University of Bergen
Department of Informatics

Inferring Gene Expression Values In Causal Directed Acyclic Graphs Using Graph Neural Networks

Author: Bendik Akselsen Solevåg
Supervisors: Ramin Hasibi and Tom Michoel

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

August, 2023

Abstract

Inferring gene expression values is helpful in determining important characteristics about an individual. Existing methods in gene expression inference mostly rely on linear methods creating separate models for each gene. This thesis hypothesises that a graph neural network can be used to model interactions between genes, and serve as a universal approximator for gene expression values. The research goals of this thesis are stated in the following four points.

1. Does prediction accuracy improve when also providing genome variation data in the dataset?
2. Can the graph feature autoencoder architecture be applied to predict missing gene expression values in a masked dataset?
3. Can a graph neural network be applied to predict missing gene expression values in a masked sample?
4. Can dataset gene expression values be extrapolated using only genome variation data?

An experiment was set up to answer this list of questions. The results indicate that prediction accuracy does improve when providing genome data, and the graph feature autoencoder architecture was applied successfully. This thesis was not able to create a graph neural network able to predict gene expression values in a masked sample. This thesis was not able to reliably extrapolate gene expression data using only genome variation data.

Acknowledgements

You have my sincerest gratitude.

Erik Hystad,
Ramin Hasibi,
Tom Michoel,
Gutama Ibrahim Mohammad,
Malin Strøm,
And the rest of you.

Bendik Solevåg
Monday 21st August, 2023

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Gene expression levels are interesting to study due to their effect on human health. Extracting gene expression levels from sources where sampling is not trivial is therefore an active area of research. Existing methods mainly utilise whole blood gene expression levels, as well as genome mutations, to reason about tissue-specific expression levels. This thesis incorporates the interactions between gene expression levels in order to make predictions about downstream genes, a method which in gene expression inference is relatively unexplored. In this thesis, Graph Neural Networks will be utilised to model these interactions.

This chapter will provide an introduction to the background materials used in this thesis. The motivation behind the approach applied in this thesis will be presented. Furthermore, a set of research questions are presented, which the work in this thesis will aim to answer. Finally, this chapter will present the structure of the thesis.

Background

The human Genome consists of 23 pairs of chromosomes, containing over 20 000 protein-encoding genes. Proteins are responsible for the function of the cell. The level at which a protein-encoding produces its protein varies from person to person, and is measured by the gene's *expression* level. Gene expression levels are a complex interactive system of values dependent on many variables, such as the individual's sex, ethnicity, age and genome mutations. One common such mutation is known as an eQTL. DNA is a double helix structure containing pairs of nucleotides. A SNP is a common variation in a single

nucleotide pair. SNPs that are known to affect the expression levels of genes are known as eQTLs.

Gene expression levels are interesting to study due to how they affect human health. Variation in gene expression levels is an important factor in determining if an individual will develop disease. Genetic variation may contribute to disease largely through misregulation of gene expression[27]. Predicting misregulated gene expression values in tissue where obtaining sample is not trivial is therefore useful in determining a person's likelihood of developing or suffering from disease.

Motivation

Obtaining gene expression values from readily available sources such as blood is trivial. Often times, however, it is more interesting to obrain expression values from tissue in which obtaining samples is difficult. Existing methods utilise principal components of whole blood expression values as well as general information about the sample (age, sex, ethnicity), and the gene's eQTLs.

The interactions between gene expression levels, however, is largely unexplored. Genome variation data has been hypothesised to act as a causal anchor in predicting downstream gene expression values. This thesis therefore takes the approach of generating a causal graph over gene expression interactions, representing genes as vertices and causal interactions as directed edges.

Graph neural networks are then applied to reason about gene expression values as feature values of vertices in the graph. Graph Neural Networks are a subcategory of neural networks within artificial intelligence specialised in reasoning about properties of graph structures. A separate approach for graph learning is useful due to the the isomorphism property of graphs, meaning that a single graph may be represented in machine-readable format in numerous ways. The created GNN model will be benchmarked against existing methods in gene expression inference using both existing graph neural network architectures as well as linear regression.

Research Questions

This thesis presents a set of 4 research questions that the thesis experiment will attempt to provide answers for. Research questions 1 and 4 are stated generally, and will be evaluated

using all of the machine learning models defined in this thesis. Research questions 2 and 3 pertain to specific model architectures, and will place special focus on variations of graph neural networks. Overall, the research questions serve as a framework to thoroughly explore the predictive capabilities of graph neural networks.

1. Does prediction accuracy improve when also providing genome variation data in the dataset?
2. Can the graph feature autoencoder architecture be applied to predict missing gene expression values in a masked dataset?
3. Can a graph neural network be applied to predict missing gene expression values in a masked sample?
4. Can dataset gene expression values be extrapolated using only genome variation data?

Thesis structure

This chapter has introduced general topic of the thesis, and a set of research questions this thesis aims to answer. This chapter has also introduced the neural network variant applied to the research questions. Chapter 2 will introduce the necessary background material this thesis is based on, including concepts from machine learning, graph theory and biology. Chapter 3 introduces existing methods in gene expression inference, and explains the motivation behind the selected approach in this thesis. Chapter 4 provides a detailed explanation of the dataset used in this thesis, featurization process of the data, the training setup for the various models, as well as the model architectures used in this thesis. Chapter 5 contains the results obtained from the experiment detailed in chapter 4, presenting a series of statistics to provide a nuanced view of the results obtained. Chapter 6 contains a discussion of the insights gained from the experiment and its result.

Chapter 2

Background

This chapter introduces the background material relevant for the experiment which will be outlined in chapter 4. To begin the chapter, section 2.1 introduces central concepts from graph theory, which will become useful in modelling gene interactions. Section 2.2 introduces machine learning. In section 2.3, the machine learning topic is expanded upon to apply to graph structures. Section 2.4 introduces central genome concepts, such as gene expression values and genome variation data. Section 2.4.5 introduces an existing tool to generate a Directed Acyclic Graph from gene expression data.

2.1 Graphs

The work in this thesis relies on graph theory. This section begins by in 2.1.1 defining vertices and edges, the two components of a graph. Subsection 2.1.2 then introduces the following graph concepts: neighborhoods, directed graphs, connectedness, DAGs, isomorphism, and adjacency matrices. The concepts introduced will be used in the experiment setup later in this thesis.

2.1.1 Vertices and Edges

A graph consists of a pair of two sets $\mathcal{V}$ and $\mathcal{E}$ [11].

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}) \tag{2.1}$$

The items of set $\mathcal{V}$ are commonly named *vertices* or *nodes*. For the purposes of this thesis, the term *vertices* will be used. Each vertex may contain arbitrarily shaped information, giving an indication of what the current vertex represents.

$$v = [v_0, v_1, ..., v_n], v \in \mathcal{V} \tag{2.2}$$

The items of set $\mathcal{E}$ are called *edges*. Each edge must be an item in the cartesian product of the set of vertices with itself [11]. Where vertices typically aims to provide information about the graph elements, edges show some relation between the graph elements, connecting the elements. As with vertices, edges may also contain arbitrarily shaped information, giving an indication of the current edge's representation.

$$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \tag{2.3}$$

2.1.2 Central Graph Properties

Neighborhoods

The term *neighborhood* in graph theory refers to a subset of the full graph. Given a graph $\mathcal{G}$ (using the notation defined in equation (2.1) , each graph vertex's neighborhood, denoted $N_{\mathcal{G}}(v_j)$, is the subset of all vertices in which every vertex v_i shares an edge with v_j and $v_i \neq v_j$ [3]. This property fulfills equation (2.4). Depending on what a given graph is modelling, a vertex's neighborhood may be interesting to consider, as vertices that are closely connected may share similar properties.

$$\forall_{\{v_i,v_j\}\in\mathcal{V}\times\mathcal{V}}\Big(\{v_i, v_j\} \in \mathcal{E} \wedge v_i \neq v_j \leftrightarrow v_i \in N_{\mathcal{G}}(v_j))\Big) \tag{2.4}$$

Directed Graphs

A graph may be *directed* or *undirected* [11]. In an undirected graph edges have no direction. If an edge exists between v_n and v_m, an edge must also exist between v_m and v_n. One vertex's relation to the connected vertex is equal to its mirror connection. In a directed graph, this is not the case. In a directed graph, edges have a source vertex and target vertex. Observing an edge from vertex v_n to vertex v_m, from vertex v_n's perspective, v_m is defined as v_n's *child* vertex. From vertex v_m's perspective, v_n is defined as v_m's *parent* vertex[37].

Directed Graph Neighborhoods

In the directed graph case, when discussing a specific vertex neighborhood, more specification is required. Three sets could be considered to be the vertex's neighborhood. The sets are specified and mathematically defined in the following list.

1. The set of parent vertices for vertex v_j
   $\forall_{(v_i,v_j)\in\mathcal{V}\times\mathcal{V}}\Big(v_i \neq v_j \wedge (v_i,v_j)\in\mathcal{E} \leftrightarrow v_i \in N_{\mathcal{G}}(v_j)\Big)$
2. The set of child vertices for vertex v_i
   $\forall_{(v_i,v_j)\in\mathcal{V}\times\mathcal{V}}\Big(v_i \neq v_j \wedge (v_i,v_j)\in\mathcal{E} \leftrightarrow v_j \in N_{\mathcal{G}}(v_i)\Big)$
3. The set of child and parent vertices for vertex v_j
   $\forall_{(v_i,v_j)\in\mathcal{V}\times\mathcal{V}}\Big(v_i \neq v_j \wedge \big((v_i,v_j)\in\mathcal{E} \vee (v_j,v_i)\in\mathcal{E}\big) \leftrightarrow v_i \in N_{\mathcal{G}}(v_j)\Big)$

For the purposes of this thesis, the set of parent vertices is considered to be the current vertex's neighborhood.

Connected Graph

In an undirected graph, the term *connected graph* is used for graphs in which from every vertex, every other vertex is reachable from the source vertex by traversing the edges of the graph [11]. In the directed case, this definition is split in two. A graph is said to be *weakly connected* if from every vertex, every other vertex is reachable from the source vertex by traversing the edges of the graph, ignoring the edge direction. A graph is said to be *strongly connected* if from every vertex, every other vertex is reachable from the source vertex by traversing the edges of the graph in their true direction.

Directed Acyclic Graphs

The term *cycle* refers to a property found on certain graphs. A cycle exists in a graph if starting in some vertex v_n, visiting no vertices more than once, it is possible to traverse edges until the starting vertex is reached. In the directed case, edges must be traversed in their true direction until the starting edge is reached. If this traversal does not exist, the graph is a *directed acyclic graph* (DAG) [4].

When a graph is a DAG, there must exist at least one vertex with no incoming edges, as shown in appendix B. At least one vertex with no outgoing edges must also exist. It is possible to sort the graph vertices in the order of graph traversal, starting from a root vertex with no incoming edges. In this representation, a vertex with no incoming edges must be the first element in the sorted array of vertices, and a vertex with no outgoing edges must be the last. For each element in the sorted array, the vertex found in the current element may only have outgoing edges toward vertices found at a greater index in the sorted array. This kind of sorting is referred to as topological sorting.

Isomorphism

In section 2.1.1 graphs were defined as an ordered pair of a set of vertices and a set of edges. As vertices are defined as elements in a set, it follows that vertices have no order. Even in the case that a graph is a DAG, the topologically sorted order may not be unique. Any permutation of the list of vertices remains a valid representation of the graph, as long as the list of edges are permutated accordingly. Two graphs which after a permutation of their vertices and corresponding edges can turn into the same graph are called 'isomorphic'.

Adjacency Matrices

Viewing the edge definition provided in equation (2.3), one can plot a table in which each row and column is indexed by a single element subset of the vertex set. The table will then provide an overview of the cartesian product of the vertices. This representation of edges is called the adjacency matrix. Typically, each cell in the adjacency matrix table contains either a 0 or 1 value, determining if an edge exists between the pair of vertices.

2.2 Machine Learning

This section introduces the subject machine learning, as well as the main concepts required in training neural networks. The section begins with a general explanation of the term *machine learning*. Section 2.2.1 goes on to introduce linear regression. Section 2.2.2 introduces a measure of the 'wrongness' of a machine learning model's predictions. Section 2.2.3 introduces activation functions, allowing neural networks to approximate

non-linear functions. Section 2.2.4 introduces neural networks. Sections 2.2.5 and 2.2.6 introduce the concepts allowing the training of neural network model parameters. Section 2.2.7 introduces a special neural network architecture known as an *autoencoder*.

As a term, *Machine Learning* is a subcategory if the umbrella term 'Artificial Intelligence'. Machine Learning takes a set of input (training) data, and attempts to reason about some quality of the input, without the author of the program explicitly telling the program what quality of the input data we are interested in reasoning about. Instead, the author provides the Machine Learning model with their optimal target for the given input, and the model must generalize over, and update its trainable parameters to fit the target [25].

This approach can be useful in problems where discovering the target based on the model input becomes computationally intractable, or when the target cannot be determined as a direct consequence of the input.

2.2.1 Linear Regression

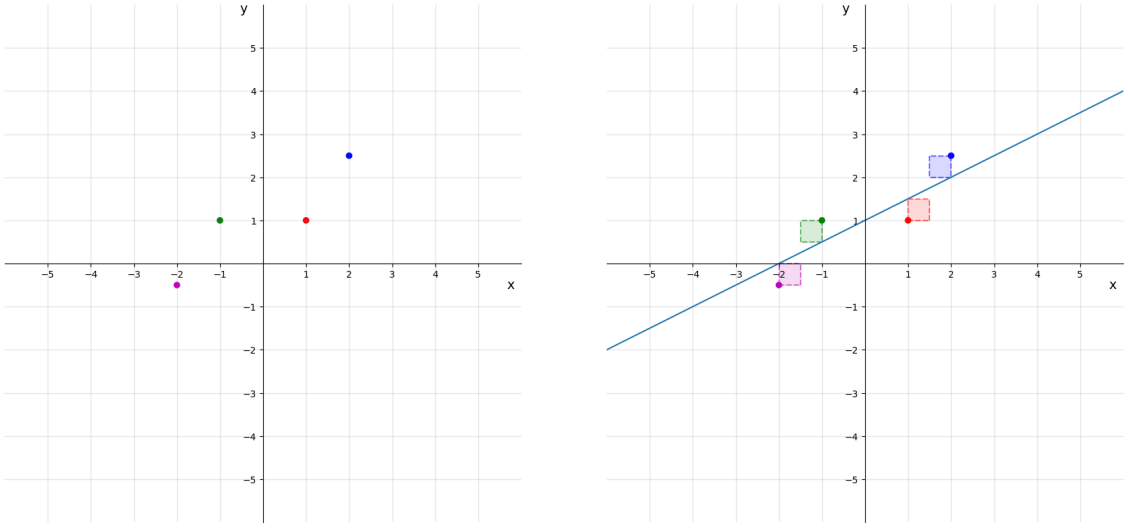
As a first study in the implementation of machine learning, linear regression is presented. An example in a two dimensional, euclidean space is considered.

$$f(x) = w_0 x + b_0 \tag{2.5}$$

Given a set of input data values X, each input value $x_i \in X$ has a corresponding output value $y_i \in Y$. Each input value can be plotted with its corresponding output value as coordinates. To describe a function that best maps the given inputs with their related outputs, introduce two single-value real number variables w_0 and b_0. Define a function to map input values to their corresponding output.

$$\mathcal{L} = \frac{1}{|X|} \sum_{i=0}^{|X|} \big(y_i - f(x_i)\big)^2 \tag{2.6}$$

The loss function is a measure of how well equation (2.5) maps input to its corresponding output is also introduced. As function variables w_0 and b_0 are tuned to better map input to its corresponding output, the measure will decrease. There exists values



(a) Input x values plotted against their corresponding y values

(b) Function (2.6) applied to function (2.5) with optimized function variables w_0 and b_0

Figure 2.1: Linear regression illustrated
The colored dots in both figures represent the data our model is attempting to approximate. The colored squares in the right figure illustrate the individual squared errors of the model predictions. The line in the right figure represent the model prediction for each value of x.

for our variables w_0 and b_0 such that the measure defined in equation (2.6) reaches its lowest possible value. $|X|$ denotes the number of elements in the set X.

$$\begin{aligned} &X \in \mathbb{R}^m, \{b\} \in \mathbb{R}^n, W \in \mathbb{R}^{nxm} \\ &f(X) = WX + b \end{aligned} \tag{2.7}$$

The function defined in equation (2.5) is arbitrary. Additional function variables can be added or removed depending on the prediction task. Equation (2.5) applies linear regression to single-dimensional inputs and single-dimensional outputs, but the same method can be applied to inputs and outputs of higher dimensions. The method can also be applied when the inputs and outputs are of different dimensions, by applying linear transformations, multiplying the input with matrices in dimensions to produce output in desired dimensions. A general formula for linear regression is defined in equation (2.7)

2.2.2 Loss

In the previous section, a measure of how 'wrong' a mapping of input points to their corresponding output points was defined. This measure is commonly referred to as the

loss function[18]. The loss function defined in equation (2.6) is named MSE loss [18], due to the fact that it is defined by the mean of the squared difference between our function's predictions and the true values.

It is possible to define different loss functions. For the purposes of this thesis, a loss function is a measure of the distance between a model's prediction and a ground truth value. For this reason, a loss function must be non-negative for all input values. Different loss functions will interpret the performance of a functions mapping of the given input data to its corresponding output in different ways. The difference can be illustrated by observing the result of applying the loss function defined in equation (2.8), typically referred to as MAE (mean absolute error) loss [18], as opposed using MSE.

$$\mathcal{L} = \frac{1}{|X|} \sum_{i=0}^{|X|} \sqrt{\left(y_i - f(x_i)\right)^2} \tag{2.8}$$

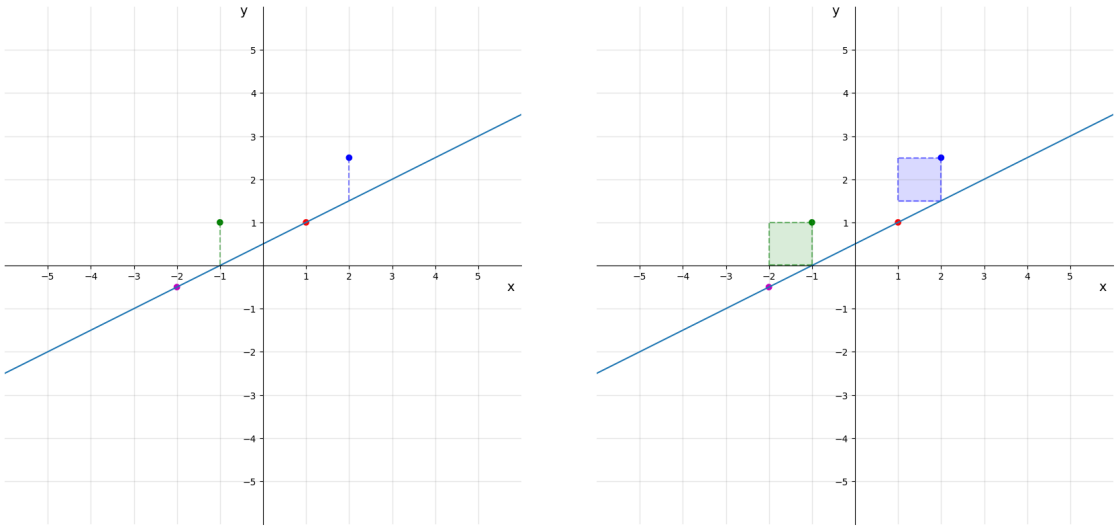
While MSE loss sums over the squares of errors, MAE sums over the absolute errors. As a consequence, MAE will place equal weight emphasis on each prediction's error. MSE will place more emphasis on predictions that are further from the true output values. Different loss functions can therefore be applied to different problems to find different solutions. Figure (2.2) illustrates an example of different loss functions being applied to the same prediction function. In this example, the optimal function parameters were found using the MAE operator. When the MSE operator is applied to the same function parameters, the MSE operator does not produce its lowest possible value.

