Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Λογικής και Επιστήμης Υπολογισμών (Co.Re.Lab)

Efficient Approaches to Deal with Oversmoothing in Deep Graph Neural Networks

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

ΤΟΥ

ΔΗΜΗΤΡΙΟΥ Κ. ΚΕΛΕΣΗ

Αθήνα, Οκτώβριος 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Λογικής και Επιστήμης Υπολογισμών (Co.Re.Lab)

Efficient Approaches to Deal with Oversmoothing in Deep Graph Neural Networks

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

ΤΟΥ

ΔΗΜΗΤΡΙΟΥ Κ. ΚΕΛΕΣΗ

Συμβουλευτική επιτροπή : Δημήτριος Φωτάκης (Καθηγητής, Ε.Μ.Π.)
Γεώργιος Παλιούρας (Ερευνητής Α', Ε.Κ.Ε.Φ.Ε. "Δημόκριτος")
Γεώργιος Στάμου (Καθηγητής, Ε.Μ.Π.)

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 8η Οκτωβρίου 2025.

.....	.....	.....
Δημήτριος Φωτάκης Καθηγητής, Ε.Μ.Π.	Γεώργιος Παλιούρας Ερευνητής Α', Ε.Κ.Ε.Φ.Ε. "Δημόκριτος"	Γεώργιος Στάμου Καθηγητής, Ε.Μ.Π.
.....	.....	.....
Αθανάσιος Βουλόδημος Επ. Καθηγητής, Ε.Μ.Π.	Ιωάννης Νικολέντζος Επ. Καθηγητής, Παν. Πελοποννήσσου	Αθανάσιος Ροντογιάννης Αναπλ. Καθηγητής, Ε.Μ.Π.

.....

Αλέξανδρος Ποταμιάνος
Αναπλ. Καθηγητής, Ε.Μ.Π.

Δημήτριος Κ. Κελέσης
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Στους γονείς μου.

Περίληψη. Τα Νευρωνικά Δίκτυα Γράφων (Graph Neural Networks - GNNs) έχουν επιδείξει εντυπωσιακή επιτυχία σε ένα ευρύ φάσμα εργασιών που αφορούν σχεσιακά δεδομένα. Ωστόσο, η απόδοσή τους επιδεινώνεται όσο αυξάνεται το βάθος τους, λόγω του φαινομένου της υπερ-εξομάλυνσης (oversmoothing), κατά το οποίο οι αναπαραστάσεις των κόμβων καθίστανται σχεδόν πανομοιότυπες μεταξύ των επιπέδων. Η παρούσα διατριβή παρουσιάζει μια συστηματική μελέτη του προβλήματος της υπερ-εξομάλυνσης και προτείνει νέες θεωρητικές και εμπειρικές προσεγγίσεις για την αντιμετώπισή του, καθιστώντας δυνατή τη σχεδίαση βαθύτερων και εκφραστικότερων αρχιτεκτονικών GNN.
Πρώτον, εισάγουμε μία νέα μετρική σε κάθε επίπεδο (layer-wise) για τη μέτρηση της υπερ-εξομάλυνσης, συνοδευόμενη από θεωρητικά όρια και εργαλεία πρακτικής ανίχνευσης [1]. Δείχνουμε ότι η υπερ-εξομάλυνση επιδεινώνεται όταν ο αριθμός των πινάκων βαρών συσχετίζεται με το βάθος της διάδοσης μηνυμάτων, και προτείνουμε το G-Reg, μια στρατηγική κανονικοποίησης που διατηρεί την ποικιλομορφία των αναπαραστάσεων.
Στη συνέχεια, μελετούμε τις residual συνδέσεις και αναλύουμε τους περιορισμούς τους στην υποστήριξη αλληλεπιδράσεων μακρινής εμβέλειας μεταξύ κόμβων. Η ανάλυσή μας δείχνει ότι, παρόλο που τα μοντέλα με residuals (π.χ. APPNP, GCNII) αντιστέκονται στην υπερ-εξομάλυνση σε τυπικά benchmarks, αποτυγχάνουν σε σενάρια που απαιτούν βαθιά και εκφραστική διάδοση πληροφορίας [2]. Για την ανάδειξη αυτού, εισάγουμε ένα συνθετικό σύνολο δεδομένων σχεδιασμένο να αξιολογεί την ικανότητα ενός GNN να αποτυπώνει μακρινές εξαρτήσεις.
Έπειτα, εξετάζουμε τη μερική εκπαίδευση (partial training) σε GNNs, όπου εκπαιδεύεται μόνο ένα επίπεδο ενώ τα υπόλοιπα παραμένουν σταθερά. Τα αποτελέσματά μας αποκαλύπτουν ότι η αύξηση του πλάτους του μοντέλου αντισταθμίζει την απουσία πλήρους εκπαίδευσης και μειώνει σημαντικά την υπερ-εξομάλυνση, ακόμη και σε βαθιές αρχιτεκτονικές. Η προσέγγιση αυτή ισοφαρίζει ή ξεπερνά πλήρως εκπαιδευμένα μοντέλα τόσο σε τυπικά όσο και σε ”cold start” σενάρια [3].
Επιπρόσθετα, προτείνουμε τη μέθοδο G-Init, μια στρατηγική αρχικοποίησης βαρών ενημερωμένη από τη δομή του γράφου, εμπνευσμένη από κλασικές τεχνικές αρχικοποίησης για βαθιά νευρωνικά δίκτυα [4]. Η G-Init λαμβάνει υπόψη την τοπολογία του γράφου και βελτιώνει τη ροή των gradients σε βαθιά GNNs, μειώνοντας την υπερ-εξομάλυνση και ενισχύοντας την επίδοση τους σε προβλήματα ταξινόμησης.
Τέλος, διερευνούμε την επίδραση της συνάρτησης ενεργοποίησης στην υπερ-εξομάλυνση. Τα θεωρητικά και εμπειρικά ευρήματά μας δείχνουν ότι η τροποποίηση της κλίσης της ReLU οδηγεί σε καλύτερη ποικιλομορφία αναπαραστάσεων και βελτιωμένη απόδοση σε βαθιά GNNs, χωρίς να απαιτούνται αλλαγές στην αρχιτεκτονική ή residual συνδέσεις [5].
Συνολικά, οι συνεισφορές αυτές προωθούν την κατανόησή μας σχετικά με τις προκλήσεις που σχετίζονται με το βάθος στα GNNs και προσφέρουν πολλαπλές, επεκτάσιμες και θεωρητικά τεκμηριωμένες λύσεις για την αντιμετώπιση της υπερ-εξομάλυνσης. Τα ευρήματα υποδεικνύουν την ανάγκη επαναπροσδιορισμού των αρχών σχεδίασης των βαθιών GNNs και ανοίγουν τον δρόμο για πιο αξιόπιστες αρχιτεκτονικές κατάλληλες για πραγματικές εφαρμογές.

