# Modelling Gene Expression by Integrating GRNs and HMs using GCNs

Shalin Patel[1,2], TBD[1,2,3], and Ritambhara Singh[2,3]

[1]*Division of Applied Mathematics, Brown University*
[2]*Center for Computational Molecular Biology, Brown University*
[3]*Department of Computer Science, Brown University*

**Abstract**

## 1 Introduction

## 2 Related Work

## 3 Method

### 3.1 Formulation for Task

In this paper, we use the same inputs and outputs as Attentive and DeepChrome while also adding a gene expression matrix for each cell line. Using the same formulation as Cheng *et al.* the task is formulated as measuring the gene expression as either up (1) or down (0) regulated. First, per cell line, a GRN is precomputed which utilizes a matrix of size $S \times G$ where $S$ denotes the number of samples in the expression matrix while $G$ represents the number of genes that were recorded.

Hence, for a sample gene, two pieces of information are fed. The first is $\mathcal{G}$ which is a graph describing gene-gene interactions for a particular cell line. In the case of this paper $\mathcal{G}$ was an adjacency list representation of the graph. Second, per gene, a matrix of size $M \times T$ was utilized where $M$ denotes the number of histone marks utilized while $T$ is the total number of bin positions taken into account around the TSS site of a gene.

Overall, for the training data of the GCN, we utilized $\mathcal{G}$ and a $N \times M \times T$ sized matrix where $N$ is the number of gene samples. The output, accordingly, is a $N$-sized vector with either 0 or 1.

### 3.2 Workflow

The workflow utilized in this paper follows three key steps to generate accurate gene expression modelling. First, the gene expression matrices are passed through a random forest based learner to determine importance scores between all the different genes in a particular cell line. Second, the HM data is fed through a 2D convolutional neural network to help capture the combinatorial interactions that can occur across HM lines as well as spatially within an HM line. This network compresses the original $M \times T$

matrix into a 1D vector. Finally, these two pieces of information are fed through a graph convolutional network to present a final two element array which, when fed through a softmax activation, determines the classification for a particular gene.

### 3.2.1 Construction of GRNs

In order to capture the effects that genes have on one another, a Gene Regulatory Network (GRN) was constructed using a gene expression matrix. This network not only captures the complex interaction between genes, but also it is able to determine the weight of these influences. The gene regulatory networks for this paper are built using the standard method of random forests. Utilizing the grnboost2 algorithm from the arboreto package on `pypi.org`, a regression task was defined for each gene in a cell line.

Let $E_i$ denote the row vector containing all samples for gene $i$ in the expression matrix. Then, specifically, for gene $i$, a random forest model $R_i$ was defined where, $L(R_i(E_{1:G\setminus i}), E_i)$ is minimized with $L$ denoting the mean square error. Once this task is completed, denote the set

$$\mathfrak{N}_i = \{imp(R_i, j) \mid j \in \{1 : G\} \setminus i\}.$$

Here, $imp(R_i, j)$ refers to the feature importance of $j$ in random forest model $R_i$. Then the final graph $\mathcal{G}$ is constructed with the neighbor list of a node $i$ being

$$\mathfrak{I}_i = \{j \mid imp(R_i, j) > \bar{\mathfrak{N}}_i + s(\mathfrak{N}_i)\}$$

in which $s(\mathfrak{N}_i)$ denotes the sample standard deviation. Hence, $\mathcal{G} = \{\mathfrak{I}_i\}, \forall i$.

### 3.2.2 HM Encoding

Due to the current requirements imposed on the inputs of GCNs, the input into the GCN layers along with a graph must be a one dimensional vector. As the HM data is provided as an $M \times T$ sized matrix, a CNN architecture is utilized to encode the information stored in the matrix to a one dimensional representation. The utilization of a CNN helps capture the combinatorial interactions that take place across HM lines as well as across multiple bins.

This paper utilized the 2DConv layers from pytorch wherein the input $X$, is a $N \times 1 \times M \times T$ sized matrix with the dimension of size one taking the role of the number of channels in the feature space. Our model utilizes three of these 2DConv layers chained together eventually giving a $N \times S$ output with $E$ representing the number of features in the encoded space.

### 3.2.3 Activation and Dropout.

Throughout these processes, nonlinear activation and dropout is applied. These help regularize the model and help avoid overfitting during the training process.The activation utilized was an elementwise $\tanh$. It is useful to utilize this transformation because $\forall x \in \mathbb{R}, -1 < tanh(x) < 1$ which kept the model coherent through layers. The dropout utilized randomly set values in the feature map to zero with probability 0.5. This assisted in simulating dead signals and increasing the robustness of the model itself.

### 3.2.4 GCN Layer

The input into the GCN layers is, thus, a graph $\mathcal{G}$ and the output from the encoding layer. Call this $H$. Let $H^{(l)}$ denote the $l$th hidden layer while $h_i^{(l)}$ is the feature matrix at node $i$ at the $l$th layer.

In this paper a combination of six layers are used all of which are implemented in the pytorch_geometric package as the `ARMAConv` layers as introduced in the following paper. Specifically, the `ARMAConv` architecture was chosen as it showed increased graph classification performance over many standard layers. The layer performs the following set of transformations from layer to layer.

$$X' = \frac{1}{K}\sum_{k=1}^{K} X_k^{(T)}$$
$$X_k^{(t+1)} = \sigma(\hat{L}X_k^{(t)}W + X^{(0)}V)$$

Here we have that $\hat{L}$ denotes the modified laplacian where $\hat{L} = I - L = D^{-1/2}AD^{1/2}$. $A$ represents the adjacency matrix of the underlying graph, $\mathcal{G}$ and $D$ represents the diagonal degree matrix of $\mathcal{G}$. Furthermore, $W$ and $V$ are trainable parameters that vary from layer to layer. These layers were all chained together eventually creating an $N \times 2$ matrix.

