Benchmarking Graph Neural Network-based Imputation Methods on Single-Cell Transcriptomics Data

Han-Bo Li
University of Cambridge
hbl26@cam.ac.uk

Ramon Viñas Torné
University of Cambridge
rv340@cam.ac.uk

Pietro Lio
University of Cambridge
pl219@cam.ac.uk

Abstract

Single-cell RNA sequencing (scRNA-seq) provides vast amounts of gene expression data. In this paper, we benchmark several graph neural network (GNN) approaches for cell-type classification using imputed single-cell gene expression data. We model the data in the Paul15 dataset, describing the development of myeloid progenitors, as a bipartite graph consisting of cell and gene nodes, with edge values signifying gene expression. We train a 3-layer GraphSage GNN to impute data by training it to reconstruct the dataset based on a downstream cell classification task. For this, we use a cell-cell graph representation on a small graph convolutional network (GCN) and an adjacency matrix predetermined by spectral clustering. When combined with the data imputation model, GNN classification performance is 58%, marginally worse than an SVM benchmark of 59.4%, however exhibits better learning and generalisation characteristics along with producing an auxiliary imputation model. Our findings catalyse the development of new tools to analyse complex single-cell datasets.

1 Introduction

GNNs offer an enticing framework for representation learning on scRNA-seq data as they can provide a natural and flexible model structure. Buterez et al. [2021] propose a variational graph autoencoder architecture with graph attention layers for dimensionality reduction and clustering of challenging scRNA-seq datasets. Wang et al. [2021] model cell-cell relationships and perform gene expression imputation with autoencoders and GNNs. They outperform existing tools for gene imputation and cell clustering on four benchmark datasets. Rao et al. [2021] combine graph convolution and autoencoder neural networks to impute drop-out events in scRNA-seq datasets, and outperform state-of-the-art techniques.

The primary goal of this paper is cell classification using gene expression data from scRNA-seq datasets[1]. We approach this task in two ways. In Section 2 we represent the scRNA-seq data using a bipartite graph, where cell samples and gene expressions are nodes, and observed gene expressions are edge attributes between these. This representation is inspired by the GRAPE framework introduced by You et al. [2020], and its application to scRNA-seq data is novel. We examine various extensions to this framework including node representations in the bipartite graph, how its performance scales with the dimensionality of the gene expression, and how adding a reconstruction penalty affects downstream cell classification.

We find that using a small graph convolutional network with adjacency based on principal component analysis (PCA) and k-nearest neighbours analysis (KNN) can achieve similar test accuracy than

[1] https://github.com/hbl4310/gnn_sc_transcript

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

the SVM benchmark with a linear kernel. When we regularise against oversmoothing in the GNN, performance improves to beat the benchmark. On a dataset with more gene expressions, the GNN models outperform the benchmark. Combining these approaches with a GNN-based data imputation framework [You et al., 2020], we achieve similar performance, but better learning characteristics.

2 Gene Expression Imputation Methods

Gene expression measurement is known to be difficult, with variability across experimental batches, intrinsic transcriptional noise, extrinsic noise from cell cycles [Dal Molin and Di Camillo, 2019]. Cell classification can be impeded by missing or incorrect gene expression data. We explore methods of imputing gene expression data with GNNs to facilitate cell classification downstream. While learning a robust data imputation method is valuable due to the difficulties of experimental techniques, another potential benefit is that the imputation training might act to regularise the cell classification model if both models are trained simultaneously ameliorating overfitting issues. This could address generalisation concerns observed in the application of GNN models directly to the gene expression data.

2.1 GRAPE

You et al. [2020] introduce GRAPE, a general framework for feature imputation and label prediction in the presence of missing data. Fundamental to GRAPE is the data representation: samples and features are represented as a fully connected bipartite graph, where edge values denote a feature occurring in a sample. Sample nodes have constant node features, and feature nodes are one-hot encoded. Embeddings for each sample and feature node are learned with a GNN (we refer to this as the data model) whose output node features are fed into an imputation model which predicts edge values, denoting feature existence and values. In the original implementation, when used for node classification, the loss from the node predictions are propagated through to the imputation and data models. The authors apply this method to various UCI datasets. Their data model for the bipartite graph is a 3-layer GraphSAGE model which accommodates edge value convolution. The imputation and node classification models are MLPs.