Λέξεις κλειδιά: Νευρωνικά Δίκτυα Γράφων, Υπερ-εξομάλυνση

Abstract. Graph Neural Networks (GNNs) have achieved remarkable success in a wide range of tasks involving relational data, yet their performance deteriorates as depth increases due to the phenomenon known as oversmoothing, where node representations become indistinguishable across layers. This thesis presents a systematic investigation into the oversmoothing problem and proposes novel theoretical and empirical approaches for mitigating it, thus enabling deeper and more expressive GNN architectures.

We first introduce a novel layer-wise metric to quantify oversmoothing, providing theoretical bounds and practical detection tools [1]. We show that oversmoothing is exacerbated when the number of weight matrices is coupled with the depth of message passing, and propose G-Reg, a regularization strategy that preserves representational diversity.

Next, we study residual connections and analyze their limitations in enabling long-range node interactions. Our analysis shows that while residual-based models (e.g., APPNP, GCNII) resist oversmoothing on standard benchmarks, they fail in settings requiring deep and expressive propagation [2]. To highlight this, we introduce a synthetic dataset tailored to evaluate the capability of a GNN to capture long-range dependencies.

We then explore partial training in GNNs, where only a single layer is trained while others remain fixed. Our results reveal that increasing model width counteracts the lack of full training and significantly reduces oversmoothing, even in deep architectures. This approach matches or outperforms fully trained models in both standard and "cold start" scenarios [3].

Building on this, we propose G-Init, a graph informed weight initialization strategy inspired by classical deep learning initialization techniques [4]. G-Init accounts for graph topology and improves gradient flow in deep GNNs, reducing oversmoothing and enhancing classification performance across tasks.

Finally, we investigate the impact of the activation function on oversmoothing. Our theoretical and empirical findings demonstrate that modifying the slope of ReLU leads to better representational diversity and improved performance in deep GNNs, without employing architectural changes or residual connections [5].

Together, these contributions advance our understanding of depth-related challenges in GNNs and offer multiple scalable, theoretically grounded solutions to overcome oversmoothing. The findings support a rethinking of GNN design principles and pave the way for more robust architectures suited to real-world problems.

Keywords: Graph Neural Networks, Oversmoothing

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά το Γιώργο Παλιούρα για τη συνεργασία μας όλα αυτά τα χρόνια, καθώς και για την πολύτιμη επιστημονική του συνεισφορά. Η εμπιστοσύνη που μου έδειξε, καθώς και η ελευθερία κινήσεων και πρωτοβουλιών που μου παρείχε, συνέβαλαν καθοριστικά στην επίτευξη των αποτελεσμάτων της παρούσας διατριβής. Μέσα από αυτή τη συνεργασία διαμόρφωσα τον ερευνητικό μου χαρακτήρα και συνειδητοποίησα την αξία της ποιοτικής και υψηλού επιπέδου έρευνας. Επίσης, ευχαριστώ θερμά το Δημήτρη Φωτάκη για τη σημαντική επιστημονική του συνεισφορά, αλλά και για τις συμβουλές και προτροπές του σχετικά με την αντιμετώπιση της έρευνας και των αποτελεσμάτων αυτής. Τα παραπάνω δύο πρόσωπα έπαιξαν καθοριστικό ρόλο στην εκπόνηση της παρούσας διατριβής.

Θα ήθελα ακόμη να ευχαριστήσω τους φίλους μου για την υπομονή και την στήριξη τους αυτά τα χρόνια (αναφέρονται με τυχαία σειρά): Γιώργος Γ., Έλενα Θ., Χάρης Π., Γιώργος Γ., Διονύσης Σ., Δανάη Ε., Γραμματική Τ., Φωτεινή Κ., Δήμητρα Κ.

Επιπλέον, θέλω να ευχαριστήσω και τα παιδιά με τα οποία συμπορευτήκαμε στο Ε.Κ.Ε.Φ.Ε. "Δημόκριτος": Ηλίας Ζ., Κωνσταντίνος Μ., Άρτεμις Δ., Νίκος Μ., Ανδρέας Σ., Βασίλης Μ., Ελίνα Σ., Βασίλης Γ., Γιώργος Γ., Χριστόφορος Ρ., Παναγιώτης Κ.

Θα ήθελα επίσης να ευχαριστήσω τον παππού μου Σπύρο και τις γιαγιάδες μου Μαρία και Σταματία για την διαρκή και απλόχερη προσφορά τους καθ'όλη τη διάρκεια της ζωής μου τόσο στον υλιστικό τομέα, αλλά κυρίως στον ηθικοπλαστικό.
Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου Κωνσταντίνο και Παναγιώτα, αλλά και την αδελφή μου Μαρία για τη διαρκή στήριξη και τον καθοριστικό ρόλο που έπαιξαν στη διαμόρφωση του ανθρώπου που είμαι σήμερα.

Δημήτρης Κελέσης
Αθήνα, Οκτώβριος 2025

Contents

Κεφάλαιο I

Εκτενής Περίληψη στα Ελληνικά

I Εισαγωγή

I.I Νευρωνικά Δίκτυα Γράφων (Graph Neural Networks - GNNs)

Τα Νευρωνικά Δίκτυα Γράφων (GNNs) έχουν αναδειχθεί ως ο ακρογωνιαίος λίθος για την ανάλυση και την εκμάθηση από δεδομένα δομημένα σε μορφή γράφου. Η ικανότητά τους να συνδυάζουν τα χαρακτηριστικά των κόμβων με την εγγενή συνδεσιμότητα των γράφων έχει οδηγήσει σε αξιοσημείωτες προόδους σε πολλούς τομείς. Τα GNNs αποτελούν μια κατηγορία νευρωνικών δικτύων ειδικά σχεδιασμένων για την επεξεργασία δεδομένων που μπορούν να αναπαρασταθούν ως γράφοι. Σε έναν γράφο, οι οντότητες μοντελοποιούνται ως κόμβοι και οι σχέσεις μεταξύ τους ως ακμές.

