Control-Inspired Graph Neural Networks

Charlie Tan[*]
University of Cambridge
ct632@cam.ac.uk

Theo Long[*]
University of Cambridge
tl559@cam.ac.uk

Abstract

Graph convolution networks can be regarded as discrete dynamical systems. Control theory provides a framework for influencing dynamical systems towards desired states. We propose novel applications of control theory to graph representation learning, with the aim of alleviating common issues such as oversmoothing, oversquashing, and overfitting. We outline a general framework for control theoretic graph neural networks, and present synthetic tasks on which this framework overcomes limitations of graph convolutions. We lastly provide empirical results for our framework on a diverse set of graph datasets and tasks, observing small improvements in performance in some cases.

Statement of contribution

We, Charlie Tan and Theo Long of the University of Cambridge, jointly declare that our work towards this project has been executed as follows:

- **Charlie Tan** contributed by writing code, setting up and running benchmark experiments, devising and implementing new methods for control, and writing the report.
- **Theo Long** contributed by writing code, devising and implementing new methods for control, designing and implementing synthetic datasets, and writing the report.

We both independently submit identical copies of this paper, certifying this statement to be correct.

GitHub repository with commit log

The companion source code for our project may be found at: https://github.com/theo-long/control_gnns.

1 Introduction

Convolutional graph neural networks (GCNs) are one of the most popular Graph Neural Network (GNN) architectures, due to their simplicity, computational efficiency, and robust performance on many graph representation learning tasks. However, graph convolution networks are known to have limited efficacy when operating on heterophilic graphs [1], graphs with bottlenecks [2], and tasks requiring long-range interactions [3]. Move expressive forms of message passing, coupled with graph rewiring can overcome such limitations, at the cost of computational efficiency.

In this work, we follow Di Giovanni et al. [4] and consider convolutional GNNs as dynamical systems. In this context, GNN layers describe interaction terms between different nodes, with the choice of layer and the parameters describing the particular relation between the features of neighboring nodes. The layers of the GNN then represent in discrete steps in time, where the features undergo an evolution according to the dynamics of the interaction terms. We propose to apply control forces into this system, via the introduction of control terms to the GNN update equations. Whilst control

[*]Equal contribution.

Charlie Tan & Theo Long, Control-Inspired Graph Neural Networks. *Representation Learning on Graphs and Networks (L45), University of Cambridge*.

typically seeks to maintain a system in a desired state, we seek to explore if control forces can overcome known limitations of graph convolutions such as over-smoothing [5], and over-squashing [2].

Control is a powerful concept for studying and influencing the evolution of graph structured data, for example in the context of temporal networks [6]. It is the dominant framework in engineering for influencing the evolution of dynamical systems, and as such we believe it is a valuable perspective to introduce in the context of GNNs.

The contributions of our work are as follows:

1. We cast graph neural network within a control theoretic framework, extending recent results connecting graph neural networks to dynamical systems.
2. We outline various possible forms of control that could be introduced to augment standard GNN architectures, and discuss how they might overcome common issues such as oversmoothing, oversquashing, and overfitting.
3. We propose a general non-linear control framework for use with GNNs, demonstrating its success on certain synthetic tasks, and studying its performance on a set of diverse graph representation learning benchmarks.

2 Technical Background

2.1 Graph Representation Learning

The goal of graph representation learning is to learn a suitable representation for solving a task such a node or graph classification. In order to process graph data efficiently, we represent the edges using an an adjacency matrix $A \in \mathbb{R}^{n\times n}$ and store any node information in a node feature matrix $H \in \mathbb{R}^{n\times d_{\text{in}}}$, where n is the number of nodes and d_{in} is the node feature dimension. Whilst a GNN can operate directly on this graph, we often employ an encoder to map from d_{in} to a suitable hidden dimension d, and require a decoder to map to the output dimension d_{out}.

