ABSTRACT

Motivation: It is known that the structure of transcription and protein interaction networks is informative of its biological function at multiple scales. However, thus far it has not been possible to systematically connect network topology to gene expression in a quantitative way.

Results: We investigated whether there is a relationship between interaction networks and gene expression values by using a graph convolutional auto-encoder and two end-to-end learning approaches for three interaction networks and hundreds of experimental conditions in the model organism E. coli. Graph neural networks use a message passing framework to learn an embedding of a graph in a continuous space, either using network topology alone, or including additional node features. We found that graph embeddings trained on transcription and PPI networks can explain more than 50 and 40 percent, respectively, of the variance in gene expression data, thus confirming the relationship between network structure and gene expression value. Additionally, for the task of predicting gene expression values using GNNs, with and without additional expression training data, we found that the message passing scheme of GNNs is able to obtain the lowest mean squared error between the tested models both in prediction of unseen test values, and in an auto-encoder scheme for reconstruction of the feature matrix of expression values.

Keywords Gene expression · Graph neural networks · Protein-protein interaction · Graph auto-encoders

1 Introduction

Cells respond to external or internal signals by activating or inactivating transcription factors (TFs), which regulate downstream gene expression programs. Transcriptional regulation is not a linear process: transcription factors are organized in complex transcription networks, and changes in target gene expression are fed back into the network, mainly via protein-protein interactions [1]. Interestingly, the structure or topology of transcription and protein interaction networks is informative of its biological function at multiple scales. Network motifs, small subgraphs of 3 or 4 nodes occurring significantly more often than expected by chance, have well-defined information-processing roles; clusters or communities of densely interacting nodes contain genes or proteins involved in similar biological processes; and at the global scale, degree distributions reflect the relative importance of genes or proteins in a cell [1, 2]. However, none of these results connect network topology to function, that is gene expression, quantitatively: does the local or global connectivity of a gene/protein in a transcription and/or protein interaction network determine or constrain its absolute or relative level of expression under a certain experimental condition?

There have been several results suggesting that the answer to this question is affirmative. In a seminal paper, Beer and Tavazoie [3] were able to assign yeast genes to coexpression clusters with similar expression profiles based on their upstream TF-binding DNA sequences alone, where the clusters were obtained from a training set of genes with both expression data and upstream DNA sequence information. Similarly, in human cells, a high proportion of variation in
gene expression among genes can be explained by the upstream DNA-binding signals of TFs \cite{4,5}. Nevertheless, these studies relate network topology to gene expression indirectly only, by incorporating information about genes sharing the same TFs (that is, parent nodes in the transcriptional network), but not any other information potentially encoded in the network.

Other studies have sought to study the relation between network topology and gene expression more directly, for instance by identifying components in the transcription networks of \textit{E. coli} and yeast where regulators and targets, or targets with shared regulators, have correlated expression profiles across multiple conditions. From a somewhat different perspective, it has been found that genes whose expression level changes in response to deletion or overexpression of a TF are indeed closely connected to the TF in transcription and/or protein-protein interaction networks \cite{6}. However, in these studies, the expression of genes is modelled as a function of the expression of TFs, and the network is only used to select which TFs to consider in the model, that is, it is not the network topology itself that determines the expression level.

The difficulty with modelling gene expression as a function of network topology directly is of course that topology by definition is discrete, and it is not obvious how the information contained in it can be included in a prediction model. In machine learning, the question of how to incorporate the structure of a graph into predictive tasks has received much attention. Traditionally, machine learning often used graph structure statistics (e.g., node degrees or clustering coefficients) as features in predictive tasks. However, these approaches were limited in flexibility and had high computational cost \cite{7,8}. Recently, with the advent of deep learning, the idea of representation learning on graphs has been introduced. In this concept, the main approach is to map nodes, subgraphs, or the entire graph into points in a low-dimensional vector space \cite{7}. In this embedding, the main goal is to preserve the local structure of the graph around each node, without having to specify in advance what “local” means, so that any subsequent machine learning task such as classification, clustering, or recommendation can be done using standard algorithms (e.g., linear regression, random forests) directly on the embedding vectors. Graph Neural Networks (GNNs) address the network embedding problem through a deep auto encoder framework, and have been shown to perform better at subsequent machine learning tasks than traditional embedding methods using non-deep learning based methods such as matrix factorization. Furthermore, GNNs are able to tackle the problem of graph analytic tasks such as graph classification, regression, etc. in an end-to-end manner rather than a separate step on network embedding which improves their ability in this area massively compared to traditional approaches \cite{9}.

