has shown promise for predictive tasks in network biology. We discussed the potential of graph convolutional networks (a sub-domain of graph machine learning) for link prediction tasks. These observations culminate in the following research question:

Primary Research Question: Can we develop an approach based on graph convolutional networks that takes as input an experimentally determined GRN and a single-cell RNA-seq dataset to predict accurately new regulatory interactions between TFs and genes? These *new* regulatory edges represent the TF-gene interactions which have not been reported in the literature.

We develop a GCN-based approach to solve this link prediction problem. Our predictor is an autoencoder. It consists of two parts: (i) an encoder that computes a low-dimensional embedding for every node in the input GRN and (ii) a decoder that when given the embeddings for two nodes, predicts a score for the existence of a regulatory link between them. We develop six variations on this theme. We compare them to a recently developed supervised method for GRN inference and two off-the-shelf link predictors on the task of predicting mouse and human GRNs. We use four scRNA-seq datasets to find the best hyperparameter values for the predictors. Using these hyperparameter choices, we compare

the accuracy of the methods on two additional scRNA-seq datasets. Our results demonstrate that the GCN-based methods quite substantially outperform the other methods. Finally, we perform additional analyses to confirm the robustness of the GCN-based algorithms.

Chapter 2

Review of the Literature

In this chapter, we provide an overview of the literature on inference of GRNs. We discuss a variety of methods and the drawbacks of each of them. Our goal is to provide a clear motivation for our decision to use supervised graph machine learning to solve the GRN inference problem. In Section 2.1, we consider association networks, which use unsupervised methods to infer GRNs. We then move on to Boolean Models in Section 2.2 followed by differential equation models in Section 2.3. We discuss methods that use unsupervised machine learning for inference of GRNs in Section 2.4, and finally we look at supervised methods in Section 2.5.

2.1 Association Networks

A broad class of unsupervised methods represent GRNs as association networks. Association networks are undirected by nature. If two genes are connected by an edge, it is not possible to determine which gene is regulating and which gene is being regulated. Similarity measures such as Pearson's correlation coefficient or mutual information are applied to determine the degree of co-expression between pairs of genes [55]. In principle, these methods may predict an edge between every pair of genes. They use statistical tests to report only pairs that have significant levels of co-expression. ARACNE [36] uses mutual information (MI) to determine the dependency among the genes and then remove indirect interactions using the data processing inequality. Weighted correlation network analysis (WGCNA) [24], an R

package describing the correlation patterns amongst genes, is used to find clusters of highly correlated genes. It uses hierarchical clustering to identify related gene modules.

One of the advantages of these methods is that they have low computational complexity [13]. A sense of direction or regulation can be added to each edge in such a graph by utilizing additional information such as which nodes correspond to TFs or by using the sign of the correlation between the connected pair of genes.

2.2 Boolean Models

Boolean models are widely used representations of GRNs. In a Boolean model, each node is in one of two states *True* or *False*. In addition, each node has a Boolean function defining the value of its state in terms of the states of its regulators. Thus, reverse engineering a Boolean networks requires not just inferring the regulatory interactions but also these functions.

REVEAL is one of the first algorithms for inferring Boolean networks [29]. It uses deterministic transition tables to infer the Boolean relationships between the variables. The algorithm works incrementally and uses mutual information analysis of state transition data (expression data) to determine if a link exists between these nodes (i.e. genes). Since this algorithm uses brute-force to find possible connection for each gene, this algorithm works best only for a low number of connections, (K), per gene (K being from 1-3).

A probabilistic Boolean network permits more than regulatory function per node. Simulation of such a network involves the selection of a node and one of its regulatory functions to update the node's state. The selection probability of a function is proportional to its coefficient of determination (COD). For network inference, the COD is used to select a list of predictors for a given gene [27].

2.3 Differential and Difference Equation models

The concentrations of the molecules in a GRN change over time [49]. Therefore, ordinary differential equations (ODEs) have been developed to model GRNs. Systems of ODEs can represent positive and negative feedback loops and use real-valued gene expression data [62]. Typically, the rate of change of expression of the genes in a GRN is modeled by the following differential equation:

$$\frac{dx_i}{dt} = f_i\big(x(t), p, u, t\big)$$

where $x(t) = (x_1(t), x_2(t), ..., x_n(t))$ is a vector of gene expression values for n genes at time t, p is a set of model parameters, u is a set of external perturbations, and f is a function that models the changes of variables x_i based on model parameters p and external perturbations u. A GRN inference model should be able to determine the function f and parameters of the model p for measured values of gene expressions x and external perturbations u at the time t [13].

These models are mainly divided into linear differential equation methods [3, 38] and non-linear differential equation methods [34]. The latter can capture more complex relationships, consistent with the order of interactions in biological systems. Although, differential equation-based methods follow simple homogeneous structure, it involves a large number of parameters of the order of $O(d^2)$; where d is the number of genes modeled [60].

SCODE uses a linear ODE model to construct the GRN directly from scRNA-seq gene expression data with timestamps. SCODE uses the Monocle algorithm [47] to assign a pseudo-time to each cell [43]. SCODE uses relational expression that can be estimated efficiently using linear regression to determine the values of the parameters of the ODE system. Non-linear ODE models, such as [34], use time-series as well as steady-state data jointly to infer GRNs and are thus computationally expensive. This approach uses time-series data to represent changes in gene expression for one gene as a function of gene expression of

other genes thus taking into account the decay rate of gene. Combination with steady-state data allows regulatory interactions between genes to be more accurately modeled [34].

2.4 Machine Learning-Based GRN Inference Algorithms

In this section, we give a brief history of machine learning-based methods to infer gene regulatory networks. These methods use classification or regression based models to fit gene expression data. They use feature engineering to obtain the subset of regulatory genes and target factors with strong correlations. To this end, they use gene expression data as input samples. The learned model assigns weights to regulatory relationships among genes. The final step is sorting pairs of genes by weight and selecting a cutoff on the weight to construct a network.

GENIE3 [18] works by solving p different regression problems to predict a regulatory network among p genes. It is a tree-based ensemble method that uses Random Forests to predict the expression profile of each target gene from the profiles of all the other genes. The significance of an input gene to predict the gene expression of a target gene is indicative of a regulatory link. GENIE3 aggregates these links over all the genes to output a ranked list of interactions. The major disadvantage of this method is that it can only be applied to steady-state data.

DynGENIE3 [9] was developed to overcome this drawback. DynGENIE3 uses a finite difference approximation to estimate the derivative of the expression level of a gene j modelled through an ordinary differential equation model that is a function of the decay rate of the gene. Since this approximation relies on the time intervals between consecutive sampling time points, DynGENIE3 will miss the regulatory interactions that happen faster than the sampling frequency. While DynGENIE3 has been shown to perform better than GENIE3 on synthetic data, the same results do not hold true for real datasets [9].

GRNBoost2 [41] is based on the GENIE3 architecture but uses gradient boosting rather than random forests to solve the regression task. Gradient boosting uses ensemble learning to integrate multiple decision trees to reduce the error between predicted and known values. GRNBoost2 trains a tree-based regression model for each gene in the dataset using the expression values of a set of candidate transcription factors. Each model produces a partial GRN with regulatory links from the best predicting TFs to the target gene. These associations associations are combined and sorted by importance to output the GRN network.

A Bayesian network is a graph model defined by a triple (G, F, q), where G denotes a directed acyclic graph, F is a set of probability distributions for each node, and q is the set of parameters for F [8]. Each node in G corresponds to one random variable in the probability distributions, while each directed edge (u, v) represents the conditional dependence of v on u.

Even though Bayesian networks can encapsulate the stochastic nature of gene expression data, they are appropriate only for steady-state and not for time series measurements data [16]. Regulatory interactions in real GRNs do not occur at the same time, there is a temporal change across the gene expression levels. Dynamic Bayesian networks overcome this drawback by including these temporal features of gene expression data [27].

Recent studies to determine GRNs construct Bayesian networks using single-cell RNA sequencing (scRNA-seq) time-series data. The GRNVBEM method [51] uses scRNA-seq data to model time according to a pseudo-temporal index generated by an autoregressive moving average. The method sorts cells according to pseudo time and then runs a first-order autoregressive moving-average model on the data. To infer the GRN, the method uses a posterior probability threshold higher than 0.5 within a variational Bayesian framework. x A drawback of Bayesian networks is that applying them to construct large-scale networks can be computationally expensive [46, 72]. One of the ways to speed up learning is to apply a Boolean model on a local network, which has been seen to reduce the calculation time

significantly [31]; however this method does not consider the global structure in its entirety.

2.5 Supervised Methods

There are a small number of publications that use supervised methods to infer GRNs with single-cell RNA-seq data. Yuan and Bar-Joseph [68] combine scRNA-sequencing and deep learning in a supervised setting to propose an encoding scheme for gene expression data called the Convolutional Neural Network for Coexpression (CNNC). For every pair of genes, this method works by first transforming their gene expression profiles into a normalized probability density function. Treating these function as images, CNNC utilize CNNs to learn relationships between genes and to infer causality.

In another paper, Yuan and Bar-Joseph [69] develop an approach called GCNG based on GCNs [22] to infer gene interactions from spatial transcriptomics data. Their approach constructs a graph where each node is a cell and an edge connects two spatially proximal cells. Using the gene expression values as the feature vectors of the cell, the method uses GCNs to identify interactions between receptors and ligands.

Wang *et al.* [61] propose an end-to-end gene regulatory graph neural network (GRGNN) approach to reconstruct GRNs from scratch from gene expression in both supervised and semi supervised settings. GRGNN starts by building a noisy skeleton of a GRN by using association measures such as Pearson's correlation coefficient and mutual information to connect nodes. This noisy skeleton works as the starting model through which the method extracts positive and negative subgraphs. It subsequently uses GNN ensemble classifiers to predict links between nodes.

Our GCN-based approach improves upon these methods in the following manner. (i) It uses the positive examples, i.e., the edges in the input GRN, to construct the graph on which we compute convolutions unlike CNNC that uses the positive examples only for loss

calculation. (ii) Since GCNG performs convolutions over a cell-cell network, it can can predict interactions between proteins expressed in differnet cells. In contrast, by using the GRN directly, our method to predict TF-gene interactions. (iii) GRGNN uses computationally-predicted edges as positive edges. These edges may be predicten in error. Our method, on the other hand, use the experimentally-discovered edges in the GRN as positive edges.

Chapter 3

Algorithms

In this chapter, we begin by formulating the problem statement. We then describe the GCN-based autoencoders. Finally, we present the other supervised methods to which we compare the GCN-based methods.

3.1 Problem Description

The inputs to the problem are a graph and a feature matrix.

(a) The first input is a directed, unweighted graph $G = (V, E)$, where each node V in the network is either a TF or a gene. An edge in the graph is directed from a TF to a gene that it regulates.

(b) The second input is a single-cell gene expression matrix X of size $|V| \times |C|$, i.e. for each $i \in V$, we have a vector of gene expression values $x_i \in \mathbb{R}^{|C|}$, measured across $|C|$ cells.

Since V contains two types of nodes (TFs and genes), we can partition it into a set T of TFs and a set B of genes such that, $V = T \cup B$ and $T \cap B = \emptyset$. Since G is a GRN, every edge in E is an element of $T \times V$. Let $\bar{E}$ be the set of edges in $(T \times V) \backslash E$. Finally, let A denote the adjacency matrix of G represented.

The supervised GRN inference problem can now be defined as follows: Given G, its adjacency matrix A, and X, compute a new adjacency matrix $\bar{A}$, where $\bar{a}_{ij}$ has a value

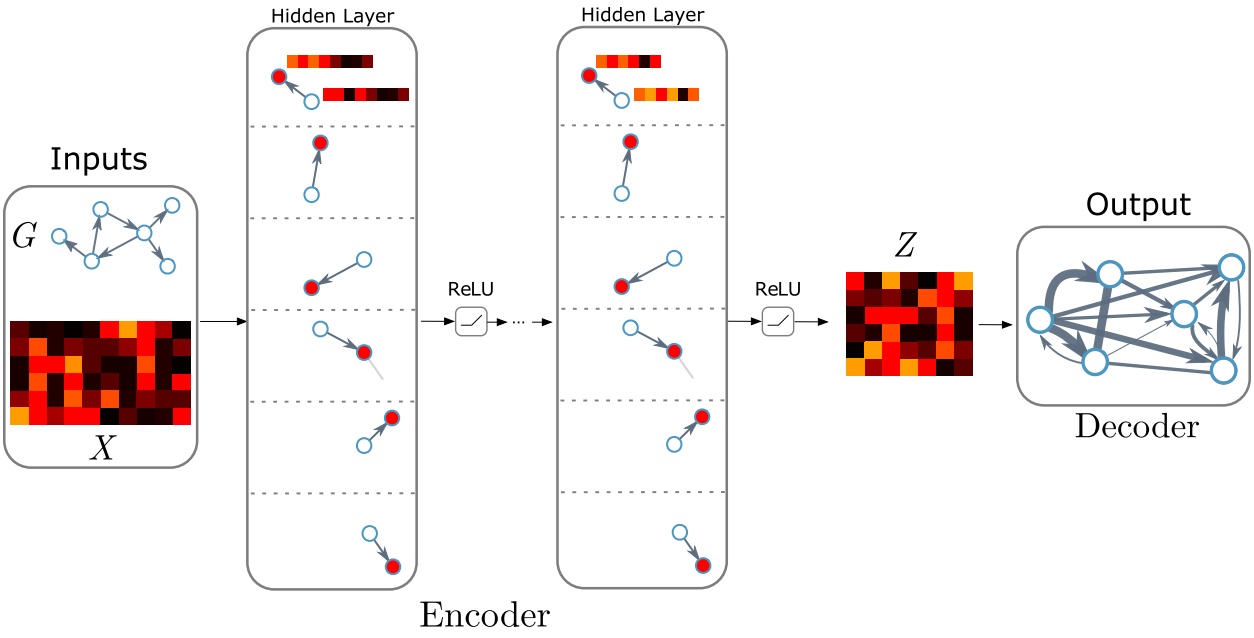
$0 \leq \bar{a}_{ij} \leq 1$ for every node pair (i, j). We can interpret the value of a_{ij} as the probability of existence of a regulatory interaction between two nodes i and j. We want to compute the matrix $\bar{A}$ such that the reconstruction loss, $\mathcal{L}$, is minimized:

$$\mathcal{L} = -\frac{1}{|E|} \sum_{(i,j) \in E} \log \bar{a}_{ij} - \frac{1}{|\bar{E}|} \sum_{(p,q) \in \bar{E}} \log (1 - \bar{a}_{pq})$$

In practice, we compute the second term for a randomly sampled subset of the edges in $\bar{E}$, as we explain in Chapter 4.

