Graph Representation Learning

Candidate number: 1071068

Date of submission: 1.1.2023

Please use this LaTeXtemplate to type your answers to the mini-project, and whenever applicable, please respect the following notation in your answers:

- $G = (V, E, \mathbf{X})$: Simple, undirected graphs with node features $\mathbf{X} \in \mathbb{R}^{|V| \times d}$.
- $N(u)$: The set of neighbors of a node $u \in V$.
- $d, d(t)$: The dimension of a vector, and the dimension of a vector at iteration t , respectively.
- $T \in \mathbb{Z}$: The total number of layers/iterations, and hence $0 \leq t \leq T$.
- σ: An element-wise non-linearity.
- $\mathbf{1}^d \in \mathbb{R}^d$: A d-dimensional vector of all 1's.
- $\mathbf{b}^{(t)} \in \mathbb{R}^d$: A bias vector.
- $\mathbf{x}_u \in \mathbb{R}^d$: The feature of a node $u \in V$.
- $\mathbf{h}_u^{(t)} \in \mathbb{R}^{d^{(t)}}$: The representation of a node $u \in V$ at time t, which is initialized as $\mathbf{h}_u^{(0)} = \mathbf{x}_u$.
- $\mathbf{z}_u = \mathbf{h}_u^{(T)} \in \mathbb{R}^{d^{(T)}}$: The final representation of a node $u \in V$ after applying T layers/iterations.
- $\mathbf{W}^{(t)}, \mathbf{W}_{\text{self}}^{(t)}, \mathbf{W}_{\text{neigh}}^{(t)} \in \mathbb{R}^{d^{(t+1)} \times d^{(t)}}$: Learnable parameter matrices at time t.

Q1a - Limitations and examples

1.Oversmoothing can occur in GNNs which use message-passing mechanism to propagate information between nodes. Node features converge to similar values such that the GNN cannot distinguish different nodes after several message passing iterations. In every layer the GNN aggregates messages from every node's neighbours, which will make the effect of previous embedding of each node contribute less towards the final classification/prediction and after passing information through all required layers, the effect of the initial features of a node will become insignificant. Additionally, if a layer uses pooling operations (i.e. mean), this can lead to further information loss from previous embeddings. This results in poor performance because nodes belonging to different classes converge to indistinguishable vectors. Consider the CORA dataset - if we train a GNN with many GCN layers, the GNN will be unable to classify papers into 7 categories and will instead classify all papers into 1 category due to convergence of feature vectors.

2.Oversquashing can occur when the task has larger problem radius (which is the minimum distance needed to solve the task), so the network has to pass messages across many layers in order to capture long-range dependencies. It happens if the output depends on embeddings of nodes far apart, interacting with each other. In such scenario the graph results in exponentially many long-range neighbouring nodes and the receptive field of each node grows exponentially with the neighbourhood radius. Information from this exponentially-growing receptive field is compressed into feature vectors of fixed-size, so messages are compressed which results in information propagation bottlenecks - because of this compression, information propagation bottlenecks increase difficulty of training deep GNNs with many layers.

3.Limits of expressiveness There are theoretical and empirical limits on what GNNs can capture. Some simple graph structures which have the same initialization and neighbors cannot be distinguished (e.g. if 2 graphs are isomorphic). The expressive power can be analysed by looking at the graph isomorphism problem for which no polynomial-time algorithm exists. MPNNs are at most as powerful as the 1-WL test in distinguishing different graph-structured features. This means that all graphs which are topologically different but cannot be distinguished by 1-WL test also cannot be distinguished by MPNNs. For example, if nodes A and B share the same subtrees then the MPNN produces the same representation for both nodes despite them being in different locations - this is because the structure of the graph is not included in the message aggregation function. This leads to issues because the location information of nodes is lost and unfortunately there are many tasks which heavily depend on the ability to distinguish location.

Q1b - Oversmoothing - quantitative measures

Performance evaluation: We can quantify the level of oversmoothing by measuring performance (e.g. by using mean-squared error) on some downstream task and measure the improvement /degradation in accuracy. We expect training, validation and testing accuracy to decrease if oversmoothing is present.

There are simple metrics that quantify oversmoothing which measure the distance between node embeddings produced by the GNN and the true labels of each node - if the distance is large it might indicate oversmoothing. Other quantitative measures include:

Mean-average distance: Calculates mean average distance among node representations and is a good proxy for similarity of nodes and smoothness of the graph - values become smaller as the representations become closer due to oversmoothing.