Τυπικά, έστω $G = (V, E, X)$ ένας μη κατευθυνόμενος γράφος, με $N = |V|$ κόμβους $u_i \in V$, ακμές $(u_i, u_j) \in E$ και $X = [x_1, \ldots, x_N]^T \in \mathbb{R}^{N \times C}$ τα αρχικά χαρακτηριστικά των κόμβων. Οι ακμές σχηματίζουν έναν πίνακα γειτνίασης $A \in \mathbb{R}^{N \times N}$, όπου το στοιχείο $A_{i,j}$ συσχετίζεται με την ακμή (u_i, u_j). Κατά τη φάση της εκπαίδευσης, είναι προσβάσιμες μόνο οι ετικέτες ενός υποσυνόλου $V_l \subset V$. Ο στόχος είναι η ανάπτυξη ενός ταξινομητή που αξιοποιεί την τοπολογία του γράφου και τα παρεχόμενα διανύσματα χαρακτηριστικών για να κάνει προβλέψεις. Τα GNNs μαθαίνουν μια συνάρτηση $F : G \to Z$, όπου $Z \in \mathbb{R}^{N \times m}$ είναι ο πίνακας των τελικών αναπαραστάσεων (embeddings).

I.I.1 Το Πλαίσιο Διάδοσης Μηνυμάτων (Message-Passing Framework)

Το κυρίαρχο πλαίσιο για τα GNNs είναι το πλαίσιο διάδοσης μηνυμάτων (ή συνάθροισης γειτονιάς). Ένα GNN αποτελείται από L επίπεδα (layers), όπου στο επίπεδο l κάθε κόμβος u διατηρεί μια κρυφή αναπαράσταση $h_u^{(l)} \in \mathbb{R}^{d_l}$. Η ενημέρωση από το επίπεδο $l-1$ στο επίπεδο l περιλαμβάνει δύο βήματα:

1. **Συνάθροιση (Aggregation)**: Κάθε κόμβος συλλέγει πληροφορίες από τους γείτονές του. Έστω $N(u)$ το σύνολο των γειτόνων του κόμβου u.

$$m_u^{(l)} = \text{AGGREGATE}^{(l)}\{h_v^{(l-1)} : v \in N(u)\}$$

   Η συνάρτηση $\text{AGGREGATE}^{(l)}$ είναι συνήθως αμετάβλητη ως προς τις μεταθέσεις (permutation-invariant), όπως το άθροισμα, ο μέσος όρος ή ένας συνδυασμός με βάση την προσοχή (attention), διασφαλίζοντας ότι η έξοδος δεν εξαρτάται από την αυθαίρετη σειρά των γειτόνων.

2. **Συνδυασμός (Combination)**: Το συναθροισμένο μήνυμα συνδυάζεται με την προηγούμενη αναπαράσταση του κόμβου για να παραχθεί η νέα κρυφή αναπαράσταση:

$$h_u^{(l)} = \text{COMBINE}^{(l)}(h_u^{(l-1)}, m_u^{(l)})$$

Στην πράξη, η $\text{COMBINE}^{(l)}$ αποτελείται συχνά από έναν γραμμικό μετασχηματισμό (μέσω ενός πίνακα βαρών) ακολουθούμενο από μια μη γραμμική συνάρτηση ενεργοποίησης (π.χ., ReLU).

Μετά από L τέτοιες επαναλήψεις διάδοσης μηνυμάτων, κάθε κόμβος u αποκτά μια τελική αναπαράσταση $h_u^{(L)}$.

I.I.2 Εκφραστική Ισχύς και Περιορισμοί

Η αναπαραστατική ικανότητα των GNNs που βασίζονται στη διάδοση μηνυμάτων έχει αναλυθεί μέσω του τεστ ισομορφισμού γράφων Weisfeiler–Lehman (WL). Έχει αποδειχθεί ότι αν δύο μη ισομορφικοί γράφοι δεν μπορούν να διακριθούν από το 1-WL τεστ, τότε κανένα τέτοιο GNN δεν μπορεί να τους αντιστοιχίσει διαφορετικές αναπαραστάσεις [6]. Ωστόσο, καθώς τα GNNs γίνονται βαθύτερα, αντιμετωπίζουν σοβαρούς περιορισμούς, όπως:

- **Υπερ-εξομάλυνση (Oversmoothing)**: Οι αναπαραστάσεις των κόμβων συγκλίνουν σε παρόμοιες τιμές, ανεξάρτητα από τις δομικές διαφορές, οδηγώντας σε απώλεια διακριτικότητας.
- **Υπερ-συμπίεση (Oversquashing)**: Η ροή πληροφορίας από απομακρυσμένους κόμβους παρεμποδίζεται, καθώς τα σήματα από εκθετικά επεκτεινόμενες γειτονιές διοχετεύονται μέσω περιορισμένης χωρητικότητας διαδρομών.

I.II Βαθιά Νευρωνικά Δίκτυα Γράφων (Deep GNNs)

Τα Βαθιά Νευρωνικά Δίκτυα Γράφων στοχεύουν στην ενίσχυση της εκφραστικής ισχύος των τυπικών αρχιτεκτονικών μέσω της στοίβαξης πολλαπλών επιπέδων διάδοσης μηνυμάτων. Κάθε επίπεδο GNN επιτρέπει σε έναν κόμβο να συναθροίζει και να μετασχηματίζει πληροφορίες από τους άμεσους γείτονές του, χτίζοντας σταδιακά μια αναπαράσταση που αποτυπώνει τόσο τα χαρακτηριστικά του κόμβου όσο και το τοπικό δομικό του πλαίσιο.

I.II.1 Κίνητρα για Βάθος

Το κίνητρο για τη χρήση βαθιών GNNs πηγάζει από την ανάγκη ενσωμάτωσης πληροφοριών από απομακρυσμένα μέρη του γράφου. Σε πολλές εφαρμογές, όπως η πρόβλεψη μοριακών ιδιοτήτων ή η ανάλυση κοινωνικών δικτύων, η υποκείμενη σημασιολογία εξαρτάται από εξαρτήσεις μεγάλης εμβέλειας (long-range dependencies). Τα ρηχά GNNs, περιορισμένα σε ένα ή δύο «άλματα» (hops), είναι ανεπαρκή για τη σύλληψη τέτοιων πολύπλοκων, μη-τοπικών εξαρτήσεων, οδηγώντας στο πρόβλημα της υπο-προσέγγισης (underreach), όπου το μοντέλο δεν έχει πρόσβαση στην κατάλληλη πληροφορία.
Επιπλέον, από θεωρητική άποψη, τα βαθύτερα GNNs είναι πιο εκφραστικά και ικανά να διακρίνουν διαφορετικές δομές γράφων, καθώς μπορούν να συλλάβουν μοτίβα που βρίσκονται πέρα από την αναπαραστατική ικανότητα των ρηχότερων μοντέλων.