2.2.3 Activation functions

Activation functions are non-linear functions that are utilised in neural networks to allow the networks to approximate non-linear functions. Any non-linear mapping of input values, defined on all real numbers, will function as an activation function.

ReLU

The (Re)ctified (L)inear (U)nit maps any given input value to a value greater than 0, with no upper limit [24]. The ReLU non-linearity has become the norm when selecting non-linearity between neural network layers[35]. One of ReLU's strengths in comparison



(a) The MAE loss function (2.8) applied to function (2.5) with optimized function variables w_0 and b_0

(b) The MSE loss function (2.6) applied to function (2.5) with function variables w_0 and b_0 optimised using loss function (2.8)

Figure 2.2: Different loss functions applied to the same prediction function

to alternative activation functions lies in its derivative. The derivative of ReLU is largely unaffected by the value of its input. This property is not found in other common activation functions, and may be a factor in creating vanishing gradients.

$$ReLU(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{2.9}$$

2.2.4 Neural Networks

The multidimensional linear regression formula defined in equation (2.7) is one common building block of neural networks, and defines what is typically referred to as a *linear layer*. A neural network can consist of a series of linear layers performing a linear transformation of the output of the previous layer. The shape of the each linear layer's weight and bias matrix is determined by the previous layer's output shape, as well as the desired input for the next layer. Each linear layer is most commonly followed by an activation function. The final layer may serve as the neural network output, or an activation function can be applied.

The neural network model becomes a better predictor of output data as a cycle of forward and backward passes is repeated until the neural network's weight matrices converge at a set of values. Due to the non-linearity introduced by the activation functions,

it is obvious that such a model has much greater expressivity than its predecessor; the linear regression model.

2.2.5 Forward Pass

During the forward pass, input data is passed through all the layers and activation functions constituting a Neural Network. Matrix multplications followed by activation functions are applied to the input data, until an output value is calculated following the last layer.

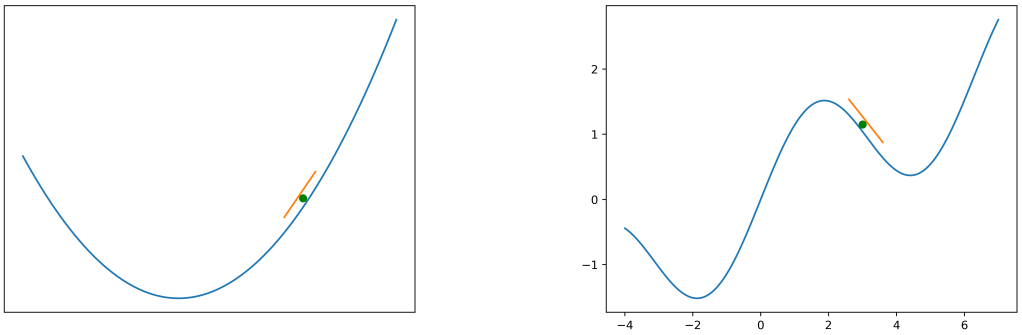
As defined in section 2.2.2, a loss function is then applied to the output of the linear layer, yielding a measure of the accuracy of the neural network's approximated output, and the desired output. The initial forward passes of any neural network with a given architecture will return a random output in dimensions specified by the architecture.

2.2.6 Optimization Step

The process of optimizing a neural network model is commonly referred to as the 'optimization step'. The goal of the optimization step is to optimize the model parameters such that for any given input, the model approximates the desired output. One common algorithm is the Gradient Descent algorithm[2]. In this thesis, the Gradient Descent algorithm will be used to discuss the implementation of the optimization step. Furthermore, successors of Gradient Descent will be introduced.

Gradient Descent

The Gradient Descent algorithm utilises the gradient of the model's output in relation to every model parameter to update each parameter. This idea is illustrated in figure 2.3. As there exists a measure of the accuracy of the neural network's prediction and the desired value, it is possible to quantify each model parameter's effect on the resulting output. This value is given by the derivative of the loss, in relation to the given model parameter.



(a) A model parameter's current value as well as it's derivative, headed towards the global minimum.

(b) A model parameter's current value as well as it's derivative, headed towards a local minimum.

Figure 2.3: Parameter Gradients

Chain Rule

Though calculating the derivative of the loss function in regard to any model parameter may seem like a difficult task, it is in fact elementary. To calculate the derivative of a given parameter, we utilise the chain rule of derivatives from calculus. Rodriguez & Fernández provides the following defintion of the chain rule [19].

Assume some function $g(c)$, differentiable at c, and some function $f(g(c))$, differentiable at $g(c)$. Then, $f \circ g$ is differentiable at c, and $(f \circ g)'(c) = (f' \circ g)(c) \cdot g'(c)$ [19] For the purposes of this thesis, with the same functions and variables as described above, this formulation is rewritten in Leibniz notation, given in equation (2.10).

$$\frac{\partial f}{\partial c} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial c} \tag{2.10}$$

Backpropagation and PyTorch Autograd

Implementations of the gradient descent algorithm should utilise dynamic programming to avoid excess computation. One such implementation is PyTorch's automatic differentiation system named Autograd.

Autograd allows circumventing the computation of the derivatives as they are needed. Instead, as input data is passed through the model, each operation performed using the model parameters are recorded.

A directed acyclic graph with edges pointing towards the root will be generated. The current iteration's training input, as well as all model parameters, will appear as leaf vertices in the tree. Mathematical operations performed on the model input, as well as the intermediate states between the operations will be represented as the child vertices of the described leaf vertices. In each intermediate vertex, the previous vertex's values will be saved in a context variable. The calculated loss will be the represented by the tree's root.

This structure allows the PyTorch library to efficiently perform the backward pass. It needs only to perform the instructions specified at each intermediate vertex. Values utilised more than once in the backward pass will only be computed once. This makes PyTorch's Autograd system an efficient implementation of calculating the gradients, while still utilising the chain rule.

Stochastic Gradient Descent

Though the Gradient Descent algorithm does converge at a set of model parameters, the algorithm is flawed. The algorithm requires the model to iterate over all samples before computing the gradient and performing the parameter update. This process is computationally expensive, and may take a long time.

Rather than iterating over training input, the Stochastic Gradient Descent selects a subset of the training data. The dataset is shuffled, and at every iteration, a selected number of samples are selected. The number of samples used in every iteration is a hyperparameter. The data is passed through the model, and the loss function is calculated over this quantity.

Subsets of input data are passed through the model, and parameter updates are performed, until all input data has been passed through the model. This process may then be repeated until the model parameters converge.

Adam

For the work presented in this thesis, the Adam optimiser[21] is used. The Adam optimiser is a successor of the stochastic gradient descent algorithm. Adam differentiates itself from the stochastic gradient descent algorithm in that it includes the momentum term, which is an exponential moving average of the gradient.

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) \tag{2.11}$$

Adam's update function is given by equation (2.11), in which ϵ is a small constant used to avoid dividing by 0. α denotes the learning rate for the algorithm, typically set to 0.001. θ denotes the trainable model parameters. $\hat{m}_t$ denotes the bias-corrected moving average of the gradients' first moment. $\hat{v}_t$ denotes the bias-corrected moving average of the gradients' second moment[21]. The exponential moving averages are calculated using the functions defined in equation (2.12), in which both m_t and v_t are functions of both the gradient at the current iteration and their respective values at previous iterations.

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \end{aligned} \tag{2.12}$$

Given an unlimited amount of time and computation resouce, any of the optimization algorithms mentioned in section 2.2.6 should converge at a locally optimal set of parameters. When creating a neural network, the choice of optimization function will affect the speed at which the network weights converge.

2.2.7 Autoencoders

An *autoencoder* is a neural network that is trained to copy its input to its output. Internally, it has a hidden layer $\mathbf{h}$ that describes a *code* used to represent the input [14]. An autoencoder typically consists of two parts. The first part is a set of neural network layers which produces a *code*, and is a learned function $\mathbf{h} = f(\mathbf{x})$, commonly referred to as the *encoder*-part of an autoencoder. The second part is another set of neural network layers which reconstructs the original input: $\mathbf{r} = g(\mathbf{h})$. This type of neural network is illustrated in figure 2.4.

The autoencoder's reconstructed input is rarely interesting on its own. The traditional use for the autoencoder architecture was to create a lower-dimensional representation with which the input data can be approximately reconstructed. Autoencoders were introduced as a non-linear alternative for traditional methods for performing principal components analysis[23]. In this thesis, a graph neural network version of the the autoencoder is used to infer missing gene expression values in a masked sample.

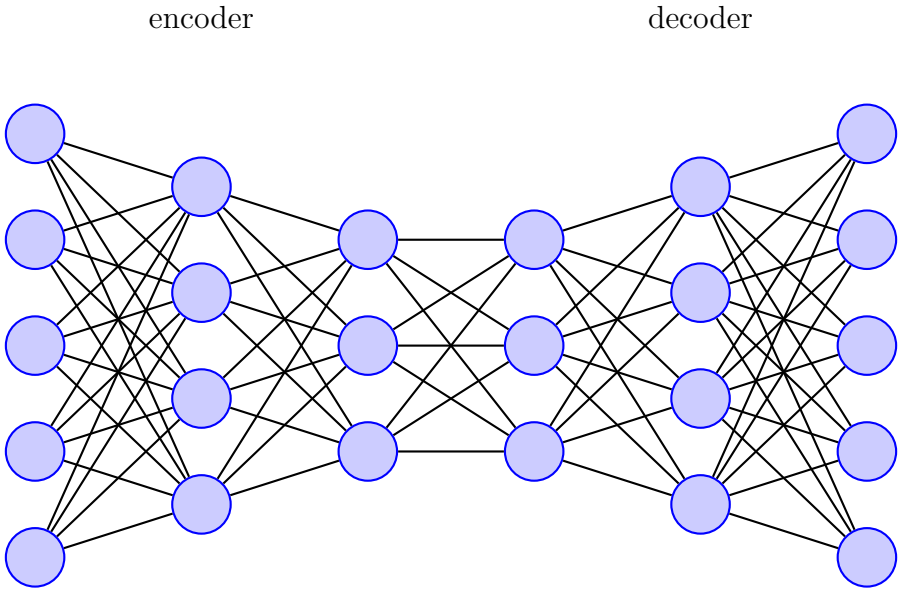


Figure 2.4: Autoencoder Architecture Illustrated

2.3 Graph Neural Networks

This section serves as an extension to section 2.2, in which a framework to perform machine learning on graph structures. The framework introduced in this section is based on Bronstein et al.'s Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges[8]. Section 2.3.1 introduces a common way to represent graphs in machine a machine readable format, and the problem with applying machine learning to these representations. Section 2.3.2 introduces permutation equivariance, a solution to the outlined permutation problem. Section 2.3.3 introduces the implementation of permutation equivariance used in this thesis.

2.3.1 Matrix Representation of Graphs

A common way to represent a graph in code form is by using a pair of matrices. One matrix represents vertex data as a $(n \times m)$-matrix, where n denotes the number of vertices in the graph, and m denotes the number of samples for each vertex.

Graph edges are often represented by edge lists. This method creates a $(2 \times |\mathcal{E}|)$ element matrix, in which $|\mathcal{E}|$ is the total number of edges in the graph. Each $i < |\mathcal{E}|$-th element in the adjacency list references the edge's parent vertex in the first list and child vertex in the second list.

This representation of graph data is possible to pass as input to a neural network consisting of linear layers, given that the architecture is designed with the two inputs in mind. This approach has a disadvantage, as it does not consider any isomorphic permutation of the input nodes to be equal to the given input nodes. This architecture therefore requires every permutation of the input data to be trained separately. This is impractical as it is computationally inefficient.

2.3.2 Permutation invariance and equivariance

As a consequence of the problem of isomorphic permutations illustrated in section 2.3.1, the neural network introduced in section 2.2.4 is no longer sufficient when attempting to learn from graph structured data. To remediate this issue, Bronstein et al. introduces the concept of permutation invariant functions [8]. A permutation invariant function has the desirable property that for any two isomorphic graphs, the output of the function is identical. This property is described by equation (2.13), in which f is a permutation invariant function, $\mathbf{X}$ is the function input, and $\mathbf{P}$ is a permutation matrix over the input.

$$f(\mathbf{PX}) = f(\mathbf{X}) \tag{2.13}$$

Permutation invariance can be easily achieved when evaluating a graph by ignoring the graph edges, and treating the vertices as a set. One method of achieving this property is by using the sum operator, as described in equation (2.14), in which ψ is a function over the individual vertex features, and ϕ is a function over the aggregated vertex features.

$$f(\mathbf{X}) = \phi\Big(\sum_{v \in \mathcal{V}} \psi(\mathbf{X}_v)\Big) \tag{2.14}$$

Equation (2.14) provides a permutation invariant output over the sum of all of the graph vertices. This formulation is not useful in many applications, where the goal is to reason about individual vertex features. As an alternative, the permutation equivariant function is introduced. This property is given in equation (2.15). The resulting vertex features should be updated in the same manner whether the permutation matrix $\mathbf{P}$ is applied to the vertices before or after the vertices are passed through function f.

$$\mathbf{P}f(\mathbf{X}) = f(\mathbf{PX}) \tag{2.15}$$

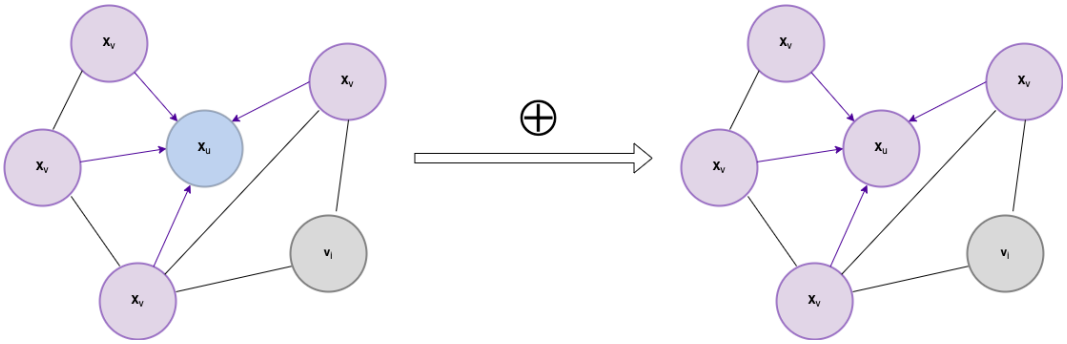


Figure 2.5: Illustration of a single aggregation layer applied to vertex u. Information is accumulated from each vertex v in u's neighborhood. Some function is applied to the aggregated node input, and u's value is updated.

This definition can be expanded to include graph edges by incorporating the adjacency matrix, as defined in section 2.1.2. A function f over graph data $\mathbf{X}$ and adjacency matrix $\mathbf{A}$ is permutation equivariant if it fulfills equation (2.16).

$$f(\mathbf{PX}, \mathbf{PAP}^\intercal) = \mathbf{P}f(\mathbf{X}, \mathbf{A}) \tag{2.16}$$

One way of achieving this property is by rather than aggregating all graph vertices, as in equation (2.14), performing a permutationally invariant function over each vertex's neighborhood, as described in section 2.1.2.

Bronstein et al. provides the general formula given in equation (2.17) as a general blueprint for neighborhood aggregating perutation equivariant function for reasoning about graph nodes. $\mathbf{h}_u$ is the updated feature value of vertex u, $\mathbf{x}_u$ is the current input vertex features, $\mathbf{x}_v$ is the current neighbor vertex's input features, ψ is a function over the individual vertex features, ϕ is a function over the aggregated vertex features, and $\bigoplus$ is some permutation invariant aggregation operator, such as the sum or mean operator.

$$\mathbf{h}_u = \phi\Big(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_\mathcal{G}(u)} \psi(\mathbf{x}_v)\Big) \tag{2.17}$$

Given input data $\mathbf{X}$ with shape $(n \times m)$, each vertex in the input data can be updated by passing it as argument to equation (2.17). At each iteration, the individual graph vertex will accumulate data from vertices at a distance equal to the number of iterations performed.

2.3.3 Aggregation layer variations

Graph Convolutional Network

The aggregation function introduced in equation (2.17) serves as a general blueprint from which more complex aggregation methods can be built. Contextualise first this blueprint by applying it to the layer which is attributed as the first in the emerging series of graph neural network layers, namely the Kipf & Welling's 2016 model *Graph Convolutional Networks* [22].