Information-to-noise ratio: Measures the quality of incoming messages to each node by looking at information-to-noise ratio which distinguishes between helpful information from neighbours (intra-class information) and harmful noise (inter-class messages).
Group-distance ratio: Computes the ratio of the sum of mean pairwise distances between node embeddings from different groups and divides it by the sum of mean distances between node embeddings of each group. If this ratio is small, the average distance between node embeddings in different groups is small, which indicates oversmoothing. High group distance ratio implies larger difference between embeddings which indicates low level of oversmoothing.

Q1b - Oversquashing - quantitative measures
Jacobian: Oversquashing can be analysed if we look at how the representation of a node is affected by some input feature of another node. Jacobian calculates sensitivity of the hidden features to the input features by using partial derivatives of the layer output of some node w.r.t. to the input features of some other node. If the values in the Jacobian are small then there is little influence of these input features on the output, signaling oversquashing.
Curvature: A measure related to edge dispersion that locally measures connectivity of neighbourhoods via edges. Given any pair of nodes A and B, if the curvature is negative, then the edge between A and B is an information bottleneck. The more negative curvature the worse is the oversquashing effect.
Ratio of negatively curved edges: In long-range tasks negatively curved edges induce information bottlenecks so observing the count of negatively curved edges normalised by all edges can provide quantitative measure of oversquashing.

Q1b - Expressive power - quantitative measures
Weisfeiler-Lehman (WL) test can be used for checking graph isomorphism - an upper bound for expressiveness of vanilla GNNs/MPNNs.

Empirical analysis on benchmark inductive and transductive datasets - measuring training accuracy on some task such as graph classification. If the model classifies graphs with low accuracy during training, it might be an indication that it cannot differentiate graphs because of the model's expressive limitations.

Measuring performance of a GNN on the EXP dataset - commonly used benchmark test which evaluates expressive power of a model based on the SAT problem. If the model fails on 1-WL indistinguishable graph pairs then it is not better than base GNN. If it does not fail on the 2-WL pairs then it has higher expressive power than GNNs.

Q1c - Addressing oversmoothing Approaches such as weight decay, dropout, dropping edges, adaptive edge optimization, MADReg (adding a regularisation term to the training objective), skip connections and JK networks (adaptive and structure-sensitive aggregation by max pooling), and others were developed. We will review **PairNorm** which is an approach inspired by the observation that graph convolution is a form of Laplacian smoothing, which after repeated application causes node feature convergence. PairNorm adds a normalisation layer between layers to prevent this convergence, by ensuring that the total pairwise feature distances are constant throughout the layers, which results in distant nodes having less similar features and close nodes having more similar features. The normalisation step is added after each GCN layer such that the output features are centred and scaled (except the last layer). It is easy to implement, can work practically on any GNN model and allows for training deep models without large degradation in performance. It has theoretical grounding and has been experimentally shown to alleviate oversmoothing in GCN, GAT, and SGC models. It does not require extra parameters or changes in the network topology. It can be applied to any layer, even if the layers have different sizes or the graph structure is changed at

each layer. The disadvantages are that this approach is not effective in case of shallow architectures with too few layers. Secondly, it only looks at features and distances between nodes so it will not decrease oversmoothing in graphs with complicated or very specific structure because the method only operates on node features without considering the graph structure. The changes it makes to the features might affect the alignment of individual features so if the graph structure plays huge importance for the task, it is not the best approach.

Q1c - Addressing oversquashing Multiple approaches address oversquashing such as adding fully-adjacent last layer, Geometric GCN approach (neighbourhoods in latent space where structurally similar nodes are put together), rewiring with positional encodings, virtual node connected to all nodes, multi-hop message aggregation, etc. We will review **Stochastic Discrete Ricci Flow** which is an algorithm based on the finding that negatively-curved edges are responsible for oversquashing which can be addressed by curvature-based rewiring. It is inspired by Ricci flow with the aim of homogenising edge curvatures and modifying negatively-curved edges. At each iteration, this algorithm adds an edge to support the graph's most negatively curved edge and then removes the most positively curved edge, thus greedily increasing curvature. It has theoretical grounding in geometry and graph topology and identifies the cause of oversquashing (negatively-curved edges). It uses a simple algorithm for rewiring which is not difficult to implement. It improves information flow without compromising computational and space complexity as well as statistical properties of the graph. Empirically it has been shown that it successfully tackles oversquashing and leads to performance improvements. Disadvantages include that this approach only focuses on graph topology and does not consider individual nodes, their node features or other information which could be extracted directly from the graph. If the node features are not taken into account while edges are being removed, this approach, although it improves oversquashing, could lead to worsening of the interactions between nodes if the wrong edges (which are important but have negative curvature) are rewired - i.e. the algorithm could remove some essential communication. In addition, it has been shown that this approach does not work well on multigraphs.