We consider a graph convolution (GCN) layer [7]. The GCN update is defined by Equation 1, where $\psi : \mathbb{R}^d \mapsto \mathbb{R}^d$ and $\phi : \mathbb{R}^{2d} \mapsto \mathbb{R}^d$ are linear projections of the form $A\mathbf{x} + \mathbf{b}$, and $c_{ij} = ((\deg(i)+1)(\deg(j)+1))^{-\frac{1}{2}}$. Here $\mathcal{N}_i$ refers to the neighbourhood of node $v_i \in \mathcal{V}$ (the nodes directly connected to v_i by an edge). For convenience we denote GCN in matrix form as $H = \text{GCN}(X, A)$.

$$\mathbf{h_i} = \phi\left(\sum_{j\in\mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right) \tag{1}$$

We further consider a simple message passing neural network (MPNN) layer [8], with update defined by Equation 2. Here $\psi : \mathbb{R}^{2d} \mapsto \mathbb{R}^d$ and $\phi : \mathbb{R}^{2d} \mapsto \mathbb{R}^d$ are multi-layer perceptrons operating on the concatenation of their inputs. Again we use a matrix shorthand $H = \text{MPNN}(X, A)$.

$$\mathbf{h_i} = \phi\left(\mathbf{x}_i, \sum_{j\in\mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right) \tag{2}$$

2.2 Graph Neural Networks as Dynamical Systems

Di Giovanni et al. [4] study graph neural networks in the context of *dynamical systems*, where the GNN layers can be thought of as a discretization of a 'flow' between nodes in the graph. In this formalism, the GNN learns an appropriate diffusion equation, which can be thought of a force terms representing the interaction of nodes in the graph. Neural networks had previously been studied as Euler discretisations of differential equations [9], establishing a connection between neural networks and dynamical systems.

In this framework, GCNs can be thought of as a particular discrete form of heat diffusion. In particular, they induce a dynamics whereby neighboring nodes have similar embeddings, behaving like a low-pass filter. In [4] they showed that if a residual term is included in the GCN, this diffusion

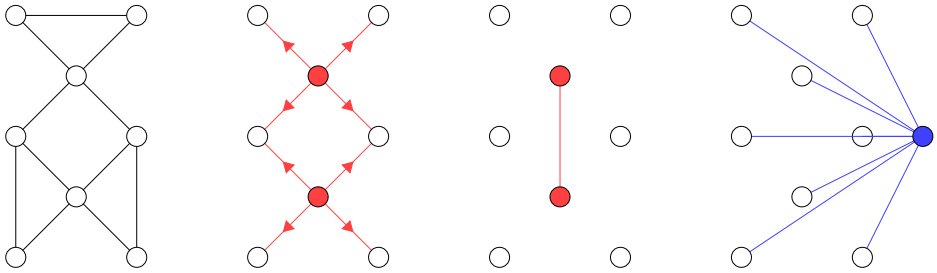

Figure 1: Illustration of various forms of control adjacency matrix. Example of control graph construction using degree as node statistic and threshold $k = 2$. The nodes 'activated' in the control term are colored in red. **Left:** the original graph $\mathcal{G}$, note the central nodes have the highest degree (4) hence will be the $k = 2$ active nodes. **Centre left:** subgraph adjacency control; where an (undirected) edge in $\mathcal{G}$ connected to an active node, we include an outgoing directed edge. **Centre right:** dense subset control; the active nodes form a fully connected graph, with no connections to other nodes. **Right:** virtual node; a new node *not* in the original graph is added and connected to all other nodes.

behaviour can be altered to behave like a high-pass filter, greatly increasing the contrast between neighboring nodes.

2.3 Oversquashing, Oversmoothing, and Overfitting

Many common GNN architectures are known to suffer from issues which limit their performance, in particular oversquashing, oversmoothing, and overfitting. In this section, we outline each of these issues. For a discussion of why our control approach might alleviate these issues see Section 3.1. For specific experiments and datasets seeking to highlight the impact of including control on these issues, please see Section 4.

