Benchmarking of Graph Autoencoder models for Gene Regulatory Network inference with prior knowledge

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Harshit Pateria

IISER PUNE

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

May, 2023

Supervisor: Antonio Scialdone

Certificate

This is to certify that this dissertation entitled Benchmarking of Graph Autoencoder models for Gene Regulatory Network inference with prior knowledge towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Harshit Pateriaat Indian Institute of Science Education and Research under the supervision of Antonio Scialdone, PhD, Institute of Epigenetics and Stem Cells (IES), Helmholtz Zentrum Muenchen , during the academic year 2022-2023.

Antonio Scialdone

Antonio Scialdone

Committee:

Antonio Scialdone

M. S. Madhusudhan

This thesis is dedicated to the journey, not just the destination.

Declaration

I hereby declare that the matter embodied in the report entitled Benchmarking of Graph Autoencoder models for Gene Regulatory Network inference with prior knowledge are the results of the work carried out by me at the Department of Data Science, Indian Institute of Science Education and Research, Pune, under the supervision of Antonio Scialdone and the same has not been submitted elsewhere for any other degree.

[illegible]

Harshit Pateria

Acknowledgments

I would like to begin by thanking the almighty coffee bean for giving me the caffeine-fueled energy to power through countless late nights and early mornings spent researching and writing. Without you, my dear coffee, I wouldn't have made it this far.
To my trusty headphones, who have blocked out countless distractions and provided the soundtrack to my academic life, thank you for keeping me focused and motivated.

To my supervisor, Dr Antonio Scialdone, and Marco Stock, a PhD student in the lab, thank you for your expertise, guidance, and unwavering support throughout this process. Your insights and feedback have been invaluable in shaping this work, and your encouragement has kept me going through the ups and downs.
Not to forget my mentor at IISER, Prof Madhusudhan, for everything he has done for me throughout my time at the institute.

To my family and friends, thank you for understanding when I had to cancel plans or miss important events because I was buried under a pile of books and papers. Your unwavering support and encouragement have been invaluable.

Lastly, I would like to thank the internet, without which this thesis would not have been possible. From online databases and scholarly articles to memes and cat videos, the internet has been a constant companion and occasional distraction. Thank you for everything, World Wide Web.

In conclusion, to all the people and things that have contributed to my academic success, thank you from the bottom of my heart (and my caffeine-addled brain).

Abstract

This thesis explores the benchmarking of graph auto-encoders for inferring gene regulatory networks (GRNs) using prior knowledge of known GRNs. Gene regulatory networks are crucial for understanding gene expression and their regulation in various biological processes. However, obtaining experimentally validated GRNs can be expensive and time-consuming. In this thesis, we propose using graph auto-encoders, a type of neural network, to learn the underlying structure of GRNs from gene expression data. We evaluate different types of graph auto-encoders, including the standard Graph Auto-Encoder (GAE), Variational Graph Auto-Encoder (VGAE), Adversarially Regularized Graph Auto-Encoder (ARGA), and Adversarially Regularized Variational Graph Auto-Encoder (ARGVA) to infer GRNs from prior knowledge. Implementation of a trainable decoder shows better results in comparison to the standard. We compare the performance of each autoencoder with respect to the performance and stability of inferred GRNs. This study demonstrates the potential of different architectures of graph auto-encoders for inferring gene regulatory networks using prior knowledge to compare performance and stability in gene network inference.

Contents

Chapter 1

Introduction

In recent years, single-cell RNA sequencing (scRNA-seq) has emerged as a powerful tool for studying gene expression patterns in individual cells. scRNA-seq data can capture the heterogeneity of gene expression across cells, enabling the identification of different cell types and states [1, 2]. However, analyzing scRNA-seq data can be challenging due to the high dimensionality and noise associated with the data. One of the significant goals of scRNA-seq data analysis is to infer the gene regulatory networks (GRNs) that control cellular processes. It provides valuable insights into the complex molecular interactions that regulate cellular processes. Accurate and reliable inference of GRNs can help identify potential therapeutic targets and improve our understanding of cellular processes, leading to novel treatments for various diseases [3].

Inferring GRNs from scRNA-seq data is a challenging problem due to several reasons:

1. scRNA-seq data is noisy and high-dimensional, making identifying true regulatory interactions between genes complex.
2. Gene regulation is complex and can involve direct and indirect interactions between genes, making it difficult to distinguish between them.
3. The regulatory landscape can vary between cell types and states, making it essential to analyze scRNA-seq data at the single-cell level.

Several computational methods have been proposed for inferring GRNs from scRNA-seq data to address these challenges. One of them is Graph autoencoder (GAE) [4], a deep learning-based approach that has shown promising results in inferring GRNs from gene expression data [5]. The GAE algorithm uses graph-based neural networks to encode gene expression data into a low-dimensional representation and then decodes it back to reconstruct the original data. The GAE algorithm can capture complex gene relationships and identify potential regulatory interactions in the data.

In the context of gene expression data, a GRN can be represented as a graph, where nodes represent genes and edges represent regulatory interactions between them. Each edge is weighted based on the strength of the regulatory interaction. The challenge in inferring GRNs from gene expression data is accurately identifying these regulatory interactions.
McCalla et al [6] showed that trying to infer GRNs from expression data alone has been a tall order for researchers to achieve. To combat this shortcoming, we employ the help of prior known GRNs. Incorporating prior knowledge into the inference process can help to reduce false positives and improve the accuracy of the inferred networks [7]. Several methods have been proposed to incorporate prior knowledge into the inference process, including network regularization and graph-based constraints. In this thesis, we investigate the use of GAE for inferring GRNs using prior knowledge of known GRNs as a guide. We focus on the Yeast dataset from Gasch et al [8], which contains single-cell RNA-seq data from yeast cells under different environmental conditions. The Yeast dataset provides a comprehensive view of the yeast transcriptome under different conditions, making it an ideal resource for network inference studies.