The model layer introced's vertex-wise formulation is given in equation (2.18), in which $\mathbf{W}$ is a trainable weight matrix, $e_{v,u}$ is the edge weight between the current neighbour and the target vertex, and d is given by $\hat{d}_i = 1 + \sum_{j \in \mathcal{N}_{\mathcal{G}}(i)} e_{j,i}$.

$$\mathbf{h}_u = \mathbf{W} \sum_{v \in \mathcal{N}_{\mathcal{G}}(u) \cup \{u\}} \frac{e_{v,u}}{\sqrt{\hat{d}_v \hat{d}_u}} \mathbf{x}_v \tag{2.18}$$

In this case, the ϕ function from blueprint (2.17) is implemented as the trainable weight matrix $\mathbf{W}$, $\bigoplus$ is implemented as the sum operator, ψ is implemented as a scaled function of the edge weight, and $\mathbf{x}_u$ is moved into the aggregation operation. In a case where edge weights are not given, equation (2.18) can be simplified as in equation (2.19).

$$\mathbf{h}_u = \mathbf{W} \sum_{v \in \mathcal{N}_{\mathcal{G}}(u) \cup \{u\}} \frac{\mathbf{x}_v}{\sqrt{\hat{d}_v \hat{d}_u}} \tag{2.19}$$

SAGEConv

While the graph convolutional network operator requires the size of every vertex neighbourhood, a variant of the SAGEConv algorithm can be viewed as an extension of the GCN framework to the inductive setting [16]. The basic variant of the SAGEConv aggregator variant is described in equation (2.20) [12].

$$\mathbf{h}_u = \mathbf{W}_1 \mathbf{x}_u + \mathbf{W}_2 \cdot mean_{v \in \mathcal{N}_{\mathcal{G}}(u)} \mathbf{x}_v \tag{2.20}$$

2.4 Genome

This section introduces central biological properties relevant in predicting gene expression values. The section begins by providing a general introduction to the human genome. Section 2.4.1 introduces the concept *gene expression*. Sections 2.4.2 and 2.4.3 introduce genetic variation. Section 2.4.4 explain gene expression values' effect on the health of an individual. Section 2.4.5 introduces a system to model gene interactions as DAG, as well as an existing framework which may be applied to create the DAG.

2.4.1 Genes

The human genome consists of 23 pairs of chromosomes[6] containing over 20 000 protein encoding genes [34]. An individual's genome is a complex system of interacting elements, which determines many things about the individual, such as eye color, height, health conditions, or the individual's response to certain medications [20]. In this section, several important elements from the genome are introduced. These elements, as well as the interactions between them, are crucial to the work presented in this thesis.

The traditional definition of the term *gene* referred only to what is today known as *protein-encoding genes*. Protein-encoding genes are sections of an organism's genome which encodes sequences of amino acids, which in turn are the building blocks of proteins. Proteins are responsible for the function of the cell[7]. The level at which each protein-encoding gene produces its corresponding protein varies from person to person. The term *gene expression level* denotes the level at which a gene produces proteins from its DNA.

2.4.2 Loci

The term *locus* (plural: *loci*), refer to a specific location in an organism's genome. Though this term in some cases is used synonymously with the term *gene*, for the purposes of this thesis, the distinction between them is clarified. Every segment in an organism's genome is associated with a locus. A locus refers to the specific location at which a genome segment is located. This segment may contain protein-coding gene, or it may contain a non-coding part of the genome.

2.4.3 SNPs and eQTLs

An organism's genome consists of a double helix structure. Each helix contains a series of nucleotide pairs, in which each nucleotide is connected to one helix. Both protein-coding genes as well as the general non-encoding genome are series of base pairs.

A Single-Nucleotide Polymorphism (SNP) is a variation at a single position in the DNA sequence in an organism. To be qualified as a SNP, the variation must occur in at least 1% of individuals in a given population. SNPs may be found in protein-coding genes, or in the non-coding part of the genome[29].

Certain SNPs are known to be associated with an altering of the gene expression of one or multiple genes. Loci associated with altered gene expression are known as Expression Quantitative Trait Loci (eQTLs) [30]. Genome-wide association studies have demonstrated that SNP variants that are associated with increased risk of disease are most commonly found in the non-coding loci in the genome, and are therefore likely to be involved in gene regulation. [30] An eQTL may affect one or multiple genes. Tong et al. have shown that genes sharing a regulatory eQTL show correlated changes in expression linked to the variant's genotype across tissues [31].

eQTLs are characterised as either *cis* or *trans*, depending on the distance between the eQTL and the gene it's affecting, measured in the number of nucleotide pairs from the eQTL to the gene. Pierce et al. state that trans-eQTLs are more difficult to identify than cis-eQTLs because trans effects are generally weaker than cis effects and because a huge number of tests must be conducted to comprehensively search the genome for trans-eQTLs, resulting in the use of stringent significance thresholds[33].

Wang and Michoel state that it is believed that genetic variation can be used to infer the causal directions of regulation between coexpressed genes, based on the principle that genetic variation causes variation in nearby gene expression and acts as a causal anchor for identifying downstream genes [40].

2.4.4 Effect on health

While the direct effects of protein-coding genes on health has been well documented, newer research places more emphasis on non-coding regions of the genome [39]. Variation in gene expression level is an important factor in determining if an individual will develop

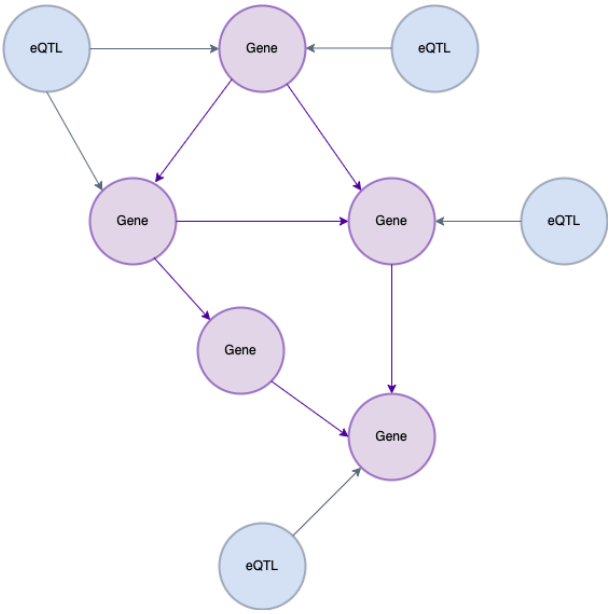


Figure 2.6: A Directed Acyclic Graph in which purple vertices represent protein-coding genes and blue vertices represent eQTLs. Graph edges refer to the source vertex's value affecting the target vertex's value.

disease, as well as in its general health. The contribution of genetic variation to disease is based on its effect on gene expression levels [27]. For instance, the *c-Myc* gene is believed to regulate the expression level of 15% of all genes[13]. In lung cancer, dysregulated gene expression levels of the c-Myc gene is frequently associated with higher mortality rate [9].

Lee & Young state that diabetes mellitus is a group of metabolic diseases in which a person has elevated blood sugar, either because the pancreas fails to produce adequate amounts of insulin, or because cells do not respond properly to the insulin that is produced. Mutations in pancreatic master transcription factors and the sequences they bind have been implicated in diabetes[27].

Being able to affect specific genes' expression level is naturally therefore an interesting problem to consider in preventing disease and developing medicine.

2.4.5 Causal Inference in Gene Expression Prediction

As stated in sections 2.4.1 and 2.4.3, an organism's protein-coding gene's expression is a complex system in which each each gene's expression value is dependent on the expression values of other genes. It is possible to create a graph in which each vertex represents an individual protein-coding gene. The node's feature value is given by the current gene's

expression levels. A graph edge is drawn if a gene's expression value is shown to affect the target gene's expression value.

Recorded eQTLs for the given genes can also be represented by vertices in the DAG, with the eQTL's given mutation as the vertex's value. As eQTLs are known to alter the expression values of their corresponding genes, edges can be drawn from eQTL vertices to their corresponding gene expression vertices. The graph will then consist of a set of protein-coding genes, and a set of eQTLs, where any edge between the sets are directed from the set of eQTL vertices to the set of gene expression vertices. The resulting graph is represented in figure 2.6.

The Findr tool

The work in this thesis requires a graph from which to reason about neighbouring vertices' expresison values. The generation of this graph is dependent on Wang & Michoel's *Findr* tool [40].

The Findr tool works by performing a series of Log Likelihood Ratio tests over a set of hypotheses, determining the existence of a causal relationship between a pair of genes and a potential eQTL. The tool performs a total of six tests, from a correlation test, to causality tests taking hidden confounders into account. The tool's workflow can be summarized in the following four points [40].

1. For robustness against outliers, every continuous variable is converted into standard normally distributed $N(0, 1)$ values using a rank-based inverse normal transformation across all samples.
2. A null- and an alternative hypothesis are proposed for each likelihood ratio test. Model parameters set to their maximum likelihood estimators to obtain the log likelihood ratio between the null- and alternative hypothesis.
3. The analytical expression for the probability density function of the log likelihood ratio when samples follow the null hypothesis is derived.
4. Log likelihood ratios are converted into posterior probabilities of the hypothesis.

A total of six likelihood ratio tests are performed. For the purposes of this thesis, tests 2 and 5 are the most interesting to consider. A and B denote a pair of genes for which causality is tested. E denotes the best eQTL for gene A.

The secondary linkage test

Figure 2.7: The Findr tool secondary linkage test

The secondary linkage tests checks whether E regulates the expression value of B. The test defines the null hypothesis as B being independent of E, while the alternative hypothesis defines B as being regulated by E, shown in equation 2.21. The relationship is illustrated in figure 2.7

$$\begin{aligned} \mathcal{H}_{null}^{(2)} &\equiv E \quad B \\ \mathcal{H}_{alt}^{(2)} &\equiv E \rightarrow B \end{aligned} \tag{2.21}$$

For $\mathcal{H}_{alt}^{(2)}$, $E \rightarrow B$ is modelled as B following a normal distribution whose mean is determined categorically by E, as shown in equation 2.22.

$$B_i | E_i \sim N(\mu_{E_i}, \sigma_B^2) \tag{2.22}$$

From the total likelihood over all samples, the MLE (Maximum Likelihood Estimate) model parameters are given in equation 2.23, in which n_j is the sample count by genotype category.

$$\begin{aligned} \hat{\mu}_j &= \frac{1}{n} \sum_{i=1}^{n} B_i \delta_{E_j} \\ \hat{\sigma}_B^2 &= 1 - \sum_{j=0}^{n_b} \frac{n_j}{n} \hat{\mu}_j^2 \end{aligned} \tag{2.23}$$

As the null hypothesis states that B is drawn from a standard normal distribution independent of E, the LLR (Log Likelihood Ratio) for test 2 can be defined as in equation 2.24.

$$LLR^{(2)} = -\frac{n}{2} \ln \hat{\sigma}_B^2 \tag{2.24}$$

The alternative hypothesis can then be accepted or rejected with posterior probability according to equation (2.25)[40]. This probability is calculated by simulating the probability of obtaining the observed $LLR^{(2)}$ by the distribution of $LLR^{(2)}$ under the null hypothesis. The distribution of $LLR^{(2)}$ is simulated by repeatedly sampling from the assumed distribution under the null hypothesis, and building a histogram based on the simulated $LLR^{(2)}$ values. The generated histogram is compared to the true distribution of the $LLR^{(2)}$. $P(\mathcal{H}^{(2)}_{null})$ can then be determined by aligning $P(\mathcal{H}^{(2)}_{null}|LLR^{(2)})$ with the real distribution $P(LLR^{(2)})$ at the $LLR^{(2)} \to 0^+$ side. All prerequisites to perform Bayesian inference to obtain the value in equation 2.25 are then provided[40].

$$P(E \to B) = P(\mathcal{H}^{(2)}_{alt}|LLR^{(2)}) \tag{2.25}$$

The control test

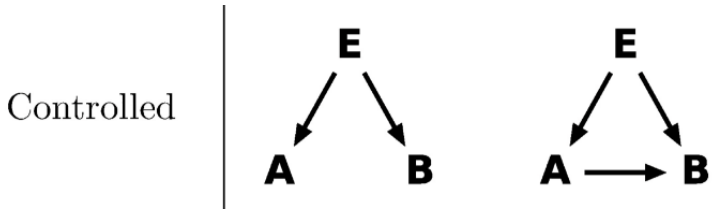


Figure 2.8: The Findr tool controlled test

Following the result of the secondary linkage test, E can be verified not to regulate A and B independently, by defining the null- and alternative hypothesis as in equation 2.26. The log-likelihood ratio can then be defined as in equation (2.27)[40]. The relationship is illustrated in 2.8.

$$\mathcal{H}^{(5)}_{null} \equiv E \to A \wedge E \to B \quad \mathcal{H}^{(5)}_{alt} \equiv E \to A \wedge E \to B \wedge A \to B \tag{2.26}$$

$$LLR^{(5)} = -\frac{n}{2} ln(\hat{\sigma}^2_A \hat{\sigma}^2_B - (\rho + \sigma_{AB} - 1)^2) + \frac{n}{2} ln \hat{\sigma}^2_A \hat{\sigma}^2_B \tag{2.27}$$

Chapter 3

Project Motivation and Goals

The chapter aims to use the background material defined in chapter 2 to contextualise the problem that this thesis attempts to solve. The list of research questions also provide a clear scope for the work in this thesis. This chapter begins by in section 3.1 explaining existing methods in gene expression inference, and the methodology applied as well as the variables used to perform inference. Section 3.2 goes on to introduce a set of research questions this thesis aims to answer. Section 3.3 outlines the two model training approaches that will be used in chapter 4.

3.1 Existing work in gene expression inference

Similar approaches using causal models and gene expression levels to infer related gene expression levels have previously been applied in related work. This section aims to break down a selection of these applications.

Predicting Tissue-Specific Gene Expression From Whole Blood Transcriptome

A first study in this causal approach Basu et al's regression approach in their paper *Predicting Tissue-Specific Gene Expression From Whole Blood Transcriptome* [5], in which it is proposed that gene expression levels in tissue may be inferred from whole-blood

expression values. Such a model is useful, as obtaining expression samples from various tissues is not a trivial task, and may involve invasive medical procedures.

The paper finds that an individual's whole blood expression levels is able to significantly— predict (based on a likelihood-ratio test with a false discovery rate threshold of 5%) tissue-specific expression levels for 60% of the genes on average across 32 tissues. "The tissue-specific expression inferred from the blood transcriptome is almost as good as the actual measured tissue expression in predicting disease state for six different complex disorders, including hypertension and type 2 diabetes, substantially surpassing the blood transcriptome" (Basu et al.)[5].

Basu et al. propose a set of three linear models to perform this task. To achieve such a high degree of accuracy, rather than providing the proposed linear models with all whole-blood gene expression levels, the top ten principal components of the whole-blood expression data (top twenty principal components for whole-blood splicing) was provided as input, as the authors found that this set of values were able to explain up to variance as features [5]. Also provided as input to the resulting linear regression model are other useful confounders, such as the given sample's age, sex and race.

The resulting, top performing linear model is a combination of all of the data described above, and can be viewed in equation 3.1, in which g denotes the current gene index, j denotes the current sample index, and k denotes the principal component index. $\hat{Y}_{gj}$ denotes the predicted gene expression value, β_g^0, β_{gk}^1, β_{gk}^2, δ_g, γ_g, and σ_g denote trainable weights. PC_{jk} represents the k'th principal components for the given sample. $WBGE$ denotes whole blood gene expression, and $WBSp$ denotes whole blood gene splicing [5].

$$\hat{Y}_{gj} = \beta_g^0 + \sum_{k=1}^{M} \beta_{gk}^1 PC_{jk}(WBGE) + \sum_{k=1}^{N} \beta_{gk}^2 PC_{jk}(WBSp) + \delta_g Age_j + \gamma_g Sex_j + \sigma_g Race_j \tag{3.1}$$

Blood-Based Multi-Tissue Gene Expression Inference with Bayesian Ridge Regression

Leng et al. present a similar motivation as the one presented in section 3.1 in their 2020 paper *Blood-Based Multi-Tissue Gene Expression Inference with Bayesian Ridge Regression* [41], citing the difficulty of collecting samples from various tissues.

Leng et al. performs dimensionality reduction to their gathered data by quantifying the linear dependencies of target genes in predicted tissue T_k on feature genes in B, and selecting the B most relevant features for each target gene. The dependencies are quantified by computing the variation of each feature, ranking them in descending order, and selecting the top 10% most varied[41].

The dataset is then used to train a bayesian ridge regression model for each targeted gene expression value, for each targeted tissue. The predicted expression value is drawn from a distribution as specified in equation 3.2, in which q denotes the current gene index, w and b denote trainable weight parameters, x denotes input data, and α denotes an error term drawn from a gaussian distribution.

$$\hat{y} \sim N(w_{(q)}^T x_{(q)}^1 + b_q, \alpha^{-1}\mathbf{I}) \tag{3.2}$$

3.2 Research Questions

Gene expression levels have been shown to be an indicator of healthy or diseased processes in organs of the human body ,refer to section 2.4.4. It is difficult obtain sample data from tissue in an individual's internal organs. It is however, possible to obtain a person's eQTL variation data from accessible sources, such as blood, skin, urine, or saliva. Existing methods in predicting gene expression values utilise this genome variation data, as well as metadata such as age, sex or ethnicity, and other samples' expression values, as causal indicators in order to make their predictions.

In section 2.4.5, a causal DAG over genes was hypothesised, and section 2.4.5 introduced an existing tool with which such a DAG can be discovered. Genome variation data has been hypothesised to be able to act as a causal anchor when predicting downstream gene expression values ,refer to section 2.4.3. This thesis therefore hypothesises that gene expression values may be inferred using a causal DAG which models gene-gene interactions as well as eQTL-gene interactions. Therefore, the aim is to answer the following list of research questions to determine if graph neural networks can be used to infer gene expression values.

1. Does prediction accuracy improve when also providing genome variation data in the dataset?

2. Can the graph feature autoencoder architecture be applied to predict missing gene expression values in a masked dataset?

3. Can a graph neural network be applied to predict missing gene expression values in a masked sample?

4. Can dataset gene expression values be extrapolated using only genome variation data?

Answering these will add to existing knowledge about graph neural networks in the context of gene expression inference, which we propose as an alternative to existing methods in gene expression inference.

3.3 Linear and Graph-based Approach Motivations

This section breaks down the need for different dataset featurization, as different regressor models are able to process different inputs. The models applied to the Geuvadis dataset, which will be introduced in section 4.1, are split into two groups. Linear models, and Graph Neural Networks.

Linear models

Linear regression is not a permutation-equivariant function. As the task that is being performed centers around inferring gene expression data based on their respective causal parents' expression data, there cannot exist a general linear model for this task, as the number of input features the linear model is required to handle is of variable length. There exists genes with no input edges, and there exists genes with multiple input edges.

One approach for solving this probem is to generate a separate linear model for each gene, as each gene has a set number of causal parents.

A common technique when featurizing the dataset for linear model usage is to perform feature extraction or dimensionality reduction, see section 3.1. For the purposes of the work presented in this thesis, this process was omitted. This decision was made due to the fact that the linear models provide a better benchmark for the graph-based models (as will be introduced in section 3.3) if their causal neighborhoods are identical. As will

be discussed in the graph-based approach section, a goal when featurizing the input data was therefore to keep some degree of similarity between the featurized datasets. When reading the results of the work presented in this thesis, this fact should be kept in mind, as the dataset was prepared to fit the training of graph-based models, not the training of linear models.