Overfitting is a common issue in many applications of machine learning, however it is especially prevalent when using more powerful GNN architectures such as those based on message-passing [8]. Though these models are extremely expressive, they can often overfit smaller or simpler graph datasets, and thus GCNs are often preferred due to their lower likelihood of overfitting [10]. However, this may be suboptimal in cases where greater expressiveness or model complexity could provide a performance benefit.

Oversmoothing is an issue commonly seen in deep GCN architectures with more than a few layers [5]. Due to the update equation of the GCN resembling a heat kernel, as outlined in [4], GCN architectures effectively "smooth out" features of neighboring nodes, giving them similar embeddings. If the GCN has many layers, this smoothing makes all node features almost identical, with only the magnitude differing by the degree of the node. This overly smoothed embedding makes node classification very difficult and limits the depth of architectures which can be used. In addition, it makes classic GCN architectures perform poorly on heterophilic datasets [1], where neighboring nodes are likely to have highly *dissimilar* features, and hence should have very different embeddings.

Oversquashing was first identified by Alon and Yahav [2] in cases where many pairs of distant nodes in a graph must all communicate through a very small 'bottleneck' in the graph. In particular, there may be a handful of nodes through which an exponential amount of information must be propagated, requiring an exponentially large hidden dimension that is not feasible in practice.

3 Control-Inspired Graph Neural Network

We propose the control-inspired GNN framework stated in Equation 3. Here, $H^{(t)}$ are the node features at layer t and A is the adjacency matrix of graph $\mathcal{G}$. The control layer $\mathrm{CON}^{(t)}$ is a GNN layer operating over B, a **control adjacency** matrix. Note that by defining $\mathrm{CON}^{(t)}$ to return the zero matrix for all t we recover our base GCN model.

$$H^{(t+1)} = \underbrace{\mathrm{GCN}^{(t)}(H^{(t)}, A)}_{\text{base layer}} + \underbrace{\mathrm{CON}^{(t)}(H^{(t)}, B)}_{\text{control layer}} \tag{3}$$

A control adjacency matrix represents a graph of modified connectivity, on which **active nodes** send unidirectional messages to other nodes; see Figure 1 for illustrations. We define active nodes as per:

Definition 1 *Active Nodes: Let $\mathcal{V}$ be the set of vertices in graph $\mathcal{G}$, and* $\mathrm{ranking}(\mathcal{V})$ *be a function that ranks the vertices by a given node statistic, such that* $\mathrm{ranking}(\mathcal{V})_i$ *is the ranking of the ith node. For a given threshold k (itself potentially depending on the cardinality of $\mathcal{V}$) we define the active nodes as:*

$$\mathcal{V}_a = \{v_i \in \mathcal{V} | \, \mathrm{ranking}(\mathcal{V})_i \leq k\}$$

With active nodes defined, we may now construct the control adjacency matrix, of which we propose two variants:

Definition 2 *Subgraph Control Adjacency: Let $\mathcal{G}$ be a graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$. Given $\mathcal{V}_a \subseteq \mathcal{V}$, a set of active nodes by a given node statistic and threshold k, we define subgraph control adjacency as:*

$$b_{ij} = \begin{cases} 1 & (i,j) \in \mathcal{E} \ \wedge \ v_j \in \mathcal{V}_a \\ 0 & \textit{otherwise} \end{cases}$$

Definition 3 *Dense Subset Control Adjacency: Let $\mathcal{G}$ be a graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$. Given $\mathcal{V}_a \subseteq \mathcal{V}$, a set of active nodes by a given node statistic and threshold k, we define subgraph control adjacency as:*

$$b_{ij} = \begin{cases} 1 & v_i, v_j \in \mathcal{V}_a \\ 0 & \textit{otherwise} \end{cases}$$

Regarding the edges $\mathcal{E}$ of the original undirected graph $\mathcal{G}$ as pairs of directed edges, subgraph control adjacency corresponds to the directed subgraph resulting from removing all edges not originating at an active node. We remark that this construction does not add new edges, and in most cases will result in a disconnected subgraph on which the control layer operates. In contrast, dense subset control adjacency corresponds to a fully-connected graph between the active nodes, with no connections to non-active nodes.