We apply the GRAPE framework to scRNA-seq data, where we regard cells as samples and gene expressions as features. The edge prediction therefore corresponds to predicting missing or incorrect gene expression measurements for cell samples. This differs from existing GNN-based data imputation methods designed for scRNA-seq data, such as the single-cell GNN (scGNN), introduced by Wang et al. [2021]. Their model uses a graph autoencoder and left-truncated Gaussian mixture model to reconstruct the data.

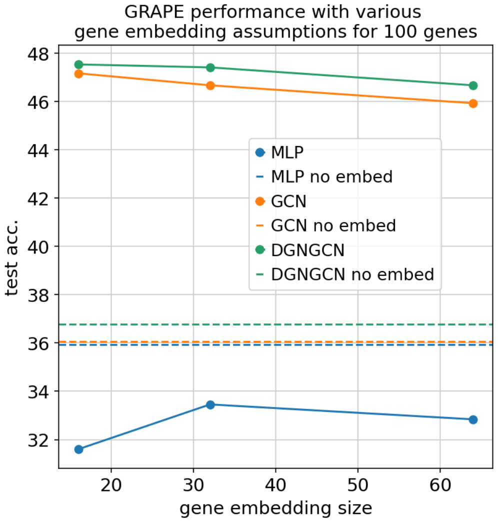

Figure 1: Using the top 100 varying genes we test the performance of the GRAPE framework on cell classification using MLP and GCN models for the prediction task. Learnable embeddings seem to improve GCN performance, but degrade MLP performance.

2.2 Experiments

We perform several experiments using GRAPE on the Paul15 dataset [Paul et al., 2015] to assess whether the augmentations to the GRAPE framework are useful. We use the 3-layer GraphSAGE model with edge updates for the data model, as this outperformed other layer types such as GCN with edge updates in testing. The imputation model is a 2-layer MLP.

Cell Classification Model

We use the 19 cell types labelled in the Paul15 dataset [Paul et al., 2015] as targets for the node classification task. Node labels are one-hot encoded and loss is calculated using cross entropy. Appendix A briefly summarises various GNN architectures. Appendix B outlines several assumptions and approaches to determining the graph adjacency for gene expression data. Appendix C describes the ablation across different GNN architectures and adjacencies to evaluate their performance for cell classification. Directly applying various convolutional GNN models to cell classification on scRNA-seq data, we find that a simple GCN with DGN and graph adjacency calculated using PCA and KNN can outperform a SVM classifier with a linear kernel, by 59.4% vs 58.6%.

We make the following modifications to the GRAPE framework to adapt it to scRNA-seq analysis:

- Since the number of potential genes is large, instead of using one-hot encoded feature nodes, we use a lower-dimension learnable embedding. The features for the cell nodes remain constant, and the same size as the gene nodes'.
- Informed by our experiments in Appendix C, we use a GCN model for the downstream node classification task. We experiment with both a 2-layer GCN without DGN and a 1-layer GCN with DGN. We compute the adjacency using PCA+KNN with $k = 30$ from the imputed data at each update step[2].
- Along with the cross entropy loss for cell classification, we also compute a reconstruction loss between the imputed gene expression values and the original data. This may have the effect of further regularising the downstream task of cell classification and aid in its generalisation. The data, imputation and prediction models are trained simultaneously and updated with both accuracy and reconstruction losses.

Throughout all experiments, the learning rate is 0.001, and the train/test split is 70/30%.

Gene Embeddings and Classification Model

To test whether using learnable embeddings improve downstream task performance, we restrict the dataset to 100 of the most highly variable genes so that one-hot encodings for gene nodes are not too large. We test learnable embeddings for gene nodes of sizes 16, 32 and 64 with different prediction models: MLP, GCN, GCN with DGN. The hidden dimensions in all models are 64. The results are shown in Figure 1. When one-hot encodings are used, test accuracies for all models are similar, at around 36-37%. When learnable embeddings are used, both GCN models perform significantly better while the MLP model performance degrades. Since the GCN without DGN outperforms the MLP for all embedding sizes tested, and performs only marginally worse than GCN with DGN, we will focus on GCN without DGN hereafter due to computational considerations.