In this study, we investigated whether GNNs are able to preserve gene expression values in network topology embedding and then assess their ability in predicting these values in an end-to-end manner for several interaction networks and hundreds of experimental conditions in the model organism \textit{E. coli}. Furthermore, we employed GNNs in an auto-encoder scheme for the reconstruction of the node features that can have multiple applications such as data imputation. The results of the paper are divided into three parts for three approaches of prediction of features based on node embedding, end-to-end learning, and also the features graph auto-encoder scheme. First in prediction of features using network embedding, we investigate whether the structure of an interaction network between genes is informative of the level of expression of gene in experiments. Then, in the end-to-end learning part, we measure how well does GNNs are able to predict these expression values based on graph structure. Finally, we show that GNNs are powerful tools when used in an auto-encoder scheme for reconstruction of the feature matrix.

2 Approach

2.1 Problem Formulation

Assume that an undirected, unweighted graph $G = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ number of nodes (genes) has an adjacency matrix $A$, where $A_{ij} = 1$ if there is an edge between nodes $i$ and $j$ and zero otherwise, and degree matrix $D$, a diagonal matrix with the degrees (number of neighbours) of each node on the diagonal. The two $N \times M$ matrix $X$ and $N \times P$ matrix $Y$, respectively called the feature matrix and the label matrix, denote the training and testing expression values, referred to as features and labels, of the $N$ nodes in $M + P$ different experiments. We have two different approaches when it comes to mapping graph into expression values. The first approach, is getting the network embedding through a deep graph auto-encoder method and the second approach is end-to-end learning of a GNN.

2.1.1 Network embedding

In this method, our aim is to learn a $N \times H$ node embedding matrix $Z$ which preserves both structural and node value features through the graph auto-encoder neural network approach and to calculate predicted features of the testing experiments through

$$Y = f(Z)$$ \hfill (1)
in which $f$ is a regression function learned on the training experiments $X$. Here we will consider linear regression (LR) and random forest regression (RF).

### 2.1.2 End-to-end learning

This approach tries to solve the prediction of the features as an end-to-end task that is done using a GNN framework. For this study, we have chosen the same semi-supervised learning framework of [10], in which the features of the unlabeled nodes (test nodes) are used in learning the labels of the labeled nodes (training nodes). In our study the labels of each node is the same as its features (expression values). In this framework, the features of each node is the aggregation of the neighbouring nodes and the prediction model is conditioned on both the node features and the adjacency matrix of the underlying graph structure. Therefore, the GNN formula for predicting the features is obtained by

$$Y = \text{GNN}(X, A)$$

### 2.2 Neural network on graphs

One of the first attempts at learning neural networks over graph structures was the convolution operation on graphs. For an input signal $x \in \mathbb{R}^N$, the spectral convolution is defined as

$$g_\theta \ast x = U g_\theta U^T x,$$

in which $U$ is the matrix of eigenvectors of the symmetric Laplacian $L = D - A = U \Lambda U^T$. $U^T x$ is called the Fourier transform of signal $x$ and $g_\theta$ is a matrix function of $\Lambda$, that is the diagonal matrix of eigenvalues of $L$.

Due to the high cost of calculating the eigenvalues in the case of large matrices, [11] proposed to use a Chebyshev series expansion truncated after the $K^{th}$ term to approximate the graph convolution operation with a $K^{th}$-order polynomial:

$$g_\theta \ast x \approx \sum_{k=0}^{K} \theta_k' T_k(\Lambda) U^T x = \sum_{k=0}^{K} \theta_k' T_k(\hat{\Lambda}) x,$$

in which $T_k(.)$ and $\theta_k'$ are the $k^{th}$-order Chebyshev polynomials and expansion coefficients, respectively. Furthermore, $\hat{\Lambda} = \frac{2}{\lambda_{\text{max}}} \Lambda - \mathbb{1}_N$ in which $\lambda_{\text{max}}$ is the largest eigenvalue of $\Lambda$. Moreover, $\hat{\Lambda}$ is obtained through $\hat{L} = U \hat{\Lambda} U^T = \frac{2}{\lambda_{\text{max}}} L - \mathbb{1}_N$.