3.1 Motivating Control

The goal of our control-inspired approach is to augment standard GCN architectures to alleviate issues such as oversquashing, oversmoothing, overfitting, and long-range interactions.

Since our control-based approach only performs full message-passing amongst a small subset of the nodes in the graph, it is potentially less likely to overfit as it only provides a small 'correction' to the GCN backbone, rather than allowing for fully-fledged message passing. In fact, this reduced message passing capability also makes the network more computationally efficient during training and inference, allowing for the use of message passing even on very large graphs.

Long-range interactions are necessary in certain graph representation learning tasks, such as molecular tasks where the interaction of atoms at opposite ends of the molecule may have a large impact on the molecule classification. In this case, extremely deep architectures are needed to propagate messages across the graph, which may lead to oversquashing in the presence of bottlenecks. One approach to allowing long-range interactions and reducing oversquashing is graph rewiring, as used in [11] and [12], where additional edges are inserted into the graph to alleviate bottlenecks. Alternatively, one can use virtual nodes to improve long-range communication [13], however this can create new bottlenecks at the virtual nodes, and moreover does not necessarily leverage the inherent structure of the graph. Our approach is perhaps similar to graph rewiring, however it allows for a more flexible approach where local neighborhoods can still be effectively learned by the GCN backbone, while allowing for targeted message passing to occur amongst particularly important or influential nodes.

One final issue control addresses is oversmoothing. Our control term can address this by learning forces within the graph that counteract the effects of oversmoothing and effectively cancel out local diffusion. In particular, the control terms operate on a subset of the adjacency graph which may locally look quite different to the underlying graph, thereby inducing asymmetry which stops or at least limits the oversmoothing tendency.

3.2 Beyond Adjacency: Alternative Control Graphs

Though our experimental data in Section 5.1 focus on a subset of possible control terms, we considered multiple other possibilities over the course of our investigation. The key choice in our framework is the construction of the control adjacency matrix. This in turn can be decomposed into a choice of node activations and a choice of edges between these nodes.

Some natural choices of node activations arise from considering node statistics on the original adjacency matrix, as mentioned above. Centrality measures such as betweeness centrality or pagerank centrality are natural in many cases for capturing which nodes are most relevant to the graph. However, another choice which may be relevant in graphs with bottlenecked structure is graph *curvature* [11]. By activating nodes connected to edges with very low curvature, this selects nodes which are likely to be bridges between communities in the graph, and might facilitate interaction between disparate communities.

We also considered multiple alternatives for constructing adjacency matrices on the activated nodes. One early idea was to simply connect every node in the graph to every activated node in a unidirectional fashion. This way, every node could 'see' what was happening at important points in the graph. However, we observed that this approach was usually worse than doing control on the original adjacency graph and often overfit the training data. Moreover, this also greatly increases the number of edges over which message passing is occurring, making the architecture much less computationally efficient for little or no gain.

When considering heterophilic datasets and those with long-range interactions, we also thought of using two-hop adjacency graphs for the control term. In this case, activated nodes are connected if they are separated by a path of length 2. Our theory was that this may aid in heterophilic node classification tasks, following the principle of "the enemy of my enemy is my friend". In addition, this would allow messages to spread more rapidly across the graph than the usual adjacency matrix, potentially improving long-range interaction. However in practice we did not see an improvement on the heterophilic datasets when using this method, and as such did not include it in our more detailed benchmarks.