Graph-based models

As an alternative to the linear models discussed in section 3.1, a possible approach involves utilising graph based learning, as was discussed in section 2.3. The advantage of this approach lies in the fact that it circumvents the problem of a graph's high number of isomorphic permutations, discussed in section 3.3. If a graph-based approach can successfully be applied, a single model may perform the same task for which hundreds are needed in the linear case. It is therefore also reasonable to think that if a graph-based approach is able to model the causal interactions between vertices in the graph, the model would be a well-performing estimator of additional vertices added to the graph, whereas the linear approach would have no way of predicting for the introduced gene without training a separate model.

Inferring gene expression values based on causal neighbours using graph neural networks was the approach made by Hasibi & Michoel in their 2021 paper *A Graph Feature Auto-Encoder for the prediction of unobserved node features on biological networks*[17]. Hasibi & Michoel present a model for missing data imputation using a graph neural network model, which is able to outperform a standard, multi-linear layer approach as described in section 2.2.

Chapter 4

Experiment Setup

In this chapter the experiment setup is explained. The chapter starts in section 4.1 by providing a detailed introduction to the Geuvadis dataset. In section 4.2, a simple analysis is performed in order to determine helpful statistical properties of the dataset. Section 4.3 details how the Geuvadis dataset is transformed from a set of matrices to the three dataset types used for training the machine learning models used in this thesis. Sections 4.4, 4.5 and 4.6 explain the training process of the different types of models used in this thesis. Section 4.7 introduces the model architectures used to perform gene expression inference.

4.1 The Geuvadis Dataset

The Geuvadis dataset[26] is a genome dataset containing both gene expression values as well as eQTL data for hundreds of samples over a group of populations. In this section, the Geuvadis dataset will be introduced. An analysis of the dataset will be provided, and central properties of the dataset will be discussed.

The 1000 Genomes project, which was an effort to provide a comprehensive description of common human genetic variation by applying whole-genome sequencing to a diverse set of individuals from multiple populations. In total, the project reconstructed the genomes of 2,504 individuals from 26 populations [1].

The Geuvadis dataset is a 462 individual subset of the 1000 Genomes dataset in which the functional variation in human genomes is characterized by RNA-sequencing[26]. The

Geuvadis dataset limited its scope to the Finnish (FIN, 93 samples), British (GBR, 94 samples), Cephalonian (CEU, 87 samples), Toscani (TSI, 89 samples) and Yoruba (YRI, 89 samples) populations.

To discover genetic regulatory variants, eQTLs were mapped to to transcriptome traits of protein-coding genes separately in the European and Yoruba populations[26]. For this reason, in the featurization step of the model pipeline, the Yoruba population was left out. Therefore, the resulting dataset contains 373 individual samples. Information related to each sample can be viewed in table 4.1.

Sample	Sex	Biosample ID	Code	Name	Code	SName	Elastic ID
HG00271	male	SAME123417	FIN	Finnish	EUR	European	FIN
HG00276	female	SAME123424	FIN	Finnish	EUR	European	FIN
HG00308	male	SAME124161	FIN	Finnish	EUR	European	FIN
HG00310	male	SAME124338	FIN	Finnish	EUR	European	FIN
HG00315	female	SAME124335	FIN	Finnish	EUR	European	FIN

Table 4.1: A selection of the samples in the Geuvadis dataset. Each row corresponds to a sample, and each column contains sample metadata.

Table 4.2 presents a subset of the gene expression data provided by the Geuvadis dataset. Gene indices are presented as table row headers, and sample indices are presented as table column headers. Each cell then corresponds to the given gene/sample pair's gene expression value.

	HG00096	HG00097	HG00099	HG00100	HG00101	HG00102	HG00103	HG00104
ENSG00000136237.12	3.78850273139249	2.05096276937706	4.00031300285966	3.27161920134705	1.79821591847768	1.51668840574531	2.50033276871929	4.50927680966078
ENSG00000247157.2	0.0884028135135502	0.0357747754321161	0.0726643060416082	0.0533120112696138	0.159740096605739	0.358860016741265	0.396625831510976	0.304475150546142
ENSG00000137411.10	10.843283852078	12.7943342635096	6.74071971232153	17.0852204261215	24.1415882515112	14.3186281484009	12.0850364077074	16.399592813196
ENSG00000099804.2	11.9757350714476	12.4334611904074	8.12649262405	14.5744648248122	13.3328763642751	12.4474527797221	14.0457935113571	11.5981889781581
ENSG00000186952.10	0.179027511445971	0.219267027613899	0.208353389630347	0.452498531975494	0.386209586874843	0.214801647294066	0.176780891869511	0.166011816642241

Table 4.2: Sample names along the columns, gene names along the rows. Each cell contains one sample's expression value for one gene.

Table 4.3 presents a subset of the genome variation data also presented to the models in some of the research tasks. eQTL indices are presented as table row headers, and sample indices are presented as table column headers. Each cell then corresponds to the given eQTL/sample pair's categorical eQTL value.

4.2 Dataset Analysis

In this section, an analysis of the dataset is performed. Various statistical properties of the dataset are visualised and explored, using histograms over gene expression values,

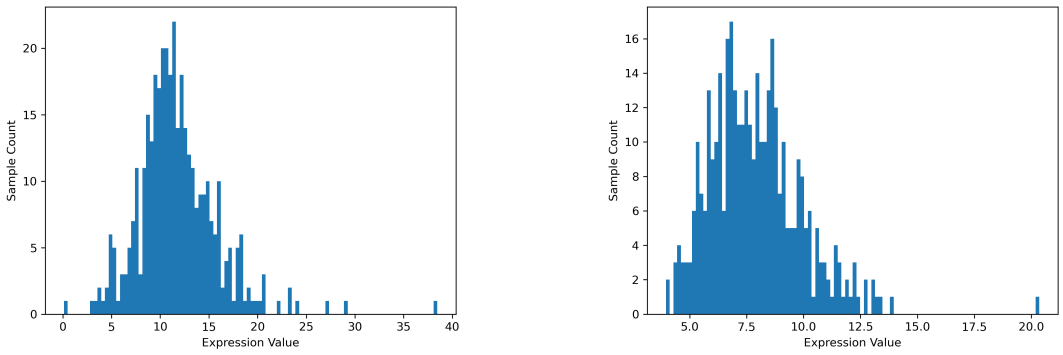
	HG00105	HG00107	HG00115	HG00132	HG00145	HG00157	HG00181	HG00308
rs12405651	0	0	0	1	0	0	0	0
rs148649543	0	0	0	0	0	0	1	0
rs138277764	0	0	0	1	0	0	1	0
rs28821228	0	0	0	0	0	0	1	0
rs28463291	1	0	0	2	2	1	1	1

Table 4.3: Sample names along the columns, eQTL names along the rows. Each cell contains one sample's categorical mutation value for one gene.

scatter plots over means and standard deviations, and the Pearson correlation coefficient. Causal reasoning between inferred properties of the dataset and their respective statistical properties are also explored, to gain insight into properties that should exist in the fitted machine learning models' output space.

Single gene expression value distribution

As a first step in the analysis of the Geuvadis dataset, the goal is to achieve some intuition of the distribution of samples for a given gene. To achieve this, a histogram of gene expression values was illustrated for a given gene. This histogram will be used later in this thesis as model predictions will be plotted against the ground truth expression value distributions.



(a) Vertex id: 199 (b) Vertex id: 200

Figure 4.1: Histograms for selected vertices and the distribution of gene expression values over all samples for the given gene.

All Gene Expression Value Distribution

The square roots of the featurized Geuvadis dataset genes' mean expression values are plotted against the square root of the genes' corresponding standard deviations. By

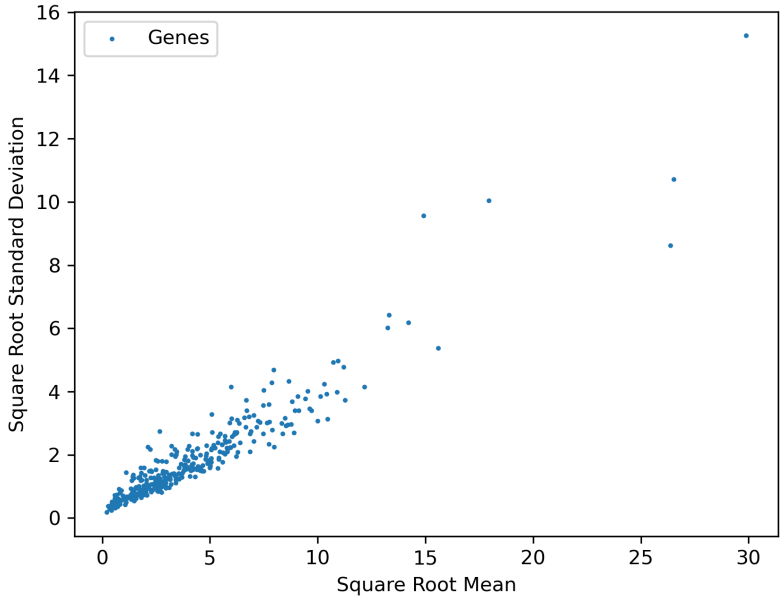


Figure 4.2: Scatter plot of each gene's mean expression value and its standard deviation

visually inspecting the figures, genes with a high gene expression value tend to also have a high standard deviation. This assumption is substantiated by calculating the Pearson correlation coefficient between the list of mean gene expression values and the mean gene standard deviation. Performing the calculation yields a correlation coefficient of 0.9257, indicating a high correlation between genes' mean expression values and standard deviations.

4.3 Dataset Featurization

This section aims to explain how this thesis organised its datasets, which the models utilised in the prediction tasks introduced in section 4.7 were trained on. The research questions specified in section 3.2 state that the dataset must be featurized in a number of ways to account for the different approaches made in their respective research questions.

4.3.1 Graph Generation

As the research questions defined in section 3.2 require training graph neural network models, a graph on which inference may be performed must be generated.

The Geuvadis data was passed to the Findr tool, as introduced in section 2.4.5. The Findr tool returns a 2-dimensional matrix in which the i-th item along the first axis and the j-th item along the second axis correspond to their respective gene indices. Each candidate edge (i, j) consists of a floating point number bound between 0 and 1. This value represents the Findr tool's predicted probability of the edge existing in the true causal graph.

The returned edge probability matrix was then passed back to the Findr tool, where the predicted edge probabilities were ranked by their significance, and only edges with a calculated probability greater than **0.80** were accepted. Edges were skipped if adding the given edge would cause a loop to appear in the graph. The generated causal DAG cannot be guaranteed to be weakly connected, as edges are selected in their order of significance. Using the NetworkX tool, the connected components of the graph were iterated over. Only the largest connected component is kept, vertices not in this component were dropped. The resulting graph is then guaranteed to be a connected DAG. In this experiment, it consists of 340 vertices and 344 edges.

Separate vertices are added to the graph to represent eQTL variation data. Edges are drawn from eQTL vertices to their affected gene expression vertices, based on the RNA sequencing data in the Geuvadis dataset.

4.3.2 Preprocessing

Train-, Validation-, and Test Split

To ensure fair testing across the 'linear', 'compact' and 'sparse' approaches, the dataset samples are split into training, validation, and test sets to ensure that the same samples appear in the datasets for the different approaches. The split was done via the `sklearn.model_selection.train_test_split()` function[32]. The resulting three datasets allocated 20% of dataset samples to the 'test' dataset, 76% to the training dataset, and 4% to the 'validation' dataset.

Scaling

Before passing gene expression data to the selected machine learning models, the training-, validation- and test datasets were scaled to have zero mean and standard deviation

equal to one. For each gene i and sample j, gene expression value $x_{i,j}$ were updated using equation 4.1. The resulting rescaled gene expression values $z_{i,j}$ were kept for use in model training. The `sklearn.preprocessing.StandardScaler()` implementation was used to perform the scaling[32].

$$z_{i,j} = \frac{(x_{i,j} - mean(\mathbf{X}))}{SD(\mathbf{X})}, \forall x_{i,j} \in \mathbf{X} \tag{4.1}$$

4.3.3 Linear Models Dataset

Having generated a graph which is guaranteed to be weakly connected, the data may be reformatted into datasets appropriate for the linear model approach. To benchmark the performance of the neural networks presented in this thesis, a featurization of the dataset appropriate for training linear models is required, as the featurizations which will be presented in the coming sections are not appropriate for models which do not have the permutation equivariance property.

The resulting output of the causal inference step performed in section 4.3.1 will provide a set of 340 genes with a variable number of causal parents. As was discussed in section 3.3, as a consequence of the variable number of inputs, there exists no single linear model which is able to make a prediction for all genes. This thesis therefore selects the approach of training a separate linear model for each gene with associated parent gene expression values, with a total of 269 models (due to the dataset containing 71 vertices with no parent genes).

In total, three distinct variations of the linear models dataset are created. Each dataset consists of in total 373 samples, available to use for training, validating and testing the models. The 'expression' linear models dataset for a given gene's input value consists of the gene's causal parents' gene expression values. Each dataset item's input value has its corresponding true output value.

The 'eQTL' linear models dataset for a given gene's input value consists of the gene's eQTL variation data for the given gene. The dataset items' corresponding output remains the same, being the true expression value of the current gene. A 'both' linear models dataset is also created for each gene, combining the inputs defined for datasets 'expression' and 'eQTL'. Again, each sample is also provided with its true output value.

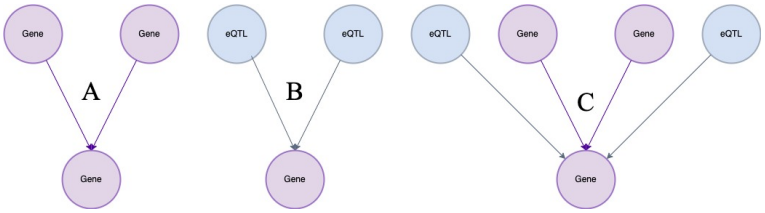


Figure 4.3: The datasets presented in section 4.3.3

A) corresponds to the 'expression' dataset. B) corresponds to the 'eQTL' dataset. C) corresponds to the 'both' dataset. In this case, the gene for which the model is predicting has two parent eQTL vertices, and two parent gene expression vertices.

4.3.4 Compact Graph-based Dataset

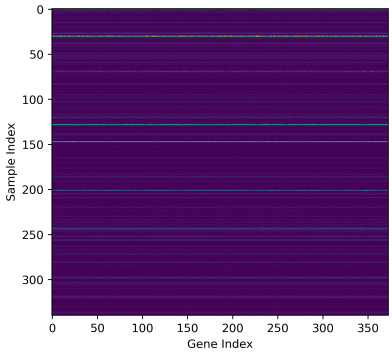
In section 3.2, research question 1) was to apply the graph feature autoencoder architecture. As the research questions differ, their corresponding dataset implementations must differ accordingly. The autoencoder-architecture transforms the set of features into a lower dimensional space (as described in section 2.2.7). To test the Graph Feature Autoencoder Architecture, a dataset in which all sample data is available to the model concurrently is required. As a consequence, where the approach in the linear models datasets specified in section 4.3.3 required creating separate datasets for each gene, the compact graph-based dataset is the same for each gene. Furthermore, whereas the each linear dataset consists of 373 samples, the compact dataset consists only of a single sample. As in section 4.3.3, the three variations keep their labels 'expression', 'eQTL' and 'both'.

Existing models which serve as a baseline for the work presented in this thesis represent their model input in a similar fashion [17]. The approach presented in this thesis differentiates itself however in including eQTL data. To achieve this, two different approaches may be used. One option consists of concatenating the two matrices along the vertical axis, and thereby creating a separate graph vertex for each for each eQTL. Though this is a sufficient solution, the a consequence of the approach is that a single model layer would have to train on two different feature types, one categorical and one real.

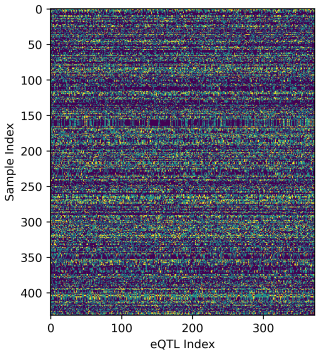
Another approach involves utilising PyTorch Geometric's *heterogeneous learning* framework[12]. In this approach, the data is kept as a separate pair of matrices. This approach involves multiple aggregation operators at each model layer, using the aggregation

operator to ensure graph features have the same shape in the latent space, concatenating the outputs of the different layer aggregation operators, and performing a linear transformation of the set to produce the layers' output. Using this approach increases the expressivity of the trained graph neural network, as it introduces a greater number of trainable parameters.

This thesis has applied this approach when performing the experiments designed to evaluate the research questions defined in section 3.2. In the 'expression' case, only the expression vertices with their respective values and $(expression - expression)$ edges are supplied. In the 'eQTL' case, eQTL-vertices with their respective values and $(eQTL - expression)$ edges are supplied. In the 'both' case, naturally, all vertices with their respective values, and all edge indices are supplied. Figure 4.4 displays the data matrices which are supplied to the neural network model in the compact case.



(a) Scaled dataset expression values in the matrix format in which they are passed to the model.



(b) Categorical eQTL values in the matrix format in which they are passed to the model

Figure 4.4: Compact graph-based dataset

4.3.5 Sparse graph-based Dataset

One further dataset is required to provide experimients that answer the research questions defined in section 3.2. The dataset defined in section 4.3.4 implicitly provides all dataset samples to the model concurrently. Unless explicitly alleviated, this means that the model may potentially learn its vertex prediction from the same vertex in a different sample. To circumvent this issue, a sparse graph-based dataset is defined. Where the dataset defined in section 4.3.4 provides expression values for all samples to the model concurrently, the sparse dataset provides expression values for a single sample at each training iteration.

The discussion of whether to use a concatenation- or heterogeneous approach from section 4.3.4 is also applicable to this dataset. The heterogeneous approach was also selected for this dataset, due to its hypothesised improvement in model expressivity.

4.4 Linear Training Setup

For each gene for which the model should be able to predict a gene expression value, the 373 item dataset is randomly split into a training-, validation- and test dataset. Using the training dataset, a linear model is instantiated for each gene in the dataset. The instantiated model's trainable parameters are fitted to best map the training dataset input to its ground truth corresponding output.

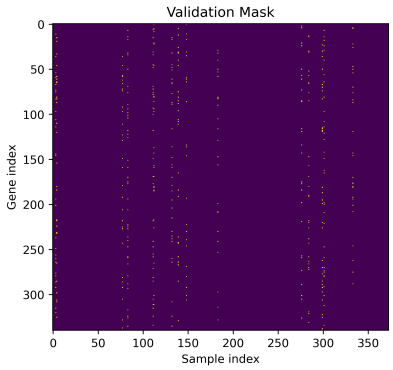
The resulting training loop is far less efficient than its graph-based counterparts, as the number of models needed to be fitted their corresponding sample data is far greater than for both the compact and the sparse approaches.