3.2 GCN-based encoders

Figure 3.1: GCN architecture



The GCN is a widely-used technique which efficiently performs parameterized neighborhood-based feature aggregation over the the nodes in the network. We use the GCN developed by Kipf and Welling [22] as the basis for our graph convolutional encoder. This encoder creates a geometric embedding of each node in G that reflects the gene expression

data in X. We describe this encoder in this section. In addition to G and X, our inputs are a number n representing the number of graph convolutional layers, and a list K representing the number of features in each layer $K = \{k_0, k_1, k_2, ..., k_n\}$. The number of features in the first hidden layer is equal to the number of cells in the expression matrix, i.e. $k_0 = |C|$.

The first layer takes X as input and the final layer produces an output matrix Z of size $|V| \times k_n$. In each layer l, the neural network learns the weights through the convolutions represented by matrix $W^{(l)}$. For every layer l, the dimensions of $W^{(l)}$ are $k_{l-1} \times k_l$. We use the rectified linear unit activation function given by $ReLU(x) = max(0, x)$ as the output of each layer. For every $1 \leq l \leq n-1$, the output of the GCN layer l is $H^{(l+1)} = f(H^{(l)}, A)$. Here, $H^{(0)} = X$, $H^{(n)} = Z$, and $f()$ is a non-linear propagation function as described below.

We use two types of encoders, differentiated by the presence or absence of directionality when considering the edges. Since GRN is a directed graph, we use a directed GCN (DGCN) to aggregate information using only the edges that enter a node. In addition, we consider undirected GCN-based encoders in this study, since it is possible that we can learn the lower dimensional embedding of a TF by aggregating information from the genes that they regulate. We describe these two encoders now.

(a) **Undirected GCN-based encoder (GCN):** In this encoder, we treat the input graph to be undirected, i.e. we make each edge $(i, j) \in E$ to be undirected. Let A' be the symmetric adjacency matrix corresponding to this undirected version of graph G. We add self-edges to A' to obtain $\hat{A}'$, i.e., $\hat{A}' = A' + I$, where I is the identity matrix. This modification is useful to incorporate information about a node's own features. We compute the diagonal node degree matrix $\hat{D}'$ of $\hat{A}'$, where the diagonal entry in row i is given by $d'_{ii} = \sum_{j=0}^{n} \hat{A}'_{ij}$. We use the following non-linear propagation function $f_u()$:

$$H^{(l+1)} = f_u(H^{(l)}, A') = ReLU\left(\hat{D'}^{-\frac{1}{2}}\hat{A'}\hat{D'}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

In other words, the vector $h_i^{(l+1)}$ associated with node i in layer $l+1$ can be obtained using:

$$h_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{d_i d_j}} h_j^{(l)} W^{(l)}$$

where $\mathcal{N}_i$ is the neighborhood nodes of node v_i, connected by an edge in the undirected version of G.

(b) **Directed GCN-based encoder (DGCN):**

Here, the input graph is fed to the encoder as it is, i.e. the edges are directed. As before, we add self-edges to integrate the features of the nodes themselves. We also compute the node in-degree matrix $\hat{D'}$ of $\hat{A'}$, where the diagonal entry in row i is given by $d_{ii} = \sum_{j=0}^{n} \hat{A'_{ij}}$. We use the following non-linear propagation function $f_d()$ at each hidden layer to represent the directed information flow from TFs to genes:

$$H^{(l+1)} = f_d(H^{(l)}, A) = ReLU\left((\hat{D}^{-1}\hat{A})^T H^{(l)} W^{(l)}\right)$$

In other words, the vector $h_i^{(l+1)}$ associated with node i in layer $l+1$ can be obtained using:

$$h_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} \frac{1}{d_j} h_j^{(l)} W^{(l)}$$

where $\mathcal{N}_i$ is set of nodes with an outgoing edge to node i in G.

3.3 Decoders

The encoders described in the above section transform the input features X of size $|V| \times |C|$ into a lower dimensional embedding Z of size $|V| \times k_n$. We next use a decoder to reconstruct the original matrix from Z. Specifically, we use Z as input and learn a decoder function $g()$ to obtain a reconstructed adjacency matrix $\bar{A}$, such that $\bar{a}_{st} = g(z_s, z_t)$, where z_s and z_t represent rows s and t of Z. We consider the following three types of decoders:

(a) **Inner Product Decoder (IP):** This is the simplest decoder where we compute the matrix $\bar{A} = \sigma(ZZ^T)$, where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid activation function. Simply put, we obtain the confidence score for the edge (s, t) as

$$\bar{a}_{st} = \sigma(\langle z_s, z_t \rangle),$$

where $\langle . \rangle$ represents the inner product of the two vectors z_s and z_t. The output adjacency matrix $\bar{A}$ here is symmetric and represents an undirected graph.

(b) **Node weight-based decoder (NW) [52, 66]:** Here, we learn a weight v_s for each node $s \in V$. We obtain the confidence score for an edge (s, t) as

$$\bar{a_{st}} = \sigma(v_s \langle z_s, z_t \rangle)$$

If the learned weights for two nodes s and t are unequal, then $\bar{A}$ is asymmetric representing a directed graph.

(c) **RESCAL decoder (RS) [44]:** We adopt RESCAL, a factorization-based approach for relational learning in large-scale data. Here, we learn a matrix of weights $M \in \mathbb{R}^{k_n \times k_n}$ to obtain the output adjacency matrix $\bar{A} = \sigma(ZMZ^T)$. The confidence score

for an edge (s, t) is given by:

$$\bar{a_{st}} = \sigma(\langle z_s \times M, z_t \rangle)$$

This approach can yield a non-symmetric adjacency matrix representing a directed graph.

In all cases, we apply a sigmoid activation function to the decoder output to obtain probability scores for edges that lie between 0 and 1.

3.4 Other Methods Evaluated

We compare the performance of the following approaches for supervised GRN inference from scRNA-seq data:

1. We consider all six variants of GCN-based autoencoders, consisting of two encoders (GCN, DGCN) and three decoders (IP, NW, RS). In Chapter 4, we explain how to adapt these methods to work in an inductive setting, i.e., when we have to predict edges among nodes that were not present in the training data.

2. **MLP**: We create an edge-feature matrix F, such that F_{uv} is the concatenation of the expression vectors of genes u and v. Each hidden layer is a fully connected neural network. We use the same parameters as in the setting of GCNs for number of hidden layers n and number of features in each layer, except that $k_0 = 2 \times |C|$. We use ReLU () activation function for the hidden layers. The output of the MLP at layer l can be written as:

$$H^{(l+1)} = ReLU\left(H^{(l)} M^{(l)}\right)$$

To obtain probability scores for edges between each pair of nodes, we add a fully connected layer $M^{(n)}$ of size $k_n \times 1$ as output and compute the final values as:

$$H^{(out)} = \sigma\left(H^{(n)} M^{(n)}\right)$$

We learn the weights for this neural network by minimizing the loss function in Equation 3.1. We use an MLP model with 1, 2, and 3 layers in this study.

3. **SVM-C**: : Here, for every pair of nodes $(u, v) \in T \times V$ we concatenate the expression vector of the two genes u and v to be its feature vector F_{uv}. This setup is similar to what we described for MLP-C. We assign the label 1 to each edge (u, v) in E and 0 otherwise. We train a linear kernel using the `LinearSVC` function in the `scikit-learn` Python package with values of the regularization parameter C taken from the set $\{0.1, 1, 10, 100\}$

4. **CNNC**: We use a convolutional neural network with NEPDF-based features as described in the original paper [68].

Chapter 4

Evaluation

A standard way to train and evaluate supervised machine learning algorithms is to split the input data into training, validation, and test sets. In this chapter, we describe the two ways in which we create train-validation-testsplits in conjunction with how we create negative examples for our experiments. We also describe the evaluation measures that we use to compare the performance of the methods discussed in Chapter 3. Finally, we list the implementation details of our experiments.

4.1 Creating Train-Validation-Test Splits

We start by considering only the genes in the expression dataset that are present in the ground-truth network. For the scRNA-seq gene expression dataset X across V genes and C cells, we partition V into a set of TFs T and set of genes B. Let $G = (V, E)$ be the graph corresponding to the ground-truth network restricted to the genes in V. We treat the edges in E as positive examples. Since G is a GRN, each edge in E is an element of $T \times V$. We sample the negative examples from the complement of this set $\bar{E} = (T \times V) \backslash E$. We now present the two strategies of splitting the dataset into training, validation, and testing data in this thesis.

Edge splits. We partition the edges in E as shown in Fig.4.1 and the negative samples of generating the three subsets from the data which we use for i) training the model; ii) val-

idating the performance of the trained model; and iii) evaluating the performance of the model.

In this setting, we divide the edges in E uniformly at random into three sets of positive examples P_{train}, P_{val}, and P_{test} based on the train, validation, and test ratios, which should sum up to 100% as shown in Fig 4.2. An important thing to note is that it is possible for a node in V to have incident edges in one or more of the training, validation and test sets. Hence, in this setting it is necessary to feed the features for *all* the nodes that are present in the graph during training.

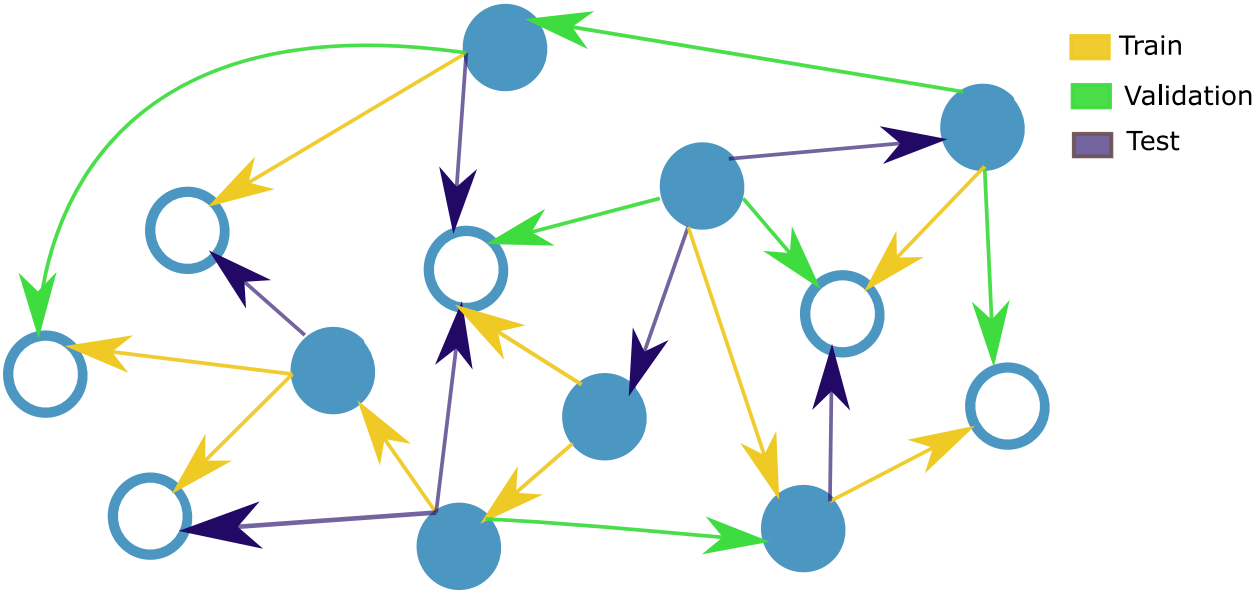
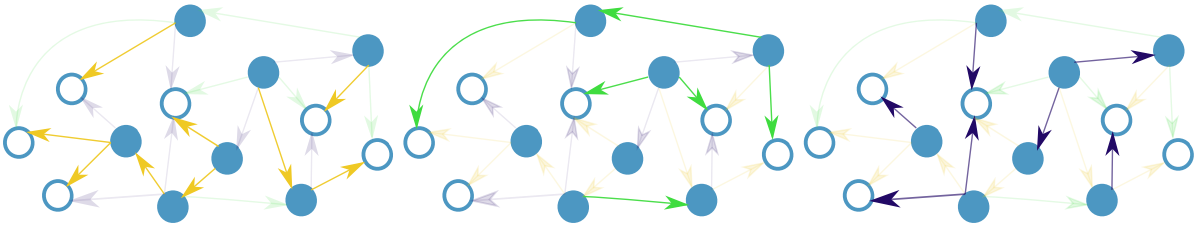


Figure 4.1: Edge splits to create training, validation and test data.

Figure 4.2: Training, Validation and Test Data for a graph split by edges.



(a) Training graph (b) Validation graph (c) Test graph

We create three sets of negative examples by sampling $\bar{E}$ uniformly at random. We create set N_{train} for negative training examples from $\bar{E}$ such that $|N_{train}| = |P_{train}|$. We create corresponding sets N_{val} and N_{test} of negative validation and testing examples of size $|P_{val}|$ and $|P_{test}|$, respectively. We ensure that these three sets are disjoint from each other and from each of the positive sets.

