GDL Reproducibility Challenge: ANTI-SYMMETRIC DGN: A STABLE ARCHITECTURE FOR DEEP GRAPH NETWORKS

GDL 2024
Group id: **A-DGN**
Project id: A-DGN

Christian Altrichter, Francesco Huber, Jury Andrea D'Onofrio
{christian.altrichter, francesco.huber, jury.donofrio}@usi.ch

Abstract

The following report incorporates the reproducibility challenge based on the paper: Anti-Symmetric DGN - A Stable Architecture for Deep Graph Networks (Gravina et al. 2023). The original paper introduces a new graph neural network architecture which leverages anti-symmetric weight matrices to increase the stability of the systems while mitigating the oversquashing problem often encountered in deep graph networks.

Our team successfully carried out the reproduction of one of the key experiments detailed in the original paper, which demonstrated the efficacy of the proposed Anti-Symmetric Deep Graph Network (DGN) in preserving information over multiple layers. By following a similar methodology and architectural settings outlined by the paper, we successfully replicated the experiment and verified the results using the same dataset, "PubMed" from Planetoid.

Furthermore, we verified the author's claim about various mathematical properties exhibited by models which leverage anti-symmetric mechanisms. We utilized an heuristic approach to check during training. This served us by comparing anti-symmetric vs regular models. Our findings corroborate the original paper's assertions clearly demonstrating that the supposed property was indeed respected. The anti-symmetric approach not only stabilized the learning process but also maintained high levels of accuracy across tasks, supporting the robustness of this architecture.

1 Introduction

As part of the course "Graph Deep Learning" at Università della Svizzera Italiana, under the tenure of Prof. Ceasare Alippi, we were given the task to understand and re-implement a paper. For the chosen paper, the goal was to perform a reproducibility challenge. A reproducibility challenge, especially when existing code is published for the paper, does not consist of rerunning the code and checking its results. Therefore, to our understanding, we approached the paper and the implementations with critical thinking and verified the claims that have been brought forward in the paper and matched them with their code implementation. Furthermore, an own implementation of the paper and the models used for comparison should ideally be provided and at the same time proving and / or disproving their results. Lastly, the problem that has been solved within the scope of the paper should be extrapolated. Individual research of common graph deep learning techniques to solve the same challenge should be explored and applied beyond the scope of exploration of the paper itself. Thus, being able to compare the true novelty of their paper. Certain sections related to mathematical proofs are beyond the scope of this project and are therefore assumed to be correct without further verification.

The paper we have chosen is called "Anti-Symmetric DGN: A stable architecture for deep graph networks", written and produced by Alessio Gravina (University of Pisa), Davide Bacciu (University of Pisa), and Claudio Gallicchio (University of Pisa). In principle, the paper addresses the issue of over-squashing. As a paper published in 2021 [1], over-squashing refers to the phenomena where graph neural networks perform poorly on long-range dependencies due to both the number of edge-counts between nodes as well as the depth (therefore temporal dimension) of the network. The paper defines long-range dependencies as messages coming from non-adjacent nodes that need to be propagated through the network without the loss of information through convolutional operations. Due to the exponential nature of graphs with respect to node-neighboring structures, a fixed size vector is therefore easily distorted. The paper introduces the notion of a stable and non-dissipative DGNs. Stable meaning invariant to vanishing or exploding gradients. The instability comes inherently from the initial state of the graph. Stability can be (as proven in the paper [2]) achieved if and only if for all time slices the real part of the max eigenvalues of the jacobian of $f_g << 0$, meaning negative but very close to 0. Should the values be < 0, thus, considerably smaller than 0, then one could observe in theory that no long term dependencies are preserved. Non-dissipative means the preservation of information throughout the layers. A common challenge is that the more layers a GNN introduces in its architecture, the more the oversquashing phenomena appears. This means that information starts to be dissipative, or in other words

that all vectors of a node start to converge to one value. Consecutively, loosing information over time. The loss of information in particular comes through the aggregation method of the message passing (e.g. addition, averaging etc.). One might argue that to solve over-squashing, a potential approach would be to increase the dimensionality of the vectors of each node to capture more information. This could however lead to challenges in memory usage as well as efficiency of processing time especially when larger graph structures are processed. Thus, another approach must be taken.