Q1c - Addressing lack of expressive power There are approaches which address the lack of expressivity such as higher-order GNNs which use higher-order representations of graphs, GNNs which involve hierarchical message passing, MPNN-RNI models which use random noise initialization to improve the graph indistinguishability problem and many others. We will review **Graph Substructure Networks**, an approach inspired by the observation that information about the neighbourhood of a node can be to some extent captured by substructures and therefore tracking occurrences of substructures provides information about the node's position within the graph. GSN uses substructures to facilitate structurally informed passing of messages - this introduces structural bias because a message going from A to B is transformed depending on structural identifiers which are assigned to both nodes. This results in creating equivalence classes for nodes/edges depending on the characteristics of the substructure shared between nodes and can be extended even across different graphs (depending on the required task). This approach was shown to be at least as powerful as MPNNs/1-WL test but strictly more expressive and able to tackle difficult graphs such as strongly regular graphs, which base GNNs cannot. Using substructures captures topological properties which enable distinguishing isomorphic graphs that are not distinguishable with 1-WL. Although this approach has theoretical and experimental improvements on benchmark datasets and in real-world scenarios, there are cases where this approach does not help, especially if small substructures are used. There is a question of how large the substructures should be - it is not always possible to reconstruct a graph from small subgraphs. On the other hand, it is computationally expensive to use very large subgraphs. This approach also requires some knowledge about the given domain in order to select substructures which are useful.

Question 2

(a) Let $G = (V, E, \mathbf{X})$ be simple, undirected graph with node features $\mathbf{X} \in \mathbb{R}^{|V| \times d}$. We will use notation $\mathbf{h}_u^{(t)}$ to denote 1-hot-encoded feature vector of node u at iteration t and $N(u)$ to denote the set of neighbors of node u.

Define $\mathbf{W}_{\text{self}}^{(t)} = [0]_{dxd}$ to be dxd null matrix and define $\mathbf{W}_{\text{neigh}}^{(t)} = I_{dxd}$ to be dxd identity matrix (with 1s on the diagonal). Define bias vector $\mathbf{b}^{(t)} = 0_{dx1}$ to be a vector of zeroes with dimensions $dx1$ and let the non-linearity σ be XOR operator.

We will show that this parametrization of the base GNN Eq1 is equivalent to function $\mathbf{f}$:

Base GNN:
$\mathbf{h}_u^{(t+1)} = \sigma\left(\mathbf{W}_{\text{self}}^{(t)}\,\mathbf{h}_u^{(t)} + \mathbf{W}_{\text{neigh}}^{(t)}\,\sum_{v \in N(u)} \mathbf{h}_v^{(t)} + \mathbf{b}^{(t)}\right)$

Base GNN after substitution of parametrization:
$\mathbf{h}_u^{(t+1)} = XOR\left([0]_{dxd}\mathbf{h}_u^{(t)} + I_{dxd}\sum_{v \in N(u)} \mathbf{h}_v^{(t)} + 0_{dx1}\right) = XOR\left(I_{dxd}\sum_{v \in N(u)} \mathbf{h}_v^{(t)}\right) = XOR\left(I_{dxd}\mathbf{c}_{dx1}^{(t)}\right) = XOR\left(\mathbf{c}_{dx1}^{(t)}\right) = \text{XOR}\left(\sum_{v \in N(u)}\sum_{v_i \in v}(v_i > 0)\right) = XOR(c * (v_i > 0)) = XOR(c)$

Dimensions check:
$\mathbf{W}_{\text{self}}^{(t)} = dxd$, $\mathbf{W}_{\text{neigh}}^{(t)} = dxd$, $\mathbf{h}_u^{(t)} = dx1$, $\mathbf{h}_v^{(t)} = dx1$, $\mathbf{b}^{(t)} = dx1$