Scaling the number of genes

Next, we increase the number of genes and examine the performance of the GRAPE framework with a GCN prediction model. The results are displayed in Figure 2. We can see that performance improves as the number of genes in the dataset increases. Further, as the number of genes increases, greater accuracy is achieved with larger gene embedding sizes, suggesting that the optimal embedding size might be proportional to the number of genes. We can compare the classification performance against the SVM and MLP baselines, shown in Figure 7 in the Appendix. With the full dataset, the GRAPE framework, using a gene embedding size of 64, matches the performance of the SVM baseline and the performance of a GCN (without GRAPE). Memory constraints inhibited testing performance with larger embedding sizes.

Reconstruction

We now turn our attention to measuring the performance of the gene expression imputation of the GRAPE framework, and the effect of incorporating reconstruction error into the cost function along with a classification loss for the downstream task. We use a MSE loss function between the imputed data and the original data, looking only at cell samples which are in the training data. This is a similar approach to Wang et al. [2021], who introduce scGNN, however we do not use auxilliary regularisation such as gene regulatory networks. The reconstruction loss is attenuated with a constant multiplier, chosen such that the initial penalty is roughly 10% of the classification penalty. Figure 3 shows that performance over training epochs using the reconstruction loss is more stable, and that the generalisation gap between train and test performance is much smaller. The final test performance

[2] We use `torch.pca_low_rank` and `torch_cluster.knn` to perform this step to preserve gradient information, however, doing so introduces a minor source of randomness through the low-rank SVD step.

using the reconstruction penalty is lower, however we did not focus on optimising the regularisation strength parameter.

Also following Wang et al. [2021], we look at a simulated dropout reconstruction metric, along with downstream accuracy to assess the performance of the framework. This involves randomly dropping out certain cell gene expression values with a fixed probability, and measuring the divergence between the original dataset and the reconstructed dataset built from the dropped out dataset. Figure 4 shows MSE reconstruction loss for varying data dropout rates. The model with reconstruction loss achieves much lower and consistent losses. As a comparison, we run the VAE scGNN with no gene regulatory network regularisation to impute the data, and on higher dropout rates, it outperforms GRAPE.

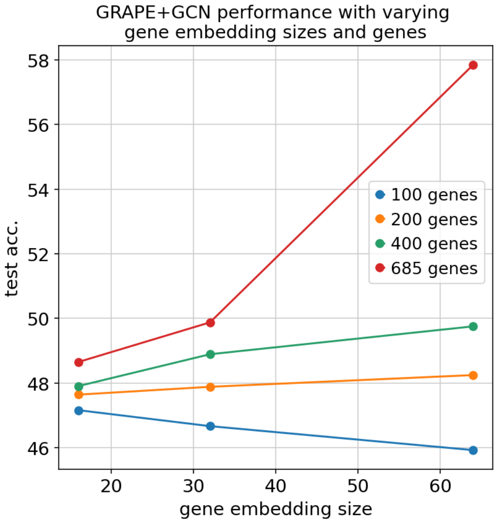

Figure 2: Test accuracies for GRAPE with GCN prediction model, trained on datasets of varying size, and with varying gene embedding sizes. 685 genes is the full pre-processed dataset.

3 Discussion

GNNs offer a promising avenue for single-cell transcriptomics as they provide a flexible and intuitive framework for representing interesting relationships. We apply the GRAPE framework to the task of data imputation and cell classification, and suggest several augmentations to the procedure to accommodate cell classification using scRNA-seq data. The resultant GNN classification model test performance is similar but worse than that of a SVM benchmark model and a GNN classification model applied without an imputation model. The GRAPE framework, with learnable embeddings for gene expressions, has a regularising effect on the GCN cell prediction model and can achieve similar performance to the GCN models applied directly to the data, but with better learning characteristics. With further research, this modular framework has potential to harness the benefits of more sophisticated GNN architectures to various scRNA-seq analysis tasks.