I.II.2 Προκλήσεις των Βαθιών GNNs

Παρά τα θεωρητικά τους πλεονεκτήματα, τα βαθιά GNNs αντιμετωπίζουν μια σειρά από κρίσιμους περιορισμούς που εμποδίζουν την πρακτική τους χρηστικότητα. Πέρα από ένα ορισμένο βάθος, η απόδοση συχνά επιδεινώνεται αντί να βελτιώνεται. Αυτή η υποβάθμιση αποδίδεται σε διάφορα φαινόμενα:

1. **Υπερ-εξομάλυνση (Oversmoothing)**: Καθώς αυξάνεται το βάθος, οι αναπαραστάσεις των κόμβων τείνουν να συγκλίνουν σε ένα σταθερό σημείο, οδηγώντας σε απώλεια της ικανότητας του μοντέλου να διακρίνει διαφορετικές κλάσεις. Αυτό αποτελεί το κύριο εμπόδιο στην κατασκευή αποτελεσματικών βαθιών GNNs [7].

2. **Υπερ-συμπίεση (Oversquashing)**: Το φαινόμενο αυτό εμποδίζει την ακριβή μετάδοση εξαρτήσεων μεγάλης εμβέλειας, καθώς τα σήματα από απομακρυσμένους κόμβους παρεμβάλλονται ή χάνονται εντελώς λόγω του φαινομένου της «συμφόρησης» (bottleneck effect) [8].

3. **Εξαφάνιση και Έκρηξη Κλίσεων (Vanishing and Exploding Gradients)**: Όπως και σε άλλες βαθιές νευρωνικές αρχιτεκτονικές, η εκπαίδευση γίνεται δύσκολη λόγω ασταθών κλίσεων, ένα πρόβλημα που επιδεινώνεται από τις μη γραμμικές συναρτήσεις ενεργοποίησης και τα επίπεδα κανονικοποίησης.

4. **Ομογενοποίηση Κλίσεων (Gradient Homogenization)**: Οι κλίσεις που σχετίζονται με διαφορετικούς κόμβους γίνονται όλο και πιο παρόμοιες, περιορίζοντας την ικανότητα του μοντέλου να εστιάζει σε χρήσιμες πληροφορίες.

5. **Απώλεια Τοπικότητας (Loss of Locality)**: Ενώ τα βαθιά δίκτυα συναθροίζουν περισσότερη παγκόσμια πληροφορία, μπορεί να χάσουν την ευαισθησία τους σε σημαντικά τοπικά χαρακτηριστικά, καθώς τα μηνύματα από απομακρυσμένους κόμβους κυριαρχούν στην αναπαράσταση.

6. **Περιορισμοί Κλιμάκωσης και Πόρων (Scalability and Resource Constraints)**: Τα βαθύτερα GNNs συνεπάγονται υψηλότερο υπολογιστικό κόστος και κόστος μνήμης, ειδικά σε γράφους μεγάλης κλίμακας.

I.III Υπερ-εξομάλυνση (Oversmoothing): Ορισμός και Σημασία

Η υπερ-εξομάλυνση αναφέρεται στο φαινόμενο κατά το οποίο οι επαναλαμβανόμενες λειτουργίες διάδοσης μηνυμάτων ή συνάθροισης σε ένα GNN οδηγούν τις αναπαραστάσεις χαρακτηριστικών των κόμβων να γίνονται όλο και πιο παρόμοιες ή ακόμα και μη διακριτές σε όλο τον γράφο.
Τυπικά, έστω $H^{(l)} \in \mathbb{R}^{N \times d_l}$ ο πίνακας των αναπαραστάσεων των κόμβων μετά από l επίπεδα ενός GNN. Σε πολλές αρχιτεκτονικές GNN, ο κανόνας ενημέρωσης μπορεί να εκφραστεί ως:

$$H^{(l)} = \sigma\left(\tilde{A}H^{(l-1)}W^{(l-1)}\right)$$

όπου $\tilde{A}$ είναι ένας κανονικοποιημένος πίνακας γειτνίασης, $W^{(l-1)}$ είναι ένας πίνακας βαρών προς εκμάθηση, και $\sigma(\cdot)$ είναι μια μη γραμμική συνάρτηση ενεργοποίησης. Καθώς το l αυξάνεται, ο επαναλαμβανόμενος πολλαπλασιασμός με τον $\tilde{A}$ συναθροίζει πληροφορίες σε μεγαλύτερες γειτονιές, με αποτέλεσμα οι γραμμές του $H^{(l)}$ να ευθυγραμμίζονται με τα κυρίαρχα ιδιοδιανύσματα του $\tilde{A}$. Όταν οι αναπαραστάσεις συγκλίνουν με αυτόν

τον τρόπο, αντανακλώντας μόνο την τοπολογία του γράφου και αγνοώντας τα χαρακτηριστικά εισόδου, η διακριτική ικανότητα των κόμβων χάνεται προοδευτικά. Αυτή η κατάρρευση της αναπαραστατικής διακύμανσης ονομάζεται υπερ-εξομάλυνση [9].
Το φαινόμενο αυτό χαρακτηρίστηκε αρχικά από φασματική σκοπιά, εξετάζοντας τη συστολή του ιδιοχώρου από την επαναλαμβανόμενη συνέλιξη γράφου: κάθε βήμα διάδοσης μειώνει τη διασπορά των αναπαραστάσεων, ωθώντας τις προς έναν κοινό υποχώρο. Η υπερ-εξομάλυνση αποτελεί το πιο θεμελιώδες και μελετημένο εμπόδιο στην κλιμάκωση των GNNs σε μεγαλύτερο βάθος.

II Υπόβαθρο και Ανασκόπηση Βιβλιογραφίας

II.I Βασικές Έννοιες και Αναπαράσταση Γράφων

II.I.1 Βασικές Έννοιες Γράφων

Ένας γράφος είναι μια θεμελιώδης δομή δεδομένων, οριζόμενη ως ένα διατεταγμένο ζεύγος $G = (V, E)$, όπου V είναι το σύνολο των κορυφών (ή κόμβων) και $E \subseteq V \times V$ είναι το σύνολο των ακμών. Ο αριθμός των κορυφών συμβολίζεται με $N = |V|$.
Οι γράφοι μπορούν να ταξινομηθούν ανάλογα με την κατεύθυνση των ακμών:

- **Μη Κατευθυνόμενοι Γράφοι**: Κάθε ακμή $\{u, v\} \in E$ αντιπροσωπεύει μια αμφίδρομη σύνδεση. Ο πίνακας γειτνίασης A είναι συμμετρικός ($A_{ij} = A_{ji}$).
- **Κατευθυνόμενοι Γράφοι**: Κάθε ακμή $(u, v) \in E$ είναι ένα διατεταγμένο ζεύγος που υποδηλώνει σύνδεση από τον κόμβο u στον κόμβο v. Ο πίνακας γειτνίασης είναι γενικά ασύμμετρος.