4.5 Compact Training Setup

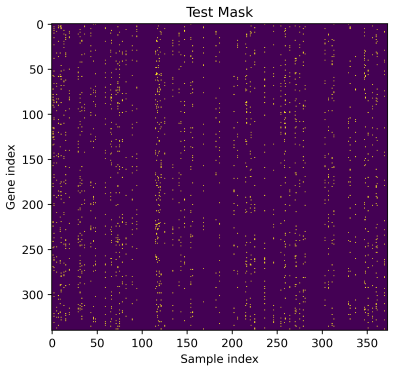
The featurized the compact model dataset in section 4.3.4 can be used to define an approach for training the compact model. The training approach used in this model follows the approach used to train Hasibi & Michoel's Graph Feature Autoencoder [17]. A boolean mask is generated in the same shape as the expression value matrix from figure 4.4a. Each data point's probability of being included in the boolean mask is given as a hyperparameter, the value of which can be found in Appendix C.

A copy is made of the boolean mask. A filter is then applied to the copied mask, where the mask cells in all vertices in all samples that do not appear in the validation set from the preprocessing step (as described in section 4.3.2) are set to the value 0.

As one goal of this thesis is to use a vertex's causal parent to reason about its feature values, predicting the expression values for vertices with no causal parents will be useless. As this thesis utilises the causal DAG generated in section 4.3.1, at least one vertex with no incoming edges must exist. The 'in'-degree of vertices is therefore studied, and set to 0 for all sample values in all masks if the vertex has no incoming 'expression' edges.



(a) Mask applied to the model output. Validation is calculated over datapoints at the yellow dots.



(b) Mask applied to the model output. Test loss is calculated over datapoints at the yellow dots.

Figure 4.5: Gene expression values over which validation- and test loss is calculated.

The resulting boolean mask is referred to as $\mathbf{M}_{validation}$. This process is repeated using the samples that do not appear in the test set. The resulting boolean mask is referred to as $\mathbf{M}_{test}$. The masks are illustrated in figure 4.5.

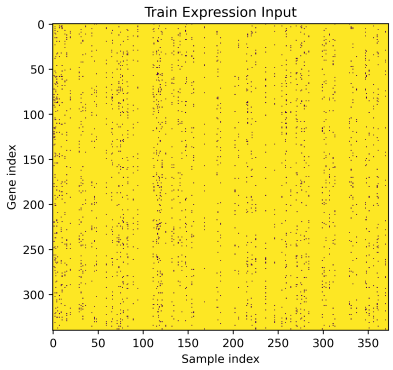
The logical *or* operation is applied to $\mathbf{M}_{validation}$ and $\mathbf{M}_{test}$. The logical *not* operation is then applied to the joint mask. The resulting mask will be applied to the expression data and is used as the model input. This mask is referred to as $\mathbf{M_input}$. Then, the mask value is set to 0 for all samples values where a vertex has no incoming 'expression' edges. The resulting mask is referred to as $\mathbf{M}_{train}$

By defining $\mathbf{M}_{cause}$ as a boolean mask in the same shape as the previous masks, with value 1 for all samples in vertices with no incoming 'expression' edges, 0 otherwise, the crucial property of equation (4.2) can be guaranteed.

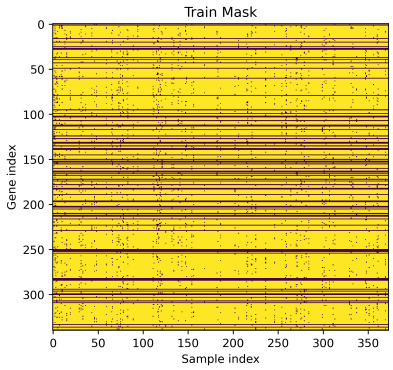
$$\mathbf{M}_{train} \equiv \neg(\mathbf{M}_{val} \cup \mathbf{M}_{test} \cup \mathbf{M}_{cause}) \tag{4.2}$$

Loss Calculation

Recall from section 4.7 describing model architectures, that a feature autoencoder is used in the compact case. As a consequence of this, the graph features are projected into a lower-dimensional feature space. The goal during model training is to reconstruct the given input data. Loss is therefore calculated as the mean of squared difference over the



(a) Yellow dots represent expression values provided to the model at every training iteration.



(b) Yellow dots represent expression values over which training loss is calculated.

Figure 4.6: Model input mask and training loss mask

predicted gene expression values and actual expression values over only the true vertices in the boolean train mask $\mathbf{M}_{train}$.

The compact graph-based dataset differentiates itself from the typical machine learning dataset in that rather than providing a single sample at a time, all sample data is provided to the model at every training iteration, until the model parameters converge. At each training step, validation loss is also calculated. Validation loss is calculated over the same output as training loss, however the validation mask is applied instead. At every epoch, the current model parameters replace the current 'best' model if the calculated validation loss is less than the 'best' validation loss.

In the evaluation step, having trained the best possible model with the given input and model architecture, the model is once again supplied with the same model input. In this scenario however, the test loss is evaluated only over the datapoints with corresponding true values in the test mask $\mathbf{M}_{test}$.

4.6 Sparse Training Setup

In sections 4.4 and 4.5, two different approaches in how to train their corresponding machine learning models were discussed. In this section, a final training approach will be discussed. In this experiment, a graph neural network model in which only gene expression data from the current sample is available to the model is required, in order to meed research questions **2. Can a graph based model be applied to predict**

missing gene expression values in a masked sample and **4. Can dataset gene expression values be extrapolated using only genome variation data?**.

The training setup made in the sparse graph-based case is one which bears similarity to both the approaches made in sections 4.4 and 4.5. The sparse graph-based dataset consists of 373 samples all containing the same graph structure (refer to section 4.3.5), but with differing expression values, and differing eQTL values if they are relevant for the current task. As in the linear training setup from section 4.4, the dataset is split into three subsets, known as the train-, validation- and test set.

Rather than initialising a separate linear model for each 'expression'-vertex in the dataset, a single graph-based model is initialised. Similarly to the compact case, there is still no obvious prediction target. This approach therefore also leverages the boolean mask approach made in the compact case, albeit somewhat simpler. The compact graph-based case is set up to utilise the autoencoder-architecture, and the measure of successfully training a model is therefore in how well the model is able to reconstruct itself. The sparse graph-based model cannot utilise an autoencoder-architecture, as each vertex contains only a single feature. The sparse graph-based training setup therefore bases itself on a more traditional prediction task. Where the compact graph-based case was able to maintain the same set of boolean masks for the training-, validation and test cases, the sparse graph-based case will have to generate a new boolean mask for each training iteration, as the model should be able to make a good prediction of any vertex in the graph, as long as at least one causal parent exists for the given vertex. In the work presented in this thesis, a series of masking approaches are attempted in an effort to discover one with good convergence properties, and which performs well in model evaluation. The results of applying the different approaches will be discussed in chapter 5.

Mask Percentage

A boolean mask $\mathcal{M}^{train}$ in the same shape as the expression value matrix is generated, in which a random subset of the vertices are given the value 0, and the rest are given the value 1. The generated mask selects a percentage of vertices with causal parents. The percentage values is given as a hyperparameter. The values of all hyperparameters can be found in Appendix C. The current sample's expression value matrix is multiplied with the generated boolean mask, and masked input is generated. This masked input is passed to the model, the model predicts some value for the masked vertices. A loss function is applied to the difference between the predicted values and the true values

found in the sample. The calculated loss is passed to the optimizer, which calculates the gradients, and updates the weights accordingly. The loss function used in training the sparse graph-based model was MSE Loss as defined in section 2.2.2. The optimizer used was the Adam optimizer, as defined in section 2.2.6.

This process repeated over a number epochs defined as a hyperparameter, until convergence. Though this process deviates from the training stage for the compact graph-based case, it is identical to how the compact graph-based case calculates its validation- and test loss.

$$\mathcal{L} = \sum_{v \in V} (\hat{y}_v - y_v)^2 \cdot ((\mathcal{M}_v^{train} + 1) \mod 2) \tag{4.3}$$

An edge case of the % mask is setting the percentage hyperparameter to 0. In this case, a one-hot mask will be generated.

4.7 Models and Architectures

Section 4.3 defined a set of datasets with which the research questions defined in section 3.2 can be answered, a set of machine learning models are needed. The following list provides an overview over the models which were used in attempting to fulfill the research questions proposed in section 3.2.

Many of the models presented require a set of hyperparameters which affect model performance. For each model, a list of possible hyperparameter combinations were created. Each model was initialised, trained and produced a loss measure over the validation set a series of times, at each iteration testing a separate set of hyperparameters. This process is known as Grid Search [28]. All hyperparameters for all models can be found in Appendix C.

- Linear Models
  - Linear Regressor
- Compact Graph-based Models
  - Graph Feature Autoencoder
  - Heterogeneous Graph Feature Autoencoder

- Sparse Graph-based Models
  - Homogenous Sparse Graph Neural Network
  - Heterogeneous Sparse Graph Neural Network

For linear regression, the implementation from the scikit-learn library was used[32]. For all graph based models, the PyTorch Geometric library was used[12].

Homogenous Graph Feature Autoencoder

The compact graph-based model for the 'expression' compact dataset is a graph feature autoencoder model. This architecture is taken from Hasibi & Michoel's Graph Feature Autoencoder. The implemented Graph Feature Autoencoder consists of four graph-aggregating layers and a single linear layer.

- The first layer is a GraphSAGE layer, first performing the aggregation function, then performing a linear transformation changing the dimension of the 373 features-per-vertex input, projecting it to a 64 features-per-vertex output. The ReLU function is then applied to the output.
- The second layer performs the same operation as the first. It performs the aggregation function, then projects the 64 features-per-vertex input to a 32 features-per-vertex output. The ReLU function is then applied to the output.
- The third layer is a linear layer is applied to the encoded data, keeping the 32 features-per-vertex dimensionality. The ReLU function is then applied to the output.
- The fourth layer is a GraphSAGE layer, performing the aggregation function over the encoded data, then performing a linear transformation of the aggregated data to a 64 features-per-vertex output. The ReLU function is then applied to the output.
- The fifth layer is a GraphSAGE layer, aggregating the output of the previous layer, then performing a linear transformation of the data to a 373 features-per-vertex output. This output serves as the output of the GNN.

Heterogeneous Graph Feature Autoencoder

The compact graph-based model shared for the 'both' and 'genotype' compact dataset are also based on the Graph Feature Autoencoder. The main difference is that each aggregation layer adds a separate aggregator for genotype data.

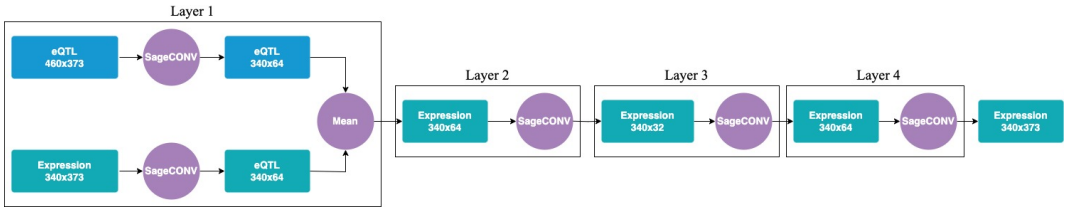


Figure 4.7: Compact heterogeneous model architecture
A larger version of this illustration is available in Appendix E.

- The first layer is a GraphSAGE layer, for each vertex performing the aggregation function over the incoming 'expression' vertices, then performing a linear transformation changing the dimension of the 373 features-per-vertex input, projecting it to a 64 features-per-vertex output. Then, using a separate GraphSAGE layer, for each 'expression' vertex, performing the aggregation function over incoming 'eQTL' vertices, changing the dimension of the 373 features-per-vertex input to a 64 features-per-vertex output. The two generated outputs are concatenated, transformed into a single 64 features-per-vertex output. The ReLU function is applied to this output.
- The following layers function the same way as the homogeneous model described in section 4.7. In the second layer, the output of the previous layer is aggregated, and a linear transformation projects the 64 features-per-vertex input to a 32 features-per-vertex output. The ReLU activation function is then applied to the output.
- The third layer is a linear layer applied to the encoded data, keeping the 32 features-per-vertex dimensionality. The ReLU function is applied to the output.
- The fourth layer is a GraphSAGE layer, performing the aggregation function over the encoded output of the linear layer, then performing a linear transformation transforming the 32 features-per-vertex aggregated input to a 64 features-per-vertex output.
- The fifth layer is a GraphSAGE layer, performing the aggregation function over the 64 features-per-vertex output over the previous layer. A linear transformation then projects the input to a 373 features-per-vertex output. This serves as the output of the GNN.

Homogeneous Sparse GNN

The homogeneous sparse GNN is designed to fit the sparse 'expression' dataset. It can be viewed as a simplification of the autoencoder-based models described in sections 4.7 and 4.7.

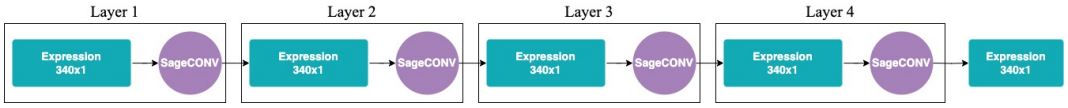


Figure 4.8: Sparse homogenous model architecture.
A larger version of this illustration is available in Appendix E.

- The fist layer is a GraphSAGE layer. As the sparse dataset splits samples, the layer performs the aggregation function over a 1 features-per-vertex input. The aggregated input is then subject to a linear transformation, keeping its single dimension in the output.
- The above layer is repeated 4 times, with the ReLU activation function applied between the layers. The final layer serves as the output of the GNN.

Heterogeneous Sparse GNN

Figure 4.9: Sparse heterogeneous model architecture.
The updated gene expression value is provided as input to each model layer. The ground truth eQTL data is provided at each model layer. A larger version of this illustration is available in Appendix E.

- The first layer consists of two GraphSAGE layers. Using one GraphSAGE layer, for each 'expression' vertex, the aggregation function is performed over the incoming 'expression' vertices. A linear transformation is applied to the aggregated data, preserving its single dimension. Using a separate GraphSAGE layer, for each 'expression' vertex, the aggregation function is performed over the incoming 'eQTL' vertices. A linear transformation is applied, preserving the aggregated data's single dimension. The two generated outputs are concatenated and transformed into a single features-per-vertex output. The ReLU function is applied to this output.
- The second layer consists of a single GraphSAGE layer. For every 'expression' vertex, the aggregation function is performed over all incoming 'expression' vertices. A linear transformation is then applied to the aggregated data, preserving its single dimension. The ReLU function is applied to this output.
- The third layer consists of a single GraphSAGE layer. For every 'expression' vertex, the aggregation function is performed over all incoming 'expression' vertices. A

linear transformation is then applied to the aggregated data, preserving its single dimension. The ReLU function is applied to this output.

- The fourth layer consists of a single GraphSAGE layer. For every 'expression' vertex, the aggregation function is performed over all incoming 'expression' vertices. A linear transformation is then applied to the aggregated data, preserving its single dimension. This serves as the GNN's output.

Chapter 5

Experiment Results and Evaluation

In this chapter the results from the experiment from chapter 4 are presented. First, in section 5.1 the MSE error over all vertices is presented. To provide some nuance to the general MSE statistic, in section 5.2 the median MSE error per vertex is presented. Section 5.3 presents the R^2 statistic, both overall and per vertex. The chapter ends by plotting the model prediction distributions in section 5.4, and comparing them to the ground truth distribution of gene expression values.

5.1 Evaluation of MSE Score

Table 5.1 displays each model's mean squared error loss over each vertex in the test set with at least one incoming vertex. The table rows each correspond to one of the models tested in the experiment. In parentheses, the model category of the given model is also denoted, being one of 'linear', 'sparse' and 'compact'. The table columns shows the task category for the given model. For the sparse and compact models, a single model is not able to make predictions for all three task categories. Task categories for which a model cannot make a prediction is denoted by a single '-'.

The table shows that over all models tested over all tasks, the best performing model is the set of linear regression models. As expected, the linear regression models performed better when providing both expression data and genotype data to the model, due to an increased number of input values to use in output prediction. Though the overall performance of the various machine learning models vary, this statement remains true for most models tested.

Model	Expression	Genotype	Both
Linear Regression	0.0366	0.0298	**0.0282**
Compact Homogenous	0.0453	-	-
Compact Heterogeneous	-	1.2659	0.0523
MLP	0.0769	-	-
Sparse Homogenous	1.2791	-	-
Sparse Heterogeneous	-	1.2296	1.3137

Table 5.1: Mean Squared Error loss over all vertices.

Table 5.1 also shows that the trained sparse GNN models' performance do not seem to be able to converge towards a set of parameters capable of performing helpful approximations of gene expression values.

5.2 Evaluation of MSE Per Vertex

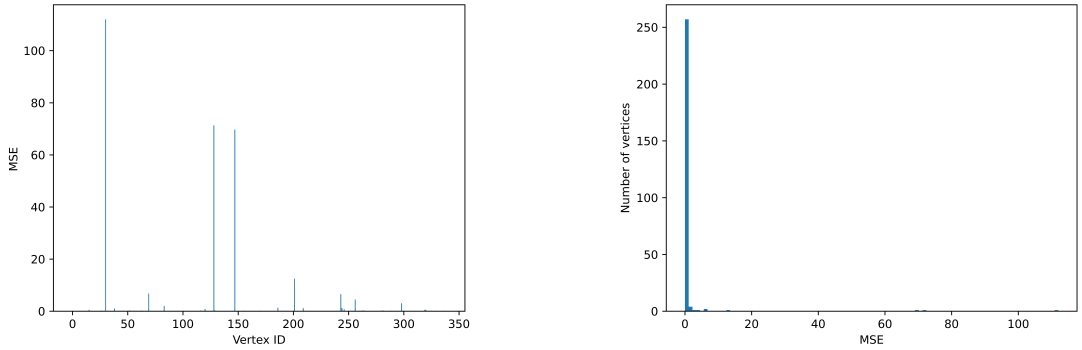
The result identified in section 5.1 is not without nuance. As the goal of this project has been to create a universal approximator of gene expression values, it is interesting to analyze the models' prediction accuracy per vertex. Rather than viewing MSE loss over all vertices, table 5.2 displays the median MSE loss grouped by target vertex. Viewing this statistic as well as the total MSE over all vertices gives some additional insight into the distribution of error over individual vertices.