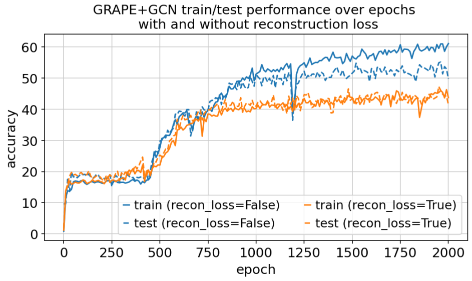

Figure 3: Performance over epochs for GRAPE+GCN model with gene embedding dimension of 64, with and without reconstruction loss penalty. The top 400 genes are used.

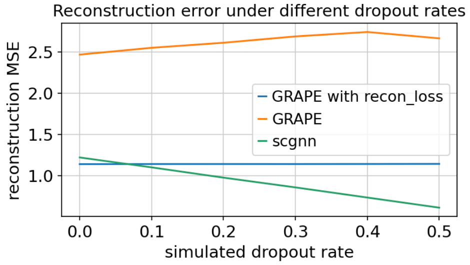

Figure 4: Reconstruction loss on simulated data dropout using GRAPE+GCN model with embedding size of 64 and trained on 400 genes.

References

David Buterez, Ioana Bica, Ifrah Tariq, Helena Andrés-Terré, and Pietro Liò. CellVGAE: an unsupervised scRNA-seq analysis workflow with graph attention networks. *Bioinformatics*, 38 (5):1277–1286, 12 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab804. URL `https://doi.org/10.1093/bioinformatics/btab804`.

Juexin Wang, Anjun Ma, Yuzhou Chang, Jianting Gong, Yuexu Jiang, Ren Qi, Cankun Wang, Hongjun Fu, Qin Ma, and Dong Xu. scgnn is a novel graph neural network framework for single-cell rna-seq analyses. *Nature communications*, 12(1):1–11, 2021.

Jiahua Rao, Xiang Zhou, Yutong Lu, Huiying Zhao, and Yuedong Yang. Imputing single-cell rna-seq data by combining graph convolution and autoencoder neural networks. *Iscience*, 24(5):102393, 2021.

Jiaxuan You, Xiaobai Ma, Yi Ding, Mykel J Kochenderfer, and Jure Leskovec. Handling missing data with graph representation learning. *Advances in Neural Information Processing Systems*, 33: 19075–19087, 2020.

Alessandra Dal Molin and Barbara Di Camillo. How to design a single-cell rna-sequencing experiment: pitfalls, challenges and perspectives. *Briefings in bioinformatics*, 20(4):1384–1394, 2019.

Franziska Paul, Ya'ara Arkin, Amir Giladi, Diego Adhemar Jaitin, Ephraim Kenigsberg, Hadas Keren-Shaul, Deborah Winter, David Lara-Astiaso, Meital Gury, Assaf Weiner, et al. Transcriptional heterogeneity and lineage commitment in myeloid progenitors. *Cell*, 163(7):1663–1677, 2015.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020a.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

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

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. *Advances in Neural Information Processing Systems*, 33:4917–4928, 2020b.

Reuben Moncada, Dalia Barkley, Florian Wagner, Marta Chiodin, Joseph C Devlin, Maayan Baron, Cristina H Hajdu, Diane M Simeone, and Itai Yanai. Integrating microarray-based spatial transcriptomics and single-cell rna-seq reveals tissue architecture in pancreatic ductal adenocarcinomas. *Nature biotechnology*, 38(3):333–342, 2020.

Sophia K Longo, Margaret G Guo, Andrew L Ji, and Paul A Khavari. Integrating single-cell and spatial transcriptomics to elucidate intercellular tissue dynamics. *Nature Reviews Genetics*, 22(10): 627–644, 2021.

A Graph Neural Networks

A graph $G = (\mathcal{V}, \mathcal{E})$ consists of nodes $\mathcal{V}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The graph edges define the adjacency matrix A and degree matrix D. GNNs propagate features along a graph structure, and aggregate and transform neighbourhood information to form node representations, and sometimes edge embeddings, best suited to a given task.

A simple GNN convolutional layer is defined as

$$h_i = \phi \left(x_i, \bigoplus_{j \in \mathcal{N}_j} \psi(x_j) \right) \tag{1}$$

where h_i represents the embedding and x_i are the features of node i, and $\bigoplus$ is some permutation invariant aggregation operation. In this report we will focus on homophilic graph representations, where edges describe some form of interaction or relationship between nodes and so connected nodes are assumed to have similar representations.