Note the dimensions of matrices/vectors allow us to multiply $\mathbf{W}_{\text{self}}^{(t)}$ and $\mathbf{W}_{\text{neigh}}^{(t)}$ with the node features $\mathbf{h}_u^{(t)}$, sum of $\mathbf{h}_v^{(t)}$ and add the bias term $\mathbf{b}^{(t)}$ in GNN model. We set $\mathbf{W}_{\text{self}}^{(t)}$ to null matrix because we do not need the features of $\mathbf{h}_u^{(t)}$ and similarly we do not need any bias $\mathbf{b}^{(t)}$. The function only needs to count the number of neighbors of node u which can be done by summing all one-hot encoded neighbor vectors $\mathbf{h}_v^{(t)} \in N(u)$ which will give us count of 1s in every dimension and then applying multiplication with identity matrix to produce a vector of counts in every dimension. Finally we apply XOR function which returns 1 $\iff$ sum of counts is odd, otherwise it returns 0. Since we assume each node is one-hot encoded, meaning it has all 0s and a single element set to 1 (in one of the d dimensions), we know that sum of counts in each dimension and then summing over all dimensions will give us the number of neighbors of node u. Applying XOR will determine parity of the total sum and therefore parity of total number of summed neighbors. This parametrization of GNN layer with XOR function is equivalent to f because it returns 1 $\iff$ count of ones is odd $\iff$ count of neighbours of node u in $N(u)$ is odd.

(b) The parametrization considers local neighbourhood of each node (so size of graph does not matter for this parametrization) and if $G_1 = (V_1, E_1, \mathbf{X1})$ is undirected simple graph which has different number of vertices, edges and node features than G, then the parametrization should work but we need to change the dimensionality of $\mathbf{W}_{\text{self}}^{(t)}$, $\mathbf{W}_{\text{neigh}}^{(t)}$ and $\mathbf{b}^{(t)}$ to match the dimensionality of the new node feature vectors $\mathbf{X1}$. In other words, if each node $u \in V_1$ has different number of features than node $u \in V$ that we used in part a, we will need to adjust the sizes of matrices and the size of bias vector appropriately to match the new dimensionality d_1 which is the number of initial features of each node on graph G_1. Assuming the same one-hot encoding convention, after adjusting the sizes of matrices and size of bias vector we should be able to continue using

this parametrization. If we do not change the sizes then it will not work for the cases when $dim(X1) \neq dim(X)$. If however G_1 is a different type of graph (for example multigraph where we allow multiple edges between nodes) then this parametrization might not work as it will count each neighbor multiple times due to allowing multiple edges between 2 nodes (theoretically it can still work but it depends how we define neighbourhood of each node).

(c) Let $G = (V, E, \mathbf{X})$ be a graph with node features set to $x_u = [0, 1]^T$ or $x_u = [1, 0]^T$ for all $u \in V$. Let the feature vector for any vector $u \in V$ be referred to as $x_u = [u_1, u_2]^T$.

Define boolean function $g : V \to \mathbb{B}$ as the following:

$$g(x) = \begin{cases} 1, & \text{if } u \mid (u \in N(x) \text{ and } u_1 == 1) \\ 0, & \text{otherwise} \end{cases}$$

Claim: Boolean function g(x) can be represented by base GNN (Eq 1) and cannot be represented by self-loop GNN (Eq 2).

Proof: Firstly note that g(x) evaluates to 1 (true) $\iff$ there exists at least 1 node u, such that u is a neighbor of x (it cannot be x itself as x is by definition not in N(x)) and its first element u_1 is set to 1.

If we set $\mathbf{W}^{(t)}_{\text{self}} = [0, 0]$, $\mathbf{W}^{(t)}_{\text{neigh}} = [1, 0]$, $\mathbf{b}^{(t)} = [0, 0]$ and apply threshold non-linearity then we can represent g(x) using base GNN (Eq 1). This parametrization will return 1 $\iff$ there exists at least 1 neighbor of node x which has its first element $u_1 = 1$ because it is the only way how to get a value which is strictly greater than 0 for the threshold non-linearity to return 1. Since there is only 1 non-zero weight in the parametrization, in $\mathbf{W}^{(t)}_{\text{neigh}}$, we ensure that this parametrized base GNN and g(x) are equivalent.