In Graph Convolution Networks (GCN) [10], further approximations were done by setting $K = 1$, $\lambda_{\text{max}} \approx 2$, and $\theta = \theta_0 = -\theta_1$. As a result, (4) was transformed into

$$g_\theta \ast x \approx (\mathbb{1}_N + D^{-\frac{1}{2}} AD^{-\frac{1}{2}}) x.$$  

Repeated application of $g_\theta$ resulting in high powers of $D^{-\frac{1}{2}} AD^{-\frac{1}{2}}$ can cause numerical instabilities. As a result, [10] suggested to set the diagonal elements of $A$ to 1 (add self-loops) and also to recompute $D$ according to the updated adjacency matrix. Therefore, they used the symmetrically normalized adjacency matrix $\hat{A}$ in their convolution layer which is obtained through

$$\hat{A} = D^{-1/2} A D^{-1/2}.$$  

Thus, the forward operation in a single convolution layer for $M$ input signals arranged in a $N \times M$ matrix $X$ is computed as

$$Z = \sigma(\hat{A} \text{ReLU}(\hat{A}XW_0)W_1)$$

with weight matrices $W_1$ containing the trainable weights for each input feature. Additionally, $\sigma$ is a non-linear function such as softmax [12].

Additional studies in GNNs show that a GNN can be viewed as a message-passing approach based on graph structure, where every node’s message for its neighbours is the aggregation of the neighbourhood information. [13]. This aggregation can be done through a trainable neural network which makes the classification or regression problem, an end-to-end learning task allowing a better generalization compared to network embedding. This framework is also known as Message Passing Neural Network (MPNN) and the forward pass in such network has two phases of message passing and readout. In the message passing phase, each node’s message is obtained through a message function and the hidden representation of each node is calculated through an update function which aggregates the gathered messages with the previous node representation according to:

$$h^k_i = \gamma^k(h^{k-1}_i, \sum_{j \in N(i)} M(h^{k-1}_i, h^{k-1}_j, e_{ij})),$$
in which, $h^k_i$ is the hidden state of node $i$ in layer $k$ with $h^0_i$ being the node $i$’s features and $e_{ij}$ is the edge attribute between nodes $i$ and $j$. Additionally, $\gamma$ and $M$ are both differentiable functions called the update and message functions, respectively. Furthermore, $N(i)$ denotes the set of neighbouring nodes of node $i$.

In the readout phase, the feature vector of the graph is computed using some learnable, differentiable readout function $R$ according to

$$Y = R(\{h_i| i \in V\}).$$ (9)

### 2.3 Graph representation learning for node embedding

For the task of learning the representation of graph structure, we have chosen the Non-probabilistic Graph Auto-Encoder model of [14]. The embedding matrix $Z$ is calculated by

$$Z = \text{GCN}(X, A),$$ (10)

where $\text{GCN}(X, A)$ is a two layer Graph Convolutional Network on the input $A$ and $X$ (the features of the graph). $\text{GCN}(X, A)$ is obtained by setting $\sigma$ in (7) to a linear function and thus is calculated as

$$\text{GCN}(X, A) = \hat{A} \text{ReLU}(\hat{A}XW_0)W_1.$$ (11)

The weights are trained by measuring how well the embedding reconstruct the graph adjacency matrix, where the reconstructed adjacency matrix $\hat{A}$ is defined as

$$\hat{A} = \sigma(ZZ^T).$$ (12)

Moreover, the cross-entropy error over all the edges in the matrix is used as a loss function,

$$L = - \sum_{n=1}^{N} A_{n} \ln \hat{A}_{n},$$ (13)

in which, $A_{n}$ is the adjacency row of the $n$th node in $A$ and $\hat{A}$. The training of the neural network is done by gradient descent and stochasticity added by dropout rate.