Stochastic forms of control are also a possibility. DropEdge [14] is a method for reducing oversmoothing and overfitting in deep GNNs, inspired by the success of Dropout [15] in standard NN architectures. Although DropEdge randomly removes edges from the graph along which messages are passed, we can instead randomly *include* edges in our control adjacency graph. This can directly alleviate oversmoothing by asymmetrically disrupting the diffusion process in a stochastic fashion, thus reducing the amount of label smoothing which occurs. We tested the performance of this stochastic control term on a few datasets but found no meaningful improvement in performance. However, our experiments were limited and questions still remain: should the control adjacency be a random subset of edges? Should edges be chosen uniformly, or through some random walk procedure? Should there be a pre-specified set of activated nodes, with random edges chose between them? Reformulating DropEdge as a stochastic control term where we *add* edges seems to be an interesting idea for future exploration.

Finally, one can consider cases where the nodes in the control term are higher-level features. For example, one might consider message passing between nodes representing graph motifs such as cycles, cliques, or stars. This may be especially relevant in molecular datasets, where particular atom structures are known to interact as units. However, constructing such 'higher-order' control requires significant familiarity with the dataset at hand to maximize task-specificity, and is difficult to do in a general fashion. As such, we focus on more general approaches applicable to a variety of graph datasets.

4 Synthetic Tasks

In order to demonstrate the potential of control-inspired GNNs to overcome issues such as oversmoothing and oversquashing, we studied their performance on two simple synthetic tasks for which standard GNN architectures perform poorly.

The first of these is a linear clique graph dataset, illustrated in Figure 2. In this task, the goal is to learn a linear order on the cliques which compose the graph. The node features are set uniformly to 1, so the network is only able to classify the cliques using the graph structure. Neither a GCN

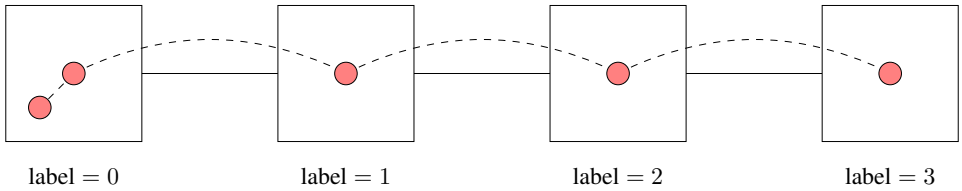

Figure 2: Illustration of Linear Ordering task. The goal is to learn node labels which form a linear order on the communities. Each of the boxes represent a complete graph on 20 nodes. Note that due to the symmetry of this graph, standard GCN architectures fail to learn an ordering. By introducing a control term on the adjacency graph (dotted line) on the activated nodes (orange), the symmetry is broken and the GNN is able to learn the desired order.

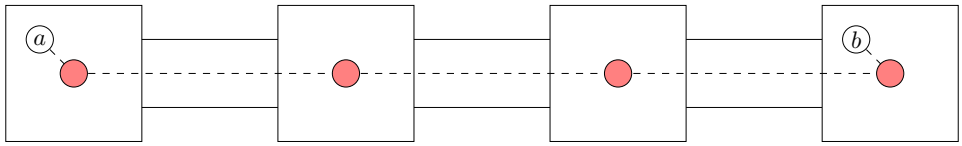

Figure 3: Illustration of Label Propagation task. The goal of this task is to propagate a label at one end of the graph (denoted by a) to a node at the other end of the graph (denoted by b). Similar to the previous task, the graph consists of linearly connected dense cliques with sparse 'bridge' edges between them. A standard GCN fails on this task due to oversmoothing issues.

nor an MPNN is capable of learning this linear order due to the symmetry of the graph - since every community is a clique, nodes which are an equal distance from either of the 'end' communities have exactly the same n-hop neighborhoods. We introduced an asymmetric graph on which we apply the control, which is composed of all of the nodes on the bridge' i.e. the linear subgraph, and one additional node from the first clique. With this asymmetric control term, the GNN is able to break this symmetry and therefore learn the desired order.