The thesis derives from the work done in the lab (unpublished) [5] establishing the use of basic GAE architecture outperforms majority of the existing algorithms. We explore the performance of different GAE architectures on the Yeast dataset. We evaluate the performance of the inferred networks using metrics such as average precision and area under the precision-recall curve and compare the results using the Wilcoxon test. One of the major contributions of our thesis is the development of a trainable decoder algorithm that outperforms the conventionally used inner product graph decoder, which is a linear decoder, in this setting. Our results demonstrate that GAE can accurately infer GRNs from single-cell RNA-seq data when prior knowledge of known GRNs is incorporated into the inference process. Our study highlights the potential of GAE as a powerful tool for inferring GRNs from single-cell RNA-seq data and provides insights into the factors that influence performance. Using prior knowledge of known GRNs can improve the accuracy of the inferred networks and reduce false positives, which is essential for downstream applications such as identifying therapeutic targets. Hence, our findings can have important implications for advancing our understanding of cellular processes and developing novel disease treatments.

In this thesis, we comprehensively explore and analyze the various architectures used in our research and discuss the unique properties and benefits that each one brings to the table. Additionally, we delve into a couple of different layer types and their respective advantages. We also present a new decoder architecture and explain the thought process and reasoning behind its design.
All of our models are benchmarked and we detail the process used to identify the best-performing model. Furthermore, we employ a robust method to validate our findings in a generalized environment and provide in-depth insights on the results to better understand our conclusions.

Chapter 2

Constructing graph autoencoder models

In this chapter, we provide an in-depth overview of the basic components of our model. These components, when combined in various configurations, form the basis of the different models that we will be examining throughout this thesis.
The GAE algorithm, Figure 2.1, consists of two main parts: an encoder and a decoder. The encoder takes as input the gene expression data and produces a low-dimensional embedding of the data. This embedding captures the most important features and relationships present in the data and serves as a compact representation of the original information. The decoder, on the other hand, takes the embedding produced by the encoder as input and uses it to generate a reconstructed version of the prior knowledge in the form of an adjacency matrix.

One of the key components of the GAE algorithm is the use of graph neural networks (GNNs) to implement the encoder. GNNs are a type of neural network that is specifically designed to operate on graph-structured data. They learn representations of nodes and edges in a graph by aggregating information from their neighbors and combining it with their own features. In the case of GAE, the GNNs operate on the gene expression data graph, with nodes representing genes and edges representing regulatory interactions between them.

Another important aspect of GAE is the loss function used to train the model. The loss function is typically composed of two parts: a reconstruction loss and a regularization term. The reconstruction loss measures the difference between the original data and the reconstructed data produced by the model, while the regularization term penalizes large weights in the model to prevent overfitting and improve generalization.

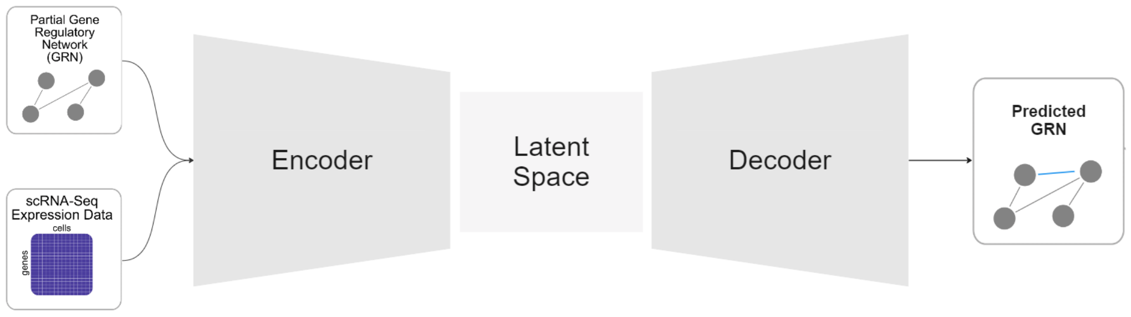
Partial Gene Regulatory Network (GRN)
scRNA-Seq Expression Data
cells
genes
Encoder
Latent Space
Decoder
Predicted GRN

Figure 2.1: The schematic shows the algorithm which takes partial known Gene Regulatory Network (GRN) and single cell RNA sequencing (scRNA-Seq) as inputs and predicts a GRN using a Graph Autoencoder structure.

2.1 Encoder

The Encoder module, Figure 2.2, consists of 2 graph layers. The encoder network takes the gene expression data X and a graph representation G of the gene regulatory network as input. The graph convolution operation is defined as:

$$H[l+1] = f(G, H[l], W[l])$$

where H[l] is the output of the l-th graph convolutional layer, f is a non-linear activation function (here, ReLU), W[l] is a weight matrix for the l-th layer, and G is a sparse adjacency matrix that encodes the regulatory relationships between genes. The weight matrix W[l] is learned during training by minimizing a reconstruction loss between the input data X and the decoded data ,here, the adjacency matrix A', produced by the decoder network. The encoder network learns to encode the input data into a lower-dimensional representation Z, by identifying the most informative features in the gene expression data that capture the underlying structure of the data.

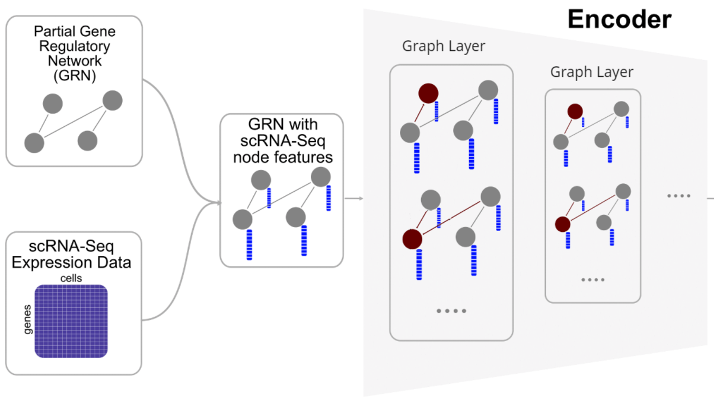
Encoder
Partial Gene Regulatory Network (GRN)
scRNA-Seq Expression Data
cells
genes
GRN with scRNA-Seq node features
Graph Layer
Graph Layer
GRN with latent embeddings

Figure 2.2: The Encoder representing the two Graph Layers that encodes the Gene Regulatory Network (GRN) and single cell RNA sequencing (scRNA-Seq) to a latent space [5]

2.2 Decoders

The decoder, Figure 2.3, takes the encoded representation Z, as input and generates a reconstruction of the input adjacency matrix A'. The decoder is trained to minimize the difference between the input data X and its reconstructed adjacency matrix A'. The sections below describe the structutre of the decoders used where the MLP and MLP(r) decoders are trainable.