Model	Expression	Genotype	Both
Linear Regression	**0.0008**	**0.0008**	0.0009
Compact Homogenous	0.0013	-	-
Compact Heterogeneous	-	0.0942	0.01
MLP	0.008	-	-
Sparse Homogenous	0.0183	-	-
Sparse Heterogeneous	-	0.2057	0.3933

Table 5.2: Median Mean Squared Error loss, grouped by vertex.

The table shows that the set of linear regression models remain the top performer. The table illustrates the large difference between the total MSE, and the median MSE

per vertex for all tested models. This comparison suggests that there exists a subset of vertices for which making predictions is more difficult than others.



(a) (b)

Figure 5.1: Sparse Heterogeneous MSE per node and histogram.

To further re-affirm this assumption, figure 5.1 is presented. Figure 5.1a illustrates the MSE loss for each predictable vertex in the test set using the *sparse heterogeneous* approach. The vertex indices in figure 5.1a remain the same as displayed throughout section 4.2. The gene names for each vertex index can be found in appendix A. Figure 5.1b illustrates the same data, but instead presents the data as a histogram over vertex loss. From the figure, it is obvious that the model is able to make good predictions for the majority of vertices, while it performs terribly for a smaller subset of vertices.

5.3 Evaluation of R^2 Score

Though a number of the models have been shown to be able to make predictions with relatively high accuracy, this measure does not capture the distribution of errors in comparison with the ground truth distribution of gene expression values for a given vertex. The term *determination* refers to the model's ability to mirror any vertex's distribution of true gene expression values in the model's predictions.

$$R^2 = 1 - \frac{\sum_i (y_i - f(x_i))^2}{\sum_i (y_i - \bar{y}_i)^2} \tag{5.1}$$

As a measure of model determination, the R^2-score for each model is presented, commonly referred to as the 'coefficient of determination'. The formula for R^2 is presented in

equation (5.1). A mean approximator, having no variance in its predictions, will produce a R^2-score of 0. A model which generates a squared error of 0 for each test sample, thereby exactly mirroring the distribution of the ground truth dataset, will achieve a R^2-score of 1. In this case, 100% of the variance in the ground truth dataset is modelled by the model.[36] By this definition, a model which produces a prediction distribution with greater squared error than a mean estimator will achieve a negative R^2-score.

As with the MSE case from sections 5.1 and 5.2, though the linear models continue to be the top performing models, the R^2-score must be presented in a series of ways to provide more insightful analysis of the actual result.

Table 5.3 shows the calculated R^2-score using all predictions and true values, regardless of which vertex any prediction belongs to. Given this presentation, the linear models are able to achieve a score of **0.9767**. However, the linear models are dependent on a large set of models to be able to provide this level of determination.

Viewing table 5.4 provides a better insight into the predictive performance of each individual linear model's determination, with the linear regression given the 'both' dataset being the top performing linear model, achieving a mean R^2-score of ***0.14***. Though the set of linear regression models remains the top performing model, this is no longer what would commonly be considered an acceptable R^2 for a predictive model.

This difference between the overall R^2 value over all vertices and the per vertex median R^2 value is illustrated in figure 5.2. The variation in the distribution of each individual gene's expression values is less than the distribution of gene expression values over all vertices. Therefore, though by the definition of R^2, a mean approximator should produce an R^2 of 0, measuring the R^2 value over all genes using separate mean approximators for each gene will produce an artificially high R^2-score.

The tendency observed in the top performing linear regression model also holds for the compact graph-based model. While the compact heterogeneous model is able to achieve an R^2-score of ***0.9601*** using the 'expression' dataset and ***0.9569*** when using the 'both' dataset, when measuring by all vertices' predictions and true values. When calculating the R^2-score by grouping the predictions and true values for each vertex, the mean R^2-score achieved is indicative of a model with minimal determination. Regarding this result as it correlates to the compact heterogeneous 'both'-result from table 5.2 indicates that the predictive determination of this model is poor.

This statement remains true when observing the R^2-scores generated by the sparse models. Though the models are able to achieve acceptable prediction accuracy for a

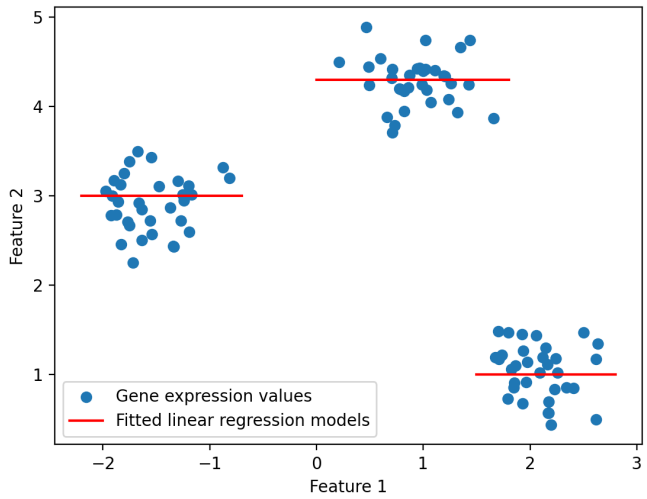


Figure 5.2: Illustrated difference between overall R^2 and median R^2

subset of vertices as per the result from table 5.2, the models' determination are obviously unacceptably low when viewing tables 5.3 and 5.4.

Model	Expression	Genotype	Both
Linear Regression	0.9697	0.9753	**0.9767**
Compact Homogenous	0.9601	-	-
Compact Heterogeneous	-	-0.0018	0.9569
MLP	0.9397	-	-
Sparse Homogenous	-0.0575	-	-
Sparse Heterogeneous	-	-0.0157	-0.0856

Table 5.3: R^2-score over all predictions

5.4 Prediction distributions

In section 4.2, when illustrating the distribution of gene expression values, any vertex's distribution of expression values was shown to roughly follow a gaussian distribution. It therefore follows that given the definition of the term *determination*, an optimally determined model's predictions should therefore also follow a gaussian distribution as the number of test samples grows indefinitely. In this section the term *expressivity* refers

Model	Expression	Genotype	Both
Linear Regression	0.0183	-0.0027	**0.1451**
Compact Homogeneous	-0.2472	-	-
Compact Heterogeneous	-	-94.952	-8.2793
MLP	-5.2656	-	-
Sparse Homogeneous	-16.6702	-	-
Sparse Heterogeneous	-	-196.5096	-437.243

Table 5.4: Mean R^2-score, grouped by vertex

to the predictive distribution for a given model. This term differs from the *determination* term defined in section 5.3, as in this seciton, only the variance of a given model's predictions are studied, while errors are ignored.

Throughout this section, each model's predictive distribution over an example vertex is illustrated and discussed. As the figures illustrate tendencies in model predictions, model performance should not be judged based only on this measure. As the GNN models aim to generalize over all vertices, any given vertex's prediction distribution may be a good example for one model, while a poor example for another. Overall model expressivity should therefore not be based on the following illustrations, but rather tables 5.3 and 5.4

Linear Regression

As an study in the tested models' expressivity, the distribution of vertex predictions over the test set using the 'both' dataset and the linear regression models is illustrated in figure 5.3. Vertex 200 was selected to illustrate this distribution because its expression values are one of the distributions illustrated in section 4.2. From the illustration it is obvious that even using the best scenario, the approach is unable to reconstruct the ground truth distribution of gene expression values.

Figure 5.4 shows the distribution of predicted gene expression values for vertex 200 using the linear regression model and the 'genotype' dataset. As the vertex has only a single categorical eQTL parent value, it comes as no surprise that the model is less expressive than its 'both'-dataset counterpart from figure 5.3.

Figure 5.5 displays the distribution of predicted gene expression values for vertex 200 sing the linear regression model and the 'expression' dataset. As one may hypothesise

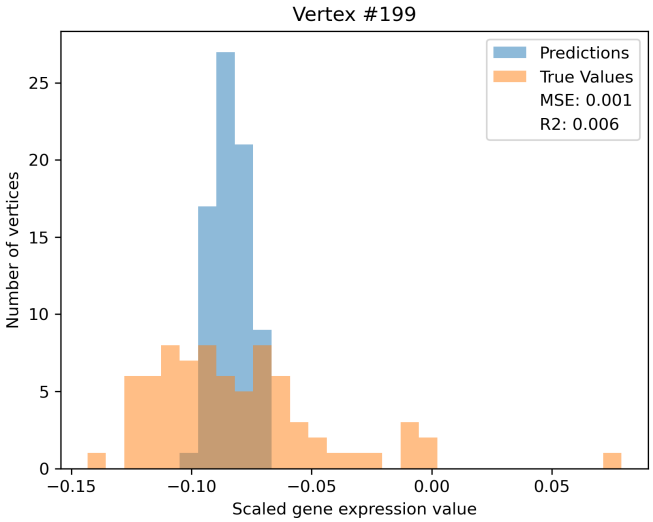


Figure 5.3: Linear Regression model's prediction distribution against the ground truth distribution of gene expression values using the 'both' dataset.

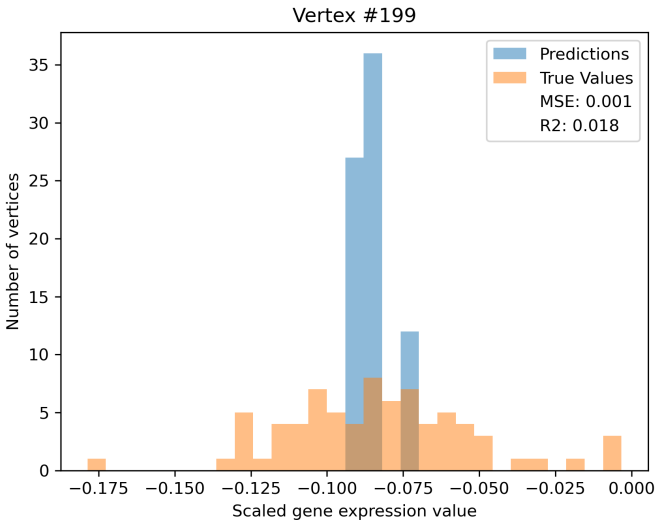


Figure 5.4: Linear Regression model's prediction distribution against the ground truth distribution of gene expression values using the 'genotype' dataset.

from table 5.4, the model is able to more accurately reconstruct the distribution of vertex 199 using the 'expression' dataset than the 'genotype' dataset, but not as well when using the 'both' dataset.

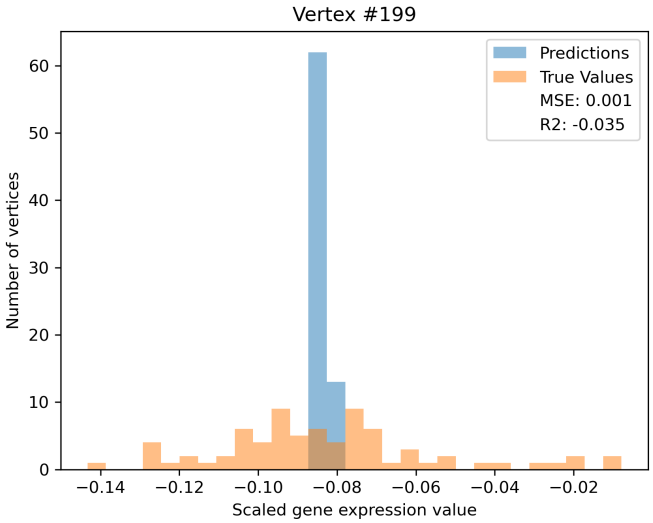


Figure 5.5: Linear Regression model's prediction distribution against the ground truth distribution of gene expression values using the 'expression' dataset.

Compact GNN

Figures 5.6, 5.7 and 5.8 display the compact GNN approach's prediction distribution using respectively the 'both', 'genotype', and 'expression' datasets. A pattern similar to what was found using the linear models is found in these distributions. Again, the model is best able to reconstruct the expression distribution using the 'both' dataset

Sparse GNN

Figures 5.9, 5.10 and 5.11 shows that the sparse GNN approach is not able to produce a distribution of predictions which mirrors the ground truth distribution of gene expression values when given any of the 'both', expression' or 'genotype' datasets. This conclusion supported by tables 5.3 and 5.4's calculated R^2-scores.

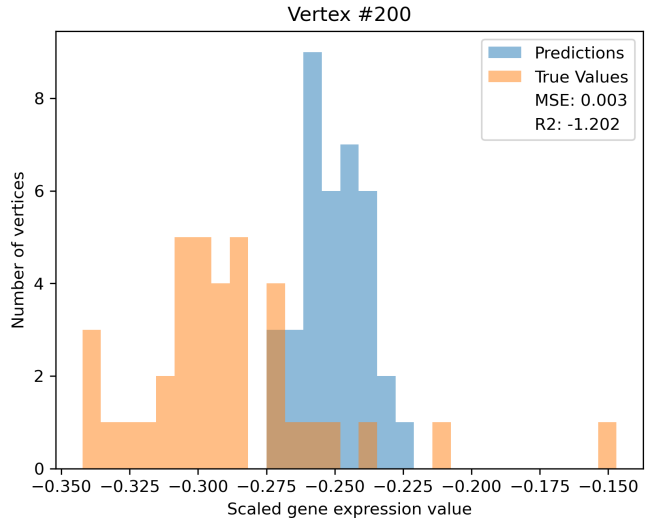


Figure 5.6: Heterogeneous Compact GNN model's prediction distribution against the ground truth distribution of gene expression values using the 'both' dataset.

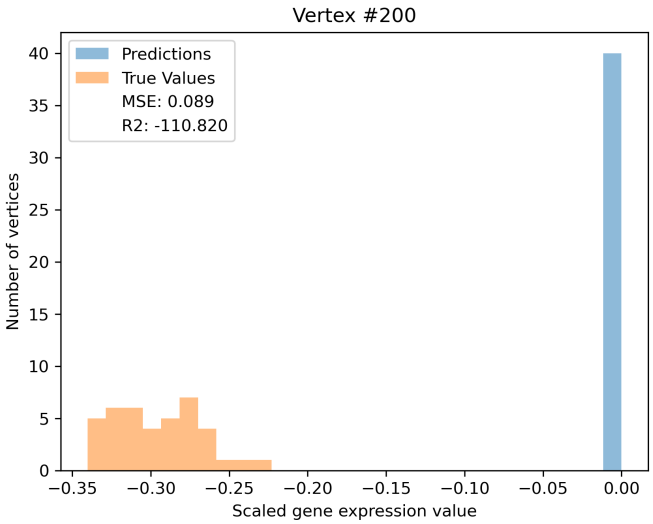


Figure 5.7: Heterogeneous Compact GNN model's prediction distribution against the ground truth distribution of gene expression values using the 'genotype' dataset.

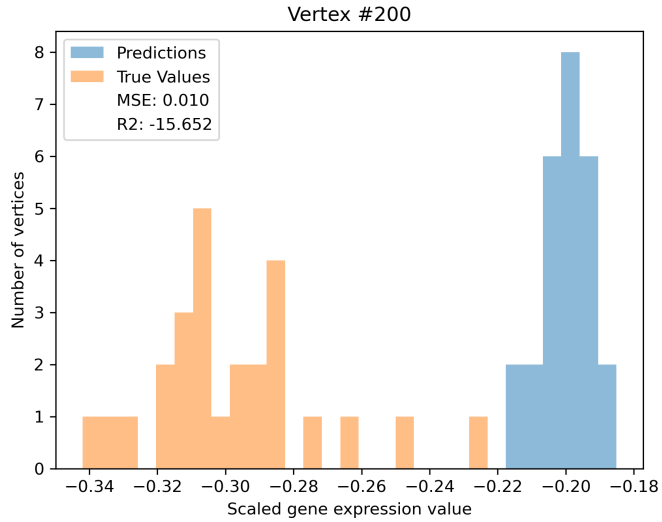


Figure 5.8: Heterogeneous Compact GNN model's prediction distribution against the ground truth distribution of gene expression values using the 'expression' dataset.

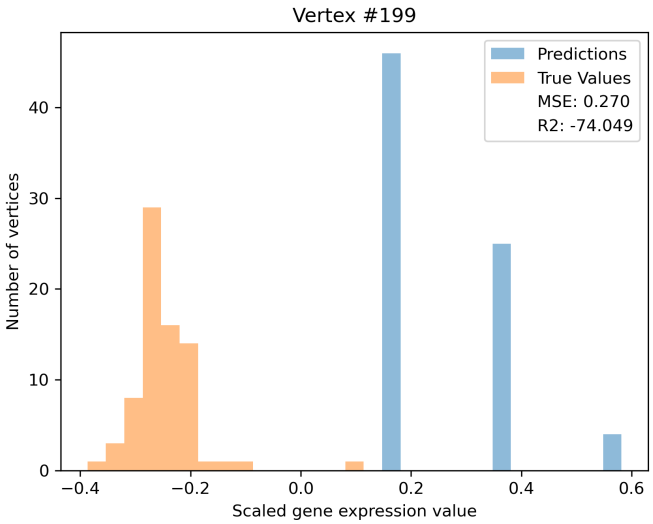


Figure 5.9: Heterogeneous Sparse GNN model's prediction distribution against the ground truth distribution of gene expression values using the 'both' dataset.

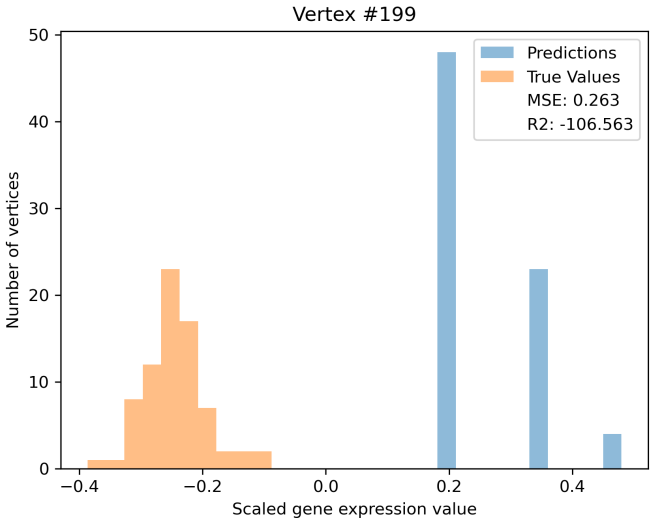


Figure 5.10: Heterogeneous Sparse GNN model's prediction distribution against the ground truth distribution of gene expression values using the 'genotype' dataset.