Node splits. In this method, we divide the nodes in V uniformly at random into three different sets $T_{train}, T_{val}, T_{test}$ using the train:val:test ratios as earlier, shown in Fig 4.3. We include an edge in E in the training set P_{train} of positive edges if and only if both endpoints are elements of T_{train}. In other words, the edges in P_{train} are exactly those in the induced graph of T_{train} in G, i.e., $P_{train} = \{(t, v) \in E | t, v \in T_{train}\}$. We define the other two subsets of positive examples similarly: $P_{val} = \{(t, v) \in E | t, v \in T_{val}\}$ and $P_{test} = \{(t, v) \in E | t, v \in T_{test}\}$. In contrast to edge splits, some edges in E do not belong to any of the sets P_{train}, P_{val}, or P_{test} as shown in Fig 4.5.

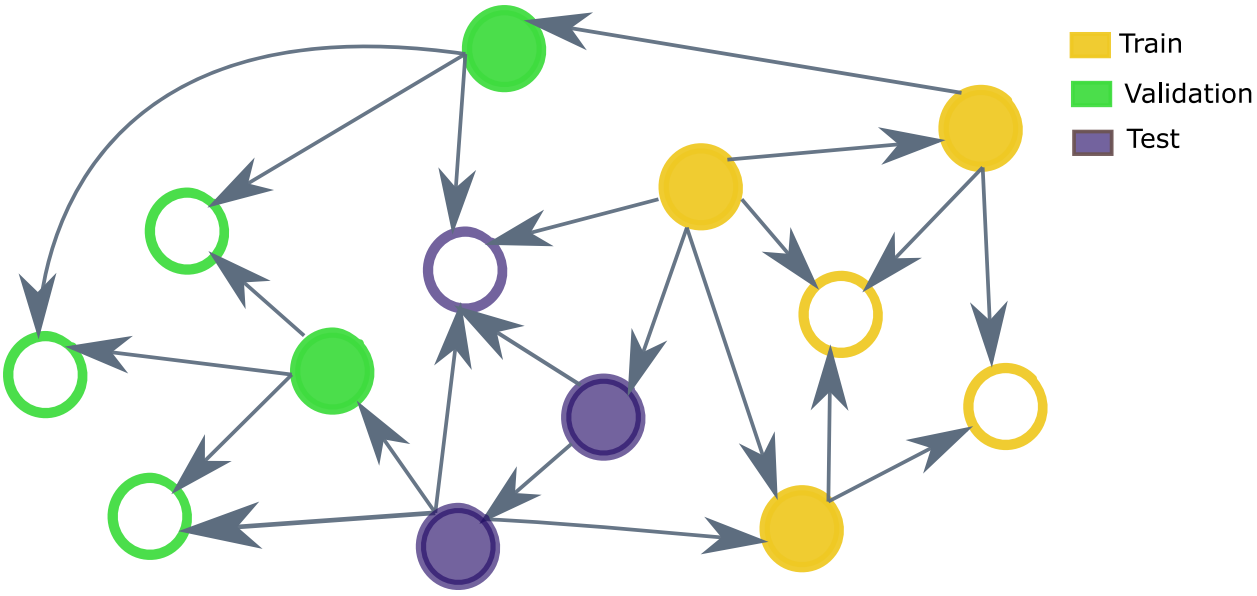
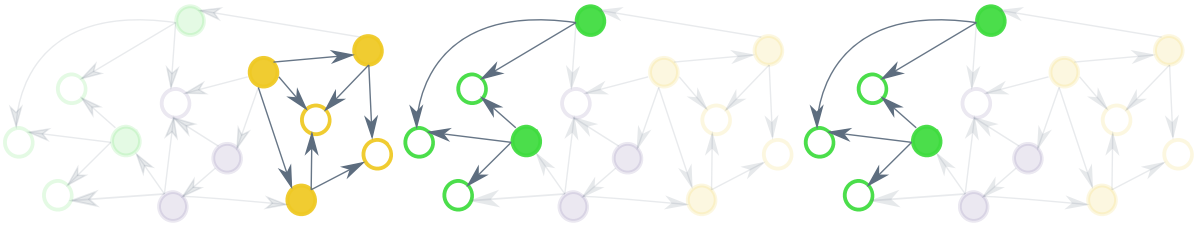


Figure 4.3: Node splits to create training, validation and test data.

To sample the negatives, we define $\bar{E}_{train} = (T_{train} \times T_{train}) \backslash P_{train}$, $\bar{E}_{val} = (T_{val} \times$

Figure 4.4: Training, Validation and Test Data for a graph split by nodes.



(a) Training graph (b) Validation graph (c) Test graph

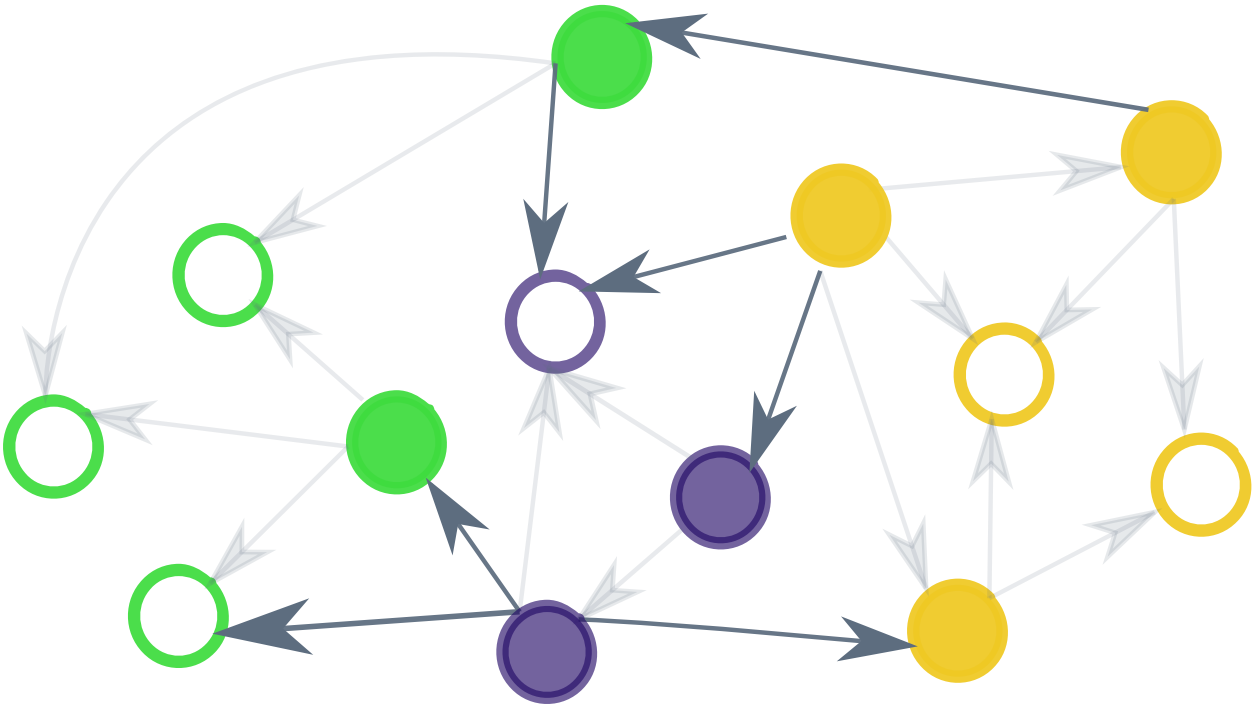


Figure 4.5: Dropped edges during node-split evaluation which do not belong to training, validation or test set

$T_{val)} \backslash P_{val}$, and $\bar{E}_{test} = (T_{test} \times T_{test}) \backslash P_{test}$. We then sample uniformly at random from $\bar{E}_{train}$, $\bar{E}_{val}$, and $\bar{E}_{test}$ to obtain the negative training (N_{train}), validation (N_{val}), and test (N_{test}) examples respectively, such that $|N_{train}| = |P_{train}|$, $|N_{val}| = |P_{val}|$, and $|N_{test}| = |P_{test}|$. By construction, none of the positive and negative tests overlap with each other as shown in Fig 4.4.

Since our goal is to predict regulatory interactions, the edge split approach is a

standard way to train a model and test its performance. However, there is a risk of leakage of information from the training step to the validation or testing steps since a node may have incident edges in more than one of the three sets. Therefore, we adopted the node split method, which ensures that for each node, any of its incident edges are present in only one of the training, validation, or test sets or not used at all. This approach is more challenging as it simulates a scenario where very few or no regulatory interactions are known for some TFs. We also do not need to know information about all the nodes in the graph during training. Thus, a model trained using node splits can be applied to unseen graphs.

4.2 Evaluation Measures

We ensure that each predictor outputs the edges in decreasing order of score so that we can compute precision-recall curves. We used two complementary measures, the early precision and the AUPRC, to measure the accuracy of the inference methods. Early precision (EP) is defined as the fraction of true positives in the top-k edges [46], where k is the number of positive edges in the test set. We repeat each experiment 10 times and report these values.

4.3 Implementation Details

We use the `PyTorch-Geometric` package [6] to implement the different GCN architectures. We set the maximum number of epochs to be 2,000. We use an early stopping criterion to terminate the training in case the validation loss does not improve for 100 epochs. To minimize the loss function, we use the Adam optimizer [21] with a learning rate of 0.001. We performed a parameter search for the number of GCN layers in $n = \{1, 2, 3, 4, 5\}$. We fixed the number of dimensions in the final layer to be 128. If we used k layers in total, for each $1 \leq i \leq k$, we set the dimensions of the ith layer to be $128(k-i+1)$. After each

layer, we introduced a dropout of 0.2 to regularize the network. The MLP predictor also had 128 dimensions in its final layer. We performed a hyperparameter search with 1, 2, and 3 layers. For SVM-C, we performed hyperparameter search for the regularization parameter C in $[0.1, 1, 10, 100]$. For all the experiments except Section 6.4, we use train:validate:test split ratio of 80:10:10.

We trained and tested the MLP, CNNC and GCN-based autoencoders on a Telsa P40 graphics card manufactured by NVIDIA. We used CUDA v11.0. The GPU processor has a 12 GB of GDDR5X memory. The operating system was Ubuntu 18.04.5 long term support (LTS).

Chapter 5

Datasets and Features

In this chapter, we describe the sources and formats of the datasets that we use. We elaborate on the nature of the scRNA-seq data, and the ground-truth mouse and human network data that we use in our experiments. We conclude by detailing how we preprocess the raw data and convert them to cell-line specific feature vectors for use in our predictors.

5.1 Data Sources

Experimental single-cell RNA-seq datasets. We obtained six different scRNA-seq datasets for training and evaluating GRN inference algorithms. These datasets are from four different mouse cell types and two different human cell types. These datasets have the following characteristics:

1. mouse embryonic stem cells (**mESC**) [12]: This dataset contains scRNA-seq expression for 421 primitive endoderm cells across 896 genes.

2. mouse hematopoietic stem and progenitor Cells (**mHSC**): Nestorowa *et al.* [14] used scRNA sequencing to profile single hematopoietic stem and progenitor cells. This dataset consists of $4,158$ genes and $3,175$ cells.

3. human definitive endoderm cells (**hESC**) [5]: This dataset is from a time course scRNA-seq experiment derived using a protocol to produce differentially expressed

cells from human embryonic stem cells, measured at 0h, 12h, 24h, 36h, 72h and 96h. This dataset contains expression measurements for 1,142 genes across 758 cells.

4. mouse bone-marrow derived macrophages (**mMac**) [1]: The mouse scRNA-seq dataset collected by Alavi *et al.* [1] consists of uniformly processed 43,261 expression profiles from over 500 different scRNA-seq studies. There are 4,126 are dendritic cells, and 6,283 are bone marrow-derived macrophage cells and 20,463 genes. We use the bone-marrow derived macrophage cells for our study. We selected this dataset since the publication on CNNC also used it for evaluation [68].

5. mouse intestinal epithelium cells (**mEpi**) [10]: This dataset contains 13,353 cells from three regions of the mouse intestinal epithelium.

6. human Placenta Cells from the maternal-fetal interface (**hPlacenta**) [59]: This dataset consists of 4,353 cells. We use this dataset with different numbers of highly-varying genes to study the effect on the performance of GCN-based autoencoders.

5.2 DataPre-processing and Normalization

In this section, we describe the steps for pre-processing and normalizing the data in Section 5.1.

1. **mESC, mHSC, hESC**: We obtained these three datasets from the BEELINE framework [46]. Using the pseudotime values for the cells in each dataset, the authors of BEELINE computed how much the expression of each gene in the dataset varied along pseudotime as well as the statistical significance of this variance. They selected the TFs and genes using the following two steps:

(a) These included all the TFs whose variance had p-value at most 0.01. This approach enabled the GRN inference methods to consider TFs that may have a modest variation in gene regulation but still regulate their targets.

(b) They added all the genes with p-value less than or equal to a threshold. They selected a threshold so that the number of genes selected was 1000 genes.

Readers may read the Methods section of [46] for further details.

2. **mMac**: We retained only the genes and TFs that are expressed in at least 30% of the cells.

3. **mEpi, hPlacenta**: For these two datasets, we employ the following steps suggested as best practices by Leuken and Theis [32].

(a) As a quality control step, we filter out cells that have total RNA-seq counts below a certain threshold (1,500) as well as total RNA-seq beyond a certain threshold (40,000). This step helps to remove cells with skewed distributions of gene expression.

(b) We also eliminate cells with a high count of mitochondrial reads. A high value for this number may indicate cell stress, as there is a lower proportion of nuclear mRNA in the cell [32].

(c) We keep only the cells that have a minimum number of genes expressed.

(d) We then control the gene quality in the dataset by eliminating the genes that are detected in less than a threshold number of cells. [32].

(e) We use the R package *scran* [33] to perform counts per million (CPM) normalization. First, for each cell, we divided each gene's expression by the sum of the mRNA counts for the genes expressed in the cell. Next, we multiplied each

count by 10^6. This step ensures that every cell has the same total count after normalization. We then log transform the data for further use. We standardize this data, i.e., for each gene, we divide its gene expression in each with the sum of gene expression in all the cells.

(f) Genes whose expression has high variance across the cells are informative of the underlying biological variation in the data. These are termed as highly varying genes. We use the *scanpy* package [63] to extract the k-most highly varying genes for our study, for different values of k.