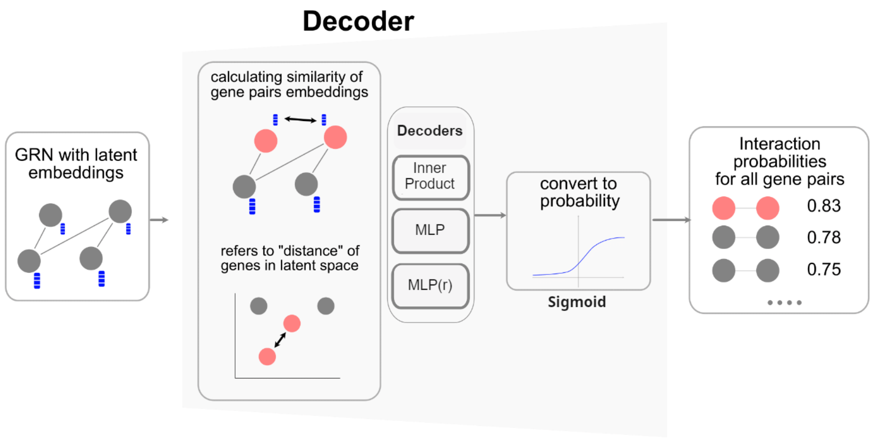
Decoder
GRN with latent embeddings
calculating similarity of gene pairs embeddings
refers to "distance" of genes in latent space
Decoders
Inner Product
MLP
MLP(r)
convert to probability
Sigmoid
Interaction probabilities for all gene pairs
0.83
0.78
0.75

Figure 2.3: The Decoder module decoding the latent space to Gene Regulatory Network [5]

2.2.1 Inner Product Decoder

The Inner Product decoder is a simple and commonly used decoder for GAE models. The reconstructed adjacency matrix A' represents the predicted gene regulatory network, where each element A'[i,j] represents the predicted regulatory interaction between genes i and j. A value close to 1 indicates a strong regulatory interaction, while a value close to 0 indicates no regulatory interaction. The Inner Product decoder computes the reconstructed adjacency matrix A' as follows:

$$A' = ZZ^T$$

where, Z is the encoded gene expression data of dimension N x d, where N is the number of genes and d is the dimensions of the encoded representation. The matrix multiplication ZZ^T produces a symmetric N x N matrix, which represents the reconstructed adjacency matrix A'.

2.2.2 MLP decoder (MLP)

The MLP decoder, which stands for Multi-Layer Perceptron decoder, is a key component of GAE models. It is used to decode the encoded graph data and reconstruct the original graph structure. The MLP decoder typically consists of multiple layers of perceptrons, which are simple computational units that take input values and apply a non-linear function to them. In our implementation, we use an MLP decoder with 3 layers containing 40, 20, and 1 perceptrons respectively. The first two layers use a ReLU activation function, while the last layer uses a sigmoid activation function.

In the context of GAE models, the MLP decoder takes the encoded graph data as input. Specifically, the latent encoding of two nodes is concatenated and fed into the decoder as input. The decoder's task is to predict the probability of there being an edge between the two nodes based on their latent encoding. The output of the decoder is bound between 0 and 1 by a sigmoid activation function at the last layer. A value close to 1 indicates a strong regulatory interaction between the two nodes, while a value close to 0 indicates no regulatory interaction.

During training of the model, these outputs are learned by minimizing the reconstruction loss. This loss measures the difference between the original graph structure and the reconstructed graph structure produced by the model. By minimizing this loss, the model learns to accurately decode the encoded graph data and reconstruct the original graph structure.

2.2.3 MLP decoder with multiple update (MLP(r))

In this implementation, we make use of the same architecture as described above, with one key difference: the frequency of update of the decoder during training. In order to speed up the learning of the decoder and improve its performance, we modify the training cycle such that the decoder is updated multiple times for each forward pass of the algorithm. The frequency of update is set as a hyperparameter to be optimized, with values ranging from 1 to 3.

By updating the decoder more frequently during training, we aim to accelerate its learning and enable it to more quickly adapt to changes in the encoded graph data. This can result in faster convergence and improved performance of the model overall. The optimal frequency of update for the decoder is determined through hyperparameter optimization, which allows us to find the best balance between speed and accuracy.

2.3 Layers

Convolutional layers and attention layers are two of the most important building blocks of graph neural networks (GNNs). These layers play a crucial role in enabling GNNs to effectively learn representations of nodes and edges in a graph by aggregating information from their neighbors and combining it with their own features.

Both convolutional and attention layers are powerful tools for learning representations of graph-structured data and play a crucial role in the success of GNNs.

2.3.1 Convolution (Conv)

A convolutional layer in a graph neural network (GNN) operates in a manner similar to a convolutional layer in a traditional convolutional neural network (CNN). However, instead of applying a convolution operation to an image, it applies the operation to a graph. This involves aggregating information from the neighborhood of each node in the graph and using it to update the node's representation.

The aggregation operation used by a convolutional layer in a GNN can take many different forms. For example, it can be a simple sum or mean of the features of neighboring nodes, or it can be a more complex operation such as a weighted sum or a message passing scheme. The weights of the convolutional filters are learned during training using backpropagation, allowing the model to adapt to the specific characteristics of the data.

By convolving over the graph and aggregating information from neighboring nodes, convolutional layers in GNNs enable the model to learn local patterns and relationships within the graph. This allows the model to effectively capture the underlying structure of the data and make accurate predictions [9].

2.3.2 Attention (GAT)

An attention layer in a graph neural network (GNN) is a powerful mechanism that allows the model to selectively focus on certain parts of the graph when updating the representations of its nodes. This is achieved through the use of an attention mechanism, which assigns weights to the nodes based on their importance in the current context.