Convolutional GNN layers can be attributed to a general taxonomy [Zhou et al., 2020a] with two categories: spectral and spatial. Spectral approaches define the convolution operation in the graph spectral domain. The graph signal is transformed with a graph Fourier transform $\mathcal{F}(x) = U^\top x$, a convolution is applied, and the resultant signal is transformed back into the graph domain with an inverse Fourier transform, $\mathcal{F}^{-1}(x) = Ux$. U is the matrix of eigenvectors of the normalised graph Laplacian, $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. The convolution operation can be expressed with $\psi(X) = U g_w U^\top X$, where g_w is a learnable diagonal matrix.

ChebNet Defferrard et al. [2016] propose approximating g_w with a Chebyshev polynomial truncated to order k. The k-th order Chebyshev polynomial is defined recursively as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with initial conditions $T_0(x) = 1$ and $T_1(x) = x$. They write the convolution operation with

$$H = \sigma \left(\sum_{j=0}^{k} T_j(\tilde{L}) X W_j \right) \tag{2}$$

where $\tilde{L} = \frac{2}{\lambda_{max}} L - I, \;\lambda_{max}$ is the largest eigenvalue of L, and W_i is a learnable weight matrix. This operation results in information passing between nodes which are connected by k hops in the graph.

GCN Kipf and Welling [2016] propose a simplification of the operation in ChebNet to

$$H = \sigma((I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) X W) \tag{3}$$

by truncating at $k = 1$, assuming $\lambda_{max} = 2$, and reducing the number of weight parameters. This is further reduced to $H = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X W)$ to address exploding gradients with $\tilde{A} = I + A$ and $\tilde{D}$ the corresponding degree matrix.

Spatial approaches define the convolution operation in the graph domain directly. GCNs can also be regarded as spatial convolutions.

GraphSAGE Hamilton et al. [2017] introduce a framework which uniformly samples within node neighbourhoods and applies some aggregation. With mean aggregation, the convolutional update is

$$h_i = \sigma \left(W_1 x_i + \frac{W_2}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} x_j \right) \tag{4}$$

With this aggregation, GraphSAGE can be viewed as an inductive version of the GCN layer, though other aggregations can be used.

GAT Veličković et al. [2017] introduce an attentional spatial convoultional. They incorporate attention into the propagation step, computing nodes' hidden states by attending to their neighbours. With linear attention, node embeddings are updated according to

$$H = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{i,j} W x_j\right) \tag{5}$$

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}(a^\top W x_i \| W x_j)\right)}{\sum_{k\in\mathcal{N}_i\cup\{i\}} \exp(\text{LeakyReLU}(a^\top W x_i \| W x_k))} \tag{6}$$

where a is a weight vector of a MLP.

GIN Xu et al. [2018] introduce the Graph Isomorphism Network, which is as powerful as the the Weisfeiler-Lehman graph isomorphism test.

$$H = \sigma\left(f(A + (1+\epsilon)I)X\right) \tag{7}$$

where f is some neural network.

B Adjacency Assumptions

There are several options for the graph adjacency representation when either none is provided, or when the provided adjacency is deficient in some way. Naïvely, we can assume either full or no adjacency. With no edges, any GNN variant reduces to Deep Sets [Zaheer et al., 2017] since embeddings are updated using only features of the receiver node. With all edges, convolutional GNNs are equivalent to Deep Sets as the neighbourhood for each convolution is the entire graph. Alternatively, we can precompute an adjacency using a standard algorithm such as KNN. This approach is relatively inexpensive computationally, however relies on the gene expression data, and requires an extra hyperparameter, k, the number of neighbours to consider. The usefulness of this adjacency may be sensitive to the k, and choosing an optimal value systematically may be expensive. In a similar vein, we could infer the adjacency from a distance metric between cell gene expressions, assigning an edge for nodes with a distance less than a certain threshold.