The second synthetic task is also performed on a similar set of linear graphs, however the conditions are relaxed slightly so that the graph is composed of a linearly connected set of dense communities, with a few "bridge" edges between them. A single node in the rightmost community is given a label between 1 and 10, and the goal is to propagate that label all the way to another node in the leftmost community, see Figure 3. This task is challenging due to a combination of factors: long-range interactions, oversmoothing within the dense communities, and potentially oversquashing on the bottlenecks between communities. Our control-inspired architecture is able to overcome this easily by providing dense connections between the different communities, where we 'activate' a single node in each community. Figure 4 demonstrates the failure of the plain GCN to learn, while the control model continues learns effectively.

5 Dataset Benchmarking

5.1 Experiment Configuration

We evaluate the proposed control-inspired GNNs on three common graph representation learning datasets; Chameleon [3] and PubMed [16] for node classification, and Enzymes [17] for graph classification. These represent a mix of heterophilic, homophillic, node and graph classification tasks, allowing us to assess the effectiveness of control in a variety of situations. We perform five-fold cross-validation, choosing 5 different train/validation/test splits. We train for 100 epochs on each split, and perform early stopping based on the lowest validation loss achieved.

For each dataset, we trained a GCN architecture both with and without control terms. In all cases, the model consisted of two linear encoding layers for the node features, a variable number of GCN layers (potentially with a control term), and two linear decoding layers. For the graph classification task an additional sum pooling layer is placed before the decoder. Our control uses the MPNN layer defined in Equation 2. Additionally, all linear and convolution layers are followed by a layer norm and

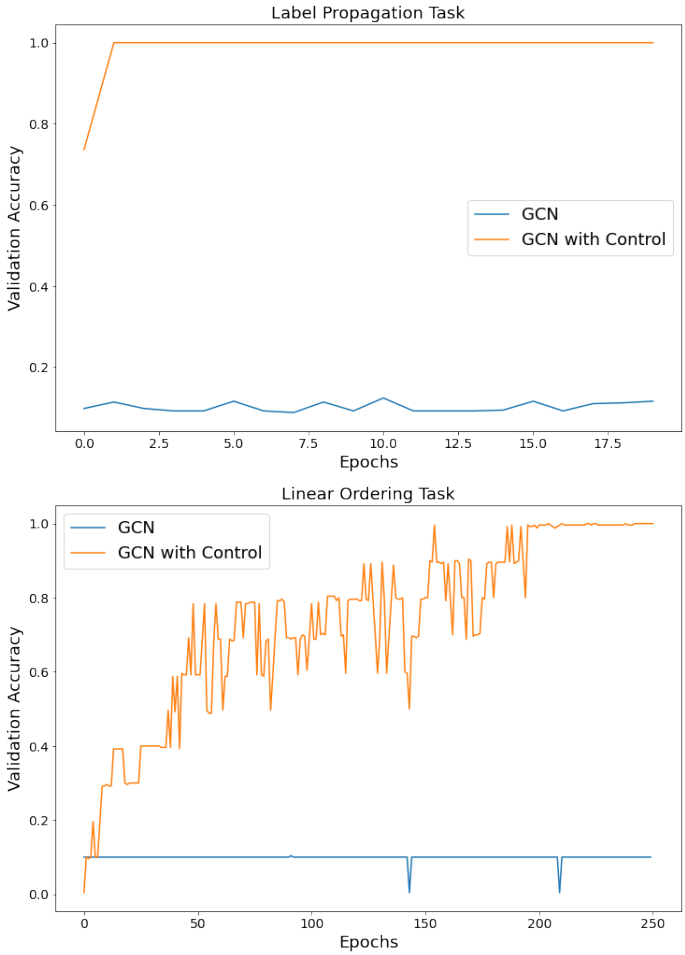

Figure 4: Training run for synthetic dataset tasks. **Top: Label Propagation** We see that the standard GCN architecture fails to learn, likely due to oversmoothing occurring at the depths required to propagate the desired label. In contrast, the control achieves perfect performance almost immediately. **Bottom: Linear Ordering** The goal of this task is to learn a linear order on a set of 10 connected cliques. Due to the symmetry of this graph, with many isomorphic n-hop neighborhoods and uniform node features, the GCN is unable to learn informative embeddings. In contrast, the slightly asymmetric control term is enough to overcome these limitations and learn an appropriate ordering.