The attention mechanism computes weights for each node by evaluating its similarity to its neighbors. Nodes that are more relevant to the task at hand are assigned higher weights, while nodes that are less relevant are assigned lower weights. These weights are then used to compute a weighted sum of the representations of the neighboring nodes.

The resulting weighted sum is used to update the representation of the current node. This allows the model to selectively incorporate information from the most relevant parts of the graph and effectively capture the underlying structure of the data. By using an attention mechanism, GNNs can learn to focus on the most informative parts of the graph and make more accurate predictions. [10]

2.4 Models

2.4.1 Graph Auto-Encoder (GAE)

GAE learns a low-dimensional graph representation by training an encoder-decoder neural network to reconstruct the adjacency matrix of the input graph. The encoder maps the input graph into a low-dimensional latent space, and the decoder maps the latent representation back to the adjacency matrix. The loss function is the binary cross entropy between the reconstructed and original adjacency matrices. [4]

2.4.2 Variational Graph Auto-Encoder (VGAE)

Variational Graph Autoencoder (VGAE) is an extension of Graph Autoencoder (GAE) that incorporates a probabilistic model to capture the uncertainty in the latent space. Specifically, VGAE learns a Gaussian distribution over the latent space and introduces a Kullback-Leibler (KL) divergence term in the loss function to encourage the learned distribution to be close to a unit Gaussian. This enables VGAE to model the distribution of the latent space and to generate new graphs that have similar properties to the input graph. [4]

2.4.3 Adversarially Regularized Graph Auto-Encoder (ARGA)

Adversarially Regularized Graph Autoencoder (ARGA) [11] is a deep learning-based approach for graph generation that combines a Graph Autoencoder (GAE) with an adversarial network. The objective of ARGA is to learn a latent space representation that captures the structural properties of the input graphs while generating realistic new graphs. ARGA consists of two main components:

1. The Graph Auto-Encoder, which encodes the graph structure into a lower-dimensional latent space representation and then decodes the latent representation to reconstruct the original graph.
2. The Adversarial Network, which discriminates between the reconstructed graphs and real graphs to ensure that the generated graphs are indistinguishable from real graphs.

2.4.4 Adversarially Regularized Variational Graph Auto-Encoder (ARGVA)

The Adversarially Regularized Variational Graph Auto-Encoder (ARGVA) model is similar to the Adversarially Regularized Graph Autoencoder (ARGA) model in that it consists of the same two components. The difference between the two models lies in the Auto-Encoder part. In ARGVA, a Variational Graph Auto-Encoder (VGAE) is used instead of a regular Graph Auto-Encoder. [11]

All model configurations

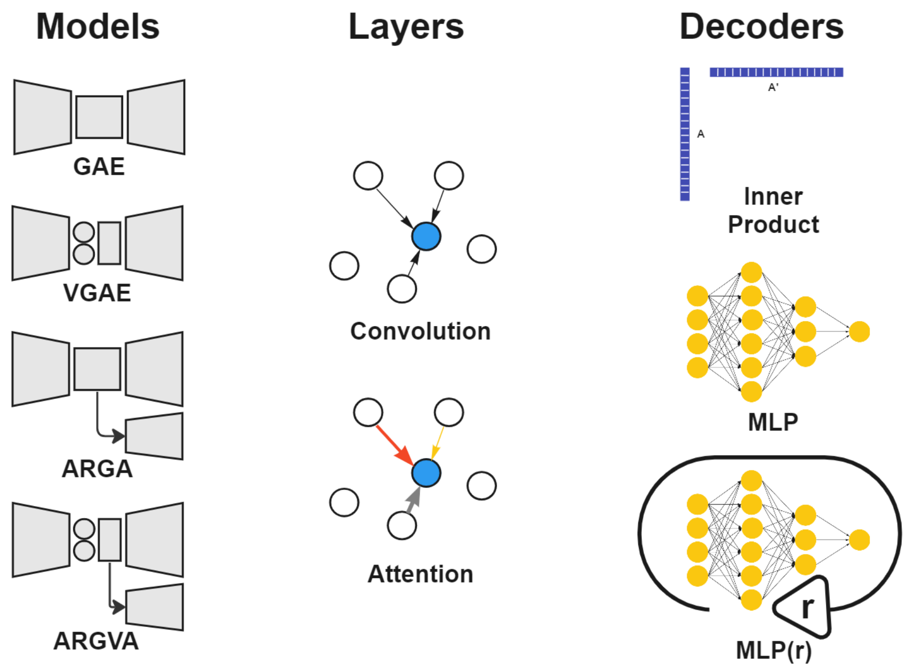

Figure 2.4: All components to create the models. Model Architecture:Graph Auto-Encoder (GAE), Variational Graph Auto-Encoder (VGAE), Adversarially Regularized Graph Auto-Encoder (ARGA), and Adversarially Regularized Variational Graph Auto-Encoder (ARGVA). Layers:Convolution and Attention. Decoders:Inner Product, MLP and MLP(r).

The entries in the Figure 2.4 are organized in a specific manner. The first part provides information about the model architecture. The second part indicates the type of layer that is employed in the model. Finally, the last part defines the type of decoder that is utilized. All the models used comprise of selecting one element from each of the three parts.

Chapter 3

Benchmarking graph autoencoder models

Benchmarking is a crucial step in the process of evaluating the performance of different models and methods. It provides a systematic and rigorous way to compare the effectiveness of different approaches and identify their respective strengths and weaknesses. By conducting a benchmarking analysis, we can gain valuable insights into the suitability of each method for different scenarios and make informed decisions about which approach to use.

In this section, we will describe in detail the benchmarking process that we used to evaluate the performance of different auto-encoder models. This includes a description of the dataset used for benchmarking, the evaluation metrics that we employed, and the specific models that we chose to compare. The models selected for benchmarking cover a wide range of possibilities currently used in the field, as well as new implementations of the decoder architecture.

By conducting a benchmarking analysis, we can gain a deep understanding of the relative performance of different auto-encoder models. This allows us to identify the most effective methods for different scenarios and provides valuable insights into the strengths and weaknesses of each approach.