5.3 Ground Truth Networks

In GRN inference experiments, it is a common practice to evaluate the accuracy of the resulting network by comparing its edges to an existing database that consists of TFs linked to their targets. We use the same ground truth network for both mouse and human dataset as is used in BEELINE [46]. Specifically, we selected non-cell-type-specific networks from the BEELINE evaluation. These networks curate and integrate TF-gene edges observed across multiple cell-types.

Table 5.1: Statistics on scRNA-seq datasets.

Dataset	**#Cells**	**#Nodes**	**#Edges**	**#TFs**
mESC	471	896	6893	516
mHSC	3,175	4,158	17,309	445
mMac	6,283	7,428	35347	747
hESC	758	1,142	4597	292

Table 5.1 provides details on each scRNA-seq dataset and the number of nodes, edges, and TFs in the corresponding ground-truth networks. Note that these counts change from one scRNA-seq dataset to another, since we considered only those genes or TFs whose expression was measured in that dataset and that were present in the ground-truth network.

Table 5.2: Statistics on mEpi dataset with different number of highly variable genes

#Highly variable genes	#Cells	#Nodes	#Edges	#TFs
2000	12,457	800	1467	92
4000	12,457	2,154	5,163	220
5000	12,457	2,863	7,597	294
6000	12,457	3,872	11,163	369

Table 5.3: Statistics on hPlacenta dataset with different number of highly variable genes

#Highly variable genes	#Cells	#Nodes	#Edges	#TFs
2000	4,146	801	1,400	71
4000	4,146	2,226	6,495	194
5000	4,146	3,020	10,660	278
6000	4,146	3,861	15,799	374

Similarly, we include the statistics for mEpi and hPlacenta datasets in Table 5.2 and Table 5.3 respectively.

Chapter 6

Results

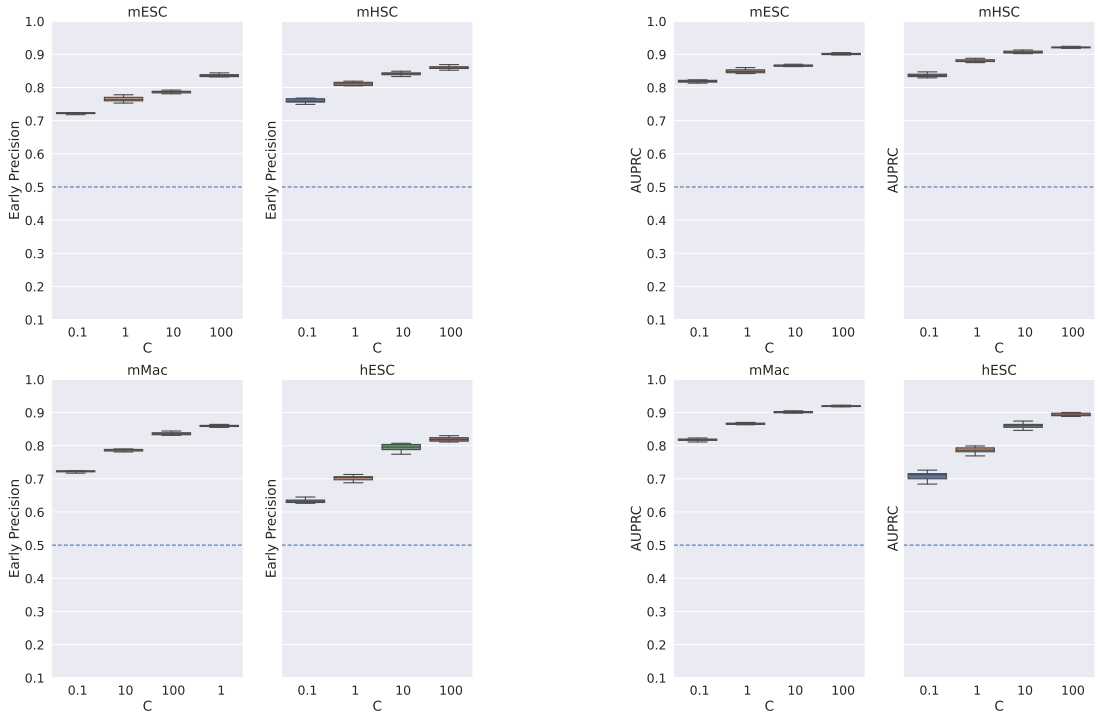
In this chapter, we lay out the results yielded by the GRN inference techniques described in Chapter 3, using the evaluation techniques defined in Chapter 4 and applied to the datasets from Chapter 5. We also note empirical evidence for some of the design choices we made for model architecture.

We show results both for edge splits and for node splits, repeated 10 times. We begin by describing the results for hyperparameter search for edge splits and for node splits on mESC, mHSC, mMac, and hESC datasets in Section 6.1 and Section 6.2 respectively. We select the best hyperparameter(s) for each GRN inference method. Subsequently, we train and test each method with the selected hyperparameters on two different datasets (mEpi and hPlacenta) and compare the methods. Finally, we assess the performance of the best performing GCN architecture on node split evaluation by varying the train-val-test ratio.

6.1 Hyperparameter Search for Edge Split Evaluation

In this section, we present the results of hyperparameter search for each of the methods described in Section 3.4 for edge-split evaluation as explained in Section 4.1. For CNNC, we have considered the same architecture as used by Yuan *et al.* [68]. We evaluate our models on four datasets: mESC, mHSC, mMac, and hESC. To obtain these results, we used a train:validate:test ratio of 80:10:10.

Figure 6.1: Early Precision and AUPRC scores for SVM-C with different values of $C = [0.1, 1, 10, 100]$ for edge split evaluation.



(a) Early Precision (b) AUPRC

6.1.1 SVM-C

In Figure 6.1, we show the performance of SVM-C for four different values of cost parameter C: $\{0.1, 1, 10, 100\}$. The cost parameter C is a regularization parameter that penalizes the model for misclassification. The larger the value of C, the heavier is penalty. We observed that, as we increased the value of C beyond 100, the training algorithm failed to converge. In this and subsequent figures, we display the performance of a random classifier using a blue line. Since we select equal numbers of test positive and test negative examples, a random classifier will have an expected EP and AUPRC of 0.5.

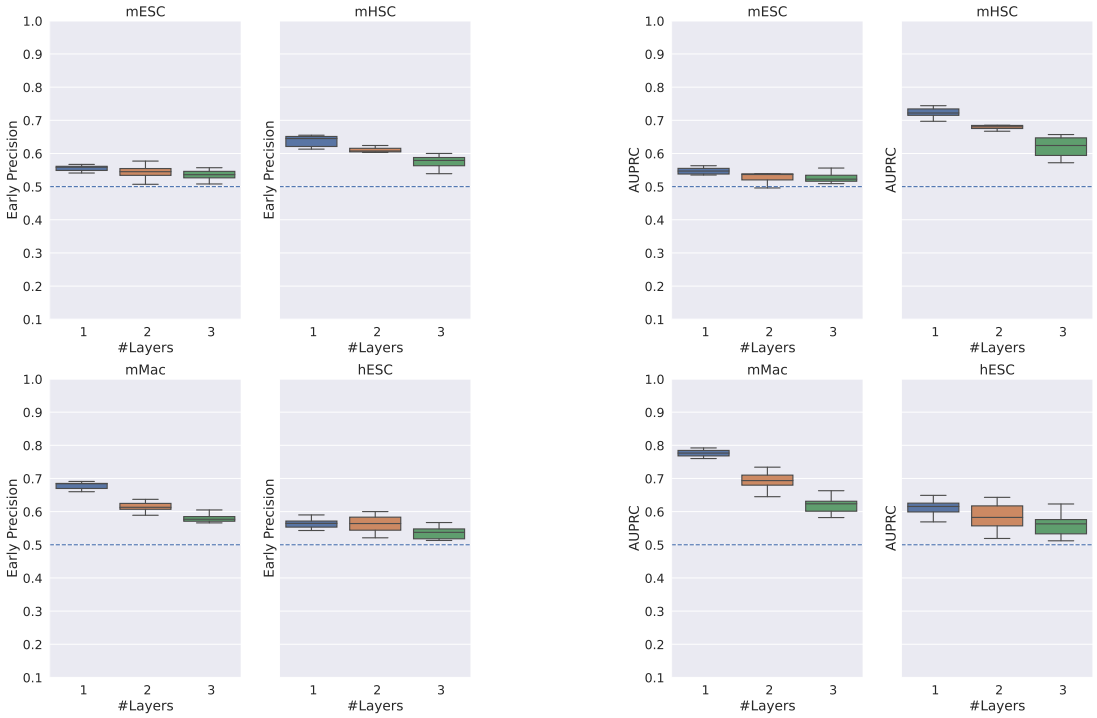
We note from Figures 6.1a and 6.1b that the trends for early precision (EP) and AUPRC are consistent within each dataset. It is also fairly evident that, as we increase the value of C, the EP and AUPRC scores improve substantially for each of the four datasets.

This trend is not surprising since higher values of C penalize the model stricter than lower vales of C for misclassification. Therefore, we choose $C = 100$ for SVM-C for subsequent edge split evaluations.

6.1.2 MLP

We evaluated the performance of MLP for 1, 2, and 3 layers in Figure 6.2. Similar to SVM, we noted that the trends are consisted among EP and AUPRC for each dataset. As the number of layers increases, for each dataset, both the EP and AURPC values decrease. While this decrease is marginal for smaller datasets (mESC, hESC), it is more substantial for the larger number of edges in the ground truth for the mHSC and mMac datasets. Therefore, we opted to use the MLP network with 1 layer for subsequent edge split evaluation.

Figure 6.2: Early Precision and AUPRC scores for MLP evaluated on 1, 2 and 3 layers for edge splits.



(a) Early Precision

(b) AUPRC

6.1.3 GCN-based Encoders and Decoders

Figure 6.3 and 6.4 display the EP and AUPRC results, respectively, for the six encoder-decoder combinations evaluated on five different numbers of hidden layers and four datasets. Since the trend for EP and AUPRC scores are similar, we used the EP values (Figure 6.3) to guide our selection of hyperparameters.

Firstly, we noted that, for all the datasets, the performance with NW decoder was erratic as we change the number of hidden layers. We were not able to determine the cause for this irregularity. Therefore, we eliminated the NW decoder, i.e., the GCN-NW and DGCN-NW models from further evaluation. For the other architectures, we observed that the performance of GCN encoders was steadily and substantially better than that of DGCN encoders.

Comparing the GCN-IP and GCN-RS models, we visually observed that the performance increase was not significant as we increase the number of layers beyond 2. For each number of layers, we compared the distributions of the EP values for GCN-RS and GCN-IP. The values for GCN-RS were significantly larger (p-value of Mann-Whitney test $< 10^{-5}$). We obtained similar results when we compared the AUPRC values. Therefore, we selected the GCN-RS combination with two layers as performance increase was not significant beyond two layers in our subsequent analyses.

Figure 6.3: Early Precision scores for six GCN-based encoders and decoders: GCN-IP, GCN-NW, GCN-RS, DGCN-IP, DGCN-NW and DGCN-RS evaluated on mESC, mHSC, mMac and hESC for $1, 2, 3, 4$, and 5 hidden layers. These results are for edge splits.