## 3.3 Training

As a whole the entire network can be described as

$$\hat{y_i} = \text{softmax}(f_{gcn}(f_{conv}(X_i), \mathcal{G}))$$

Suppose that all of the learnable parameters in the model, which are initially random, are in the vector $\Theta$. Then using the standard cross entropy loss, we can calculate the loss with respect to $\Theta$

$$L(\Theta) = \sum_{i=0}^{N} \text{cross\_entropy}(y_i, \hat{y_i})$$

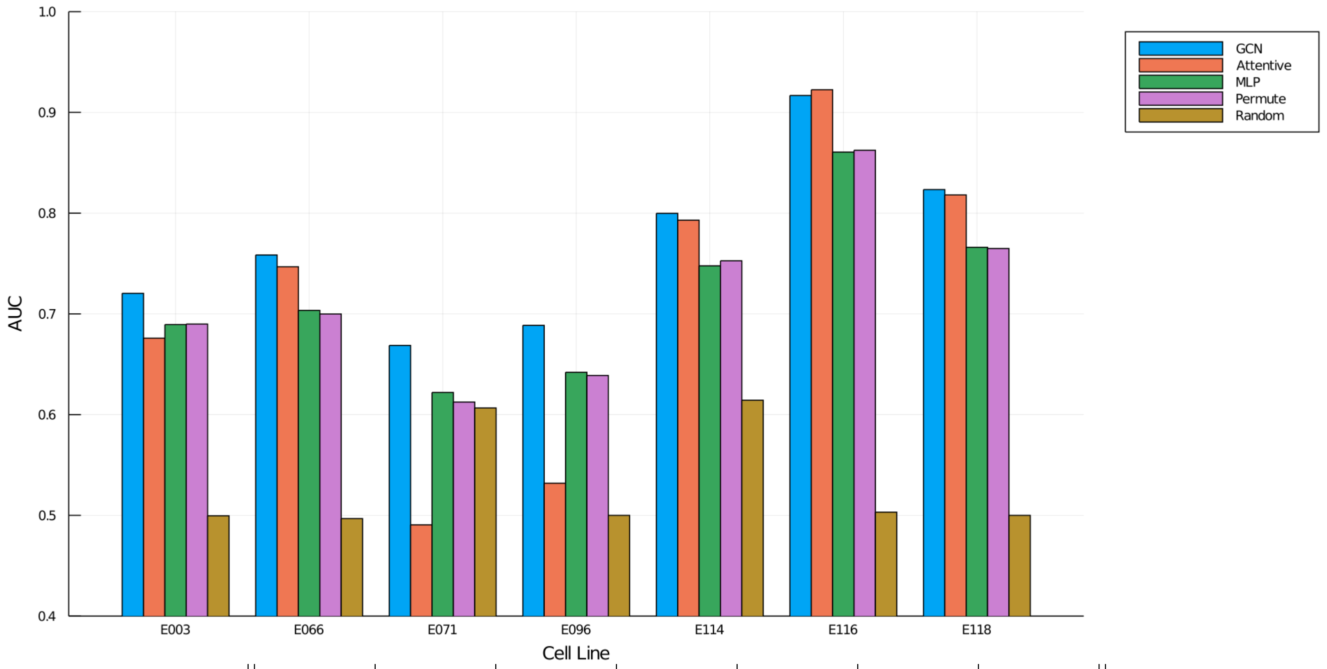
To train these parameters, and thus the entire network the ADAM optimizer is used. This is possible because all steps in the network are differentiable, and thus, the gradient of $L$ can be calculated with respect to $\Theta$. Thus, ADAM can use backpropagation to train the weights of the model.

## 4 Experimental Setup

## 5 Results

## 5.1 Performance Evaluation

The performance of the model was checked against four baselines across seven different cell lines and is summarized in Figure 1.



| | E003 | E066 | E071 | E096 | E114 | E116 | E118 | *Average* |
|---|---|---|---|---|---|---|---|---|
| GCN | **0.7204** | **0.7585** | **0.6687** | **0.6886** | **0.7998** | 0.9168 | **0.8234** | **0.7655** |
| Attentive | 0.6759 | 0.7468 | 0.4906 | 0.5319 | 0.7930 | **0.9225** | 0.8181 | 0.7113 |
| MLP | 0.6894 | 0.7035 | 0.6220 | 0.6420 | 0.7477 | 0.8606 | 0.7661 | 0.7188 |
| Permute | 0.6899 | 0.6999 | 0.6125 | 0.6389 | 0.7527 | 0.8624 | 0.7649 | 0.7094 |
| Random | 0.4996 | 0.4968 | 0.6066 | 0.5000 | 0.6143 | 0.5032 | 0.5000 | 0.5229 |

Figure 1: AUC Performance Across Cell Lines and Baseline Methods

Quite clearly, the GCN model outperforms almost all of the other baseline methods, including the state of the art AttentiveChrome, across the cell lines. The only cell line where the GCN model was beaten was in the E116 cell line which was high performing across models. It is clear, though, that the GCN model shows great consistency and proves performance gains in lower performing cell lines such as E071. This is encapsulated in the average AUC performance metric across cell lines where the GCN model has a significantly higher average than all other models and in Figure 1. Note that AUC scores above are calculated using the weighted method to correct for any potential class imbalances in the datasets.

## 6 Discussion

**Acknowledgments** .

**Funding**

## References