3.1 Dataset

In order to conduct a benchmarking analysis of different graph auto-encoder models for inferring gene regulatory networks (GRNs), we used dataset sourced from McCalla et al [6]. This dataset, which was originally produced by Gasch et al [8]., contains gene expression data for the yeast Saccharomyces cerevisiae.

The dataset covers a total of 163 cells and includes information on the expression levels of 3,847 genes. This provides a rich and detailed view of the gene expression patterns in yeast and allows us to accurately evaluate the performance of different graph auto-encoder models. The dataset covers cells in two conditions, stressed and unstressed. The Figure 3.1 shows the gene expression values of Environmental Stress Response (ESR) genes under the two conditions. The rows represent transcripts and each column is an individual cell, with expression values according to the key.

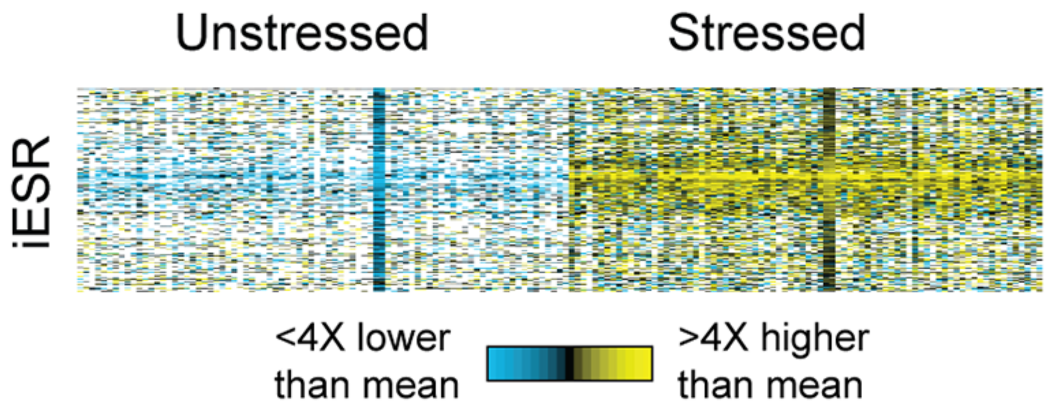

Figure 3.1: Mean-centered log2(read counts) for Environmental Stress Response (ESR) gene groups before and after induced stress. [8]

In addition to the gene expression data, the McCalla et al yeast dataset also includes a ground-truth GRN obtained by collating the results of TF perturbation experiments. This GRN contains a total of 34,282 edges and provides a reliable reference against which we can compare the inferred GRNs produced by different models. The edges in the ground-truth GRN are taken to be undirected. By using this high-quality dataset for benchmarking, we can ensure that our analysis is rigorous and reliable, and that our conclusions are well-founded.

3.1.1 Preprocessing

In order to ensure the quality and reliability of our gene expression data, we applied several preprocessing steps to the dataset. These steps were designed to filter out low-quality data and ensure that the remaining data was suitable for use in our analysis.

One of the key preprocessing steps we applied was filtering the expression data using the scanpy package [12]. This allowed us to remove cells that contained fewer than 10% of the total genes (384) and genes that were present in fewer than 10% of the cells (16). By applying this filtering step, we were able to ensure that our dataset only contained high-quality data that was suitable for use in our analysis.
In addition to filtering the data, we also scaled the expression matrix using a min-max scalar. This step ensured that all of the values in the expression matrix were on a common scale, making it easier to compare and analyze the data.
By applying these preprocessing steps to our dataset, we were able to ensure that our analysis was based on high-quality and reliable data.

3.2 Evaluation

3.2.1 Evaluation Metric

In this study, the prior known gene regulatory network is split into three parts: 50% is used for training the models, 25% is used for validation, and the remaining 25% is used for testing. The split is performed such that the training split does not include edges in validation and test splits; and the validation split does not include edges in the test split. During the hyperparameter optimization process, the performance of the models is measured using the Average Precision value of the validation split.

Average Precision is a commonly used metric for evaluating the performance of binary classification models. It calculates the average precision value for all possible recall values. In other words, it measures how well a model can accurately predict positive instances while minimizing false positives. This metric is particularly useful when dealing with imbalanced datasets where one class is significantly more prevalent than the other.

3.2.2 Hyperparameter optimisation

In order to find the best set of hyperparameters for our models, we utilized the Weights and Biases [13] webserver as a platform for hyperparameter optimization. We set the seed to zero and searched through a range of hyperparameters with the goal of maximizing the Average Precision (AP) as our target metric.

The hyperparameters we considered included weight decay, lambda L1, dropout rate, learning rate, latent dimension, and layer ratio. For each hyperparameter, we specified a range of values to search within. The table 3.1 contains all the ranges for the hyperparameters used.

By searching through these ranges of hyperparameters and evaluating the performance of our models using the AP metric, we were able to identify the optimal set of hyperparameters for our case. This allowed us to maximize the performance of our models and obtain more accurate and reliable results. The Weights and Biases framework has a feature called Bayesian hyperparameter search which uses a Gaussian Process to model the relationship between the parameters and the model metric and chooses parameters to optimize the probability of improvement. This method was used to perform our search across the ranges given for 500 runs.

Hyperparameter	Search Range
weight_decay	0.0 to 0.25
lambda_l1	0.75 to 1.0
dropout_rate	0.0 to 0.25
learning_rate	[0.1, 0.01, 0.001, 0.0001, 0.00001]
latent_dimension	[4, 8, 12, 16, 20]
layer_ratio	1.0 to 4.0

Table 3.1: Hyperparameter search ranges

3.2.3 10 fold validation

In order to ensure the robustness and validity of our models' performance, we subjected each model to rigorous testing for generalization and performance. This was achieved by calculating a key metric, the average precision value. The metrics provide a comprehensive measure of how well each model is able to accurately predict gene regulatory networks.

To account for variability in results and ensure that our findings were not influenced by chance, we repeated the model training process for 10 different randomly chosen seed values, excluding 0. These seeds values are 14, 25, 73, 89, 42, 7, 69, 56, 98 and 33. This allowed us to obtain a more accurate and reliable estimate of each model's performance. The models were trained and evaluated using the optimized hyperparameter set for seed 0.