Ο πίνακας γειτνίασης $A \in \{0, 1\}^{N \times N}$ κωδικοποιεί την παρουσία ή απουσία ακμών: $A_{ij} = 1$ αν υπάρχει ακμή από τον κόμβο i στον κόμβο j, και 0 διαφορετικά.
Ο πίνακας βαθμού $D \in \mathbb{R}^{N \times N}$ είναι ένας διαγώνιος πίνακας του οποίου τα στοιχεία αντιστοιχούν στους βαθμούς των κορυφών. Για μη κατευθυνόμενους γράφους, ο βαθμός του κόμβου i ορίζεται ως $\deg(i) = \sum_{j=1}^{N} A_{ij}$, και $D_{ii} = \deg(i)$.

II.I.2 Χαρακτηριστικά Κόμβων και Αναπαραστάσεις

Σε πολλές εφαρμογές, οι κορυφές ενός γράφου συσχετίζονται με διανύσματα χαρακτηριστικών που κωδικοποιούν εγγενή ή περιβαλλοντικά χαρακτηριστικά. Αυτά συλλέγονται σε έναν πίνακα χαρακτηριστικών $X \in \mathbb{R}^{N \times C}$, όπου κάθε γραμμή $x_i \in \mathbb{R}^C$ αντιπροσωπεύει τα χαρακτηριστικά της κορυφής i, και C είναι η διάσταση των χαρακτηριστικών εισόδου.
Μια αναπαράσταση (embedding) είναι ένας μετασχηματισμός που προβάλλει αυτά τα διανύσματα χαρακτηριστικών σε έναν χώρο χαμηλότερης (συνήθως) διάστασης, διατηρώντας παράλληλα τις δομικές και τις βασισμένες σε χαρακτηριστικά σχέσεις. Ο στόχος είναι οι γραμμές $h_i \in \mathbb{R}^d$ του πίνακα αναπαραστάσεων H να κωδικοποιούν τόσο τα τοπικά μοτίβα συνδεσιμότητας όσο και τα αρχικά χαρακτηριστικά της κορυφής i.

II.II Γεωμετρική Μάθηση (Geometric Learning)

Η Γεωμετρική Μάθηση (ή Γεωμετρική Βαθιά Μάθηση) είναι η μελέτη και ανάπτυξη μοντέλων μηχανικής μάθησης που εκμεταλλεύονται ρητά τη γεωμετρική και τοπολογική δομή

που είναι εγγενής σε πεδία δεδομένων που δεν αναπαρίστανται φυσικά στον Ευκλείδειο χώρο.
Ενώ τα παραδοσιακά μοντέλα βαθιάς μάθησης (όπως τα CNNs για εικόνες) έχουν σχεδιαστεί για την επεξεργασία δεδομένων με κανονικές δομές, η γεωμετρική βαθιά μάθηση επεκτείνει αυτές τις έννοιες (π.χ., συνέλιξη, ομαδοποίηση, συνάθροιση χαρακτηριστικών) για να χειριστεί δεδομένα που βρίσκονται σε μη Ευκλείδειους χώρους, συμπεριλαμβανομένων γράφων με ακανόνιστη συνδεσιμότητα.
Στον πυρήνα της, η γεωμετρική μάθηση στοχεύει στον εντοπισμό και την επιβολή κατάλληλων επαγωγικών προκαταλήψεων (inductive biases), συνήθως με τη μορφή αναλλοίωσης (invariance) ή ισομεταβλητότητας (equivariance) σε μετασχηματισμούς που υπαγορεύονται από την υποκείμενη γεωμετρία [10].

II.II.1 Κατηγορίες Γεωμετρικής Μάθησης

Η γεωμετρική μάθηση μπορεί να χωριστεί σε πέντε αλληλένδετες κατηγορίες, γνωστές ως τα «πέντε G's» [10]:

1. **Grids (Πλέγματα)**: Κανονικά πλέγματα (π.χ., εικόνες), όπου οι κλασικές αρχιτεκτονικές συνέλιξης εκμεταλλεύονται τη μεταφορική αναλλοίωση.

2. **Groups (Ομάδες)**: Συμμετρικές ομάδες και οι δράσεις τους, όπου οι συνελίξεις ομάδων επεκτείνουν την ιδέα της κοινής χρήσης βαρών σε πιο γενικές ομάδες μετασχηματισμού (π.χ., περιστροφές).

3. **Graphs (Γράφοι)**: Διακριτές σχεσιακές δομές, όπου οι μέθοδοι βασισμένες σε γράφους γενικεύουν την αρχή ορίζοντας λειτουργίες συνέλιξης μέσω σχημάτων διάδοσης μηνυμάτων.

4. **Geodesics (Γεωδαισιακές)**: Συνεχείς πολλαπλότητες με Ριμάνιες μετρικές, όπου οι μέθοδοι ορίζουν πυρήνες συνέλιξης σε σχέση με γεωδαισιακές αποστάσεις.

5. **Gauges**: Τοπικά πλαίσια αναφοράς σε πολλαπλότητες, όπου οι αρχιτεκτονικές ισομεταβλητότητας βαθμίδας μοντελοποιούν ρητά τις αναπαραστάσεις ως πεδία που μετασχηματίζονται κατάλληλα υπό αλλαγές τοπικών πλαισίων αναφοράς.

II.II.2 Εφαρμογές

Η ανάγκη για γεωμετρική μάθηση προέκυψε από την παρατήρηση ότι πολλά δεδομένα του πραγματικού κόσμου είναι εγγενώς μη Ευκλείδεια (π.χ., κοινωνικά δίκτυα, μοριακές δομές, 3D πλέγματα). Η κωδικοποίηση γνωστών συμμετριών και αναλλοίωτων απευθείας στην αρχιτεκτονική του δικτύου μειώνει τον χώρο υποθέσεων, βελτιώνοντας την αποδοτικότητα του δείγματος και μειώνοντας την υπερ-προσαρμογή (overfitting).
Οι μέθοδοι γεωμετρικής μάθησης έχουν επιδείξει κορυφαία απόδοση σε ένα ευρύ φάσμα εργασιών, όπως:

- Πρόβλεψη μοριακών ιδιοτήτων.
- Ταξινόμηση και τμηματοποίηση σημειακών νεφών (point-cloud).
- Ανάλυση κοινωνικών δικτύων.
- Πρόβλεψη κυκλοφορίας σε γράφους οδικών δικτύων.