### 2.4 Message passing neural network scheme for predicting features

In order to have a thorough comparison between different message passing schemes we used three popular message passing networks for finding the hidden representation of the nodes, as well as introducing our own, for the specific task of predicting expression values of genes based on their neighbourhood information. These three methods are inductive GCN, GraphSAGE by [15], and the GNN operator from [16] (from here on out referred to as GraphConv). According to [13], the formula for inductive GCN is

$$h^k_i = \sum_{j \in N(i) \cup i} \frac{1}{\sqrt{\text{deg}(i) \ast \text{deg}(j)}} (W^k_{h^k_j - 1}).$$ (14)

Additionally, the formula for GraphSAGE is

$$h^k_i = W \cdot (h^k_{i - 1} || \text{Mean}_{j \in N(i) \cup i}(h^k_{j - 1})),$$ (15)

in which $(.||.)$ is the concatenation function. Furthermore, the GraphConv operator is calculated through

$$h^k_i = W_1 h^k_{i - 1} + \sum_{j \in N(i)} W_2 h^k_{j - 1}.$$ (16)

Our version of MPNN deals with prediction of features of each node based on its own features and its neighbours. Therefore, in our model we first obtain a representation of the node features and its neighbours by running them through a linear layer and getting the aggregation of messages by a mean aggregator. Then for the update function we concatenate the node features with its aggregated message representation and run them through a shared weight network which determines how important each of these values are in predicting the features of the node. The formula for our Expression GraphConv is as follows

$$h^k_i = W_1 (h^k_{i - 1} || \text{Mean}_{j \in N(i) \cup i}(W_2 h^k_{j - 1})).$$ (17)
Table 1: Hyper-parameters of the graph neural network

| Hyper-parameter       | Node Embedding | MPNN |
|-----------------------|----------------|------|
| Epochs                | 200           | 250  |
| initial learning rate | 0.01          | 0.001|
| first hidden layer size | $2 \times H$ | 64   |
| second hidden layer size | $H$           | 32   |

3 Experimental setup

3.1 Node embedding methods

We used the PytorchGeometry implementation of the graph auto-encoder provided by [17]. For our approach, the normal auto-encoder provided in the package was used, and the variational auto-encoder was omitted. The hyper-parameters of all the experiments are available in Table 1. For each of the experiments, the dimension of the embedding, $K$, is equal to the dimension of the output layer of the auto-encoder and the embedding is obtained using the adjacency matrix of the whole graph including all the nodes. In order to demonstrate the eligibility of the numerical results, we compare three different models of graph node embedding.

3.1.1 Random graph embedding

For generating random graphs, we used the random graph generator of the Python3 package NetworkX, using the Erdős–Rényi model [18]. According to this model, a random graph $G(n, p)$ has $\binom{n}{2}p$ edges placed at random. The degree distribution of each node is calculated through:

$$P(\text{deg}(v) = k) = \binom{n - 1}{k} p^k (1 - p)^{n - 1 - k},$$

which is the binomial distribution. We adjusted the value of $p$ so that the random graphs would approximately have the same number of edges as the real networks. In this approach the $Z$ is calculated by

$$Z = \text{GCN}(\mathbb{1}_N, \mathbf{A}_{\text{rand}}),$$

in which, $\mathbb{1}_N$ and $\mathbf{A}_{\text{rand}}$ represent an identity matrix of size $N \times N$ and adjacency matrix of the random graph, respectively.

3.1.2 Graph embedding without node features

In order to embed a graph using only topological information, we only took into account the adjacency matrix $\mathbf{A}$, and none of the feature values were fed into the auto-encoder. In this approach, the embedding matrix $Z$ is calculated by (10), except that instead of $\mathbf{X}$, an identity matrix $\mathbb{1}_N$ of size $N \times N$ is fed into the model as features:

$$Z = \text{GCN}(\mathbb{1}_N, \mathbf{A}).$$

3.1.3 Graph embedding with node features

In order to learn the embedding of graph nodes, there is an option to feed the features of nodes into the model. In our approach, we used a subset of the available expression data as the feature matrix $\mathbf{X}$ to input to the graph auto-encoder. The embedding $Z$ in this case, is calculated through (10).