Wilcoxon test

The Wilcoxon test [14] is a non-parametric statistical hypothesis test used to compare two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ. It is often described as the non-parametric version of the two-sample t-test. The null hypothesis of the Wilcoxon test is usually taken as equal medians. We use the wilcoxon test to compare all the different models after all ten seeds are trained.

Pearson Correlation Coefficient

The Pearson correlation coefficient [15] is a widely used statistical measure that quantifies the strength and direction of the linear relationship between two variables. It ranges from -1 to 1, with values close to -1 indicating a strong negative correlation, values close to 1 indicating a strong positive correlation, and values close to 0 indicating no correlation.

$r = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$ where:
$\text{cov}(X, Y)$ is the covariance between the two variables X and Y
σ_X is the standard deviation of variable X
σ_Y is the standard deviation of variable Y

To evaluate the performance of our models, we this simple linear measure as a baseline for comparison. Specifically, we calculate the Pearson correlation coefficient for the gene pairs in the test split. This provides us with a measure of the strength of the linear relationship between the expression levels of each gene pair.
We then calculate the Average Precision values for these gene pairs and repeat this process

for all ten seeds. By taking an average of these values, we obtain a single performance metric that reflects the ability of a simple linear measure to capture the relationship between gene pairs.
This value serves as a benchmark against which we can compare the performance of our more sophisticated models and assess their ability to accurately model the complex relationships between gene expression levels.

3.3 Benchmarking Results

This chapter presents the results of our benchmarking analysis on the Gasch et al. dataset. We evaluated 24 models from 4 architectures: GAE, VGAE, ARGA, and ARGVA. Each architecture offers a unique approach to modeling data and has its own set of advantages and disadvantages. The models use convolutional and attention layers to extract local features from data and focus on specific parts of the input. The decoders used are Inner Product, MLP and MLP(r) decoders, responsible for reconstructing the output from the model's latent representation.

We selected Average Precision as our metric and conducted tests using seed zero. For seed zero, hyperparameters were optimized on the validation set and Average Precision was calculated on the test set to evaluate model performance. We validated the results using 10 other seeds with the hyperparameters optimized for seed zero. This will probide us with inforamation on the generalisation capabalities of the models. Our analysis provides valuable insights into each system's strengths and weaknesses to guide future improvements.

3.3.1 Hyperparameter optimisation for single seed

In order to evaluate the performance of our model, we conducted hyperparameter optimization for seed zero on the validation split. This process allowed us to fine-tune the model and ensure that it was performing at its best. Once the optimization was complete, we proceeded to evaluate the model's performance on the test split. The results of this evaluation are presented in Figure 3.2, which displays the average precision value calculated on the test split for all models in the form of a heatmap. This visualization provides a clear and concise representation of the performance of each model and allows for easy comparison between them.

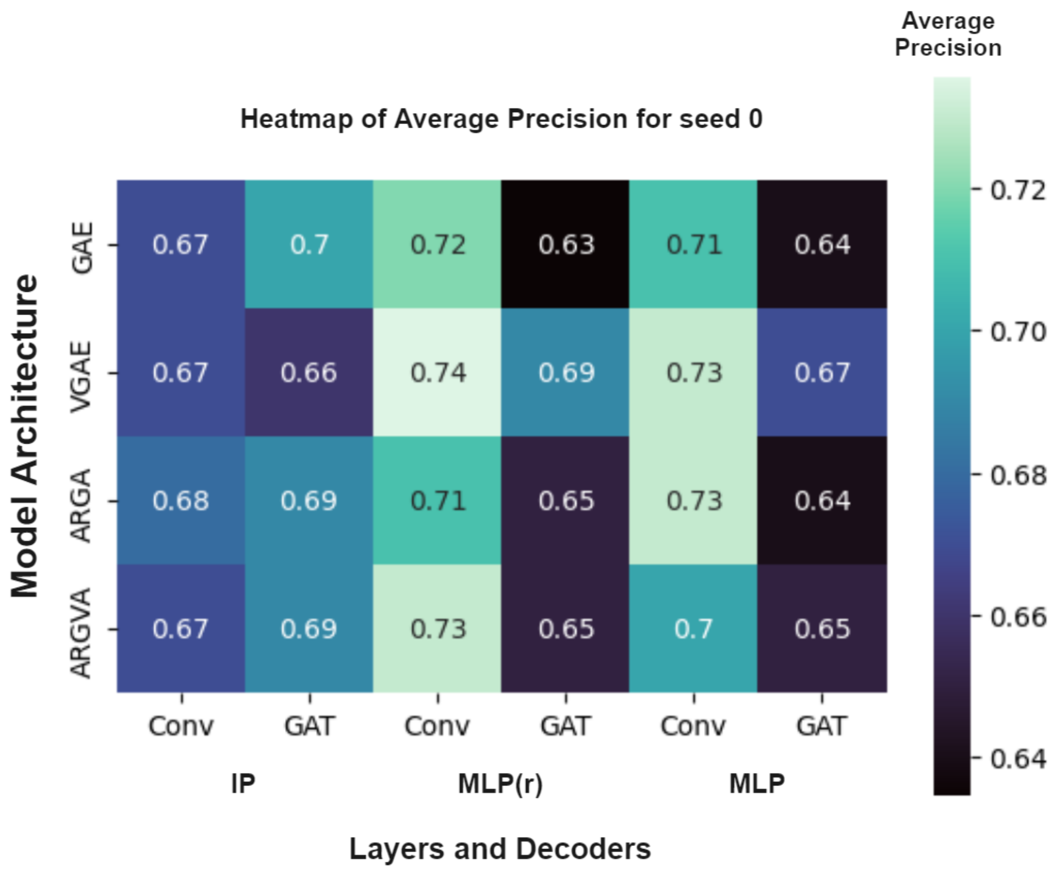

Figure 3.2: Heatmap of Average Precision values obtained after evaluating the models on the test set for seed 0

One of our key observations was that the choice of architecture for the convolution layers did not have a significant impact on the overall performance of the model. In fact, we found that the convolution layers consistently outperformed or performed at least as well as their attention layer counterparts, regardless of the decoder used.
The attention layers perform especially poorly for the MLP and MLP(r) decoder in comparison to the Inner Product decoder.