Model	Enzymes	Pubmed	Chameleon
GCN	0.617 ± 0.044	0.879 ± 0.004	0.442 ± 0.029
GCN + ASG B-Centrality	0.567 ± 0.093	0.887 ± 0.003	0.442 ± 0.036
GCN + DSS B-Centrality	0.657 ± 0.063	0.887 ± 0.005	0.441 ± 0.040
GCN + ASG Curvature	0.537 ± 0.025	0.888 ± 0.006	0.439 ± 0.017
GCN + ASG B-Centrality (Post)	0.553 ± 0.104	0.886 ± 0.002	0.443 ± 0.025

Table 1: Results of Control GCN Benchmark. Each entry corresponds to mean and standard deviation of the best performing hyperparameter configuration across all 5 splits. ASG = Adjacency Subset Control. DSS = Dense Subset Control.

dropout layer, and all models have hidden dimension of 128. We employ the ADAM [18] optimiser, where for graph classification we use a batch size of 128.

We evaluated two different control graphs: subgraph adjacency control matrix and dense subset control. In all cases, we activated 5% of the nodes in the graph, chosen according to one of two metrics: betweeness centrality and Ollivier-Ricci Curvature [19]. These provide relatively different subsets of activated nodes, in order to highlight the differences the choice of activation metric can have on performance. Finally, we also evaluated a 'post-GCN' training run, where the parameters of the GCN model were trained first, and then held fixed while the control parameters were trained (starting from an initialization with very small magnitude).

For each model, dataset, and training split, we perform a four-dimensional grid search over learning rate $= \{10^{-4}, 10^{-3}, 10^{-2}\}$, weight decay $= \{0, 10^{-6}, 10^{-5}\}$, dropout $= \{0, 0.2, 0.4\}$ and depth $= \{2, 3, 4\}$. Results shown below are mean and standard deviation of the best hyperparameter setting across all 5 splits.

5.2 Results

Overall, our benchmark experiments fail to show meaningful improvements over the GCN baseline in most cases. It seems that it provides a small boost on the PubMed dataset, however more extensive experiments would need to be done to confirm that this is statistically significant. Additionally, given the significant added expressivity of the MPNN, this improved performance seems relatively underwhelming given the greatly increased parameter count and computational expense.

We note that in one case (the Enzymes dataset), dense subset control with betweeness centrality does seem to provide a meaningful boost to performance. This might perhaps be due to the increased long-range interactions which the control term allows, which might be especially relevant for molecular datasets. However, our control term does not seem to improve the capability of GCNs on heterophilic datasets, as evidenced by the poor performance on Chameleon.

6 Discussion and Conclusion

Given the motivations that we outlined earlier, and the ability to outperform other architectures on synthetic datasets, we believe control-inspired GNN architectures may yet prove to be useful in certain contexts. However, as our benchmark results demonstrate, finding good metrics and control graphs to effectively learn control dynamics is a challenging task. One-size-fits all approaches may be highly unlikely to succeed, and careful analysis of specific GCN diffusion patterns on particular tasks may be needed to successfully leverage the power of control-inspired GNNs.

Moreover, the breadth of possible control terms and architectures is large, and we have only explored a small subset here. More complex control graphs, perhaps leveraging higher-order features of the original data graph may be necessary in some cases. In others, more carefully crafted node statistics may be useful for deciding which nodes to activate, or perhaps even allowing the control activations to be learned as well.