The graph adjacency can also be calculated dynamically, rather than input statically. Wang et al. [2019] introduce Dynamic Graph CNN (DGCNN) which computes adjacencies at each layer of the network based on the node embeddings at that point. They use KNN to construct the adjacency based on the embeddings and propagate information with this. The convolution operator is

$$h_i = \sigma\left(\sum_{j\in\mathcal{N}'(i)} f(x_i \| x_j - x_i)\right) \tag{8}$$

where f is a neural network taking as input the concatenation of the receiver node features and its difference with features of nodes in its KNN adjacency. Using a dynamic adjacency may allow GNNs more flexibility in learning.

C GNN Classification Experiments

In order to assess various GNN designs on classifying cells with gene expression data, we apply the models to a standard dataset. We use the 19 cell types labelled in the Paul15 dataset [Paul et al., 2015] as targets for the node classification task. Node labels are one-hot encoded and loss is calculated using cross entropy. We apply a variety of convolutional GNN layers, described in Appendix A, to cell classification using the cells' measured gene expressions as node features and test models' performances under various graph adjacency assumptions outlined in Section B. For the KNN adjacency, we apply the clustering algorithm on the raw data, as well as the top 40 principal components. For the node similarity adjacency, we use the Euclidean distance metric and define the threshold for edge existence by the quantile, q, over all distances. The various hyperparameters for the adjacency methods are chosen such that the three methods produce similar levels of graph sparsity (equivalently, similar numbers of edges). $k = 10$ for KNN after PCA produces an adjacency sparsity of 0.6%, which is close to $k = 20$ for KNN on the gene expression data and $q = 0.5\%$ for the Euclidean similarity threshold. $k = 40$ for KNN after PCA produces an adjacency sparsity of 1.7%, which is close to $k = 50$ for KNN on the gene expression data and $q = 2\%$ for the Euclidean similarity threshold. Note that we do not include the full adjacency due to computational constraints.

As baselines, we use a support vector machine (SVM) with a linear kernel, and a multi-layer perceptron (MLP) with a hidden dimensionality of 64 and 3 layers. For all analyses we take a random 70-30% train-test transductive split, and run GNN models 5 times over 2000 epochs per run for each setting with a constant learning rate of 0.001.

C.1 Results

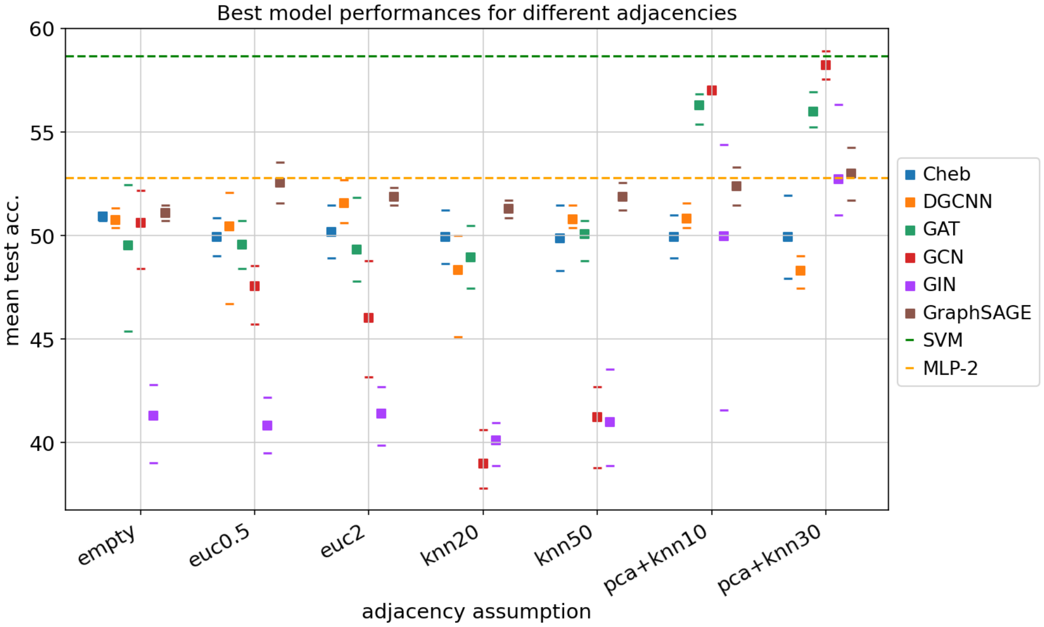

Figure 5: Convolutional GNN model performances based on different adjacency assumptions. Boxes are mean final test accuracies over 5 training runs while the single dashes of the same colour represent the highest and lowest final test accuracies over those runs. The dashed lines signify the performance of the benchmark techniques.