Η ιστορική εξέλιξη της γεωμετρικής μάθησης περιλαμβάνει τρεις φάσεις:

1. **Πρώτη Φάση**: Εστίαση σε πρώιμες αρχιτεκτονικές βασισμένες σε γράφους, όπως τα επαναλαμβανόμενα GNNs και οι φασματικές μέθοδοι που βασίζονται στον Λαπλασιανό του γράφου.

2. **Δεύτερη Φάση**: Άνοδος των χωρικών GNNs, με κυρίαρχο το παράδειγμα διάδοσης μηνυμάτων.

3. **Τρίτη Φάση**: Χαρακτηρίζεται από προσπάθειες ενοποίησης αυτών των προσεγγίσεων υπό ένα ενιαίο μαθηματικό πλαίσιο, αναγνωρίζοντας κοινές έννοιες συμμετρίας και ισομεταβλητότητας.

II.III Σχετικές Εργασίες για την Αντιμετώπιση της Υπερ-εξομάλυνσης

Η αντιμετώπιση της υπερ-εξομάλυνσης αποτελεί κεντρικό ερευνητικό ζήτημα, με τη βιβλιογραφία να προτείνει λύσεις που μπορούν να κατηγοριοποιηθούν σε τρεις κύριες κατευθύνσεις: (i) Τροποποιήσεις στην Αρχιτεκτονική, (ii) Τεχνικές Εκπαίδευσης και (iii) Μέθοδοι Διάδοσης.

II.III.1 Τροποποιήσεις στην Αρχιτεκτονική

Αυτές οι μέθοδοι στοχεύουν στην αλλαγή του τρόπου με τον οποίο τα επίπεδα του GNN αλληλεπιδρούν μεταξύ τους, ώστε να διατηρείται η πληροφορία των κόμβων σε μεγαλύτερο βάθος.

- **Residual και Skip Συνδέσεις**: Η πιο διαδεδομένη προσέγγιση είναι η ενσωμάτωση συνδέσεων παράκαμψης (skip connections), όπως οι residual συνδέσεις, οι οποίες επιτρέπουν την απευθείας ροή πληροφορίας από τα αρχικά επίπεδα στα βαθύτερα. Αυτό βοηθά στη σταθεροποίηση των κλίσεων και στην άμβλυνση της υπερ-εξομάλυνσης, καθώς η τελική αναπαράσταση είναι ένας συνδυασμός της τοπικά συναθροισμένης πληροφορίας και της αρχικής πληροφορίας του κόμβου. Παραδείγματα περιλαμβάνουν τα GCNII [11] και PPRGNN [12], τα οποία υιοθετούν το πλαίσιο του ResNet.

- **Ειδικά Σχεδιασμένα Επίπεδα**: Άλλες αρχιτεκτονικές εισάγουν επίπεδα που έχουν σχεδιαστεί για να περιορίζουν την εξομάλυνση. Το PairNorm [13] χρησιμοποιεί κανονικοποίηση για να διατηρήσει τη μέση απόσταση μεταξύ των ενσωματώσεων των κόμβων σε κάθε επίπεδο. Το DropEdge [14] διαγράφει τυχαία ακμές κατά την εκπαίδευση, μειώνοντας έτσι την πυκνότητα του γράφου και επιβραδύνοντας τη διάδοση της πληροφορίας, γεγονός που καθυστερεί την υπερ-εξομάλυνση.

II.III.2 Τεχνικές Εκπαίδευσης και Βελτιστοποίησης

Αυτές οι μέθοδοι εστιάζουν στον τρόπο με τον οποίο εκπαιδεύεται το μοντέλο, αντί να αλλάζουν την ίδια την αρχιτεκτονική.

- **Κανονικοποίηση (Regularization)**: Η χρήση όρων κανονικοποίησης στη συνάρτηση κόστους μπορεί να ενθαρρύνει τις αναπαραστάσεις να παραμείνουν διακριτές, καθώς προστίθεται ένας όρος που τιμωρεί τη μείωση της διακύμανσης των αναπαραστάσεων.

- **Αρχικοποίηση Βαρών**: Η σωστή αρχικοποίηση των βαρών είναι κρίσιμη για τη σταθερότητα της εκπαίδευσης σε βαθιά δίκτυα. Μέθοδοι όπως η G-Init (που προτείνεται στην παρούσα διατριβή) λαμβάνουν υπόψη τη δομή του γράφου για να εξισορροπήσουν τη διασπορά των σημάτων και των κλίσεων, επεκτείνοντας τις αρχές των [15] σε μη Ευκλείδειους χώρους.

- **Μερική Εκπαίδευση (Partial Training)**: Η ιδέα της εκπαίδευσης μόνο ενός υποσυνόλου των επιπέδων, όπως προτείνεται στην παρούσα εργασία, λειτουργεί ως μορφή κανονικοποίησης, διατηρώντας την ποικιλομορφία των αναπαραστάσεων.

II.III.3 Μέθοδοι Διάδοσης (Propagation Methods)

Αυτές οι μέθοδοι τροποποιούν τον τρόπο με τον οποίο η πληροφορία διαδίδεται μεταξύ των κόμβων.

- **Εξατομικευμένη Διάδοση (Personalized Propagation)**: Μοντέλα όπως το APPNP [16] χρησιμοποιούν εξατομικευμένο PageRank για να διατηρήσουν την πληροφορία του αρχικού κόμβου, επιτρέποντας τη διάδοση σε μεγάλο βάθος χωρίς να χάνεται η τοπική πληροφορία.

- **Αυτόνομη Διάδοση (Decoupled Propagation)**: Ορισμένες αρχιτεκτονικές διαχωρίζουν τη διάδοση της πληροφορίας από τον μετασχηματισμό των χαρακτηριστικών. Το SGC [17] αφαιρεί τις μη γραμμικότητες και τους πίνακες βαρών μεταξύ των επιπέδων, εκτελώντας μόνο διάδοση, γεγονός που επιταχύνει την εκπαίδευση και μειώνει την υπερ-εξομάλυνση.