In contrast, we cannot represent g(x) with base self-loop GNN because there is no way to set $\mathbf{W}^{(t)}$ such that it would differentiate whether there exists a node u which is not x itself such that u is a neighbor of x and has the first element $u_1 = 1$. Since there is no way to distinguish between node x (itself) and its neighbors u (since in self-loop model they were all merged into 1 set), the function would evaluate incorrectly to 1 if there was no such neighbor u which would have $u_1 = 1$ but instead x itself had the first element $x_1 = 1$. Self-loop GNN cannot distinguish between node x and its neighbours so g(x) cannot be represented by the base self-loop GNN model (Eq 2). We can provide a concrete counterexample. If we set $\mathbf{W}^{(t)} = [1, 0]$ in Eq2, this parametrization will work for all cases except for the case where all neighbors u of node x s.t. $u \in N(x)$ have $u_1 = 0$ but node x itself has $x_1 = 1$. In this case g(x) evaluates to 1 even though it should evaluate to 0 because none of the neighbors satisfy the required condition. We can avoid this problem (of counting node x) only if we set $\mathbf{W}^{(t)} = [0, 0]$ but in this case Eq(2) will always output 0 (since all weights are 0) and therefore it will not be equivalent to g(x). Therefore, there is no way how to represent g(x) by the base self-loop GNN.

(d) We will show that function f(u) can be expressed as a logical formula $\Phi(u)$ in graded modal logic and show such formula. Then we will argue that $\Phi(u)$ is graded modal logic classifier $\implies$ $\Phi(u)$ can be captured by a base GNN (Eq 1).

Recall from lecture 12 that FOC_2 is 2 variable fragment of FO which in addition to FO allows counting quantifiers. There is a connection between FOC_2 and WL:

Firstly, note that FOC_2 is a subset of FO (since counting can be mimicked in FO by introducing more variables) $\implies$ FOC_2 classifiers are logical classifiers.

Secondly, for any graph G and nodes $u, v \in G$, the WL test colours u and v the same after any number of rounds $\iff$ u and v are classified the same by all FOC_2 classifiers. We will further restrict FOC_2 to a subset called graded modal logic guarded by Edge(x,y). This ensures that all subformulas are guarded by edge predicate and this will allow us to ensure locality property which is essential condition for the ability to represent subformulas by base GNNs.

We will now show that we can represent f(u) in graded modal logic: A vertex u satisfies f(u) if and only if u has at most 2 neighbours that have degree at least 3:

$\Phi(u) = \neg\exists^{\geq 3} y \left(E(u, y) \wedge \exists^{\geq 3} x E(y, x)\right)$.

We have a formula above which is a graded modal logic classifier. Barceló et al in their work (see references for more details about the paper) showed that each graded modal logic classifier can be captured by a base GNN (Eq 1) where each of the vector's elements represents one subformula of the captured classifier. Thus, if a feature in a node is 1 then the node satisfies the corresponding sub-formula. The features corresponding to subformulas will get updated in every iteration and after k iterations (and passing through k layers), the latest component in every node will be 1 $\iff$ given node satisfies subformula. We then combine all elements and output final classification after applying a non-linearity.

Formally, the procedure which will allow us to simulate the formula Φ with a GNN is:
1. Enumerate all subformulas
2. Ensure that feature vectors are constructed in such a way that every component represents 1 subformula
3. Every component corresponding to a subformula gets value 1 $\iff$ the subformula is satisfied in node u
4. Ensure the number of iterations (number of layers) is sufficient to cover the scope of quantifiers (i.e. for $\Phi(u)$ we need at least 4 layers if we count input/output layers)

If we set initial node features to $\mathbf{1}^d$ we can use the steps outlined above to simulate f(u) using the base GNN (Eq 1) model as required.

(e) Define boolean function $g : V \to \mathbb{B}$ as the following:

$$g(u) = \begin{cases} 1, & \text{if } y \mid (y \notin N(u) \wedge |N(y)| \geq 5) \\ 0, & \text{otherwise} \end{cases}$$

Claim: Boolean function g cannot be represented by the base GNN model (Eq 1) on a graph $G = (V, E, \mathbf{X})$ regardless of whether we set the initial node features to $\mathbf{1}^d$ or whether we use 1-hot encoded initial features.

Proof/reasoning:
We can tranfsorm fucntion g to a formula in FO_2 logic:

$\Omega(u) = \exists y \left(\neg E(u, y) \wedge \exists^{\geq 5} u(E(y, u))\right)$

Following-up from Q2d, consider a similar boolean classifier formula $\Omega(u)$, which represents our defined function g. Observe the difference between $\Phi(u)$ (from 2d) and $\Omega(u)$ is that $\Omega(u)$ is not

guarded by Edge(x,y) $\implies$ this formula is in FO_2 logic but not in graded modal logic $\implies$ boolean classifier $\Omega(u)$ cannot be captured by the base GNN model (Eq 1) without global readout.