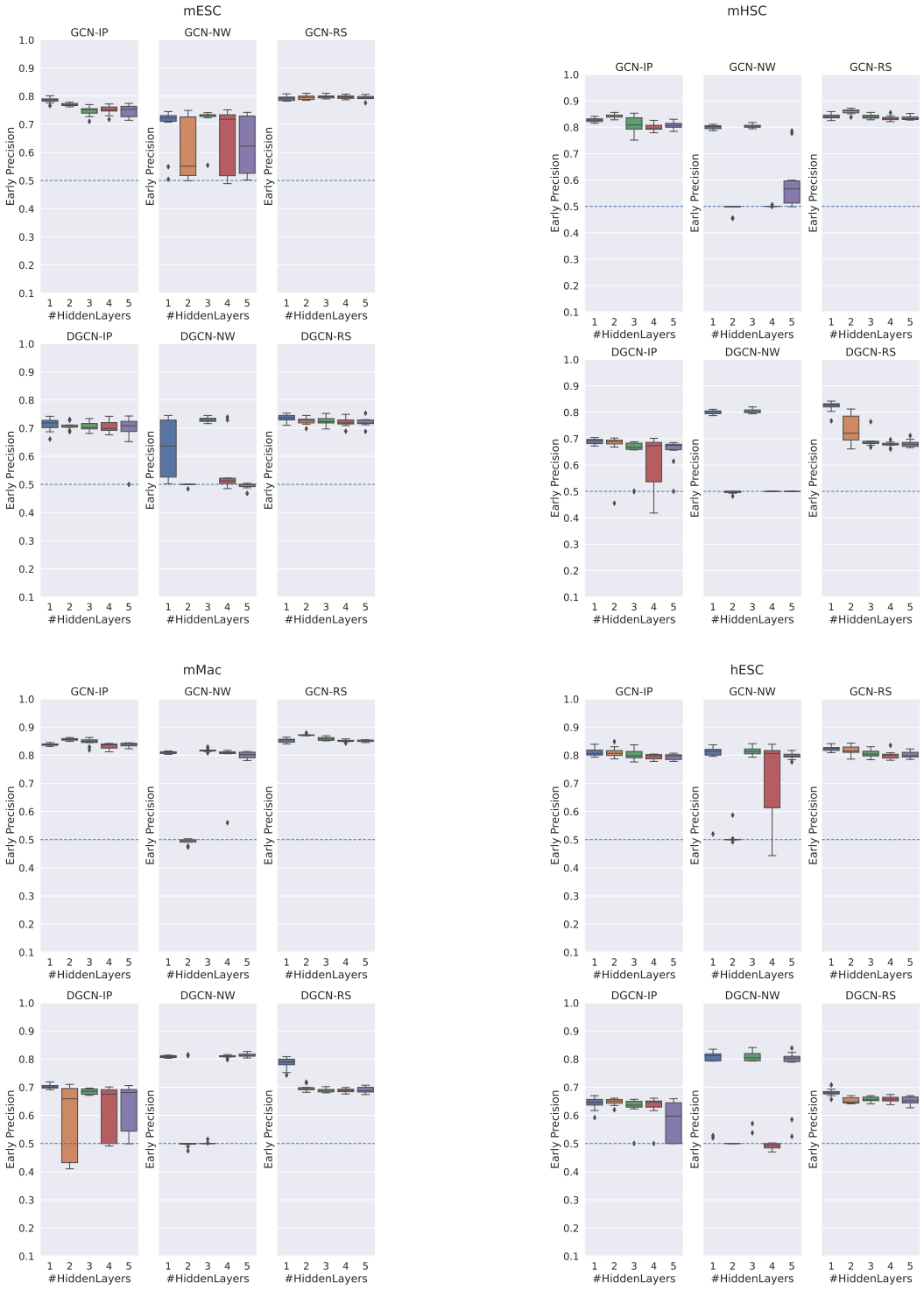
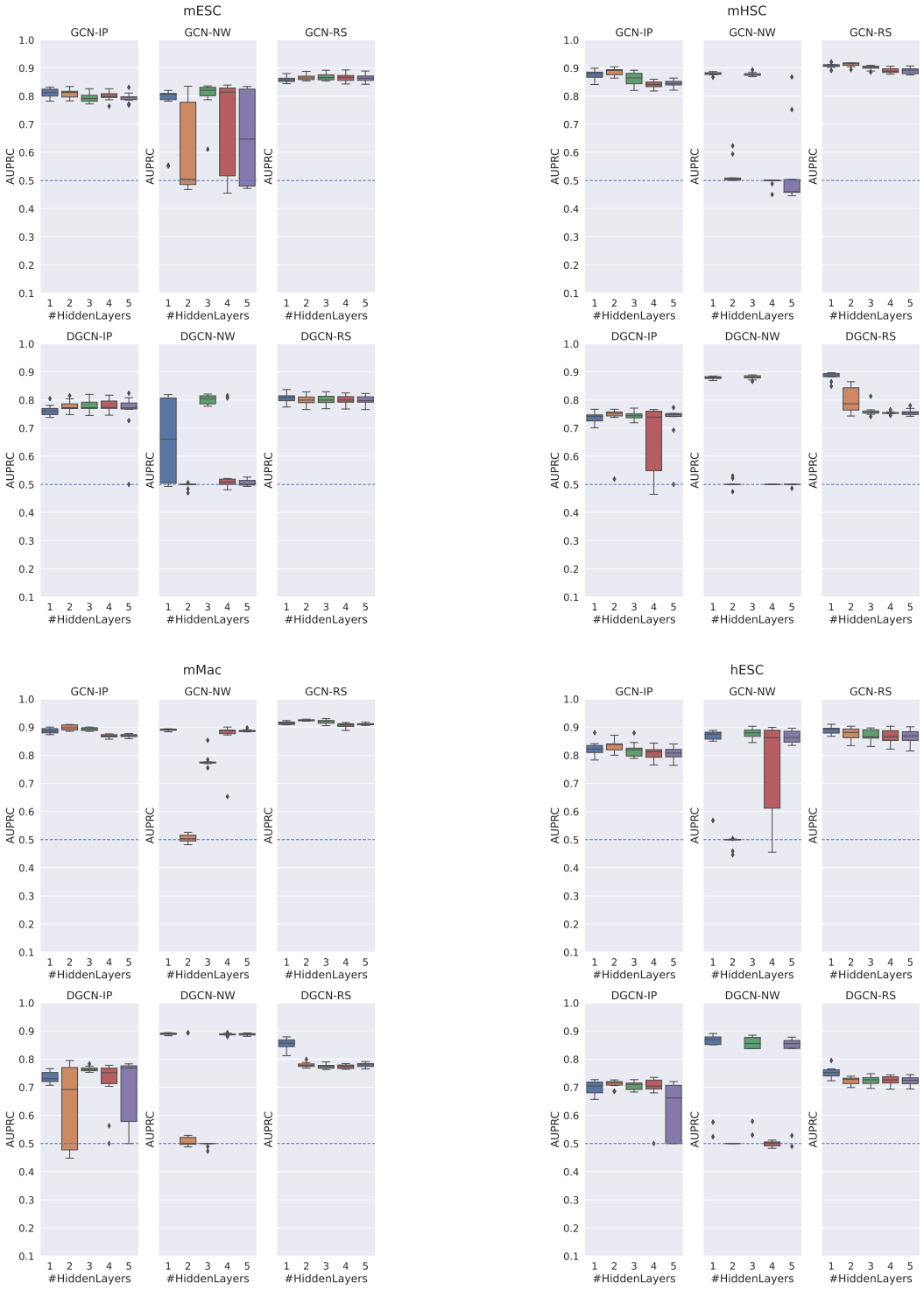


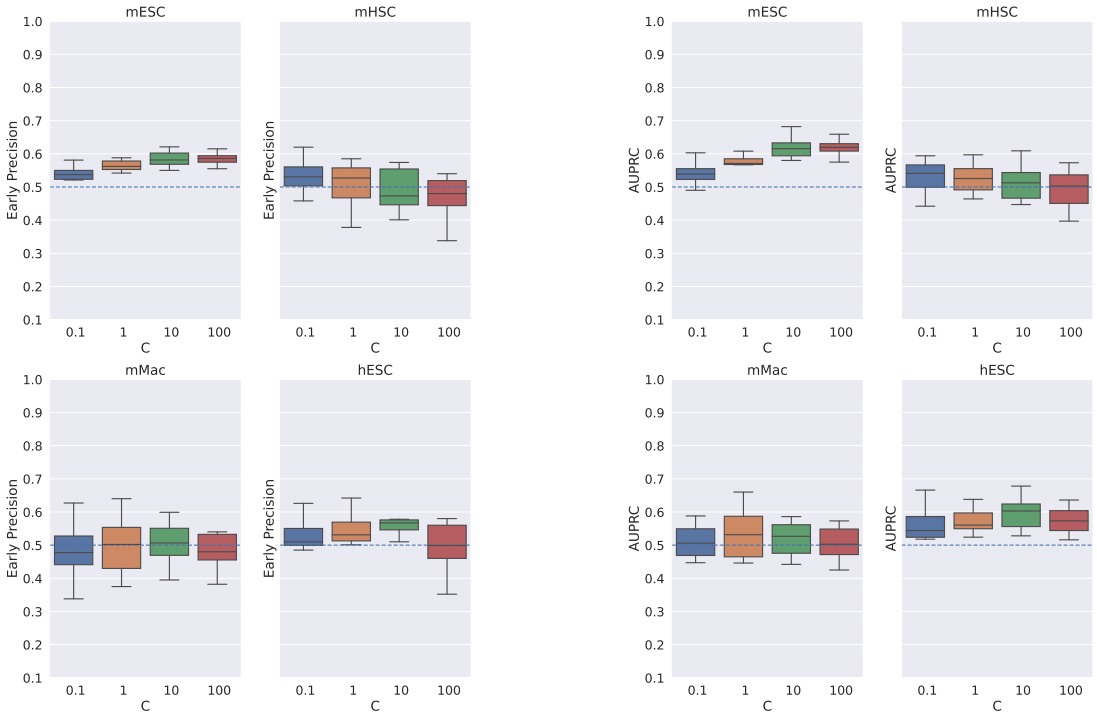
Figure 6.4: AUPRC scores for six GCN-based encoders and decoders: GCN-IP, GCN-NW, GCN-RS, DGCN-IP, DGCN-NW and DGCN-RS evaluated on mESC, mHSC, mMac and hESC for for $1, 2, 3, 4$, and 5 hidden layers. These results are for edge splits.



6.2 Hyperparameter Search for Node Split Evaluation

In this section, we present the results for node-split evaluation as discussed in Section 4.1. Similar to edge-split evaluation, we report our results on four datasets: mESC, mHSC, mMac, and hESC and a train:validate:test ratio of 80:10:10.

Figure 6.5: Early Precision and AUPRC scores for SVM-C evaluated for different values of $C = [0.1, 1, 10, 100]$ for Node-split evaluation.



(a) Early Precision

(b) AUPRC

6.2.1 SVM-C

We explore Figure 6.5a containing the EP scores for different values of cost parameter C. We would like to note here that the range of EP values are significantly worse than the range for EP (and AUPRC) for Edge-split evaluation (refer Figure 6.1). While the median EP scores for Edge-split evaluation is anywhere between 0.7-0.85 for different values of cost

parameter, in node-split evaluation, we see that the median EP scores struggle to cross beyond a modest 0.1 over random. We observe that from $C = 0.1$ to 10, median EP improves but only marginally. At $C = 100$, median EP decreases, except for mESC.

With these observations and the fact that C controls the trade-off between errors on training data and margin maximization, we can say that SVM-C struggles to find a *good* hyperplane that divides the data without loss of generalization. We should also make a note here that the node-split evaluation is a challenging setting that provides limited information about other nodes due to the nature of the split, which otherwise may have been connected by *some* edges in the original graph. We consider $C = 1$ for our analyses on the performance of these predictors for highly variable genes in node-split evaluation.

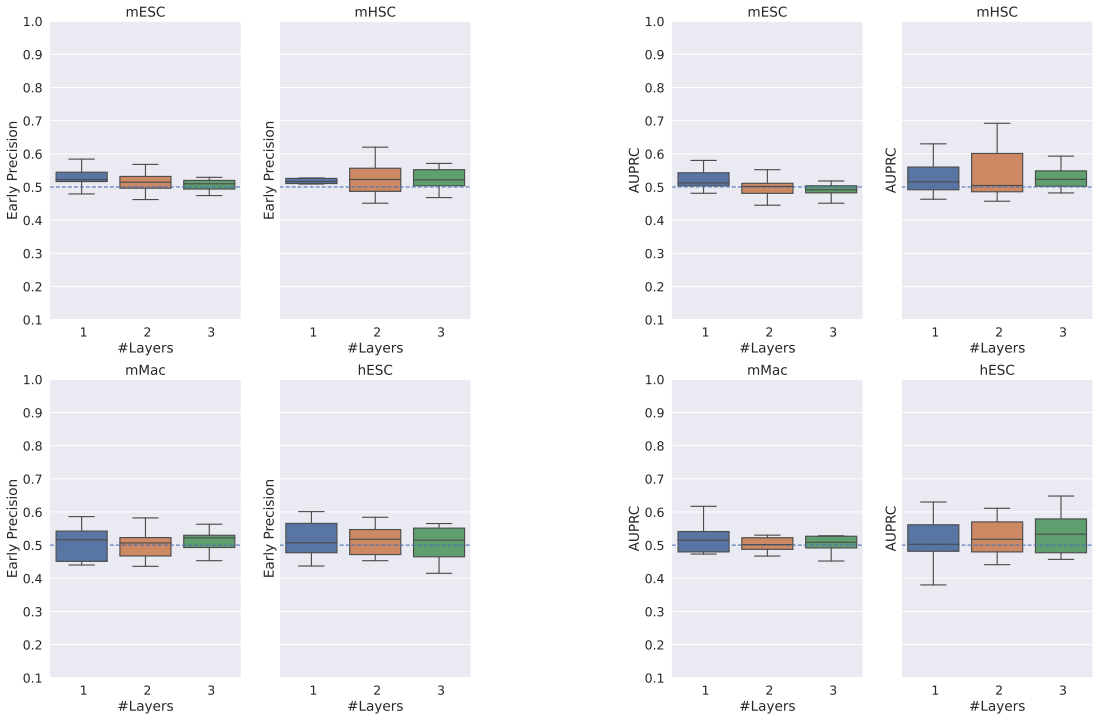
6.2.2 MLP

We refer to Figure 6.6a for this subsection. As we increase the number of layers for the MLP model from 1 to 3, median early precision improves minimally. We make an observation that performance of MLP is undistinguished in both of the settings (*node-split* and *edge-split*) of our evaluation. We use a 1-layer MLP due to the lack of improvement as we stack on more layers for the next analysis.

6.2.3 GCN-based Encoders and Decoders

For GCN-based encoders and decoders, we considered directed and undirected GCN as the encoders and two decoders, namely, IP and RESCAL. We refer to early precision scores in Figure 6.7. It is clear from the plots that DGCN encoders perform better than their GCN counterparts. Considering the decoders, DGCN-RS outperforms DGCN-IP for all the four datasets. Therefore, we focused our attention on selecting the number of hidden layers for DGCN-RS for subsequent analyses. We noticed that the results for two layers were better

Figure 6.6: Early Precision and AUPRC scores for MLP evaluated on 1, 2 and 3 layers for Node-split evalution.



(a) Early Precision (b) AUPRC

than for one layer. There seemed an improvement in the performance even when we increased the number of layers further. We perform a Mann-Whitney U Test to that effect. The EP distributions for three layers and four layers were not statistically distinguishable (p-value 0.05 for the Mann-Whitney test). The AUPRC distributions were not distinguishable either (p-value 0.02). Therefore, we used a three-layer DGCN-RS going forward.

Figure 6.7: Early Precision scores for four GCN-based encoders and decoders: GCN-IP, GCN-RS, DGCN-IP and DGCN-RS evaluated on mESC, mHSC, mMac and hESC for hidden layers in $[1, 2, 3, 4, 5]$. The results are reported for Node-split evaluation setting.