Η παρούσα διατριβή, όπως περιγράφεται στην αρχική περίληψη, συμβάλλει σε όλες αυτές τις κατηγορίες, εισάγοντας μια νέα μετρική διάγνωσης (MASED), αναλύοντας τους περιορισμούς των residual συνδέσεων, προτείνοντας μια νέα τεχνική εκπαίδευσης (Μερική Εκπαίδευση), μια νέα αρχικοποίηση βαρών (G-Init), και μια τροποποίηση στη συνάρτηση ενεργοποίησης (slope-ReLU).

III Κεφάλαιο 3: Ανάλυση της Επίδρασης των Νορμών των Αναπαραστάσεων και των Ιδιαζουσών Τιμών στην Υπερ-εξομάλυνση

Το παρόν κεφάλαιο εισάγει μια θεμελιώδη προσέγγιση για την ποσοτικοποίηση και τη μείωση της υπερ-εξομάλυνσης (oversmoothing) στα βαθιά Νευρωνικά Δίκτυα Γράφων (GNNs). Η υπερ-εξομάλυνση αποτελεί ένα κρίσιμο εμπόδιο στην ανάπτυξη εκφραστικών, βαθιών αρχιτεκτονικών GNN. Για την αντιμετώπιση αυτής της πρόκλησης, προτείνεται μια νέα μετρική απόστασης, η Μέση Τετραγωνική Ευκλείδεια Απόσταση (Mean Average Squared Euclidean Distance - MASED), η οποία ποσοτικοποιεί τον βαθμό της υπερ-εξομάλυνσης σε ένα δεδομένο μοντέλο.

Η ανάλυση του MASED περιλαμβάνει την εξαγωγή των άνω και κάτω ορίων του, αποκαλύπτοντας τη σχέση μεταξύ των αποστάσεων των αναπαραστάσεων των κόμβων και της δομής του πίνακα βαρών. Συγκεκριμένα, αναδεικνύεται ο κεντρικός ρόλος των νορμών των αναπαραστάσεων των κόμβων (node embedding norms) και των μικρότερων ιδιαζουσών τιμών (smallest singular values) των πινάκων βαρών, παράμετροι που είχαν παραβλεφθεί σε μεγάλο βαθμό. Η διατήρηση της διακύμανσης των αναπαραστάσεων των κόμβων και η αύξηση του κάτω ορίου του MASED επιτυγχάνεται μέσω της αύξησης αυτών των παραμέτρων.

Επιπλέον, η μετρική MASED έχει αποδειχθεί ότι μπορεί να προβλέψει την έναρξη της υπερ-εξομάλυνσης νωρίς στη διαδικασία εκμάθησης. Μια ταχεία μείωση του MASED, συνοδευόμενη από συρρίκνωση των ιδιαζουσών τιμών του πίνακα βαρών, οδηγεί σε κακή απόδοση του μοντέλου.

Ως στρατηγική μετριασμού, προτείνεται μια νέα μέθοδος κανονικοποίησης, η G-Reg, η οποία στοχεύει στη μείωση της συγγραμμικότητας μεταξύ των γραμμών των πινάκων βαρών, αυξάνοντας έτσι τις μικρότερες ιδιάζουσες τιμές τους. Η G-Reg δοκιμάζεται πειραματικά σε βαθιά GCNs, residual GCNs και SGCs, με έως και 32 επίπεδα, μειώνοντας την υπερεξομάλυνση και επιδεικνύοντας οφέλη σε σενάρια περιορισμένης πληροφορίας, όπως το “cold start”. Τέλος, διερευνάται η επίδραση της μείωσης του πλήθους των πινάκων βαρών σε συναθροίσεις πολλαπλών αλμάτων (multi-hop aggregation) μέσω της χρήσης λιγότερων πινάκων βαρών από τον αριθμό των αλμάτων γειτονιάς.

IV Κεφάλαιο 4: Ανάλυση της Επίδρασης των Residual Συνδέσεων στην Υπερ-εξομάλυνση

Το κεφάλαιο αυτό διερευνά τον ρόλο των residual συνδέσεων στον μετριασμό της υπερεξομάλυνσης και στη δυνατότητα χρήσης βαθιών GNNs για εργασίες που απαιτούν αλληλεπιδράσεις μεγάλης εμβέλειας. Ενώ οι συνδέσεις παράκαμψης (skip connections) τύπου residual προσφέρουν έναν απλό μηχανισμό για τη διατήρηση της ποικιλομορφίας των χαρακτηριστικών σε πολλά επίπεδα, η παρούσα εργασία εξετάζει τους περιορισμούς τους.

Συγκεκριμένα, αναλύεται μια οικογένεια μοντέλων που χρησιμοποιούν residual συνδέσεις, όπως τα APPNP, GCNII και PPRGNN. Η ανάλυση αποδεικνύει ότι η προσθήκη residual συνδέσεων σε κάθε επίπεδο ενός GNN καθιστά το μοντέλο ισοδύναμο με το σταθμισμένο άθροισμα των αναπαραστάσεων από μοντέλα διαφορετικού βάθους, με εκθετικά μειούμενους συντελεστές.

Ένα βασικό εύρημα αφορά την Απεικόνιση Ταυτότητας (Identity Mapping), όπου αποδεικνύεται ότι η προσθήκη του πίνακα ταυτότητας σε κάθε πίνακα βαρών ισοδυναμεί με ένα σταθμισμένο άθροισμα δυνάμεων (weighted powerset sum) των πινάκων βαρών του μοντέλου. Αυτό το αποτέλεσμα ρίχνει φως στη σχέση μεταξύ της απεικόνισης ταυτότητας και της υπερ-εξομάλυνσης.

Επιπλέον, το μοντέλο APPNP αναδιατυπώνεται και αποδεικνύεται ότι ανάγεται σε μια επανάληψη δύναμης (power iteration), με ρυθμό σύγκλισης που εξαρτάται από την παράμετρο ισχύος του residual (α). Εξάγεται ένας τύπος που συνδέει την παράμετρο α, το βάθος και την ανοχή του μοντέλου, όπου η ανοχή μετρά την εγγύτητα των αναπαραστάσεων που δημιουργούνται από διαδοχικά επίπεδα.

Τα πειράματα, ειδικά σε σενάρια με μειωμένη πληροφορία (όπως το “cold start”) και σε συνθετικά σύνολα δεδομένων με ελεγχόμενες αλληλεπιδράσεις μεγάλης εμβέλειας, αποκαλύπτουν τους περιορισμούς των μεθόδων που χρησιμοποιούν residual συνδέσεις. Αυτές οι μέθοδοι δίνουν έμφαση στα αρχικά χαρακτηριστικά των κόμβων, μιμούμενες έτσι ρηχές αρχιτεκτονικές και περιορίζοντας την ικανότητα του μοντέλου να συλλαμβάνει τις απαραίτητες αλληλεπιδράσεις μεγάλης εμβέλειας.