$\Omega(u)$ captures non-local properties (since there is no edge between u and y) so with the base GNN (without global readout) it cannot be represented. It does not matter whether we initialise the features as $\mathbf{1}^d$ or use one-hot encoded initialisation because if there is no edge there is no communication between the nodes. The information about neighbourhood of node y might never reach node u since there is no edge between them. Only if we allow an extra term for a global feature computation on each layer of the GNN which allows us to aggregate current feature vectors of all the nodes in a graph (not only neighbours) then we can express such a formula (this reasoning is supported by theory outlined in paper from Barceló et al, please see references for more detail).

Question 3

Research question: Does stochastic scaling of node features and gradients, as a form of regularisation, help to improve performance of GAT networks?

Motivation: Multi-head attention mechanism in GAT networks potentially introduces a form of redundancy which is evidenced by the fact that GAT networks often learn similar distributions of attention coefficients and if multiple heads are used and/or multiple layers, GAT networks are more likely to over-fit to the training data [11]. In addition, if deeper architectures are used, there is a risk of oversmoothing due to convergence of node features [1]. Both of these problems hinder performance. We want to investigate stochastic scaling of features and gradients as a form of regularisation which could address these problems. Stochastic regularisation scales node features in each layer by a scalar factor sampled from beta distribution (in forward pass) and scales gradients during backward propagation. Our hypothesis is that this approach should tackle over-fitting and break feature convergence (oversmoothing) by introducing randomness. We will explore whether GAT networks achieve better performance if we introduce this form of regularisation.

Approach and goals: We design a set of experiments in which we compare the performance of base GAT network without regularisation vs. the performance of GAT network with stochastic regularisation in a form of: Scaling node features (SR-F), Scaling gradients (SR-G), Combining scaling node features and gradients (SR-C).

Our setup includes experiments which test the hypothesis by comparing behaviour and performance of models (Base GAT, GAT SR-F, GAT SR-G, GAT SR-C) by looking at different aspects of regularisation on 3 benchmark datasets: Cora, CiteSeer and PubMed.

Our goal is to determine whether stochastic scaling improves performance of GAT networks and whether it helps to alleviate over-fitting and oversmoothing when tested in various scenarios.

Methodology: We conduct 3 different experiments, each focusing on one aspect of observing how robust the GAT models are when we introduce regularisation. Each of the experiments is run on 3 datasets and compares performance of 1 base GAT and 3 regularised GAT models. All experiments are run 100 times and we show the average performance across 100 runs.

Hyperparameters:
epochs: 200 (EX1), 50 (EX2,EX3) || repeats: 100 || layers: 2 (for EX1,EX3) || hidden units: 8 || heads:[8,1] || learning rate: 5e-3 || lambda = 0.05 (for EX2,EX3)

Experiment 1: Lambda parameter
We compare the performance of base GAT network to 3 regularised GAT models. We vary lambda (the hyperparameter used to draw a factor from beta distribution which is then used for scaling) and measure its effect on performance.

Experiment 2: Number of layers
We compare the performance of base GAT network to 3 regularised GAT models and observe behaviour as we increase the number of GAT layers.

Experiment 3: Introduction of noise We compare the performance of base GAT network to 3 regularised GAT models in a setup that introduces noise. We introduce noise to the datasets by randomly adding new edges. We measure the performance of models at different levels of noise.

Assumptions: We fix hyperparameters for all models such that in each experiment we only vary 1 independent variable. Everything else constant, we assume that any differences in performance between base GAT and regularised GAT must be due to introduced regularisation. Before starting

experiments, all hyperparameters were set/tuned to reasonable values and fixed - we use similar hyperparameters as were used in Graph Attetion Networks paper (Veličković et al 2018). In models we did not implement dropout or early stopping which are a form of regularisation as it could interfere with stochastic regularisation. We only apply scaling during training (not evaluation).