When comparing the performance of different decoders, we found that the MLP and

MLP(r) decoders outperformed the conventional Inner Product decoder. Among these two decoders, MLP(r) emerged as the best-performing decoder overall.
Except the Inner Product decoder, VGAE does marginally outperform the other models. The VGAE architecture, which utilized a convolution layer and an MLP(r) decoder, achieved the best performance out of all the models tested.

In order to further verify our findings and ensure their robustness, we conducted a 10-fold validation performance verification using 10 different seeds. This analysis allowed us to confirm our initial observations and provided additional evidence to support our conclusions. By taking into account multiple seeds and conducting a rigorous validation analysis, we were able to demonstrate that our findings were not simply due to chance or random variation.

3.3.2 10 fold validation of optimised models

All the models are trained for all 10 seeds using the hyperparameters optimised for seed zero and the set of average precision values for all the seeds are used to calculate the statistical significance using the Wilcoxon test. This is done to check for the generalisation capabilities of the models used. All the models are compared to GAE-Conv-IP called the base model henceforth.

We also report the average of the Pearson correlation coefficient on the test set for all the seeds comes out to be 0.55. We use this value as a baseline value to keep in mind while commenting on the performance of our models.

Similar to seed zero, Figure 3.3 contains the average of average precision values of all seeds in a heatmap. The major trends observed for results for seed zero follow through even after the rigorous criteria for testing. The p values are demarcated as asterisks (*). A single asterisk (*) if $p \leq 0.05$ and two asterisks (**) if $p \leq 0.01$.

In order to draw stronger and more robust conclusions about the generalisation and performance of different models, we conducted a test to provide us with statistical significance for comparing their results, called the Wilcoxon test. This allowed us to go beyond the limitations of relying solely on the results obtained using seed zero and to gain a deeper understanding of the relative strengths and weaknesses of each model.

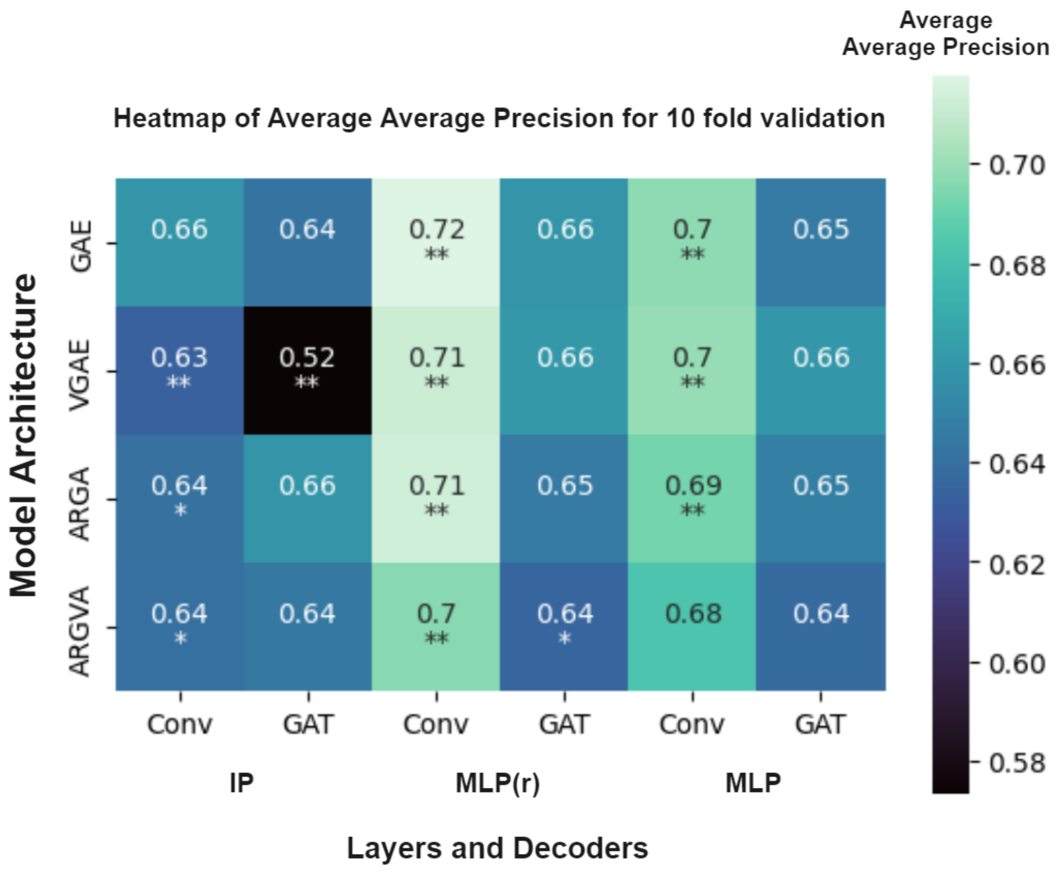

Figure 3.3: Heatmap of Average of Average Precision values obtained from evaluation the models on ten seeds apart from 0.

Following our results for seed zero, our analysis revealed that for the Inner Product decoder, changing the model from GAE resulted in worse performance for convolution layers and similar or worse performance for attention layers. This suggests that the choice of model architecture can have a significant impact on performance when using this particular decoder. Furthermore, we observed that attention layers consistently performed worse than convolution layers for both the MLP and MLP(r) decoders, regardless of the architecture used. This indicates that the choice of decoder can also play a crucial role in determining the overall performance of the model.
In contrast to the seed zero results, the worse performance of attention layers for MLP and MLP(r) decoder does not translate after the 10 fold validation. Similarly, our results for

VGAE for seed 0 do not translate, moreover VGAE architecture using attention layer with the Inner Product Decoder performs the worst.

All of the models utilizing the MLP and MLP(r) decoders in convolution layer configurations outperformed the base model with a high degree of statistical significance to support this claim. This provides strong evidence to suggest that these decoders, when used in conjunction with convolution layers, are highly effective at improving model performance for our use case of infering GRNs.
The GAE architecture, which utilized a convolution layer and an MLP(r) decoder, achieved the best performance out of all the models tested.