3.2 End-to-end learning of features using MPNNs

For the MPNN modules of inductive GCN, GraphSage, and GraphConv we used the already available implementations from the package PytorchGeo. Furthermore, we used MessagePassing class of this package to implement our proposed Expression GraphConv. The hyper-parameters of all the MPNNs are available in Table 1. The architectures of all the models are a two layer message passing layers followed by a fully connected layer as the readout function with ReLu as the activation function of the message passing layers. Furthermore, same as node embedding task, both the cases of with node features and without node features are used in the training of the MPNN. Therefore in the case of node without features, $\mathbb{1}_N$ is used thus turning (2) into

$$Y = \text{GNN}(\mathbb{1}_N, \mathbf{A}).$$
3.3 Node features graph auto-encoder

Typically, graph auto-encoders are used to reconstruct the structure of the graph such as the one introduced in [14]. However, in this study, we propose the use of GNNs in a feature based auto-encoder scheme in which the MPNN models that were used to predict expression values of nodes, are employed in an auto-encoder manner such that the GNN tries to reconstruct \( \hat{X} \). As a result the feature matrix and the label matrix are the same and (2) is turned into

\[
\hat{X} = \text{GNN}(X, A),
\]

in which, \( \hat{X} \) is the reconstructed version of \( X \).

3.4 Train-test splitting of features and nodes

In order to have an unbiased comparison of results between different models in using node embedding and end-to-end learning, we split the nodes in 67%-33% splits for training and testing. As a result, a regression model is trained on the embedding of the 67 percent of nodes for each of the features (experiments) and is used to predict the feature for the test nodes. Furthermore, we randomly divide experiments in 67%-33% training and test feature sets (labels). For each column in the test features, a regression model is trained on the training features of the training node embedding and is used to predict the test features of the test nodes. Additionally, for assessing the feature auto-encoder scheme, the elements of the feature matrix are divided into 75-25 % portions of training and testing. The aim is to try to train to reconstruct the training values and measure the error on the test values. The reason of choosing the split proportions different in different experiments is that in the first task of prediction of test features, the number of features and the number of genes are not high. Therefore our aim is to have larger testsets to measure the ability of the methods employed. In the second task of auto-encoder, the number of values in the feature matrix are high (\( N \times (M + P) \) expression values). Therefore, 25 percent of the values is a large enough testset to assess the ability of the methods.

All the operations of splitting the datasets have been done using Python3 package sklearn with the initial random seed of 42.

3.5 Variance explained on features

For an embedding matrix \( Z \in \mathbb{R}^{N \times H} \) and feature matrix \( X \in \mathbb{R}^{N \times M} \), we calculated the amount of variance in \( X \) explained by \( Z \) as

\[
\mathbb{V}_Z = \frac{\text{tr}(P_Z X^T X)}{\text{tr}(X^T X)},
\]

where \( P_Z \) is the projection matrix onto the subspace of \( \mathbb{R}^N \) spanned by the columns of \( Z \),

\[
P_Z = Z(Z^T Z)^{-1} Z^T.
\]

Note that if we write the eigendecomposition of \( X^T X \) as \( X^T X = V^T \Delta V \), then the columns of \( V \) corresponding to the nonzero eigenvalues in \( \Delta \) are the principal components of \( X \). If \( Z \) consist of the \( i^{th} \) principal component, then eq. [23] reduces to the familiar variance explained by this component, \( \Delta_i / (\sum_j \Delta_j) \). If \( Z \) consists of a single vector \( z \in \mathbb{R}^N \) with unit length, \( \|z\| = 1 \), then eq. [23] reduces to

\[
\mathbb{V}_z = \sum_{i=1}^{N} \frac{\Delta_i}{\sum_j \Delta_j} (z^T u_i)^2,
\]

a weighted sum of the variances explained by each principal component, with weights determined by the extent of overlap between \( z \) and each principal component. Eq. [23] generalizes this to summing the variances explained by multiple vectors simultaneously that need not be mutually orthogonal.

3.6 Datasets

We evaluated the performance of our method on data for the organism Escherichia Coli (E. coli).