Results

Experiment 1:

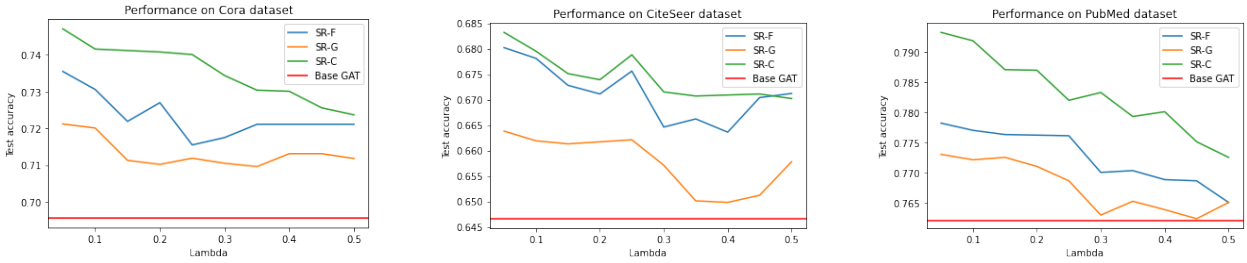

Figure 1: Performance of SR-F, SR-G and SR-C for different lambda parameter vs. base GAT

3 regularised models show gains in test accuracy compared to base GAT (red line) as we vary the regularisation parameter lambda. As we increase lambda, the performance tends to decrease, although it does not decrease below base GAT (even for very large lambda > 100). After grid-search for best range of lambda we report values 0.05-0.5 as the most optimal range. SR-C (scaling features and gradients) performs the best, SR-F (scaling features) performs the second best and SR-G (scaling gradients) performs the worst, although all 3 approaches improve the base model.

Experiment 2:

n. layers	Base GAT	SR-F	SR-G	SR-C
2	0.7912	0.8151	0.8121	0.8164
4	0.7817	0.7930	0.7880	0.7992
8	0.7467	0.7606	0.7498	0.7677
12	0.7396	0.7560	0.7540	0.7666
16	0.6322	0.7488	0.7204	0.7476
20	0.2513	0.2899	0.2862	0.2888
24	0.1591	0.2664	0.2564	0.2732

(a) Cora

n. layers	Base GAT	SR-F	SR-G	SR-C
2	0.6968	0.6996	0.7002	0.7048
4	0.6582	0.6658	0.6666	0.6628
8	0.5892	0.6164	0.6143	0.6364
12	0.5976	0.5992	0.5918	0.6042
16	0.5548	0.5946	0.5946	0.6082
20	0.3190	0.5780	0.5799	0.5823
24	0.2540	0.3892	0.3850	0.3962

(b) CiteSeer

n. layers	Base GAT	SR-F	SR-G	SR-C
2	0.7746	0.7790	0.7782	0.7738
4	0.7484	0.7528	0.7544	0.7574
8	0.7425	0.7423	0.7424	0.7462
12	0.7464	0.7596	0.7552	0.7580
16	0.6754	0.7366	0.7402	0.7516
20	0.4349	0.5312	0.5276	0.5338
24	0.3832	0.4750	0.4888	0.4723

(c) PubMed

Table 1: Model performance with increasing number of layers

With increasing number of layers, performance decreases across all models as expected due to over-fitting and oversmoothing [2]. The results show that regularised models are able to maintain higher test accuracy than base GAT even with increasing number of layers. In particular, for the model with 24 layers, all 3 forms of regularisation were able to perform by more than 10% better than base GAT model. This suggests that stochastic scaling might help to reduce oversmoothing. The best overall performance across models is achieved by SR-C model. The second and third best by SR-F and SR-G model respectively. This suggests that scaling the node features is more important than scaling gradients when attempting to break node feature convergence.

Experiment 3:

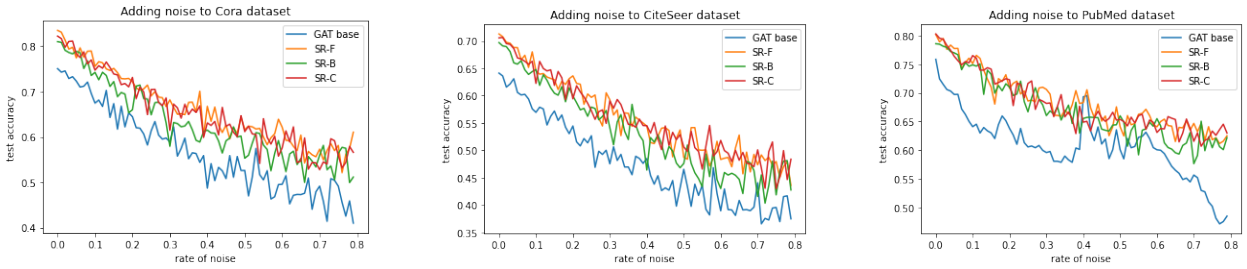

Figure 2: Comparison of performance of SR-F, SR-G and SR-C in the presence of noise

With introduction of noise, the classification accuracy of all models decreases but regularisation helps in the presence of noise - this is a sign that the regularised models over-fit less to the training data due to their better generalisation capabilities. On PubMed, we see that after reaching noise 0.6+, the performance of base GAT worsens significantly (< 0.5) while the regularised models maintain accuracy around 0.65 even at large levels of noise. This suggests that stochastic regularisation helps to alleviate over-fitting by making GAT models more robust and adaptable to noise.