The performances of standard convolutional GNN layers are presented in Figure 5. We look at the average model performances over all runs for each setting, and select the best hyperparameter settings from a narrow hyperparameter search for each model. None of the GNN models outperform SVM (58.65% test accuracy) on average, and most do not outperform a 2-layer MLP (52.8% test accuracy). Only the GCN model achieves similar performance to SVM in settings with an adjacency computed from PCA+KNN.

Adjacency Assumptions

We tested models with a variety of hyperparameters, including with 1 and 2 layers (of the same type). The models' hidden dimensions, where relevant, are set constant to 64. The DGCNN models with 1 layer do not use dynamically calculated adjacency. The GCN model is also tested with a 4 layer setting. Interestingly, most model performances are relatively consistent across all adjacency choices, except GCN and GAT, which perform significantly better on PCA+KNN adjacency methods. GIN models underperform their peers in most adjacency settings suggesting it may be better suited to graph level tasks rather than node level tasks. A 2-layer GraphSAGE model performs the best out of GNN models for empty, Euclidean and KNN adjacencies, while a 2-layer GCN and a 1-layer GAT model with 32 attention heads outperform with PCA+KNN adjacencies. Compared to the GCN model, the GAT model has roughly 10 times the number of parameters, and took roughly twice as long to train and test. We also observe that while the 2-layer GCN outperforms the 4-layer variant on PCA+KNN, the 4-layer variant performs to best on other adjacency settings. This might be an indication of over-smoothing, where adding more layers to the model leads to node representations becoming more similar and thus hindering node classification.

We ran further experiments for models with multiple layers, where the first layer was a DGCNN, and its calculated adjacency was used for subsequent layers. These models did not see any marginal improvement.

Oversmoothing

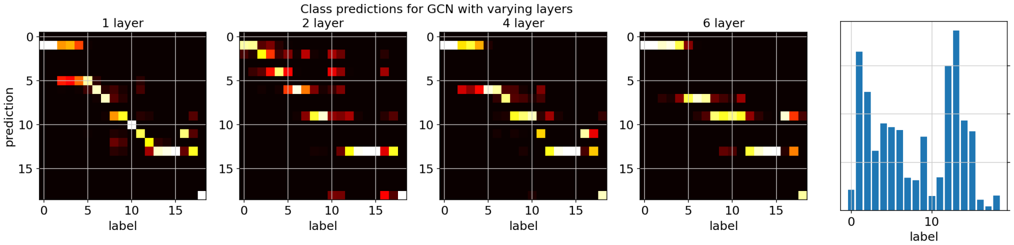

Figure 6: Model predictions are displayed as a heat map, with white-hot squares representing high prediction of a particular label for a particular ground truth. As the number of layers increases, the predictions collapse to the most common labels. A histogram of labels in the data is shown on the right for reference.

We briefly look for evidence of mode collapse by inspecting the distribution of label predictions for each cell label, using the GCN model with varying layers. From Figure 6 we can see that as the number of GCN layers increase, predictions collapse to the most over-represented class labels in the training data, with the 6-layer model only predicting 5 unique labels out of the 19 in the data.

This mode collapse is indicative of oversmoothing, where node features become more similar as information is propagated and averaged. Several remedies exist to address oversmoothing of node features in GNNs. Skip connections can partly alleviate the problem as earlier node representations may be more distinct than those after several convolution operations. Zhou et al. [2020b] introduce Differentiable Group Normalisation (DGN) which normalises nodes within the same group, and separates node distributions from different groups. The number of groups is user-defined. This simultaneously ensures representations of similarly labelled nodes remain smooth, while representations for differently labelled nodes are distinct.