- **Network datasets**: For our network datasets we used the transcription network dataset RegulonDB [19] which was extracted from the TF - gene interactions dataset file. All the positive and negative regulatory effects regardless of their degree of evidence of existence (strong or weak) were used to construct the adjacency matrix. A PPI and genetic interaction network were obtained from BioGRID [20][21]. For each of the networks
Table 2: Summary description of benchmark datasets

| Type      | Dataset | Nodes | Edges | Expressions |
|-----------|---------|-------|-------|-------------|
| Network   | TF net  | 1559  | 3184  | -           |
|           | PPI     | 1929  | 11592 | -           |
| Genetic   | 3688    | 147475|       | -           |
| Expression levels | M3DB | -     | -     | 466         |

we extracted the interactions from the file "BIOGRID-ORGANISM-Escherichia coli_K12_W3110-3.5.180" and considered the “physical” and “genetic” values of 'Experimental System Type' column for each of the PPI and genetic networks, respectively.

- **Expression level dataset:** We used the Many Microbes Microarray Database (M3DB) of expression levels of genes in *E. coli* [22, 23]. All the expression levels of the experiments from the file "avg_E_coli_v4_Build_6_exps466probes4297" were used to construct the feature matrix.

For each of the networks, the common genes between the M3DB dataset and the network were extracted from the datasets and an adjacency matrix and a matrix of features were constructed from the network and expression level datasets, respectively.

The detailed description of each of the networks is available in table[2]

3.7 Computational resources

All the experiments were done on google colab Jupyter notebook environment with Python 3. The GPU resources of colab was used to train the GNN and they are automatically chosen by colab itself from the available list of GPUs that often include Nvidia K80s, T4s, P4s and P100s.

4 Results

4.1 Node embedding results

Note that in all of the figures Linear Regression is shown as "LR" and Random Forest is shown as "RF". Additionally, Transcription Factor network is refered to as "TF". Furthermore, "Random Network" represents the graph embedding of a random graph described in 3.1.1. Moreover, 'without features' and 'with features' refers to embedding methods described in 3.1.2 and 3.1.3 respectively.

4.1.1 Network topology explains a significant percentage of gene expression variance

To test the overall feasibility of predicting gene expression from network topology, we measured the total variance of a large set of 466 diverse gene expression profiles in *E. coli* explained by various embeddings of three networks, the transcription, protein-protein and genetic interaction networks (Section 3.6). We compared network embeddings that used network topology alone, network topology with training features, and random networks (Section 3.1), using a range of embedding dimensions: {8, 16, 32, 64, 128, 256, 512}. As illustrated in Fig. 1.(a), network topology, with and without features included, in all of the real networks explained significantly more variance than the random network.

4.1.2 Network topology predicts expression of individual experiments accurately

In order to measure the ability of the embedding matrix in predicting real expression values, we trained linear regression (LR) and random forest regression (RF) models to predict the values of the test experiments using the values of the embedding of the training genes. (For the method of graph embedding with features included, the training genes were used to train the graph neural network.)

We compared real predictions to a random sampling approach, as learning embeddings and training models for an ensemble of random networks for all parameter combinations was too time and resource consuming. In this approach, we sampled training genes’ training features randomly as the prediction of the test genes’ test features. We confirmed that the prediction of linear regression on random graph embeddings for the PPI network and random sampling of the genes gave approximately the same results (Fig. 1.(b)).

We predicted real gene expression values by training a linear regression and a random forest of 20 estimators and maximum depth of 5 on the embedding of the graph using two different methods of with and without features. In Fig.
Figure 1: Results of prediction of test features from node embeddings. (a): Variance explained for different number of embedding dimensions. (b): Comparison between random sampling and prediction on random network. (c): Histogram of Pearson Correlation for transcription regulatory network. (d): Histogram of Pearson Correlation for PPI network. (e): Histogram of Pearson Correlation for genetic interaction network.
1.(c), Fig. 1.(d), and Fig. 1.(e) the normalized histogram of the Pearson correlation values between the predicted and real (unseen during training) expression values for the test genes in the test experiments of the different models is shown, for the transcription, PPI, and genetic interaction networks, respectively. As expected, in all of the networks, the embedding approach that uses network topology and training features (expression data) is able to make the best predictions, and since the random forest is able to find non-linear relations, it performs better than simple linear regression. However, predictions that are based on embeddings based on network topology alone also significantly outperformed the random sampling.