Here, the paper introduces the notion of ordinary differential equations (ODE) to solve the over-squashing problem, combined with an anti-symmetric weight application within the ODE. A problem typical neural networks face is the discretization of time steps through layers. However, as typical graph neural network applications deal with continuous time (e.g. prediction of energy consumption for a city), ODEs can be used to model the data more accurately. As a consequence, we are able to map nodes over time as continuous function. This results in stability and non-dissipative properties.

In general, we are able to verify the claim that the ADGN framework proposed is non-dissipative compared to classical graph convolutional neural networks. We furthermore have proven the stability through random checks of the models mathematical properties. It can be criticized that the comparison with baseline methods that address the oversquashing problem are inadequately explored. Thus, our own comparison had to further be brought forth. We have observed certain logical errors when comparing for instance the ADGN with a Graph Attention Network (GAT). Despite our limited understanding of the mathematical proofs provided in the paper, it becomes quickly evident that an in depth explanation of the proposed methods and the reasoning behind the usage is lacking. Lastly, the extensiveness of their approach to solving the challenges is not a wise usage of computational power, e.g. running 1500-10000 epochs for each model (with early stopping).

2 Related works

In this section, we will briefly touch upon related work introduced by the paper and extend to our own research. First, the notion of message passing is introduced [3]. The message passing function takes into consideration for every node its neighbors at time t, performs some kind of aggregation and writes this information to node at time $t + 1$. Graph Neural Networks mainly differ in the way they perform the aggregation function.
As part of our reproducibility, we have in detail analyzed the following model architectures besides the ADGN proposed by the paper:

1. GCN - Graph Convolutional Network: This model utilizes convolutional operations on graphs to capture information. Convolution can be seen as the degree of message passing layers applied to the graph. This in theory enhances the extraction of features. However, a renowned problem is the over-squashing phenomena. [4].
2. GAT - Graph Attention Network: This model architecture makes use of the self-attention mechanism which enables the node to selectively attend to neighbors. This enhances flexible and adaptive information aggregation.
3. GGNN - Gated Graph Neural Network: Similar to NLP, long term dependencies have been an issue with RNNs. With the introduction of LSTMs and the concept of a cell structure, models have been able to preserve long term dependencies. [5]. Thus, we explored the effects of using cell structures moved by an interest in assessing their potential to preserve long term dependencies with regards to graphs. We selected the architecture of GGNNs which employs gated recurrent cells to propagate information across graph structures. [6].

3 Methodology

As briefly mentioned in our introduction, the state of the art of the paper was achieved via a novel approach of designing and applying mathematical constraints onto existing graph convolutional networks. In particular the authors leveraged the natural property of neural networks to be expressed as systems of ODEs, which can be thought as continuous mappings of the states of each node of the graph. This mapping in turn makes it possible to extrapolate and prove methods to solve the oversquashing problem analytically. The solution proposed by the authors originates from reasoning on the underlying ODE of a particular neural network which in turns is defined by a Cauchy Problem on graphs where each temporal slice through the ODE represents a graph state of our GCN as displayed by Figure 1.

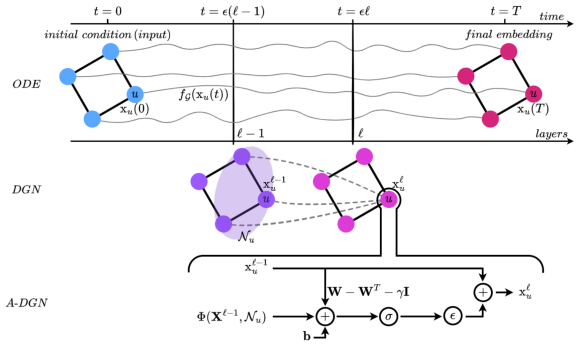

Figure 1. From ODE to DGN

The authors have been able to propose changes to the forward function which makes it able to sustain long term dependencies while also reducing the dependency on one particular input and at the same time enable better generalization capabilities. The changes to the forward pass include the transformation of the learnable weigh matrix into an anti-symmetric variant and adding a mechanism analogous to skip connections by having an additional term

which represents the input of layer $l-1$ for a given node at time-step l, as shown in the following formulas:

Regular DGN state update rule:

$$x_u^l = \sigma(x_u^{l-1}W + \Phi(X^{l-1}, \mathcal{N}_u) + b) \quad (1)$$

Revised A-DGN state update rule:

$$x_u^l = x_u^{l-1} + \epsilon\sigma(x_u^{l-1}(W - W^T - \lambda I) + \Phi(X^{l-1}, \mathcal{N}_u) + b) \quad (2)$$

The validity of this approach was proved mathematically, granting the authors to affirm the formulation efficiency to the over-squashing problem and non-dissipativeness of information.

In order to further validate their claim, the authors conducted a series of tests which explored several parameter configurations, datasets and convolutional architectures. In terms of hardware, the tests have been carried out on a Dell server equipped with 4 Nvidia A100 GPUs.

The framework proposed in the original paper has been evaluated according to a suite of trials pertaining to both *graph property prediction task* and *graph benchmarks*.
The graph property prediction tasks consisted of two node level tasks and one graph level task, namely the prediction of: single source shortest path, node eccentricity and graph diameter; the correct solution of such scenarios assumes the ability of preserving and identifying long term-dependencies between all nodes of the graph. The three datasets used for the above tasks have been obtained via a generator using specific seeds.
The A-DGN model has been tested with both weight sharing, layer dependent weights configuration and 2 aggregation strategies. The complete set of models used to carry out the performance comparison against the A-DGN framework includes: GCNII, GCN, GAT, GraphSAGE, GIN and two neuralODE-based models (GRAND and DGC).

As highlighted in Figure 2, all models are implemented to have an initial encoding to a latent space via linear layer, followed by a graph convolutional structure specific to the model being implemented and lastly a 'readout' layer which maps the convolution output to extract the results.

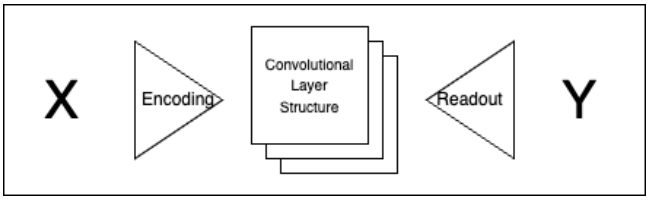

Figure 2. Paper's Common Architectural GCN Implementation

All models have been tuned via grid search before running the comparison. Each model was trained using Adam optimizer for 10'000 epochs.

The same architectural designs and configurations were then used for the subsequent test suite: 'graph benchmark setting'. For this context, 5 popular datasets were selected:

- PubMed
- Coauthor CS
- Coauthor Physics
- Amazon Computers
- Amazon Photo

Similarly to graph property prediction, hyper-parameter tuning was achieved by means of grid search, using a period of 1'500 epochs.

4 Implementation and Experimental Set Up

4.1 Datasets

Our team considered various benchmark datasets, ultimately the choose 'PubMed' as it belonged to both Planetoid which was seen during lectures and also used in the paper. The PubMed dataset is part of the Planetoid collection, which includes several benchmark datasets commonly used in graph learning research. PubMed is used for evaluating the performance of graph-based semi-supervised learning models, particularly in node classification tasks. Semi-supervised means that only a certain number of nodes have labels / can be used for training.
The graph is structured as follows:

1. **Nodes**: 19'717
2. **Edges**: 88'648
3. **Features**: 500
4. **Classes**: 3

To visualize the node prediction tasks, we have taken two models (one performing well and one not so well) and did a TSNE reduction to visualize the nodes on a two dimensional euclidean space with the predicted classifications. Here are the results:

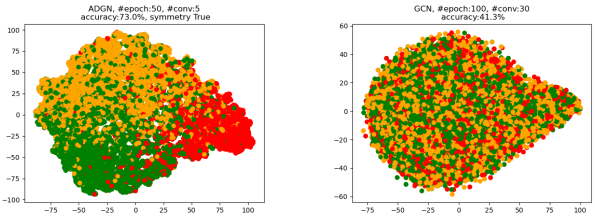

Figure 3. PubMed Clustering Examples: On the *left* ADGN with Accuracy of 73.0% and on the *right* GCN with Accuracy of 41.3%

4.2 Choice of tasks

We have opted to (due to the scope of this project) to focus only on the graph benchmark task of one Dataset. This enables us to prove and / or disprove their results. Furthermore, we deemed it important to transfer concepts and knowledge seen in class (e.g. the PubMed Dataset) to use this for the reproducability task.

4.3 Code Review

First, we started revising their provided code. We have identified the following challenges within their code which made the provided code unusable:

1. Missing dependencies in the yml file (prettytable)
2. Undeclared variables used (compile error): AntiSymmetricDGN/graph_benchmark/models/dgn.py the variable self.conv_name was not instantiated. We tried figuring out if it is inherited from the super class Module through `print(dir(self))` and `print(dir(self.__getattr_))` but it is not there. So we proceeded to change it to `self.conv_layer` (without fully knowing the implications of this code change). However, when we changed it we could successfully run training.
3. The `.sh` command is not properly parsed so we need to run the `python3 main.py` command for the individual models.

This necessitated thorough debugging to resolve undeclared variables, ensuring the code's reliability and effectiveness. Ultimately, due to libraries that are definitely used with large computational availability (e.g. ray for multiprocessing on GPUs) we opted to re implement everything from scratch at adjust to our computational availability and prove and / or disprove only certain aspects of the paper.

4.4 Basics of GNN

Starting with the implementations, we focused first to manifest the basics. For that, we began with implementing a basic GNN and a basic custom convolution / message passing. Through such, we were able to translate theoretical concepts into practical code. These implementations can be viewed in our project structure under *'GNN_basics'*.

4.5 Hyperparameter search

After having thorough understanding of graph deep learning techniques, we proceeded to select 3 models to compare ADGN to. The aim was to choose models that are best aligned with our research goals. Here, the following models were implemented:

1. Trivially we have re-implemented the paper of the ADGN without weight sharing across the layers.
2. One of the most common architectures used as baseline is the Graph Convolutional Networks (GCNs).
3. As an alternative to capture long term dependencies and deviate from the convolutional framework, we have opted to choose the Graph Attention Networks. Thus, giving us a true insight into their efficiency of their results.
4. Lastly, we have taken the over-squashing problem and have transferred the challenge of long term memory of NLPs to the graphs. Thus, we expected good results from a Gated Graph Neural Network.

Implementing these models have included producing the training loop for all of them with the Dataset 'PubMed' from Planetoid Dataset.

One of the main claims the paper states is that with an increase of layers the accuracy of models deteriorate, which is solved by the ADGN. As they have given their hyper parameters that they have used for the result production, we first wanted to check that claim and verify its generalizability. Thus, we have looked across only two out of the four models, namely the GCN and the ADGN for a good hyperparameter set. The limited hyperparameter search was due to the computational availability. We explored the following hyperparameters for both models:

1. Convolutional Layers: [1, 2, 3, 5, 10, 12, 20, 30]
2. Learning Rates: [0.1, 10e-3, 10e-4, 10e-5]
3. Hidden Layers: [4, 8, 12, 24, 48, 64, 128]

We conducted extensive hyperparameter searches, systematically exploring the impact of varying parameters on model performance. This rigorous analysis aimed to either substantiate or challenge existing assumptions regarding layer configurations and their implications for model efficacy. Interestingly, but not surprisingly, the accuracy can be upheld with increasing number of layers by adjusting carefully the learning rate and increasing the number of hidden dimension. The results can be seen in **??** and **??**. Here, as can be seen the higher the number of layers, the more we will have to increase the hidden dimensions to capture better the long term dependencies.
The authors of the paper have done their own hyperparameter search over a larger search space by for instance including the epsilon and gamma search space of the ODE.