Discussion: The results show that all 3 regularised models implementing stochastic scaling improve performance compared to base GAT model on the benchmark datasets. We were able to achieve lower rate of performance degradation with increasing number of layers which suggests that this approach alleviates oversmoothing. This is a promising result because there are many tasks with large problem radius requiring use of several layers but in literature many state-of-the-art GAT models only use small number of layers due to otherwise reduced performance because of oversmoothing [2]. We were able to achieve lower rate of performance degradation with increasing noise thus alleviating over-fitting - this is important as most of the real-life datasets, unlike benchmark datasets, contain noise and therefore it is advantageous that models can learn in the presence of noise. The study shows experimental positive results fully supporting our hypothesis as the improvements in performance were significant compared to the standard deviations in test accuracies which were recorded across 100 runs.

Although the study shows performance improvement, we are not able to provide a proof that the improvement is due to addressing the oversmoothing issue - for this we would need to use some quantitative measure of oversmoothing rather than measuing performance. In addition, overfitting in graph neural networks often happens due to over-parametrization. Unfortunately, our regularisation technique introduces another parameter - lambda, which needs to be tuned. This is problematic because the most optimal lambda will vary depending on the model, number of layers, type of dataset, etc. Since we are sampling from transformed beta distribution there is lack of interpretability of how lambda parameter should be tuned and what effect it has on the model [12].

Additional studies: Firstly, we could run the same set of experiments and extend the comparison to different types of regularized models (i.e. dropout, DropEdge, early stopping, weight decay) and observe how those methods perform in comparison to stochastic scaling. Secondly, we could measure if there is less oversmoothing by using measure such as information-to-noise ratio or group distance measure to make more convincing argument about oversmoothing. Thirdly, we could do more exploration into interpretability of hyperparameter lambda by deriving theoretical results.

Link to repo: https://anonymous.4open.science/r/GRL-EXAM-TASK3-REPO-38ED/

References

[1] Q1: Zhao Lingxiao and Akoglu Leman, PairNorm: Tackling Oversmoothing in GNNs, 2019

[2] Q1: Chen Deli and Lin Yankai and Li Wei and Li Peng and Zhou Jie and Sun Xu, Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View, 2019

[3] Q1: Zhou Kaixiong and Huang Xiao and Li Yuening and Zha Daochen and Chen Rui and Hu Xia, Towards Deeper Graph Neural Networks with Differentiable Group Normalization, 2020

[4] Q1: Topping Jake and Di Giovanni Francesco and Chamberlain Benjamin Paul and Dong Xiaowen and Bronstein Michael M., Understanding over-squashing and bottlenecks on graphs via curvature, 2021

[5] Q1: Michael Bronstein, Over-squashing, Bottlenecks, and Graph Ricci curvature, 2021 (published in Towards Data Science)

[6] Q1: Bouritsas Giorgos and Frasca Fabrizio and Zafeiriou Stefanos and Bronstein Michael M, Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting, 2020

[7] Q1: GRL Lecture slides 10,11,12

[8] Q2: Pablo Barceló and Egor V. Kostylev and Mikael Monet and Jorge Pérez and Juan Reutter and Juan Pablo Silva, The Logical Expressiveness of Graph Neural Networks, International Conference on Learning Representations, 2020

[9] Q2: C. Morris, M. Ritzert, M. Fey, W. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks, 2019

[10] Q2: GRL Lecture slides 13

[11] Q3: Veličković Petar and Cucurull Guillem and Casanova Arantxa and Romero Adriana and Liò Pietro and Bengio Yoshua, Graph Attention Networks, 2018

[12] Q3: Haimin Zhang and Min Xu and Guoqiang Zhang and Kenta Niwa, SSFG: Stochastically Scaling Features and Gradients for Regularizing Graph Convolutional Networks, 2021

[13] Q3: GRL Lecture slides 8