Theoretical approaches to control may also prove fruitful in creating better control-inspired GNNs. In particular, the gradient flow framework of [4] could be augmented with control-theoretic tools such as the Z-Transform, a discrete-time analogue of the Laplace Transform. Though this can only

be used in cases where the underlying differential equation is linear and time-invariant, [4] showed that expressive GNN architectures are still possible within this framework.

Acknowledgements

We thank Dr Francesco Di Giovanni and Mr James Rowbottom for their insightful and encouraging supervision during this project. We further thank Prof Pietro Liò and Dr Petar Veličković for their teaching this term.

References

[1] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods, October 2021. URL http://arxiv.org/abs/2110.14446. arXiv:2110.14446 [cs, stat]. 1, 3

[2] Uri Alon and Eran Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications, March 2021. URL http://arxiv.org/abs/2006.05205. arXiv:2006.05205 [cs, stat]. 1, 2, 3

[3] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric Graph Convolutional Networks, February 2020. URL http://arxiv.org/abs/2002.05287. arXiv:2002.05287 [cs, stat] version: 2. 1, 6

[4] Francesco Di Giovanni, James Rowbottom, Benjamin P. Chamberlain, Thomas Markovich, and Michael M. Bronstein. Graph Neural Networks as Gradient Flows: understanding graph convolutions via energy, October 2022. URL http://arxiv.org/abs/2206.10991. arXiv:2206.10991 [cs, stat]. 1, 2, 3, 8, 9

[5] Chen Cai and Yusu Wang. A Note on Over-Smoothing for Graph Neural Networks, June 2020. URL http://arxiv.org/abs/2006.13318. arXiv:2006.13318 [cs, stat]. 2, 3

[6] A. Li, S. P. Cornelius, Y.-Y. Liu, L. Wang, and A.-L. Barabási. The fundamental advantages of temporal networks. *Science*, 358(6366):1042–1046, November 2017. doi: 10.1126/science.aai7488. URL https://www.science.org/doi/10.1126/science.aai7488. Publisher: American Association for the Advancement of Science. 2

[7] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. July 2022. URL https://openreview.net/forum?id=SJU4ayYgl. 2

[8] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. URL http://arxiv.org/abs/2104.13478. arXiv:2104.13478 [cs, stat]. 2, 3

[9] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations, December 2019. URL http://arxiv.org/abs/1806.07366. arXiv:1806.07366 [cs, stat]. 2

[10] Kuangqi Zhou, Yanfei Dong, Kaixin Wang, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Understanding and Resolving Performance Degradation in Graph Convolutional Networks, September 2021. URL http://arxiv.org/abs/2006.07107. arXiv:2006.07107 [cs, stat]. 3

[11] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature, November 2022. URL http://arxiv.org/abs/2111.14522. arXiv:2111.14522 [cs, stat]. 4, 5

[12] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion Improves Graph Learning, April 2022. URL http://arxiv.org/abs/1911.05485. arXiv:1911.05485 [cs, stat]. 4

[13] Trang Pham, Truyen Tran, Hoa Dam, and Svetha Venkatesh. Graph Classification via Deep Learning with Virtual Nodes, August 2017. URL http://arxiv.org/abs/1708.04357. arXiv:1708.04357 [cs, stat]. 4

[14] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification, March 2020. URL http://arxiv.org/abs/1907.10903. arXiv:1907.10903 [cs, stat]. 5

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. ISSN 1533-7928. URL http://jmlr.org/papers/v15/srivastava14a.html. 5

[16] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings, May 2016. URL http://arxiv.org/abs/1603.08861. arXiv:1603.08861 [cs]. 6

[17] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs, July 2020. URL http://arxiv.org/abs/2007.08663. arXiv:2007.08663 [cs, stat]. 6

[18] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs]. 8

[19] Yann Ollivier. Ricci curvature of metric spaces. *Comptes Rendus Mathematique*, 345(11): 643–646, December 2007. ISSN 1631-073X. doi: 10.1016/j.crma.2007.10.041. URL https://www.sciencedirect.com/science/article/pii/S1631073X07004414. 8