The results and the implementation can be viewed in our project structure under *'Hyperparameter'*. Overall, to compute the results for one of these took about 9 hours on an Intel Macbook Pro for each hyperparameter search.

5 Results

The goal of the reproducibility challenge is not to verify and check every claim, but to choose the few most important aspects of the paper and run separate tests on them. Thus, we have opted to choose the two main claims *Stability and non-dissipative behavior* of the models. The first we can prove by comparing the maximum real part of the eigenvalues of the ODE when using antisymmetric weights matrices compared to non-antisymmetric weight matrices. The latter result to be proven or disproven is the non-dissipative behaviour which can be shown through an accuracy measurement across models with different settings while increasing the depth of the network architectures. At the same time, we will gain insights into the effectiveness of their implemented novel approach.

5.1 Verification of base claim of stability through ODE

We then proceeded to verify the claim of the paper denoted as *Proposition 1*. Seeking mathematical correctness, we employed Jacobian analysis to assess the stability and sensitivity of the ADGN model with respect to the ODE. This analytical approach provided insights into the models' behavior and their robustness. We compared the ADGN with the anti-symmetric weight construct and without. Here, we expected that the model without anti symmetric weight sharing and ODE implementation would suffer from instability by showing either strongly negative or positive

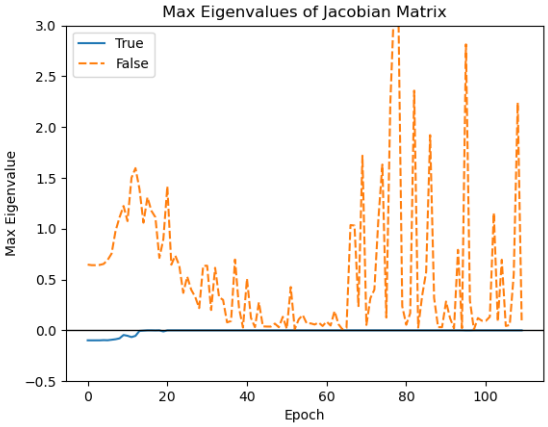

Figure 4. Max eigenvalues of ADGN with anti-symmetric weight matrix and without

values of the maximum value of the eigenvalue of the Jacobian matrix. It is important to note that we have not verified the fundamental assumption that stability is given if and only if the real part of each Jacobian's eigenvalues of f_g is equal or slightly below than 0. As enunciated in the following proposition 3 formula:

$$Re(\lambda_i(\mathbf{J}(t))) \lessapprox 0 \; \forall i = 1, ..., d. \tag{3}$$

The results can be observed in the following figure 4. Clearly, when setting the anti-symmetric weight matrix to true, we observe stable behaviour (in blue) whereas when setting it to false, we observe non-stable behaviour. Thus, we can verify the claim of stability brought forth in the paper.

It is important to note that within the scope of this project and with our limited computational capability, we have decided to preform the aforementioned verification by randomly selecting nodes within each epoch. This approach was rendered necessary by the real-time implications of checking each node (about a minute per node) by computing the Jacobian Matrix and its the max eigenvalues verification for all 19 thousand nodes and the multiple epochs. Concretely, our method selects and verifies 1 node per epoch during the training loop.

The results and the implementation can be viewed in our project structure under *'Jacobian'*.

5.2 Reproduction of non-dissipative property

Here, we have tested each model with increasing depth in their architecture. The code can be viewed within our project under *'Conv_layer'*. The goal here was to increase the depth of the architecture by adjusting the number of convolutional layers (where applicable). Ultimately, we aimed at reproducing the following image of the paper:
We used the parameters proposed by the paper, which are as follows:

1. convolutional layers: [1,5,10,20]
2. learning rates: [0.003]
3. hidden layers: [10,20,30]