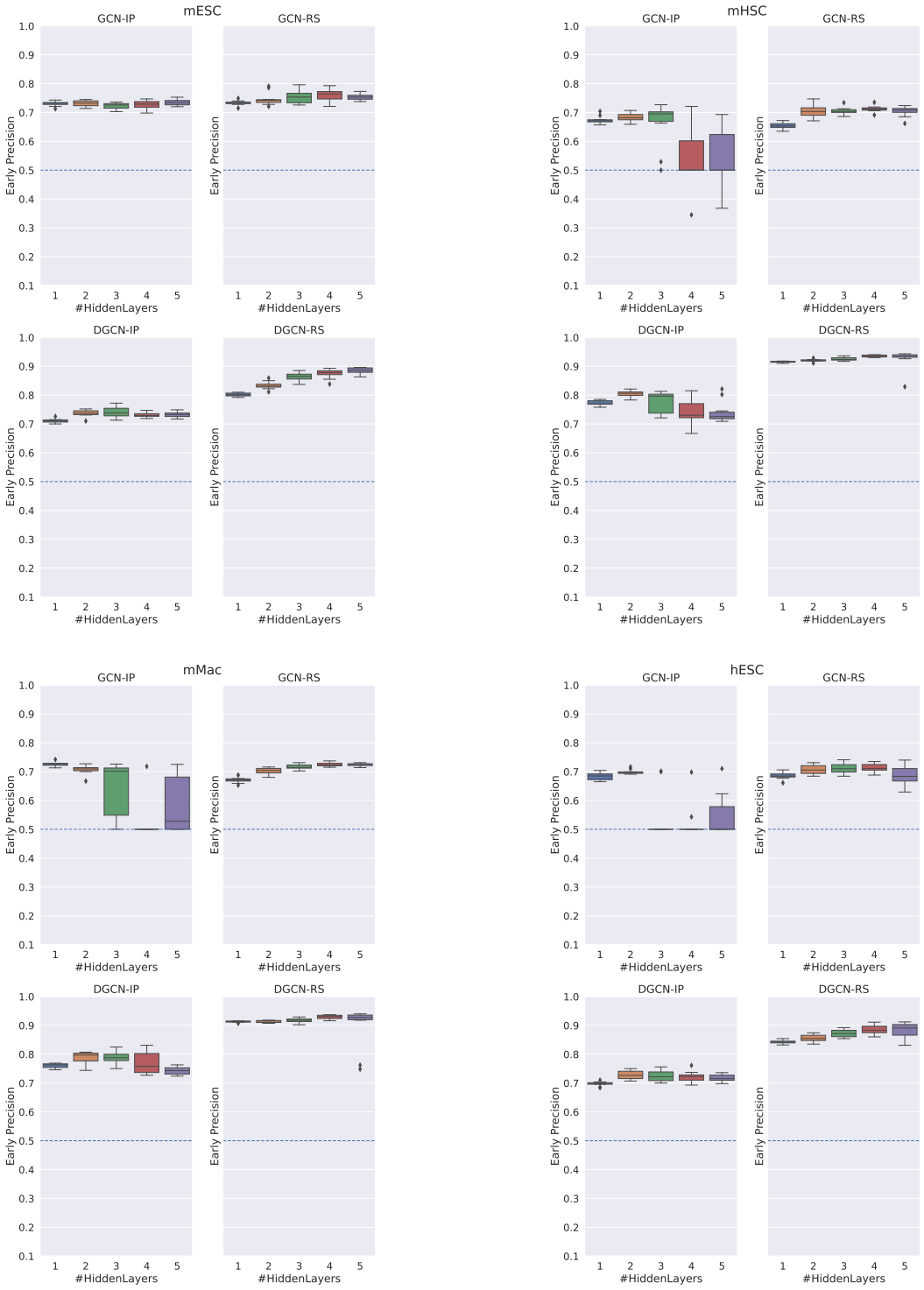
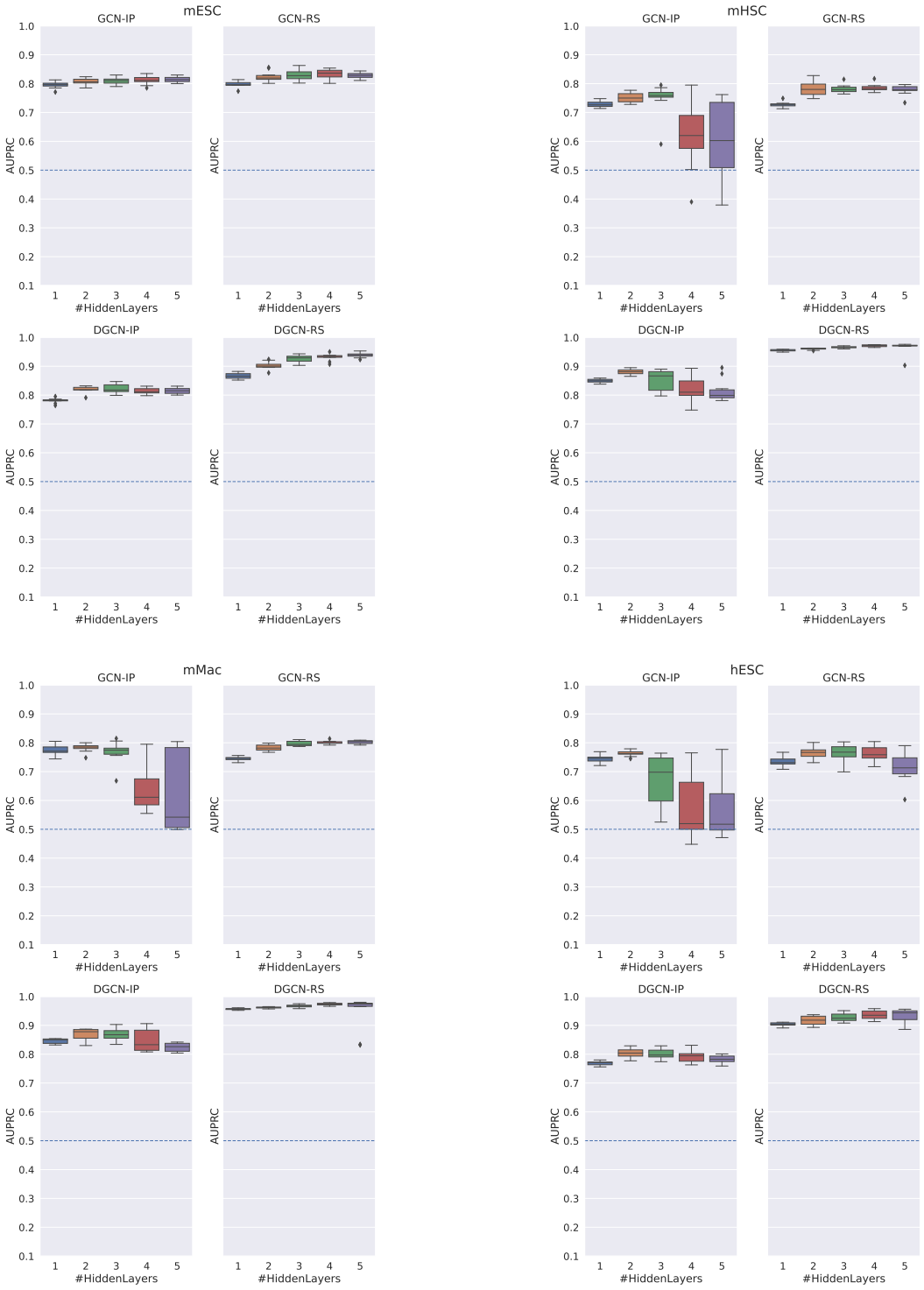


Figure 6.8: AUPRC scores for four GCN-based encoders and decoders: GCN-IP, GCN-RS, DGCN-IP and DGCN-RS evaluated on mESC, mHSC, mMac and hESC for hidden layers in $[1, 2, 3, 4, 5]$.



6.3 Evaluation on Independent Datasets

In the previous analysis, we used four datasets to establish the best GCN encoder-decoder combination and to select hyperparameter(s) for this architecture and for SVM-C and MLP. Now we compare the performance of these methods and of CNNC on two independent datasets: mEpi and hPlacenta. For each dataset, we selected the 2,000 most highly variable genes as discussed in Section 5.2 and inferred GRNs for the subset of TFs and genes that were also members of the corresponding ground-truth network.

We then selected the best performing model for 2,000 genes in the node-split evaluation. To assess the performance of this model on the number of genes variation, we evaluated it as we varied the number of HVGs from 2000 to 4000, 5000 and 6000.

6.3.1 Performance for Edge-split Evaluation and 2,000 Genes

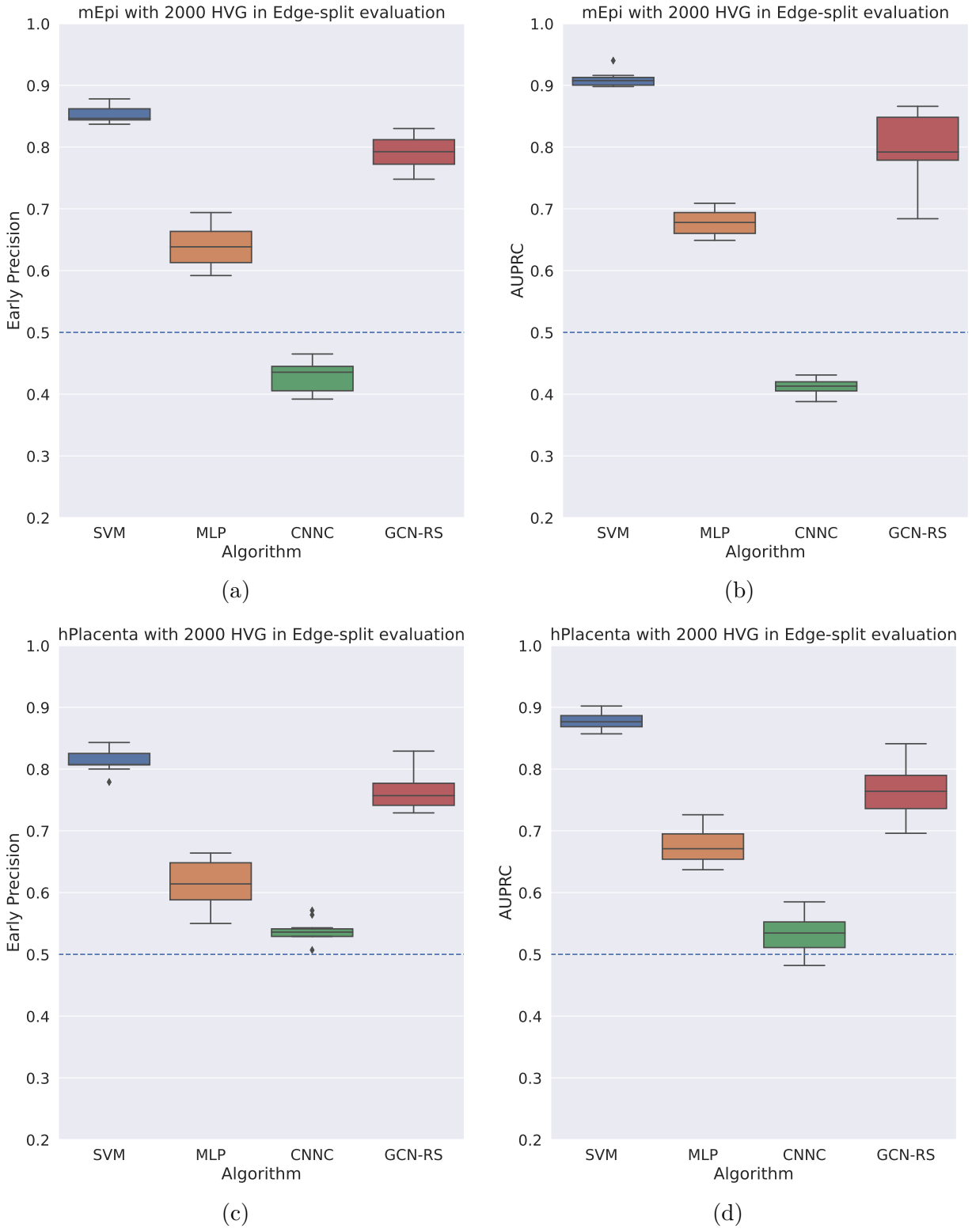
Figures 6.9a and 6.9c display the EP scores for SVM-C with $C = 100$, 1-layer MLP, CNNC and GCN-RS with 2 hidden layers, evaluated on mEpi and hPlacenta dataset with 2,000 highly varying genes.

We observe from these plots that SVM-C is the best performing model with median EP between 0.8 and 0.85, for both datasets. GCN-RS has the second best performance with median EP close to 0.8. CNNC and MLP perform close to a random predictor with CNNC performing worse than random for mEpi. The trends are similar when we refer to the AUPRC scores.

It is interesting to see that SVM-C outperforms CNNC, a supervised method developed specifically for GRN inference from scRNA-seq datasets. Since SVM-C uses a linear kernel to find linearly separable data in high-dimension, we may say that the SVM-C is able to find a good high-dimensional representation of the edges of the graph that is separable by a hyperplane. GCN-RS, which is based on convolutions specifically for graph structured

data comes second close to SVM-C. It was interesting that an off-the-shelf classifier such as SVM succeeded in outperforming deep learning-based classifiers such as GCNs and CNNC.

Figure 6.9: Early Precision and AUPRC scores for SVM-C, MLP, CNNC and GCN-RS for mEpi and hPlacenta dataset on edge-split evaluation.