The authors introduce the group distance ratio (GDR) to measure oversmoothing, which is a ratio of inter-group distance over intra-group distance using the Euclidean metric. Lower GDR may be indicative of greater oversmoothing. Using the group distance ratio described in Zhou et al. [2020b], the 1, 2, 4, 6-layer GCN models produce metrics of 0.0630, 0.0640, 0.0624 and 0.0640 respectively. These metrics do not appear to correlate to the mode collapse shown in Figure 6 which might be due to the number of unique labels predicted decreases (10, 7, 7, 5) as the number of layers increase.

We retrain the GCN models with an appended DGN layer with groups set to 10, on the PCA+KNN adjacencies with $k = 30$. With DGN, the 1-layer GCN marginally outperforms the SVM benchmark. The 1-layer model achieves an average final test accuracy of 59.44%, which marginally outperforms the SVM benchmark of 58.65%. The 2-layer model achieves an average final test accuracy of 58.48%, which is only slightly better than the 2-layer GCN model without DGN normalisation.

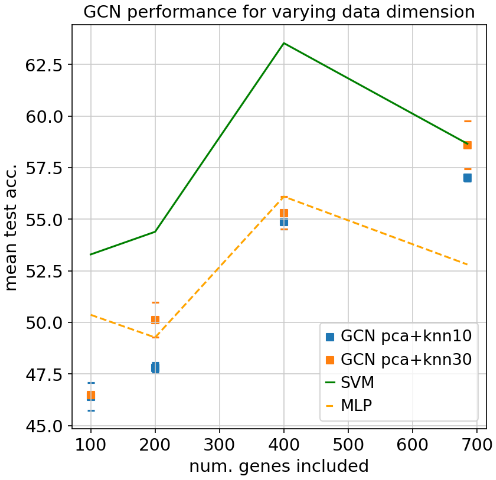

Figure 7: Performance of GCN model vs benchmarks for datasets with varying numbers of genes. Genes are selected during the preprocessing stage according to their expression variance. GCN performance scales better than benchmarks despite underperforming until the full dataset is used.

Scaling the number of genes

To understand how the GNNs' performances scale with the number of different gene expressions, we select the top n genes by their expression variance in the preprocessing stage. We train a GCN model with 1, 2 and 4 layers with PCA+KNN adjacency for $k = 10$ and $k = 30$ on the reduced datasets. The results are displayed in Figure 7 where the result from using all 685 genes is displayed for comparison. The performance of the GCN model appears to scale better than the benchmarks', as the gap in performances decreases as the dimensionality of the gene expression increases.

The preprocessed PBMC3K data provided by the ScanPy API has 2638 cell samples with 1838 genes and 8 provided cell labels. Fitting 1, 2 and 4 layer GCN models to this unseen dataset, we find that all settings outperform the SVM and MLP benchmarks. The GCN models achieve test accuracies of 95.2%, 95.6% and 95.6% respectively, while SVM achieves 94.2% and the best MLP setting achieves 90.3%. While the GNN outperformance on this dataset may be due to the higher gene expression dimensionality, it might be confounded by the reduced number of target classes. Performance with DGN is similar to without, suggesting that oversmoothing is less of an issue, possibly due to fewer target classes.

C.2 Discussion

Directly applying various convolutional GNN models to cell classification on scRNA-seq data, we find that a simple GCN with DGN can outperform a SVM benchmark. Performance of other models was mixed, relative to benchmarks. It is unclear why GCN and GAT models perform better with adjacencies determined by PCA+KNN. This is more marked in the GCN case, where performance under KNN adjacency is dramatically different to PCA+KNN adjacency. Since the PCA+KNN is a form of spectral clustering, one might think that it might favour spectral convolutions, however ChebNet models did not see a significant performance gain, and GAT models are spatial. These results suggest that finding an informative adjacency for GNN models in cell classification using scRNA-seq data is challenging. Further work could be done on incorporating PCA+KNN adjacency, instead of KNN, to dynamic adjacency techniques such as DGCNN.

Alternatively, future work could explore combining cell spatial transcriptomics data with scRNA-seq data in a GNN model to improve cell classification performance. Several recent attempts have been made integrating the data [Moncada et al., 2020, Longo et al., 2021], however the application of GNNs to this augmented data has not been studied. The spatial data would provide a natural adjacency structure which could be augmented with gene expression node features.