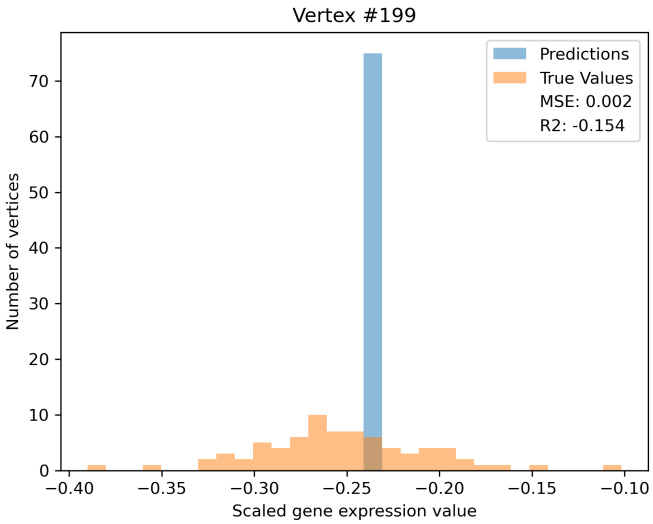


Figure 5.11: Heterogeneous Sparse GNN model's prediction distribution against the ground truth distribution of gene expression values using the 'expression' dataset.

Chapter 6

Disucssion

This chapter concludes the work presented in this thesis. The chapter begins by in section 6.1 analysing the poor performance of the sparse GNN model. The research questions defined in section 3.2 outlined the requirements of the experiment described in chapter 4, the results of which was presented in chapter 5. These results provide sufficient groundwork to answer the research questions posed, given the approach made by this thesis. The questions are answered in section 6.2. Section 6.3 discusses what steps may be taken in the future to improve the gene expression inference capabilities of the created models.

6.1 Sparse GNN Parameter Convergence

This section provides an analysis into the poor performance achieved utilising the sparse graph-based approach. Studying the reason behind this model's lack of performance is essential, as a major goal of this thesis was to use graph neural networks to predict gene expression values based only on its causal parents' expression values. The analysis will be made by studying the trained parameters used predicting the test vertices.

In an effort to create a universally approximating graph neural network model for gene expressions using the 'sparse' dataset, a varied set of model architectures were explored. In the graph, the longest path contains 5 edges. As a consequence, any Graph Neural Network with greater than 5 layers will not aggregate any unseen vertex information, but may still serve as a non-linear transformation of the aggregated vertex information.

For this reason, during the experiment to determine the optimal number of model layers, network models with depths ranging from 1 to 6 layers were tested. The testing showed no substantial difference between the number of layers. The four layer model was used as it was the marginally highest performing model.

6.2 Research Questions

Does prediction accuracy improve when also providing genome variation data in the dataset?

Illustrated in tables 5.1 and 5.3, this thesis concludes that providing genome variation data does improve prediction accuracy for the linear models and the compact graph-based model. Providing genome variation data does not improve prediction accuracy in the sparse graph-based model.

Can the Graph Feature Autoencoder architecture be applied to predict missing gene expression values in a masked dataset?

In answering this question, the 'compact' rows from tables 5.3 and 5.4 are studied. Over all predictions, the compact heterogeneous model is able to produce an R^2-score of **0.96** using the 'both' and 'expression' datasets. This result indicates a model in which **96%** of variation in the dependent variable can be described by the model. From tables 5.1 and 5.2, we are able to tell that this model suffers the problem of being a good predictor of most vertices, while being unable to make good predictions for a smaller subset of vertices discussed in sections 5.1 and 5.2. The compact heterogeneous model achieves a MSE of **0.04** when using the 'expression' dataset. As previously mentioned, the median MSE per vertex value achieved is far better. Grouped by vertex, the median MSE the model achieves is **0.0013**.

This research question is meant to serve as a reference to Hasibi & Michoel's 2021 paper *A Graph Feature Auto-Encoder for the prediction of unobserved node features on biological networks* [17]. The proposed Graph Feature Autoencoder proposed in this paper is able to serve as a universal approximator of masked graph vertices, obtaining MSE loss values of **0.025**, **0.023**, **0.025**, and **0.003**, depending on the dataset used.

Though this thesis provides a model which makes accurate predictions for a large number of vertices, this thesis has not been able to fit a universal approximator to the dataset created. This thesis must therefore conclude that using the approach described in chapter 4, the graph feature autoencoder architecture cannot be applied to predict missing gene expression values in a masked dataset in this experiment.

Can a graph neural network be applied to predict missing gene expression values in a masked sample, only given its causal parents' values?

A central research question posed in section 3.2 is whether a graph based model was able to predict missing gene expression values in a masked sample, only given its causal parents' gene expression and genome variation values. Tables 5.1, 5.2, 5.3 and 5.4 forms the basis of this thesis' conclusion.

As a universal approximator for all vertices in the causal graph identified in section 4.3.1, the sparse graph-based approach fails. From figure 5.1a, the sparse graph-based approach is shown to perform poorly for a subset of vertices. This causes the overall MSE error of the sparse graph-based approach to be the worst of all models tested. Though this overall performance is nuanced by the fact that a majority of vertices are predictable by the model, this thesis cannot conclude that the project detailed in this thesis resulted in a universal approximator for all vertices.

In explaining why the sparse graph-based approach was unable to serve as a universal approximator for all vertices, the model as it was described in section 4.7 is studied. The model parameters used in prediction over the test vertices are presented in table 6.1.

Layer	Weight	Bias
Layer 1	-0.7956	0.3972
Layer 2	0.4583	-0.4583
Layer 3	0.0334	-2.0066
Layer 4	-1.5201	-0.7233

Table 6.1: Sparse Heterogeneous model's trained model parameters using the 'expression' dataset.

This thesis concludes that the downside of having only two trainable parameters at each network layer is the leading cause of the poor performance of the graph based sparse model. The limited number of parameters is a consequence of the featurization of the dataset, with each vertex containing only a single feature, as a consequence of limiting

the model input to a single individual's gene expression values. Note that the weight parameter at layer 3 is close to 0. This tendency suggests that the model in a large degree discards the propagated expression values found in layers 1-3 and largely bases its output prediction on a normalized guess.

Can dataset gene expression values be extrapolated using only genome variation data?

To answer the question of whether gene expression values may be extrapolated using only genome variation data, the results presented in tables 5.3 and 5.4 forms the base of this thesis' conclusion.

The linear regression model using only genome variation data achieved a mean R^2-score of **-0.0027**. This means that on average, none of the variance of the prediction targets in the test set is accounted for using the linear models approach. This thesis therefore concludes that using the method outlined in chapter 4, gene expression values cannot be extrapolated only using causal parents' genome variation data.

In regards to the graph-based models, the sparse approach achieved a R^2-score of **-196.5096** when only supplied with genome variation data. The compact approach achieved a R^2-score of **-94.95**. This thesis therefore concludes that using the experiment outlined in section 4.6, a graph based model could not accurately extrapolate gene expression values for a given gene given only its causal parents' genome variation data in this experiment. We hypothesise that the categorical input space, limits the possible expressivity in the output space. This limitation is expanded by the aggregation function performed over eQTL vertices.

In our experiment to determine if graph neural networks are an effective alternative to existing linear approaches in gene expression inference, we found that the graph neural networks proposed in this experiment were not a viable replacement to existing methods.

6.3 Further work

Inferring gene expression values in tissue where obtaining samples is not trivial remains an interesting topic of research, as it may be helpful in determining an individual's probability of developing disease. The experiment outlined in this thesis was unsuccessful in

inferring masked gene expression values. It is possible to make a number of changes in the experiment setup outlined in chapter 4.

The changes suggested by this thesis largely pertain to the graph structure used to make predictions. In this thesis, a DAG was used. The same experiment could be attempted using an undirected graph in which cycles were allowed. In this case, vertices may be able to receive feedback on messages passed to downstream genes, mimicking real life biological networks.

In the featurization step, only the largest connected graph component was kept when building the graph. This step could have been omitted, and the resulting graph would have remained a valid input to the graph neural networks used in this thesis. This would provide the graph-based models with a higher number of vertices over which predictions could be made.

Related work in gene expression inference used samples' age, sex and ethnicity in the regression models in order to make predictions. These values could have been added to the graph, represented by vertices with directed edges to all 'expression' vertices.

The models used in this thesis all used the SAGEConv aggregation operator. Other operators, such as Corso et. al's Principal Neighbourhood Aggregation operator [10], could be used in order to potentially improve prediction accuracy.

Glossary

adjacency matrix A 2-dimensional matrix with number of rows and columns equal to the number of vertices in the graph. If an edge is present between a pair of vertices (i, j), the j-th column of the i-th row will be equal to 1, 0 otherwise..

causal anchor Directed Acyclig Graph vertex with no incoming vertices. eQTLs serve as causal anchors as their variation values are unaffected by the gene expression values..

eQTL Expression Quantitative Trait Loci.

Gene expression The rate at which a gene produces the protein it encodes..

Genome An organism's genetic material..

Graph Neural Networks A type of neural network specialised in performing inference on graph structures..

LLR Log Likelihood Ratio, a statistic used to compare hypotheses in statistical hypothesis testing..

locus A specific location in the genome..

MAE Mean Absolute Error.

MLE Maximum Likelihood Estimate.

MSE Mean Squared Error.

permutation equivariant A function where the result of applying any permutation matrix to a function's input yields the same output as applying the permutation matrix to the function output with the true input..

permutation invariant A function where the result of applying any permutation matrix to a function's input yields the same output..

SNP Single Nucleotide Polymorphism.

topological sorting A way so sort graph vertices in a directed graph such that at any point, moving right in the sort order gurantees the next vertex is not a parent of the current vertex..

Bibliography

[1] 1000 Genomes Project Consortium, Adam Auton, Lisa D Brooks, Richard M Durbin, Erik P Garrison, Hyun Min Kang, Jan O Korbel, Jonathan L Marchini, Shane McCarthy, Gil A McVean, and Gonçalo R Abecasis. A global reference for human genetic variation. *Nature*, 526(7571):68–74, October 2015.

[2] Solveig Badillo, Balazs Banfai, Fabian Birzele, Iakov I. Davydov, Lucy Hutchinson, Tony Kam-Thong, Juliane Siebourg-Polster, Bernhard Steiert, and Jitao David Zhang. An introduction to machine learning. *Clinical pharmacology and therapeutics*, 107(4):871–885, 2020. ISSN 0009-9236.

[3] R. Balakrishnan and K. Ranganathan. *A textbook of graph theory*, page 3–3. Springer, 2012.

[4] Jørgen Bang-Jensen. Digraphs: Theory, algorithms, and applications, 2000.

[5] Mahashweta Basu, Kun Wang, Eytan Ruppin, and Sridhar Hannenhalli. Predicting tissue-specific gene expression from whole blood transcriptome. *Science Advances*, 7 (14):eabd6991, 2021. doi: 10.1126/sciadv.abd6991.
URL: https://www.science.org/doi/abs/10.1126/sciadv.abd6991.

[6] Sarah A. Bates. Chromosome, 2023.
URL: https://www.genome.gov/genetics-glossary/Chromosome. Accessed 02-04-2023.

[7] Boundless. The genetic code - the relationship between genes and proteins, 2022.
URL: https://bio.libretexts.org/Bookshelves/Introductory_and_General_Biology/Book%3A_General_Biology_(Boundless)/15%3A_Genes_and_Proteins/15.01%3A_The_Genetic_Code_-_The_Relationship_Between_Genes_and_Proteins#:~:text=Protein%2Dencoding%20genes%20specify%20the,life%20as%20we%20know%20it. Accessed 02-04-2023.

[8] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

[9] PITHI CHANVORACHOTE, NICHARAT SRIRATANASAK, and NONGYAO NONPANYA. C-myc contributes to malignancy of lung cancer: A potential anticancer drug target. *Anticancer Research*, 40(2):609–618, 2020. ISSN 0250-7005. doi: 10.21873/anticanres.13990.
URL: https://ar.iiarjournals.org/content/40/2/609.

[10] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Velickovic. Principal neighbourhood aggregation for graph nets. *CoRR*, abs/2004.05718, 2020.
URL: https://arxiv.org/abs/2004.05718.

[11] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Nature, Berlin, Heidelberg, fifth edition edition, 2017. ISBN 3662536226.

[12] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019.
URL: http://arxiv.org/abs/1903.02428.

[13] John Gearhart, Evanthia E. Pashos, and Megana K. Prasad. Pluripotency redux — advances in stem-cell research. *New England Journal of Medicine*, 357(15):1469–1472, 2007. doi: 10.1056/NEJMp078126.
URL: https://doi.org/10.1056/NEJMp078126. PMID: 17928593.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[15] Aric Hagberg, Pieter Swart, and Daniel Chult. Exploring network structure, dynamics, and function using networkx. 01 2008.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[17] Ramin Hasibi and Tom Michoel. A graph feature auto-encoder for the prediction of unobserved node features on biological networks. *BMC Bioinformatics*, 22(1):525, 2021. doi: 10.1186/s12859-021-04447-3.
URL: https://doi.org/10.1186/s12859-021-04447-3.

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York, New York, NY, second edition edition. ISBN 0387848576.

[19] Omar Hernandez Rodriguez and Jorge Fernández. A semiotic reflection on the didactics of the chain rule. *The Montana Mathematics Enthusiast*, 7:321–332, 07 2010. doi: 10.54870/1551-3440.1191.

[20] National Human Genome Research Institute. Human genomic variation, 2023.
URL: https://www.genome.gov/about-genomics/educational-resources/fact-sheets/human-genomic-variation. Accessed: 14-08-2023.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[22] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.
URL: http://arxiv.org/abs/1609.02907.

[23] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *Aiche Journal*, 37:233–243, 1991.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. doi: 10.1145/3065386.

[25] Miroslav Kubat. *An Introduction to Machine Learning*. Springer Nature, Cham, 2nd ed. 2017 edition, 2017. ISBN 9783319639130.

[26] Tuuli Lappalainen, Michael Sammeth, Marc Friedländer, Peter Hoen, Jean Monlong, Manuel Rivas, Mar Gonzàlez Porta, Natalja Kurbatova, Thasso Griebel, Pedro Ferreira, Matthias Barann, Thomas Wieland, Liliana Greger, Maarten Iterson, Jonas Almlöf, Paolo Ribeca, Irina Pulyakhina, Daniela Esser, Thomas Giger, and Emmanouil Dermitzakis. Transcriptome and genome sequencing uncovers functional variation in humans. *Nature*, 501, 09 2013. doi: 10.1038/nature12531.
URL: https://www.researchgate.net/publication/256611581_Transcriptome_and_genome_sequencing_uncov

[27] Tong Ihn Lee and Richard A Young. Transcriptional regulation and its misregulation in disease. *Cell*, 152(6):1237–1251, March 2013.

[28] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas. 2019.

[29] nature.com. Snp.
URL: https://www.nature.com/scitable/definition/snp-295/. Accessed 02-04-2023.

[30] Alexandra C Nica and Emmanouil T Dermitzakis. Expression quantitative trait loci: present and future. *Philos. Trans. R. Soc. Lond. B Biol. Sci.*, 368(1620):20120362, May 2013.

[31] Tong P, Monahan J, and Prendergast JGD. Shared regulatory sites are abundant in the human genome and shed light on genome evolution and disease pleiotropy. *PLoS Genet 13(3): e1006673.*, 03 2017. doi: https://doi.org/10.1371/journal.pgen.1006673.

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

[33] Brandon L. Pierce, Lin Tong, Lin S. Chen, Ronald Rahaman, Maria Argos, Farzana Jasmine, Shantanu Roy, Rachelle Paul-Brutus, Harm-Jan Westra, Lude Franke, Tonu Esko, Rakibuz Zaman, Tariqul Islam, Mahfuzar Rahman, John A. Baron, Muhammad G. Kibriya, and Habibul Ahsan. Mediation analysis demonstrates that trans-eqtls are often explained by cis-mediation: A genome-wide analysis among 1,800 south asians. *PLOS Genetics*, 10(12):e1004818, 2014. doi: 10.1371/journal.pgen.1004818.
URL: `https://app.dimensions.ai/details/publication/pub.1021600128`.
https://journals.plos.org/plosgenetics/article/file?id=10.1371/journal.pgen.1004818type=printable

[34] Leslie Pray. Eukaryotic genome complexity. *Nature Education 1(1):96*, 2008.
URL: `https://www.nature.com/scitable/topicpage/eukaryotic-genome-complexity-437/#`.

[35] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sinčák. A review of activation function for artificial neural network. *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 01 2020. doi: 10.1109/SAMI48414.2020.9108717.

[36] Olivier Renaud and Maria-Pia Victoria-Feser. A robust coefficient of determination for regression. *Journal of Statistical Planning and Inference*, 140(7):1852–1862, 2010. ISSN 0378-3758. doi: https://doi.org/10.1016/j.jspi.2010.01.008.
URL: `https://www.sciencedirect.com/science/article/pii/S0378375810000194`.

[37] Kenneth H Rosen. Discrete mathematics and its applications, 2019.

[38] Robert Sedgewick and Kevin Wayne. Algorithms (4th edition), 2011.

[39] S.P. Simna and Zongchao Han. Prospects of non-coding elements in genomic DNA based gene therapy. *Current Gene Therapy*, 22(2):89–103, April 2022. doi: 10.2174/

156652322166621041909035 7.

URL: https://doi.org/10.2174/156652322166621041909035 7.

[40] Lingfei Wang and Tom Michoel. Efficient and accurate causal inference with hidden confounders from genome-transcriptome variation data. *PLOS Computational Biology*, 13(8):1–26, 08 2017. doi: 10.1371/journal.pcbi.1005703.

URL: https://doi.org/10.1371/journal.pcbi.1005703.

[41] Wenjian Xu, Xuanshi Liu, Fei Leng, and Wei Li. Blood-based multi-tissue gene expression inference with Bayesian ridge regression. *Bioinformatics*, 36(12):3788–3794, 04 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btaa239.

URL: https://doi.org/10.1093/bioinformatics/btaa239.