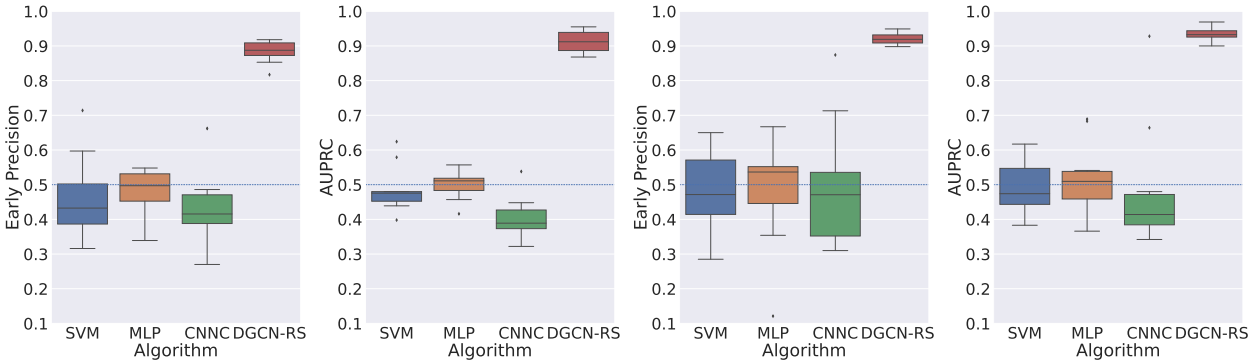
6.3.2 Performance for Node-Split Evaluation and 2,000 genes

Next, we turned our attention to the node-split evaluation. As mentioned before, the node split method of creating training, validation and test splits is more challenging and realistic than edge splitting as the graphs in each of these sets do not overlap either in nodes or in edges. Since a node in a training set is not connected to any node in the validation or the test set, we used the GCN architecture to work in this *inductive* setting. Here, we leverage the features of one set of nodes to effectively learn the weights of the GCN in order to compute embeddings for a different set of nodes. Specifically, we use the features of the nodes present *only* in the training set to learn the weights of the layers in the GCN. We then use these weights to compute embeddings for nodes either in the validation or the test set. Since all the nodes in the original graph have feature vectors of the same dimensionality, we could leverage the learned weights during training to be applied to unseen nodes.

Figures 6.10a and 6.10c display the EP scores of all the GRN inference methods. The result is exceptional as DGCN-RS with three layers has median EP values over 0.9 for both datasets, while the other algorithms perform at random or even worse than random. This suggests that DGCN-RS method is capable of applying the weights learned during training to successfully predict on unseen graphs.

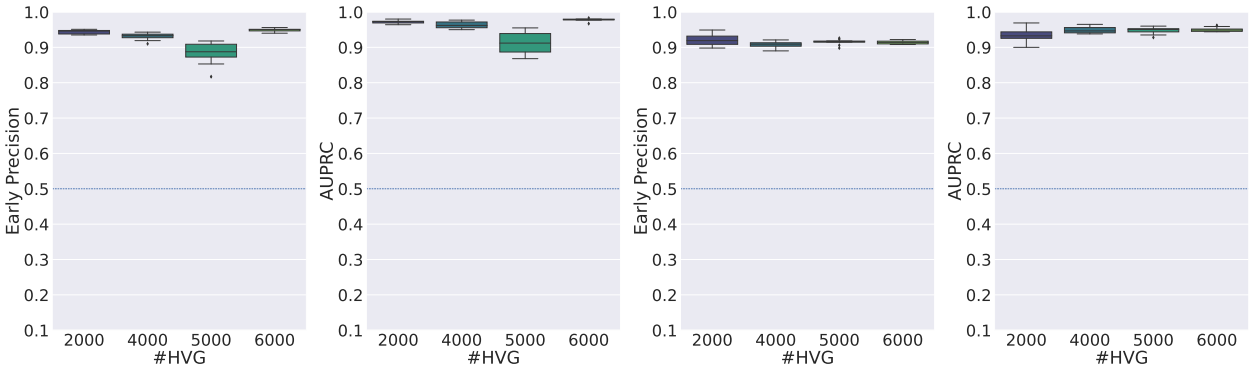
Due to the excellent performance of DGCN-RS compared to the rest of the algorithms, we decided to test it as we increased the number of highly-varying genes for both mEpi and hPlacenta. We note the results in Figure 6.11, as this number changed from 2000 to 4000, 5000, and 6000. We noted that DGCN-RS maintains a median EP close to 0.9. For the mEpi dataset, we noticed a decrease in EP and AUPRC for 5000 genes, but the drop is not significant enough to raise concerns over the method itself. Moreover, we observe that the median AUPRC scores were well above 0.95 for both the datasets (except for mEpi with 5,000 highly-varying genes). The directed GCN encoder enabling informa-

Figure 6.10: Early Precision and AUPRC scores for SVM-C, MLP, CNNC and GCN-RS for mEpi and hPlacenta dataset evaluated on Node-split method.



(a) EP values for the mEpi dataset. (b) AURPC values for the mEpi dataset. (c) EP values for the hPlacenta dataset. (d) AUPRC values for the hPlacenta dataset.

Figure 6.11: Early Precision and AUPRC scores for DGCN-RS with three layers on the mEpi and hPlacenta datasets with different numbers of highly varying genes on node-split evaluation.



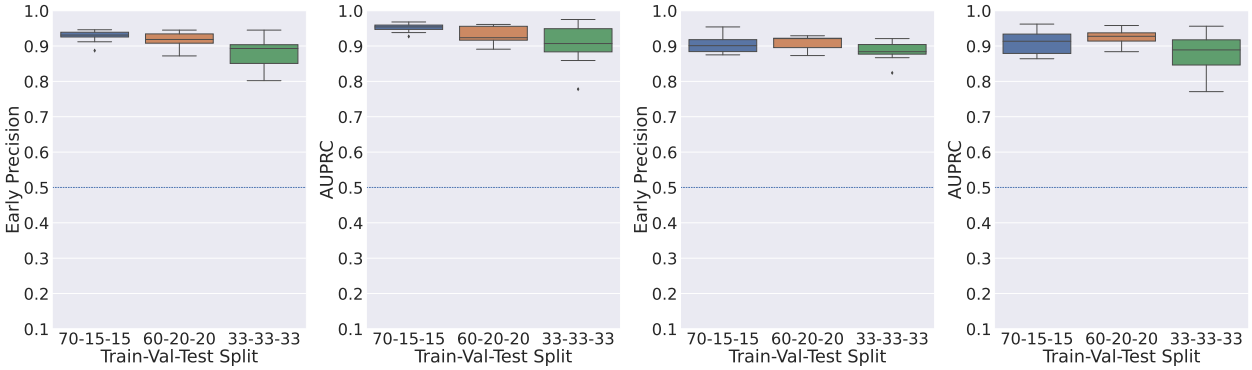
(a) EP values for the mEpi dataset. (b) AUPRC values for the mEpi dataset. (c) EP values for the hPlacenta dataset. (d) AUPRC values for the hPlacenta dataset.

tion flow in one directions between a TF and a gene and the parametric RESCAL decoder which learns the TF's targets in the lower dimensional embedding seem to be a generally good encoder-decoder for *inductive tasks* given the striking performance of DGCN-RS in this method.

6.4 Varying the Train-Val-Test Ratio in Node-Split Evaluation

In this section, we describe the effect of limiting the amount of training data on the performance of DGCN-RS. We expected the EP and AUPRC scores to decrease as we lowered the number of training examples. We followed the experimental setup described in Section 4.1 for node-split evaluation, but we varied the percentage training nodes from 80% to 70%, 60%, and even as low as 33%. We plot the remaining nodes equally into validation and test sets. We report the performance results in Figure 6.12 for mEpi and hPlacenta dataset for 2,000 highly-varying genes. As expected, we see median EP and median AUPRC scores decrease as we tighten the number of training nodes but it is notable that the drop is minimal. More so, even with as few as 33% of nodes in the training set, we observe that the median scores are still close to 0.9.

Figure 6.12: Early Precision and AUPRC scores for DGCN-RS on mEpi and hPlacenta dataset (2000 HVG) with different train:val:test ratios.



(a) EP values for the mEpi dataset. (b) AUPRC values for the mEpi dataset. (c) EP values for the hPlacenta dataset. (d) AUPRC values for the hPlacenta dataset.

6.5 Running Time of Algorithms

In this section, we note down the running time i.e. train and test time of the algorithms averaged over 10 runs for the largest dataset (mMac) as we vary the hyperparameters. We report the results in Table 6.1. For GCN encoders, the running time is less than for their DGCN counterparts. Furthermore, RS decoders take more time than IP decoders. This trend is not surprising because RS decoders learn more parameters. For SVM-C, we observe that as we increase the value of C, the running time increases. For MLP, the number of layers does not seem to have an impact on the running time. The running time for CNNC is comparable to MLP.

Similarly, we note the trends for running time as we increase the number of highly variable genes for mEpi and hPlacenta dataset in Table 6.2 and Table 6.3 respectively. It is obvious that as we increase the number of highly variable genes, the graph size increases leading to a greater running time.

Table 6.1: Running time of mMac dataset for GCN-based autoencoders, SVM-C, MLP and CNNC. Running time includes the training and test time averaged over 10 runs.

(a) GCN-based autoencoders

Algorithm	#Layers=1	#Layers=2	#Layers=3	#Layers=4	#Layers=5
GCN-IP	7.8min	11.6min	8.3min	8.1min	7.9min
GCN-RS	6min	12.3min	10.4min	11.7min	13.9min
DGCN-IP	18.7min	19.9min	20.8min	21.6min	23.3min
DGCN-RS	44.5min	46.3min	47.5min	49.2min	49.9min

(b) MLP

Algorithm	#Layers=1	#Layers=2	#Layers=3
MLP	14.28 min	15.29 min	14.5 min

(c) SVM-C

Algorithm	C=0.1	C=1	C=10	C=100
SVM-C	4.13 min	7.7 min	34.6 min	72.83 min

(d) CNNC

Algorithm	
CNNC	13.8 min

Table 6.2: Runtime for the mEpi dataset with different number of highly variable genes.

#Highly Variable Genes	**SVM-C**	**MLP**	**CNNC**	**DGCN-RS**
2000	20.8 seconds	4.7 minutes	4.69 minutes	2.89 minutes
4000	56.9 seconds	8.9 minutes	6.34 minutes	3.83 minutes
5000	84.05 seconds	9.55 minutes	7.22 minutes	8.59 minutes
6000	124.9 seconds	14.2 minutes	10.78 minutes	10.32 minutes

Table 6.3: Runtime for the hPlacenta dataset with different number of highly variable genes.

#Highly Variable Genes	**SVM-C**	**MLP**	**CNNC**	**DGCN-RS**
2,000	10.9 seconds	5.38 minutes	3.2 minutes	45.9 seconds
4,000	29 seconds	4.22 minutes	4.83 minutes	1.67 minutes
5,000	47.1 seconds	5.88 minutes	5.65 minutes	2.23 minutes
6,000	70.7 seconds	8.61 minutes	7.12 minutes	2.49 minutes

Chapter 7

Conclusions and Future Work

We began this thesis by noting that inferring GRNs accurately from experimental data is an important step to more downstream tasks in biomedicine such as disease gene prediction (Chapter 1.1). We summarized the different computational methods in the literature that have been in use to infer GRNs in Section 1.2 and described them more broadly in Chapter 2. We then examined the rise of the recent measurement technology called scRNA-seq in Section 1.2.1 and discussed how scRNA-seq measurements are a rich data to reconstruct GRNs from. We studied the drawbacks of the computational methods in Section 1.2.2. In Chapter 2, we went over the limitations of the unsupervised methods of inferring GRNs. We saw that graph machine learning and graph convolutional networks are gaining great popularity in bioinformatics and have shown potential for a diverse array of predictive tasks (see Section 1.2.3).

7.1 Summary of Contributions

Driven by these motivating observations, we developed a graph convolutional approach with different encoders and decoders (Sections 3.2 and 3.3) and studied their performance in inferring GRNs from scRNA-seq data. We described two approaches, the edge-split and the node-split method for evaluating GRN inference algorithms (Section 4.1). We benchmarked GCN-based encoders and decoders against three supervised learning approaches, SVM, MLP, and CNNC (Section 2.5). We performed hyperparameter search for SVM-C, MLP and GCN-

based encoders for both the evaluations in Section 6.1.3 and Section 6.2.3.

On further evaluation on independent datasets, we saw that in edge-split evaluation, SVM-C outperformed the other approaches with GCN-based approaches coming in a close second (Section 6.3.1). In node-split evaluation, which is a realistic and challenging setting of splitting the graph into disjoint sets, we observed that DGCN-RS outperformed all other algorithms consistently and by a large margin (Section 6.2), highlighting the effectiveness of graph machine learning over other machine learning methods.

We then tested the performance of the outstanding DGCN-RS method for different ratios of train, validation and test set sizes. We observed that even with close to just a third of dataset as the training set, the performance of DGCN-RS did not drop below 0.9 AUPRC and EP (see Section 6.4). Overall, our work provides strong evidence for the utility of graph convolutional techniques to infer GRNs from experimental data.

7.2 Future Work

The takeaways observed in the previous section open up some interesting avenues for future work. We list them briefly below before posing a challenging problem.

- In this work, we varied the train:validation:test ratio between 80:10:10, 70:15:15, 60:20:20, and 33:33:33. It will be interesting to explore the ratio of positive:negative training examples from current setting of 1:1 to 1:5 and 1:10.
- We observed a trend in our results with the NW decoder that we were unable to explain. The EP and AUPRC scores with the NW decoder for certain layers was at 0.5. This trend is contrary to what we observed for other decoders. Understanding the reason for this irregular performance may shed light on this decoder.
- In the node-split setting, we were not able to use the NW decoder, since it learns

a weight for every node in the network. It would be useful to design a version of this decoder that can be independently applied to a set of nodes not observed during training.

- We have used datasets that have been preprocessed differently owing to how we sourced them. Specifically, the datasets we took from BEELINE [46] were preprocessed differently from the other two datasets. Ideally, we would like to evaluate on datasets that are be pre-processed by the same pipeline.

- We have used GCNs [22] as our convolutional encoder in this study. There have been advances in the field such as Graph Attention Networks [58] and GraphSAGE [11]. We leave it to future work to explore the effects of these new techniques in the encoder.

In this thesis, we trained GCNs to learn non-cell type specific networks. However, TF-gene interactions can vary considerably from one cell type to another. Developing GCN-based methods that are trained on non-specific GRNs but can then be used to predict tissue specific networks is a challenging open problem.

Bibliography

[1] A. Alavi, M. Ruffalo, A. Parvangada, Z. Huang, and Z. Bar-Joseph. A web server for comparative analysis of single-cell RNA-seq data. *Nature Communications*, 9(1):4768, 11 2018.

[2] J. Ambroise, A. Robert, B. Macq, and J. L. Gala. Transcriptional network inference from functional similarity and expression data: a global supervised approach. *Stat Appl Genet Mol Biol*, 11(1):Article 2, Jan 2012.