### 4.1.3 Network topology consistently predicts the same experiments well

Fig. 1.(c), Fig. 1.(b), and Fig. 1.(c) show that predictive performance is variable across the set of test experiments. To test if the same experiments or different experiments are most 'predictable' across multiple networks, we compared the Pearson correlation values for the RF model between predicted and real expression data for each experiment across pairs of networks (Fig. 1.(f). As seen in this figure, the relative performance for most experiments is largely consistent across networks, that is, experiments tend to perform well or not irrespective of the network used, with differences between networks only reflecting the overall differences seen in Fig. 1.(c), Fig. 1.(b), and Fig. 1.(c).

### 4.2 End-to-end learning results

In order to measure the ability of MPNNs in predicting test features, a separate model was trained on training features of the training nodes and was used to predict a test feature column in Y (expression profiles). Furthermore, the Mean Squared Error (MSE) was used as a measurement of how well the model was able to predict the test features of the test nodes. Table 2 depicts the averaged MSE of predicting test features of 154 test features profiles based on training features. The multilayered perceptron model is used as baseline approach of predicting test features of test nodes directly from features.

As can be viewed in Table 3, All the MPNN models, except for GCN, are able to predict the test expression values better than the baseline of Multi-Layered Perceptron (MLP). This is possibly due to the fact that the expression values of a node is affected by its neighbours and the MPNN models are able to capture this fact better than a simple MLP. Furthermore, with the models that have the node features included, our proposed MPNN scheme of Expression GraphConv, manages to outperform other models in terms of lowest achieved MSE. However, when the prediction is based solely on the structure of the graph, the RF models that were trained on the graph embedding seem to out perform end to end schemes. This could be due to the fact that the embedding approach manages to capture the structure of the graph better compared to the end to end schemes.

### 4.3 Features graph auto-encoder results

For this part, we measured the MSE for the reconstruction of randomly selected values in the feature matrix which is the same approach that is used by auto-encoders in missing data imputation problems. The results of the reconstruction of the test values of the feature matrix is shown in Table 4.

The results in Table 4 shows that our proposed feature auto-encoder scheme is able to outperform other models in two of the networks that were put to test. Additionally, Genetics network gives better results compared to the other two networks.

### 5 Discussion

In this paper we examined whether graph embeddings inferred using graph neural networks on different types of interaction networks are able to predict gene expression in the model organism $E$. coli. As a proof-of-principle, we first

| Method | Features | Graph | Features | Graph | Features | Graph |
|--------|----------|-------|----------|-------|----------|-------|
| GCN    | 10.08±0.866 | 19.16±9.143 | 8.11±0.435 | 7.77±0.354 | 10.158 | 6.33±0.820 |
| GraphSAGE | 0.49±0.187 | 3.87±0.710 | 0.39±0.164 | 2.23±0.368 | 0.33±0.149 | 2.97±0.595 |
| GraphConv | 0.46±0.196 | 3.89±0.768 | 0.35±0.148 | 2.18±0.365 | 0.31±0.151 | 2.88±0.562 |
| Expression GraphConv (ours) | 0.42±0.176 | 4.71±0.910 | 0.34±0.145 | 3.05±0.482 | 0.30±0.149 | 4.45±0.837 |
| MultiLayer perceptron | 0.51±0.195 | 14.83±2.395 | 0.39±0.148 | 15.80±1.846 | 0.35±0.155 | 14.65±0.701 |
| LR-embedding | 10.38±2.836 | 24639±31622 | 11.80±4.743 | 38057±525287 | 6.37±1.597 | 172461±237488 |
| RF-embedding | 1.59±0.205 | 2.25±0.38 | 1.17±0.186 | 1.57±0.263 | 1.74±0.267 | 1.93±0.321 |

Table 3: The averaged MSE of predicting test features of the test nodes using different models
When we included training expression data as feature in the graph neural network training, the predictive performance with additional features (expression levels in other experiments). As expected, these end-to-end frameworks were able to consider a more realistic missing-data scenario, we used the auto-encoder approaches to predict randomly deleted TF-binding information \([4, 5]\). However, it can be argued that in the present situation where hundreds of gene expression entries from the full expression data matrix. In this case, it is not possible to learn a function directly from present to missing genes, because the missing genes are different in each experiment. Hence the network data is the only information that links present to missing data points across the entire expression matrix. We found that the auto-encoder models achieved high prediction accuracy. While this expression data imputation problem is still somewhat artificial for bulk expression data in a well-studied organism such as \textit{E. coli}, it is a real and important problem in the analysis of single-cell RNA-sequencing data. Our results suggest that graph neural networks would allow to integrate protein interaction networks and existing (bulk) gene expression data in a powerful imputation model for single-cell data.