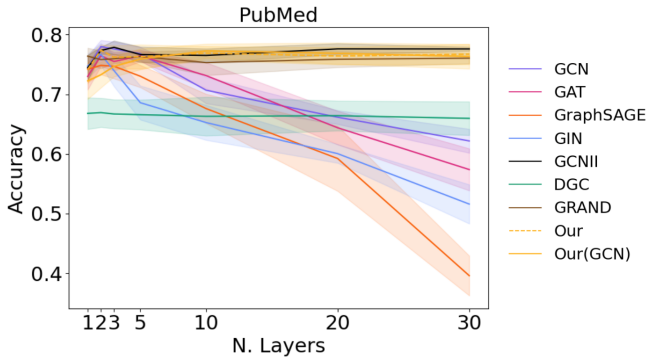

Figure 5. A-DGN paper results on PubMed dataset with increasing layer, showing non-dissipative behaviour of A-DGN

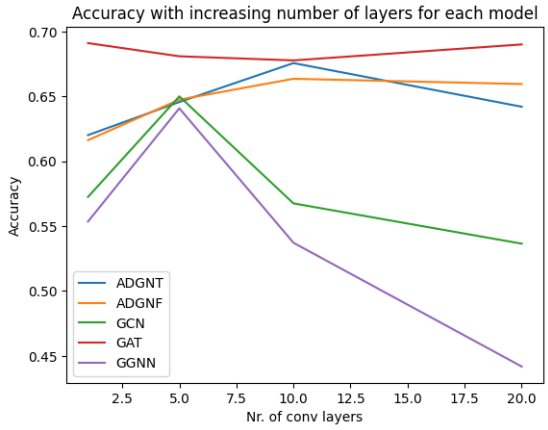

Figure 6. Different model implementations with respect to increase layer dimensionality

We ran each parameter setting for 100 epochs and took the accuracy on the test set every 10 epochs. Across all hidden dimension settings we then averaged the accuracy for a given model prediction to get an output for a convolutional layer setting. Here, the following considerations ought to be made. First, we are not sure if the paper used the max accuracy or the average accuracy on the test set achieved over the 100 epochs. Our assumption and intention to take the average was to observe the speed of learning within our statistics. Thus, when a model only achieves at later epochs a high accuracy it will be visible in an overall lower accuracy average. Second, the increase in convolutional layers does not make sense for all models. In particular, the Graph Attention Network typically replaces convolutions through the self-attention mechanism by number of heads. Thus, we for the GAT the number of layers is equal to the number of heads used in the model. The results of our test are as follows:
The results are in line with what the paper has produced. The ADGN shows a non-dissipativeness alongside the GAT. As stated and shown in the results of the paper, with an almost fixed setting of a hyperparameter and lower number of hidden dimensions, the GCN swiftly starts to deteriorate. To our surprise, this is also the case with the GGNN model. Here, we assume this to be the case due to the lack of a proper hyperparameter search. Adding multiple GRU cells after another can lead to vanishing/exploding gradient and/or over fit on the train data. Thus, it might not be beneficial to

construct the number of GRU cells according to the number of convolutional layers as done in our experiment.
Overall, to compute the results for all of these took about 3 hours on an Intel Macbook Pro. Of course there were multiple trail and errors until a clean reproducibility could be ensured.

5.3 Experimental setup

The experiments were conducted both on Macbook Pro with the M1 Pro and Intel chips using only the CPU. Furthermore, we did not use any external resources, such as Google Colab or Kaggle, that provide free GPUs in the course of our experiments. However, running on online compute required to build wheels for the repository, which, due to the large amount of libraries used took too much time and lead to an error.
Our code can be found at the following GitHub Repository.