[3] P. C. Aubin-Frankowski and J. P. Vert. Gene regulation inference from single-cell RNA-seq data with linear differential equations and velocity inference. *Bioinformatics*, 36(18):4774–4780, 09 2020.

[4] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy, 2021.

[5] L. F. Chu, N. Leng, J. Zhang, Z. Hou, D. Mamott, D. T. Vereide, J. Choi, C. Kendziorski, R. Stewart, and J. A. Thomson. Single-cell RNA-seq reveals novel regulators of human embryonic stem cell differentiation to definitive endoderm. *Genome Biol*, 17(1):173, 08 2016.

[6] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[7] Alex M Fout. *Protein interface prediction using graph convolutional networks*. PhD thesis, Colorado State University, 2017.

[8] Nir Friedman and Moisés Goldszmidt. Learning bayesian networks with local structure. *CoRR*, abs/1302.3577, 2013. URL http://arxiv.org/abs/1302.3577.

[9] Pierre Geurts et al. dyngenie3: dynamical genie3 for the inference of gene networks from time series expression data. *Scientific Reports*, 8(1):1–12, 2018.

[10] A. L. Haber, M. Biton, N. Rogel, R. H. Herbst, K. Shekhar, C. Smillie, G. Burgin, T. M. Delorey, M. R. Howitt, Y. Katz, I. Tirosh, S. Beyaz, D. Dionne, M. Zhang, R. Raychowdhury, W. S. Garrett, O. Rozenblatt-Rosen, H. N. Shi, O. Yilmaz, R. J. Xavier, and A. Regev. A single-cell survey of the small intestinal epithelium. *Nature*, 551(7680):333–339, 11 2017.

[11] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[12] T. Hayashi, H. Ozaki, Y. Sasagawa, M. Umeda, H. Danno, and I. Nikaido. Single-cell full-length total RNA sequencing uncovers dynamics of recursive splicing and enhancer RNAs. *Nat Commun*, 9(1):619, 02 2018.

[13] M. Hecker, S. Lambeck, S. Toepfer, E. van Someren, and R. Guthke. Gene regulatory network inference: data integration in dynamic models-a review. *Biosystems*, 96(1): 86–103, Apr 2009.

[14] B. Hie, B. Bryson, and B. Berger. Efficient integration of heterogeneous single-cell transcriptomes using Scanorama. *Nat Biotechnol*, 37(6):685–691, 06 2019.

[15] Yu-an Huang, Pengwei Hu, Keith CC Chan, and Zhu-Hong You. Graph convolution for predicting associations between mirna and drug resistance. *Bioinformatics*, 36(3): 851–858, 2020.

[16] Dirk Husmeier, Richard Dybowski, and Stephen Roberts. *Probabilistic modeling in bioinformatics and medical informatics*. Springer Science & Business Media, 2006.

[17] V. A. Huynh-Thu and G. Sanguinetti. Gene Regulatory Network Inference: An Introductory Survey. *Methods Mol Biol*, 1883:1–23, 2019.

[18] V. A. Huynh-Thu, A. Irrthum, L. Wehenkel, and P. Geurts. Inferring regulatory networks from expression data using tree-based methods. *PLoS One*, 5(9), Sep 2010.

[19] M. Kaern, W. J. Blake, and J. J. Collins. The engineering of gene regulatory networks. *Annu Rev Biomed Eng*, 5:179–206, 2003.

[20] S. Kauffman. Homeostasis and differentiation in random genetic control networks. *Nature*, 224(5215):177–178, Oct 1969.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[23] Aleksandra A. Kolodziejczyk, Jong Kyoung Kim, Valentine Svensson, John C. Marioni, and Sarah A. Teichmann. The technology and biology of single-cell rna sequencing. *Molecular Cell*, 58(4):610–620, 2015. ISSN 1097-2765. doi: https://doi.org/10.1016/j.molcel.2015.04.005. URL https://www.sciencedirect.com/science/article/pii/S1097276515002610.

[24] Peter Langfelder and Steve Horvath. WGCNA: an R package for weighted correlation network analysis. *BMC Bioinformatics*, 9(1):1–13, 2008.

[25] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems*, pages 253–256. IEEE, 2010.

[26] T. Lenoir and E. Giannella. The emergence and diffusion of DNA microarray technology. *J Biomed Discov Collab*, 1:11, Aug 2006.

[27] Peng Li, Chaoyang Zhang, Edward J Perkins, Ping Gong, and Youping Deng. Comparison of probabilistic boolean network and dynamic bayesian network approaches for inferring gene regulatory networks. In *BMC Bioinformatics*, volume 8, pages 1–8. Springer, 2007.

[28] Y. Li, C. Huang, L. Ding, Z. Li, Y. Pan, and X. Gao. Deep learning in bioinformatics: Introduction, application, and perspective in the big data era. *Methods*, 166:4–21, 08 2019.

[29] S. Liang, S. Fuhrman, and R. Somogyi. Reveal, a general reverse engineering algorithm for inference of genetic network architectures. *Pac Symp Biocomput*, pages 18–29, 1998.

[30] F. Liu, S. W. Zhang, W. F. Guo, Z. G. Wei, and L. Chen. Inference of Gene Regulatory Network Based on Local Bayesian Networks. *PLoS Comput Biol*, 12(8): e1005024, 08 2016.

[31] F. Liu, S. W. Zhang, W. F. Guo, Z. G. Wei, and L. Chen. Inference of Gene Regulatory Network Based on Local Bayesian Networks. *PLoS Comput Biol*, 12(8): e1005024, 08 2016.

[32] Malte D Luecken and Fabian J Theis. Current best practices in single-cell RNA-seq analysis: A tutorial. *Molecular Systems Biology*, 15(6):e8746, 2019.

[33] Aaron TL Lun, Davis J McCarthy, and John C Marioni. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Research*, 5, 2016.

[34] Baoshan Ma, Mingkun Fang, and Xiangtian Jiao. Inference of gene regulatory networks based on nonlinear ordinary differential equations. *Bioinformatics*, 36(19): 4885–4893, 2020.

[35] Jianzhu Ma, Michael Ku Yu, Samson Fong, Keiichiro Ono, Eric Sage, Barry Demchak, Roded Sharan, and Trey Ideker. Using deep learning to model the hierarchical structure and function of a cell. *Nature Methods*, 15(4):290–298, 2018.

[36] Adam A Margolin, Ilya Nemenman, Katia Basso, Chris Wiggins, Gustavo Stolovitzky, Riccardo Dalla Favera, and Andrea Califano. ARACNE: An algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC Bioinformatics*, 7(1):1–15, 2006.

[37] S. C. Materna and E. H. Davidson. Logic of gene regulatory networks. *Curr Opin Biotechnol*, 18(4):351–354, Aug 2007.

[38] H. Matsumoto, H. Kiryu, C. Furusawa, M. S. H. Ko, S. B. H. Ko, N. Gouda, T. Hayashi, and I. Nikaido. SCODE: An efficient regulatory network inference algorithm from single-cell RNA-Seq during differentiation. *Bioinformatics*, 33(15): 2314–2321, Aug 2017.

[39] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[40] Daniele Mercatelli, Laura Scalambra, Luca Triboli, Forest Ray, and Federico M. Giorgi. Gene regulatory network inference resources: A practical overview. *Biochimica*

et Biophysica Acta (BBA) - Gene Regulatory Mechanisms, 1863(6):194430, 2020. ISSN 1874-9399. doi: https://doi.org/10.1016/j.bbagrm.2019.194430. URL https://www.sciencedirect.com/science/article/pii/S1874939919300410. Transcriptional Profiles and Regulatory Gene Networks.

[41] Thomas Moerman, Sara Aibar Santos, Carmen Bravo González-Blas, Jaak Simm, Yves Moreau, Jan Aerts, and Stein Aerts. Grnboost2 and arboreto: efficient and scalable inference of gene regulatory networks. *Bioinformatics*, 35(12):2159–2161, 2019.

[42] F. Mordelet and J. P. Vert. Supervised inference of gene regulatory networks from positive and unlabeled examples. *Methods Mol Biol*, 939:47–58, 2013.

[43] H. Nguyen, D. Tran, B. Tran, B. Pehlivan, and T. Nguyen. A comprehensive survey of regulatory network inference methods using single cell RNA sequencing data. *Brief Bioinform*, 22(3), May 2021.

[44] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing YAGO: Scalable machine learning for linked data. In *Proceedings of the 21st International Conference on World Wide Web*, pages 271–280, 2012.

[45] S. Picelli, O. R. Faridani, A. K. Björklund, G. Winberg, S. Sagasser, and R. Sandberg. Full-length RNA-seq from single cells using Smart-seq2. *Nat Protoc*, 9(1):171–181, Jan 2014.

[46] A. Pratapa, A. P. Jalihal, J. N. Law, A. Bharadwaj, and T. M. Murali. Benchmarking algorithms for gene regulatory network inference from single-cell transcriptomic data. *Nat Methods*, 17(2):147–154, 2020.

[47] X. Qiu, A. Hill, J. Packer, D. Lin, Y. A. Ma, and C. Trapnell. Single-cell mRNA

quantification and differential analysis with Census. *Nature Methods*, 14(3):309–315, 2017.

[48] Zahra Razaghi-Moghadam and Zoran Nikoloski. Supervised learning of gene-regulatory networks based on graph distance profiles of transcriptomics data. *NPJ Systems Biology and Applications*, 6(1):1–8, 2020.

[49] Blagoj Ristevski. A survey of models for inference of gene regulatory networks. *Nonlinear Analysis: Modelling and Control*, 18(4):444–465, 2013.

[50] A. Sagner, Z. B. Gaber, J. Delile, J. H. Kong, D. L. Rousso, C. A. Pearson, S. E. Weicksel, M. Melchionda, S. N. Mousavy Gharavy, J. Briscoe, and B. G. Novitch. Olig2 and Hes regulatory dynamics during motor neuron differentiation revealed by single cell transcriptomics. *PLoS Biol*, 16(2):e2003127, 02 2018.

[51] M. Sanchez-Castillo, D. Blanco, I. M. Tienda-Luna, M. C. Carrion, and Y. Huang. A Bayesian framework for the inference of gene regulatory networks from time and pseudo-time series data. *Bioinformatics*, 34(6):964–970, 03 2018.

[52] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[53] T. Schlitt and A. Brazma. Current approaches to gene regulatory network modelling. *BMC Bioinformatics*, 8 Suppl 6:S9, Sep 2007.

[54] Vikash Singh and Pietro Lio. Towards probabilistic generative models harnessing graph neural networks for disease-gene prediction. *arXiv preprint arXiv:1907.05628*, 2019.

[55] Lin Song, Peter Langfelder, and Steve Horvath. Comparison of co-expression measures: mutual information, correlation, and model based indices. *BMC Bioinformatics*, 13(1):1–21, 2012.

[56] F. Tang, C. Barbacioru, Y. Wang, E. Nordman, C. Lee, N. Xu, X. Wang, J. Bodeau, B. B. Tuch, A. Siddiqui, K. Lao, and M. A. Surani. mRNA-Seq whole-transcriptome analysis of a single cell. *Nat Methods*, 6(5):377–382, May 2009.

[57] Masashi Tsubaki, Kentaro Tomii, and Jun Sese. Compound–protein interaction prediction with end-to-end learning of neural networks for graphs and sequences. *Bioinformatics*, 35(2):309–318, 2019.

[58] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[59] R. Vento-Tormo, M. Efremova, R. A. Botting, M. Y. Turco, M. Vento-Tormo, K. B. Meyer, J. E. Park, E. Stephenson, K. Polański, A. Goncalves, L. Gardner, S. Holmqvist, J. Henriksson, A. Zou, A. M. Sharkey, B. Millar, B. Innes, L. Wood, A. Wilbrey-Clark, R. P. Payne, M. A. Ivarsson, S. Lisgo, A. Filby, D. H. Rowitch, J. N. Bulmer, G. J. Wright, M. J. T. Stubbington, M. Haniffa, A. Moffett, and S. A. Teichmann. Single-cell reconstruction of the early maternal-fetal interface in humans. *Nature*, 563(7731):347–353, 11 2018.

[60] Nedumparambathmarath Vijesh, Swarup Kumar Chakrabarti, Janardanan Sreekumar, et al. Modeling of gene regulatory networks: A review. *Journal of Biomedical Science and Engineering*, 6(02):223, 2013.

[61] Juexin Wang, Anjun Ma, Qin Ma, Dong Xu, and Trupti Joshi. Inductive inference of gene regulatory network using supervised and semi-supervised graph neural networks. *Computational and Structural Biotechnology Journal*, 18:3335–3343, 2020.

[62] L. F. Wessels, E. P. van Someren, and M. J. Reinders. A comparison of genetic network models. *Pac Symp Biocomput*, pages 508–519, 2001.

[63] F. A. Wolf, P. Angerer, and F. J. Theis. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biol*, 19(1):15, 02 2018.

[64] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans Neural Netw Learn Syst*, 32(1):4–24, Jan 2021.

[65] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans Neural Netw Learn Syst*, 32(1):4–24, Jan 2021.

[66] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[67] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018.

[68] Ye Yuan and Ziv Bar-Joseph. Deep learning for inferring gene relationships from single-cell expression data. *Proceedings of the National Academy of Sciences*, 116(52): 27151–27158, 2019.

[69] Ye Yuan and Ziv Bar-Joseph. Gcng: graph convolutional networks for inferring gene interaction from spatial transcriptomics data. *Genome Biology*, 21(1):1–16, 2020.

[70] Evangelia I Zacharaki. Prediction of protein function using a deep convolutional neural network ensemble. *PeerJ Computer Science*, 3:e124, 2017.

[71] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[72] M. Zhao, W. He, J. Tang, Q. Zou, and F. Guo. A comprehensive overview and critical evaluation of gene regulatory network inference technologies. *Brief Bioinform*, Feb 2021.

[73] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications, 2021.

[74] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.