In summary, this work provides a promising first exploration of the use of graph neural networks to relate and integrate molecular interaction networks and functional genomics data, and sheds light on the close relation between network topology and gene expression levels.
References

[1] Uri Alon. *An introduction to systems biology: design principles of biological circuits*. Chapman, 2007.

[2] X. Zhu, M. Gerstein, and M. Snyder. Getting connected: analysis and principles of biological networks. *Genes & Development*, 21(9):1010–1024, April 2007.

[3] Michael A. Beer and Saeed Tavazoie. Predicting gene expression from sequence. *Cell*, 117(2):185–198, April 2004.

[4] Z. Ouyang, Q. Zhou, and W. H. Wong. ChIP-seq of transcription factors predicts absolute and differential gene expression in embryonic stem cells. *Proceedings of the National Academy of Sciences*, 106(51):21521–21526, December 2009.

[5] C. Cheng, R. Alexander, R. Min, J. Leng, K. Y. Yip, J. Rozowsky, K.-K. Yan, X. Dong, S. Djebali, Y. Ruan, C. A. Davis, P. Carninci, T. Lassman, T. R. Gingeras, R. Guigo, E. Birney, Z. Weng, M. Snyder, and M. Gerstein. Understanding transcriptional regulation by integrative analysis of transcription factor binding data. *Genome Research*, 22(9):1658–1667, September 2012.

[6] Anagha Joshi, Thomas Van Parys, Yves Peer, and Tom Michoel. Characterizing regulatory path motifs in integrated networks using perturbational data. *Genome Biology*, 11(3):R32, 2010.

[7] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[8] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[9] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[10] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[11] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129 – 150, 2011.

[12] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[13] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[14] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[15] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[16] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *CoRR*, abs/1810.02244, 2018.

[17] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[18] P. Erdős and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[19] RegulonDB. Regulondb network interactions - gene interaction, 2019. data retrieved from RegulonDB Downloadable Experimental Datasets, [http://regulondb.ccg.unam.mx/menu/download/datasets/index.jsp](http://regulondb.ccg.unam.mx/menu/download/datasets/index.jsp)

[20] BioGRID. Biogrid ppi interaction network ecoli, 2019. data retrieved from BioGRID download page available at, [https://downloads.thebiogrid.org/BioGRID/Release-Archive/BIOGRID-3.5.180/](https://downloads.thebiogrid.org/BioGRID/Release-Archive/BIOGRID-3.5.180/)

[21] Rose Oughtred, Chris Stark, Bobby-Joe Breitkreuz, Jennifer Rust, Lorrie Boucher, Christie Chang, Nadine Kolas, Lara O’Donnell, Genie Leung, Rochelle McAdam, Frederick Zhang, Sonam Dolma, Andrew Willems, Jasmin Coulombe-Huntington, Andrew Chatr-aryamontri, Kara Dolinski, and Mike Tyers. The BioGRID interaction database: 2019 update. *Nucleic Acids Research*, 47(D1):D529–D541, 11 2018.

[22] Jeremiah J. Faith, Michael E. Driscoll, Vincent A. Fusaro, Elissa J. Cosgrove, Boris Hayete, Frank S. Juhn, Stephen J. Schneider, and Timothy S. Gardner. Many microbe microarrays database. 2007. data retrieved from M3DB download page available at [http://m3d.mssm.edu/norm/](http://m3d.mssm.edu/norm/)
[23] Jeremiah J. Faith, Michael E. Driscoll, Vincent A. Fusaro, Elissa J. Cosgrove, Boris Hayete, Frank S. Juhn, Stephen J. Schneider, and Timothy S. Gardner. Many microbe microarrays database: uniformly normalized affymetrix compendia with structured experimental metadata. *Nucleic Acids Research*, 36(Database-Issue):866–870, 2008.