Chapter 4

Discussion

In conclusion, our results demonstrate that the trainable MLP(r) decoder proposed in this study performs better in the task of inferring Gene Regulatory Networks (GRNs) when incorporating prior knowledge. Furthermore, our study provides an extensive understanding of the best model architecture for this task. Our results also demonstrate that using a more complex architecture or incorporating an attention layer may not always yield better results. Instead, it is important to carefully consider the specific characteristics of the task at hand and select an appropriate approach accordingly. By carefully selecting and optimizing the components of our model, we are able to achieve improved performance on this challenging problem.

As observed in the results section, the MLP decoder performs better than the base model and all of the Inner Product decoders. We theorize that this improved performance is due to the fact that the MLP decoder learns patterns in the pairwise node embeddings in a non-linear way. This allows the decoder to more accurately classify whether an edge is present between two nodes.

Further, the MLP(r) decoder performs better than the base model and slightly better than MLP decoders. We theorize that this improved performance is due to the fact that the decoder is updated up to 3 times during each training epoch as found by the best set of hyperparameters for all the models. This frequent updating allows the decoder to learn the pattern of the latent representations of input gene pairs more quickly. By rapidly adapting to the changing representations of the input data, the MLP(r) decoder is able to more accurately infer the underlying Gene Regulatory Networks (GRNs).

Interestingly, our results show that using a more complex architecture or incorporating an attention layer, which are traditionally known to perform well on other tasks, do not yield better results on our task. This counter-intuitive finding suggests that the specific characteristics of our task may require a different approach. It is possible that the additional complexity introduced by these techniques may actually hinder performance by making it more difficult for the model to learn the relevant patterns in the data.
This difference in performance is particularly notable when examining the results of 10-fold validation. In this analysis, we found that attention layers performed significantly worse than their convolutional counterparts for MLP and MLP(r) decoders. One possible explanation for this result is that the attention layers may be unable to effectively generalize and conform to a pattern for the latent embeddings. This could hinder the decoding procedure and result in lower performance.

The best performing model in our benchmarking analysis of 10 fold validation achieved a average average precision score of 0.72, in comparison to 0.55 for Pearson correlation coefficient, indicating good performance for GRN predictions. Looking at the low value in case of Pearson correlation coefficient demonstrates the complexity of the system and the high score of our models demonstrates the effectiveness of the model in accurately predicting relationships within the GRN and provides confidence in its ability to be applied in real-world scenarios. The success of this model highlights the potential for further development and optimization to improve performance even further.

After evaluating a variety of Graph Autoencoder structures, we have established the most effective model for inferring Gene Regulatory Networks (GRNs) using prior knowledge. The next steps will involve testing these models on different datasets to assess their robustness and ability to accurately infer GRNs. Additionally, comparison of the performance of these models to existing methods in the field is imperative. By increasing the complexity of the problem and incorporating directionality into our models, we hope to gain deeper insights and further develop our approach. This will open up new avenues for research and help us continue advancing our understanding of GRNs.

Bibliography

[1] Tamim Abdelaal, Lieke Michielsen, Davy Cats, Dylan Hoogduin, Hailiang Mei, Marcel JT Reinders, and Ahmed Mahfouz. A comparison of automatic cell identification methods for single-cell rna sequencing data. *Genome biology*, 20:1–19, 2019.

[2] Betsabeh Khoramian Tusi, Samuel L Wolock, Caleb Weinreb, Yung Hwang, Daniel Hidalgo, Rapolas Zilionis, Ari Waisman, Jun R Huh, Allon M Klein, and Merav Socolovsky. Population snapshots predict early haematopoietic and erythroid hierarchies. *Nature*, 555(7694):54–60, 2018.

[3] Xuran Wang, Jihwan Park, Katalin Susztak, Nancy R Zhang, and Mingyao Li. Bulk tissue cell type deconvolution with multi-subject single-cell expression reference. *Nature communications*, 10(1):380, 2019.

[4] Thomas N. Kipf and Max Welling. Variational Graph Auto-Encoders, November 2016. arXiv:1611.07308 [cs, stat].

[5] Marco Stock et al. Gene regulatory network inference from single-cell rna-seq data by incorporating prior knowledge with graph autoencoders. Unpublished.

[6] Sunnie Grace McCalla, Alireza Fotuhi Siahpirani, Jiaxin Li, Saptarshi Pyne, Matthew Stone, Viswesh Periyasamy, Junha Shin, and Sushmita Roy. Identifying strengths and weaknesses of methods for computational network inference from single-cell RNA-seq data. *G3 Genes|Genomes|Genetics*, 13(3):jkad004, March 2023.

[7] Emma Steele, Allan Tucker, PAC't Hoen, and Martijn J Schuemie. Literature-based priors for gene regulatory networks. *Bioinformatics*, 25(14):1768–1774, 2009.

[8] Audrey P. Gasch, Feiqiao Brian Yu, James Hose, Leah E. Escalante, Mike Place, Rhonda Bacher, Jad Kanbar, Doina Ciobanu, Laura Sandor, Igor V. Grigoriev, Christina Kendziorski, Stephen R. Quake, and Megan N. McClean. Single-cell RNA sequencing reveals intrinsic and extrinsic regulatory heterogeneity in yeast responding to stress. *PLOS Biology*, 15(12):e2004050, December 2017. Publisher: Public Library of Science.

[9] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, February 2017. arXiv:1609.02907 [cs, stat].

[10] Shaked Brody, Uri Alon, and Eran Yahav. How Attentive are Graph Attention Networks?, January 2022. arXiv:2105.14491 [cs].

[11] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially Regularized Graph Autoencoder for Graph Embedding, January 2019. arXiv:1802.04407 [cs, stat].

[12] F. Alexander Wolf, Philipp Angerer, and Fabian J. Theis. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biology*, 19(1):15, December 2018.

[13] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[14] F Wilcoxon. Individual comparisons by ranking methods. biom. bull., 1, 80–83, 1945.

[15] Israel Cohen, Yiteng Huang, Jingdong Chen, Jacob Benesty, Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. *Noise reduction in speech processing*, pages 1–4, 2009.