Appendix A

Gene Names and Indices

1	ENSG00000178585.10	41	ENSG00000136280.10	81	ENSG00000164414.11
2	ENSG00000163584.13	42	ENSG00000075239.9	82	ENSG00000107537.9
3	ENSG00000158623.10	43	ENSG00000119950.14	83	ENSG00000165832.4
4	ENSG00000182512.4	44	ENSG00000133195.6	84	ENSG00000084623.7
5	ENSG00000160213.5	45	ENSG00000161217.6	85	ENSG00000090432.5
6	ENSG00000171103.6	46	ENSG00000100983.5	86	ENSG00000153933.5
7	ENSG00000131148.3	47	ENSG00000167634.6	87	ENSG00000174652.12
8	ENSG00000160781.9	48	ENSG00000177946.4	88	ENSG00000141295.8
9	ENSG00000197956.4	49	ENSG00000113812.9	89	ENSG00000164190.11
10	ENSG00000254470.2	50	ENSG00000186184.11	90	ENSG00000196267.6
11	ENSG00000112697.10	51	ENSG00000163002.8	91	ENSG00000181027.5
12	ENSG00000085415.10	52	ENSG00000115306.10	92	ENSG00000055211.7
13	ENSG00000154889.11	53	ENSG00000101444.7	93	ENSG00000180185.7
14	ENSG00000204149.4	54	ENSG00000110108.3	94	ENSG00000135336.8
15	ENSG00000197498.7	55	ENSG00000090054.8	95	ENSG00000118217.5
16	ENSG00000012124.9	56	ENSG00000031698.8	96	ENSG00000117362.8
17	ENSG00000261311.1	57	ENSG00000131042.9	97	ENSG00000179934.5
18	ENSG00000085982.8	58	ENSG00000182749.5	98	ENSG00000145386.5
19	ENSG00000013503.4	59	ENSG00000196365.5	99	ENSG00000147853.10
20	ENSG00000125445.5	60	ENSG00000125637.10	100	ENSG00000172500.8
21	ENSG00000204632.7	61	ENSG00000130513.3	101	ENSG00000233610.1
22	ENSG00000163875.11	62	ENSG00000245532.2	102	ENSG00000141759.9
23	ENSG00000169245.4	63	ENSG00000231507.1	103	ENSG00000132196.8
24	ENSG00000101181.12	64	ENSG00000259705.1	104	ENSG00000089693.6
25	ENSG00000104972.10	65	ENSG00000171045.10	105	ENSG00000109084.8
26	ENSG00000167740.3	66	ENSG00000167081.10	106	ENSG00000157557.7
27	ENSG00000168894.5	67	ENSG00000188223.7	107	ENSG00000087157.11
28	ENSG00000155463.7	68	ENSG00000086289.7	108	ENSG00000177721.3
29	ENSG00000189339.7	69	ENSG00000153898.8	109	ENSG00000167173.13
30	ENSG00000166548.10	70	ENSG00000006075.10	110	ENSG00000164253.8
31	ENSG00000204525.8	71	ENSG00000105618.8	111	ENSG00000185127.5
32	ENSG00000171469.6	72	ENSG00000105063.12	112	ENSG00000198231.6
33	ENSG00000183291.11	73	ENSG00000226742.2	113	ENSG00000105245.4
34	ENSG00000169660.10	74	ENSG00000108666.4	114	ENSG00000171940.8
35	ENSG00000197771.8	75	ENSG00000132819.12	115	ENSG00000160949.11
36	ENSG00000106771.7	76	ENSG00000155275.14	116	ENSG00000106772.11
37	ENSG00000126246.4	77	ENSG00000167965.12	117	ENSG00000147684.3
38	ENSG00000185164.10	78	ENSG00000164284.10	118	ENSG00000164933.5
39	ENSG00000104894.6	79	ENSG00000137841.7	119	ENSG00000109519.7
40	ENSG00000053501.7	80	ENSG00000148335.8	120	ENSG00000198624.7

Table A.1: Gene names and their respective indices, part 1

121	ENSG00000106153.12	161	ENSG00000183570.9	201	ENSG00000119599.12
122	ENSG00000105928.9	162	ENSG00000216937.5	202	ENSG00000140105.11
123	ENSG00000101577.4	163	ENSG00000180901.5	203	ENSG00000232703.1
124	ENSG00000040199.13	164	ENSG00000089250.13	204	ENSG00000218739.4
125	ENSG00000164978.12	165	ENSG00000221923.3	205	ENSG00000115216.8
126	ENSG00000175354.11	166	ENSG00000118640.6	206	ENSG00000108294.3
127	ENSG00000108433.10	167	ENSG00000184517.7	207	ENSG00000196756.5
128	ENSG00000144034.10	168	ENSG00000161265.6	208	ENSG00000148841.10
129	ENSG00000186468.8	169	ENSG00000165714.6	209	ENSG00000099282.4
130	ENSG00000084070.6	170	ENSG00000189409.7	210	ENSG00000136810.8
131	ENSG00000131238.10	171	ENSG00000204084.6	211	ENSG00000196367.7
132	ENSG00000196700.3	172	ENSG00000121691.4	212	ENSG00000186020.7
133	ENSG00000125430.4	173	ENSG00000177283.4	213	ENSG00000180616.3
134	ENSG00000117335.13	174	ENSG00000246273.2	214	ENSG00000205038.7
135	ENSG00000141141.9	175	ENSG00000185963.9	215	ENSG00000256771.1
136	ENSG00000224565.1	176	ENSG00000162384.7	216	ENSG00000011105.7
137	ENSG00000132423.6	177	ENSG00000171135.9	217	ENSG00000134077.10
138	ENSG00000204227.4	178	ENSG00000155307.13	218	ENSG00000101197.8
139	ENSG00000125753.8	179	ENSG00000154864.6	219	ENSG00000143149.8
140	ENSG00000159593.9	180	ENSG00000186226.7	220	ENSG00000179294.5
141	ENSG00000170542.5	181	ENSG00000122481.12	221	ENSG00000087191.7
142	ENSG00000148296.5	182	ENSG00000198171.7	222	ENSG00000111875.6
143	ENSG00000246174.2	183	ENSG00000142684.7	223	ENSG00000162396.5
144	ENSG00000182500.7	184	ENSG00000136436.9	224	ENSG00000111328.2
145	ENSG00000164615.3	185	ENSG00000185236.6	225	ENSG00000112584.8
146	ENSG00000154035.6	186	ENSG00000167182.10	226	ENSG00000179603.12
147	ENSG00000160953.8	187	ENSG00000132274.10	227	ENSG00000166529.9
148	ENSG00000168028.8	188	ENSG00000114054.9	228	ENSG00000236756.3
149	ENSG00000125319.8	189	ENSG00000102445.13	229	ENSG00000132842.8
150	ENSG00000119431.5	190	ENSG00000144199.6	230	ENSG00000143157.7
151	ENSG00000226479.3	191	ENSG00000259243.1	231	ENSG00000001630.10
152	ENSG00000006695.5	192	ENSG00000186470.7	232	ENSG00000179119.9
153	ENSG00000104835.8	193	ENSG00000141452.3	233	ENSG00000171790.11
154	ENSG00000133106.10	194	ENSG00000145934.11	234	ENSG00000146574.10
155	ENSG00000136448.6	195	ENSG00000176472.5	235	ENSG00000065000.9
156	ENSG00000117226.7	196	ENSG00000107745.10	236	ENSG00000075975.11
157	ENSG00000116983.7	197	ENSG00000174586.4	237	ENSG00000172893.10
158	ENSG00000174007.7	198	ENSG00000205181.2	238	ENSG00000134851.7
159	ENSG00000132199.11	199	ENSG00000253140.1	239	ENSG00000130309.4
160	ENSG00000177000.6	200	ENSG00000135842.12	240	ENSG00000162734.8

Table A.2: Gene names and their respective indices, part 2

241	ENSG00000138028.9	274	ENSG00000181396.7	307	ENSG00000188186.6
242	ENSG00000105875.9	275	ENSG00000196329.5	308	ENSG00000182108.5
243	ENSG00000144021.2	276	ENSG00000167748.5	309	ENSG00000105609.11
244	ENSG00000092010.10	277	ENSG00000124145.5	310	ENSG00000142102.11
245	ENSG00000104964.9	278	ENSG00000127220.3	311	ENSG00000159202.11
246	ENSG00000185187.7	279	ENSG00000078142.4	312	ENSG00000260804.1
247	ENSG00000064666.7	280	ENSG00000225339.1	313	ENSG00000118162.8
248	ENSG00000103199.8	281	ENSG00000086504.10	314	ENSG00000136982.5
249	ENSG00000085721.8	282	ENSG00000173113.2	315	ENSG00000184716.9
250	ENSG00000138035.9	283	ENSG00000181004.5	316	ENSG00000197457.5
251	ENSG00000246731.2	284	ENSG00000167720.7	317	ENSG00000148484.12
252	ENSG00000244165.1	285	ENSG00000135972.4	318	ENSG00000130758.2
253	ENSG00000122359.11	286	ENSG00000151552.7	319	ENSG00000176986.10
254	ENSG00000108389.4	287	ENSG00000105656.4	320	ENSG00000165672.5
255	ENSG00000127399.10	288	ENSG00000125450.5	321	ENSG00000180817.6
256	ENSG00000115946.3	289	ENSG00000085644.9	322	ENSG00000108592.9
257	ENSG00000204642.7	290	ENSG00000139044.6	323	ENSG00000168765.10
258	ENSG00000144655.9	291	ENSG00000046604.7	324	ENSG00000111850.5
259	ENSG00000204920.4	292	ENSG00000213246.1	325	ENSG00000165661.10
260	ENSG00000120088.9	293	ENSG00000162688.11	326	ENSG00000176912.2
261	ENSG00000228789.2	294	ENSG00000170425.2	327	ENSG00000103550.9
262	ENSG00000254614.1	295	ENSG00000134070.4	328	ENSG00000081665.7
263	ENSG00000197563.4	296	ENSG00000116514.11	329	ENSG00000196605.2
264	ENSG00000067057.10	297	ENSG00000101220.11	330	ENSG00000248099.2
265	ENSG00000010310.3	298	ENSG00000141562.11	331	ENSG00000152558.10
266	ENSG00000132623.10	299	ENSG00000171848.7	332	ENSG00000125735.5
267	ENSG00000179918.14	300	ENSG00000197863.3	333	ENSG00000065308.4
268	ENSG00000196449.3	301	ENSG00000168116.9	334	ENSG00000140403.7
269	ENSG00000116120.8	302	ENSG00000135211.5	335	ENSG00000198113.2
270	ENSG00000039650.4	303	ENSG00000181458.6	336	ENSG00000112218.5
271	ENSG00000082212.6	304	ENSG00000138801.4	337	ENSG00000003056.3
272	ENSG00000167671.6	305	ENSG00000138297.8	338	ENSG00000214447.3
273	ENSG00000105401.1	306	ENSG00000161091.7	339	ENSG00000119632.3
				340	ENSG00000144231.5

Table A.3: Gene names and their respective indices, part 3

Appendix B

Proof of the Existence of a Vertex With No Incoming Edges in a Directed Acyclic Graph

Given a directed acyclic graph $\mathcal{G}$ consisting of a set of n vertices $\mathcal{V}$ and a set of directed edges $\mathcal{E}$, each with consisting of a parent vertex v_i and a child vertex v_j, $i, j < n$. Assume a scenario in which all vertices have at least one incoming vertex.

Select an arbitrary vertex v_0 from $\mathcal{V}$. As each vertex is assumed to have at least one incoming vertex, select one parent vertex v_1 of v_0. Repeat this process until reaching vertex v_n. At this point at least one vertex must have been visited more than once. This is a contradiction, however, as the graph is acyclic.

Appendix C

All Hyperparameters

Sparse Models

Sparse Expression

epochs: 30, learning rate: 0.1, weight decay: 0, mask percentage: 0.0

Sparse Genotype

epochs: 30, learning rate: 0.1, weight decay: 0, mask percentage: 0.0

Sparse Both

epochs: 30, learning rate: 0.1, weight decay: 0, mask percentage: 0.0

Compact Models

Compact Expression

epochs: 400, learning rate: 0.05, weight decay: 0.0001, mask percentage: 0.1

Compact Genotype

epochs: 400, learning rate: 0.05, weight decay: 0.0001, mask percentage: 0.1

Compact Both

epochs: 400, learning rate: 0.05, weight decay: 0.0001, mask percentage: 0.1

Appendix D

Random Graph Generation

At an earlier point in the project, to benchmark the performance of the models introduced in this thesis, a system for generating random graphs was introduced. Supplying the models with a randomly generated graph at each training iteration, resulting in worsening predictive performance of the model, verifies that the model benefits from the graph structure.

Results obtained using the randomly generated graphs at each iteration have been omitted, in favor of reviewing the converged model parameters to determine on what basis models make their predictions.

The algorithm should return a graph with the same vertex data as the input graph, but with randomised edges. The number of edges in the output graph should be of the same order of magnitude to be a fair reflection of the input graph. The resulting algorithm was heavily influenced by Kruskal's algorithm[38], psuedocode for which can be viewed in listing D.1.

A new Graph object is instantiated using the NetworkX-framework for Python[15], with vertices imported from the DAG generated by the Findr tool. In this state, the new graph consists of a set of single vertex directed acyclic subgraphs with size equal to the number of vertices in the Findr DAG. Until the set of directed acyclic subgraphs has size 1, the following loop is performed.

Two unconnected directed acyclic subgraphs are selected from the remaining set at random. From each selected subgraph, a random vertex is selected. A new, directed, edge is drawn between the pair of selected edges, such that one subgraph becomes a parent of the other subgraph. A new subgraph is then created, which keeps the DAG-property.

When only one subgraph remains, the resulting graph is guaranteed to be a DAG. At this state, however, the randomly generated DAG's set of edges has size equal to the size of the set of vertices minus 1. To make the generated DAG comparable to the ground truth graph generated by the Findr tool, the random graph's vertices are first topologically sorted. We now utilise the property of DAGs that drawing an edge from one vertex in the graph to another will not create a cycle, if the *from*-vertex appears before the *to*-vertex in its topologically sorted order.

While the number of edges in the randomly generated graph is less than the number of edges in the ground truth Findr graph, the following loop is performed. A random vertex is selected from the topologically sorted set of vertices not containing the last vertex in the list. Another random vertex is then selected from the set of vertices appearing after the previously selected vertex in the topologically sorted order. If no edge already exists between the two vertices, an edge is drawn.

The resulting graph is a guaranteed to be a DAG, with a number of edges equal to the given ground truth graph. If a somewhat variable number of graph edges is desired, the algorithm could trivially be modified to that effect, by performing the last loop until the desired number of edges is added, decided as a hyperparameter or drawn from some distribution.

Listing D.1: Psuedocode for random graph generation

```
1
2  def generate_random_graph(ground_truth_graph):
3      """
4          @param ground_truth_graph <Graph>: Graph object returned by
               ↪ the Findr tool.
5
6          @returns random_graph <Graph>: Graph object with vertex values
               ↪ equal to the input graph, but with randomised edges.
7      """
8      vertices = ground_truth_graph.vertices
9      random_graph = new Graph(vertices)
10     while not random_graph.is_weakly_connected():
11         all_components = random_graph.weakly_connected_components()
12         from_subgraph = random.select_and_remove(all_components)
13         from_vertex = random.select(from_subgraph)
14         to_subgraph = random.select_and_remove(all_components)
15         to_vertex = random.select(to_subgraph)
16         random_graph.add_edge(from_vertex, to_vertex)
17
18     toposorted_vertices = random_graph.topologically_sorted_vertices()
19     while len(random_graph.edges) < len(ground_truth_graph.edges):
20         from_vertex_index = random.number(0, len(toposorted_vertices))
21         from_vertex = toposorted_vertices[from_vertex_index]
22         to_vertex_index = random.number(from_vertex_index,
               ↪ len(toposorted_vertices))
23         to_vertex = toposorted_vertices[to_vertex_index]
24         if random_graph.has_edge(from_vertex, to_vertex):
25             continue
26         random_graph.add_edge(from_vertex, to_vertex)
27
28     return random_graph
```

Appendix E

Model Architecture Illustrations

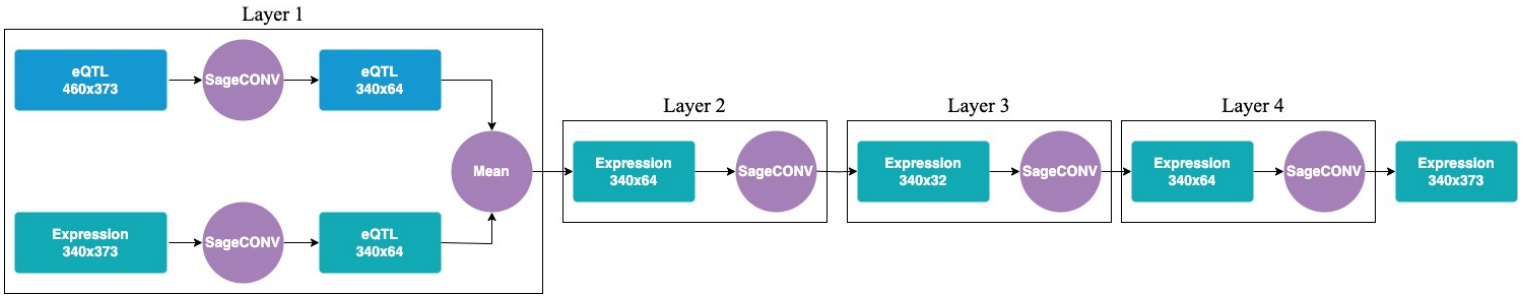


Figure E.1: Compact heterogeneous model architecture

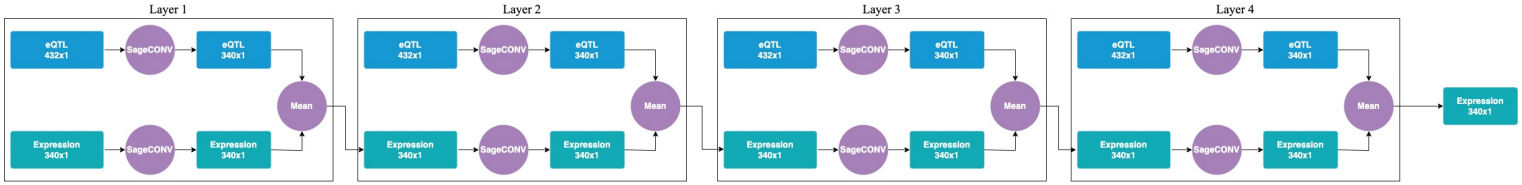


Figure E.2: Sparse heterogeneous model architecture

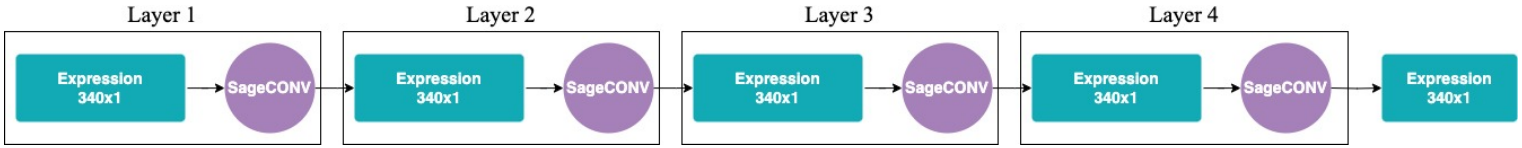


Figure E.3: Sparse homogeneous model architecture