5.4 Computational requirements

To better understand the efficiency and the efficacy of the model implementation compared to other state of the art models, we also have to have a look at the computational resource utilization at hand. Therefore, for each test performed we have run our own analysis of the computational requirements.
For context, here are the computational units that were used during the experiments:

Model	Chip	Ram	Cores	CPU
2019 Macbook Pro	Intel	16 GB	6	2.6 Ghz
2021 Macbook Pro	Arm	16 GB	10	3.2 Ghz

Table 1. Team's Hardware

Train-loop with fixed parameters Here, we have tested each models basic performance with standard setting of 3 convolutional layers 100 epochs and 32 hidden dimensions:
We cannot find a difference in CPU nor memory consumption. When looking at the memory consumption, the difference comes from the starting memory when the train loop runs.
However, when looking at the time taken, we can clearly see that ADGN outperforms all the other models quite significantly. Here, we see a speedup of approximately 1.8x from ADGN-GGNN, 1.9x from ADGN-GCN, and 2.7x ADGN-GAT at the last iteration taken. Furthermore, here we ensured that the accuracy results are all considerably comparable. Thus, all being in the range of 70.0% to 75.0% accuracy.
These tests were run on 1 the Intel Mac Book Pro.

Hyperparameter search space of ADGN The hyperparameter space search has been monitored using a 1 M1 Mac Book Pro and the configurations of 3. The entire search took approximately 24 hours for one complete search execution of one model. We monitored execution time, CPU utilization and allocated memory for each configuration (224 tested configurations). The results can be found in Figures 9.1, 9.2 and 9.3 .

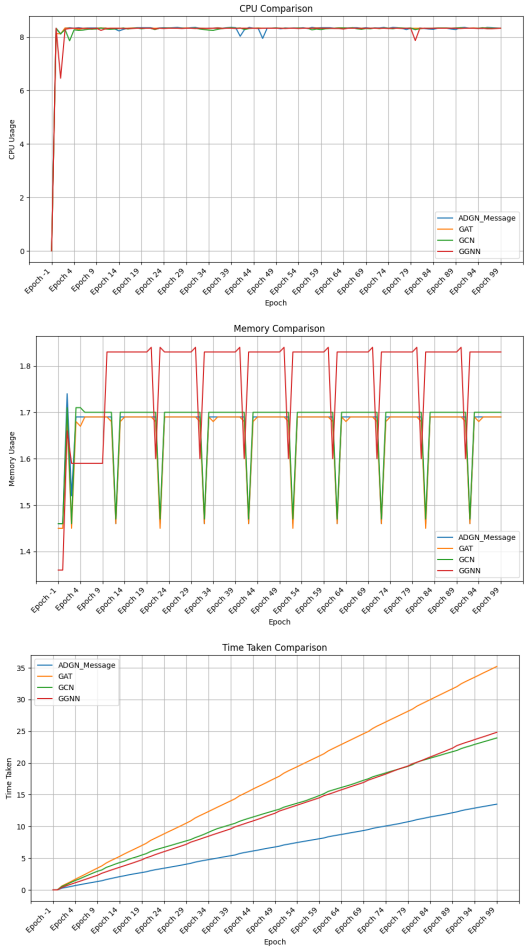

Figure 7. CPU, Memory and Time taken of different models

It can be noted that execution time is considerably affected by the amount of convolutional layers and hidden dimensions. Thus, as initially stated, we want to have the model that performs best with low dimensionality on the nodes, which as proven is the ADGN. When taking a further look at the graph **??**, the data becomes harder to interpret with regards to allocated memory as no clear pattern emerges. For what concerns CPU usage once again convolutional layers are the most impacting factor, causing bottlenecks already with 5 layers. Please refer to metric ranges in 2

Statistic	Min	Max	Metric
Execution Time	10	1431	Seconds
Allocated Memory	0.64	1.8	% of RAM
CPU Usage	61.92	98.89	% of CPU

Table 2. Grid Search ranges obtained on Mac M1

Increase of convolutional layers Lastly, we also have monitored how the different models have consumed CPU, memory and time across an increasing number of convolutional layers. A convolution layer in the graph context corresponds to the depth k of the convolution. As this directly captures the correlations to other nodes, and in particular we are interested in long term dependencies, we also have to take a look at how the models behave when performing the ultimate goal of the task at hand:

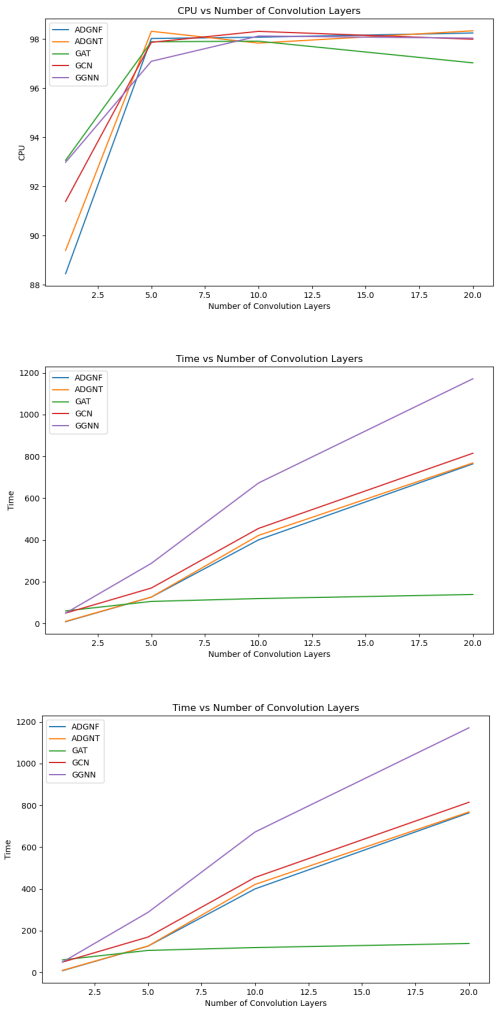

Figure 8. CPU, Memory and Time taken of different convolutions layers

Figure 9. CPU Usage, Memory Allocation and Execution Time for Hyperparameter search on ADGN

As the attention mechanism is easily parallelizable through pytorch the time taken for the GAT when scaling up is considerably lower than all other models (which we assume that pytorch does in the background as low level optimization). Furthermore, it might be the case that not all heads are useful and thus require little computational overhead. The other results are in line with our expectations, where ADGN outperforms GCN.

6 Discussion, conclusion and limitations

In conclusion despite some initial hurdle with the paper's provided code, which proved to be hard to run due to a multitude of bugs, our team decided on re-implementing a selection of models with the goal of reproducing a specific experiment in a simpler and more controlled fashion. Following a similar methodology as outlined in the paper, our team was successful in reproducing and verifying a selection of the claims of the original authors. Specifically we were able to reproduce the experiment on the PubMed dataset while also adding a different model to the mix as an interesting mean of comparison. To summarize, we were able to confirm that:

1. ADGN solves oversquashing. However, here we would need higher computational power to avoid a heurisitc approach and to be able to check the Jacobian real value for every node. For now, we assume through our applied heuristic and since we could not prove otherwise, the paper is correct.
2. ADGN solves dissipative behavior by preserving long-term dependencies over higher convolutional layers.
3. ADGN is better in computational compelxity compared to regular GCN.

As highlighted in the original work the A-DGN behaved very well as more convolutional layers were added while other models started to fail. Nevertheless the GAT architecture we used did perform remarkably better than the original plots show. We have in fact found that the way the authors used GAT was not very coherent with regards to its inner architectural design (iterating over multiple attention convolution blocks with a fixed amount of heads rather than treating heads as the convolution layer amount).

Critique on the published paper is inline with our initial assumption, for instance that:

1. The computational effort of 1'500 - 10'000 epochs was unnecessary.

2. The models benchmark used to compare ADGN to is not very useful. Despite GAT which directly addresses long term dependencies, no other useful model that directly revolutionized long term dependencies was used. Thus, an ideal place to do our own research and implementation of more novel models.

For future reference the work could be enhanced by leveraging on GPU architectures for more efficient training and testing, allowing for larger hyperparameter search spaces exploration and comparable results to the original paper.

References

[1] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*, 2021.

[2] U. M. Ascher, R. M. M. Mattheij, and R. D. Russell. *Numerical solution of boundary value problems for ordinary differential equations*. Society for Industrial and Applied Mathematics (SIAM), United States, unabridged, corr. republication edition, 1995. ISBN 0-89871-354-4.

[3] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[4] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.