V Κεφάλαιο 5: Μερικώς Εκπαιδευμένα Νευρωνικά Δίκτυα Γράφων Αντιστέκονται στην Υπερ-εξομάλυνση

Το κεφάλαιο αυτό εξετάζει τη θεωρητική και πειραματική ικανότητα αναπαράστασης των μερικώς εκπαιδευμένων GNNs, δηλαδή μοντέλων στα οποία μόνο ένα επίπεδο εκπαιδεύεται, ενώ τα υπόλοιπα διατηρούνται στην αρχική τους τυχαία κατάσταση. Η προσέγγιση αυτή βασίζεται στην παρατήρηση ότι ακόμη και ένα μη εκπαιδευμένο GCN μπορεί να παράγει χρήσιμες αναπαραστάσεις κόμβων.

Τα ευρήματα δείχνουν ότι η αύξηση του εύρους (width) των μερικώς εκπαιδευμένων GCNs ενισχύει την απόδοσή τους και τα καθιστά ανθεκτικά στην υπερ-εξομάλυνση. Η ανάλυση αυτή επεκτείνεται και σε αρχιτεκτονικές GAT και GCNII, επιβεβαιώνοντας τους ισχυρισμούς.

Η θεωρητική ανάλυση εστιάζει στις ιδιότητες των μη εκπαιδευμένων πινάκων βαρών, χρησιμοποιώντας το Νόμο Bai-Yin για να συνδέσει την αρχικοποίηση Glorot με τη μεγαλύτερη ιδιάζουσα τιμή των πινάκων. Εξετάζεται το γινόμενο των μη εκπαιδευμένων πινάκων βαρών και η κατανομή των στοιχείων τους. Αυτή η ανάλυση εξηγεί πώς η πληροφορία των αρχικών κόμβων ρέει και μετασχηματίζεται μέσω του δικτύου.

Πειραματικά, αποδεικνύεται η ισχύς των βαθιών μερικώς εκπαιδευμένων GCNs στην αντίσταση στην υπερ-εξομάλυνση. Επιπλέον, τονίζονται τα πλεονεκτήματα αυτών των δικτύων σε σενάρια περιορισμένης πληροφορίας, όπως το “cold start”, όπου τα χαρακτηριστικά των κόμβων είναι διαθέσιμα μόνο για τους επισημασμένους κόμβους. Τέλος, διερευνάται η βέλτιστη θέση και ο τύπος του εκπαιδεύσιμου επιπέδου εντός του δικτύου, με ένα απλό επίπεδο GCN να αποτελεί μια αποτελεσματική επιλογή.

VI Κεφάλαιο 6: Μείωση της Υπερ-εξομάλυνσης μέσω Ενημερωμένης Αρχικοποίησης Βαρών στα GNNs

Το κεφάλαιο αυτό αντιμετωπίζει το πρόβλημα της υπερ-εξομάλυνσης στα βαθιά GNNs μέσω της αρχικοποίησης των βαρών. Οι υπάρχουσες μέθοδοι αρχικοποίησης (όπως οι Kaiming ή Xavier) παραμελούν τις δομικές ιδιότητες των δεδομένων γράφου και τη μοναδική συμπεριφορά διάδοσης σήματος που είναι εγγενής στις αρχιτεκτονικές των GNNs.

Η εργασία γενικεύει την ανάλυση των [15] για τα GNNs, παρουσιάζοντας τύπους για τη διακύμανση των σημάτων (forward signals) και των κλίσεων (backward gradients) που ρέουν μέσα στο δίκτυο. Η ανάλυση καλύπτει ένα παραμετρικό GNN που μπορεί να συνδυάζει συνέλιξη γράφου, residual συνδέσεις, skip συνδέσεις και απεικόνιση ταυτότητας, καλύπτοντας έτσι την πλειονότητα των υφιστάμενων μοντέλων.

Με βάση αυτά τα θεωρητικά αποτελέσματα, προτείνεται μια νέα μέθοδος αρχικοποίησης βαρών, η G-Init, ειδικά προσαρμοσμένη για GCNs. Η G-Init στοχεύει στη σταθεροποίηση της διακύμανσης και αξιοποιεί τη σχέση της με τη μείωση της υπερ-εξομάλυνσης. Αυτό το αποτέλεσμα αποδίδεται στις αρχικές μεγαλύτερες ιδιάζουσες τιμές των πινάκων βαρών, οι οποίες επηρεάζουν τις τελικές ιδιάζουσες τιμές μετά τη σύγκλιση, και οι οποίες είναι γνωστό ότι σχετίζονται με την υπερ-εξομάλυνση.

Τα πειράματα επιβεβαιώνουν τα θεωρητικά αποτελέσματα, χρησιμοποιώντας βαθιά GCNs και GATs σε 8 σύνολα δεδομένων για εργασίες ταξινόμησης κόμβων. Η G-Init αποδεικνύεται ότι μειώνει την υπερ-εξομάλυνση και προσφέρει οφέλη σε σενάρια “cold start”. Επιπλέον, η επέκταση των πειραμάτων σε εργασίες ταξινόμησης γράφων δείχνει ότι η G-Init υπερτερεί των τυπικών μεθόδων αρχικοποίησης και σε αυτά τα προβλήματα.

VII Κεφάλαιο 7: Μείωση της Υπερ-εξομάλυνσης στα GNNs μέσω Αλλαγής της Συνάρτησης Ενεργοποίησης

Το κεφάλαιο αυτό διερευνά την υπερ-εξομάλυνση στα βαθιά GNNs από μια νέα οπτική γωνία, εστιάζοντας στον αντίκτυπο της συνάρτησης ενεργοποίησης και του ρυθμού εκμάθησης (learning rate) ανά επίπεδο. Αποδεικνύεται, τόσο θεωρητικά όσο και πειραματικά, ότι η επιλογή και η διαμόρφωση των συναρτήσεων ενεργοποίησης παίζουν κεντρικό ρόλο στη δυναμική εξομάλυνσης των αναπαραστάσεων των κόμβων.

Συγκεκριμένα, αναλύεται η συμβολή της τυπικής ενεργοποίησης ReLU στην κατάρρευση των αναπαραστάσεων σε βαθιά GNNs. Βασιζόμενοι σε αυτή τη διαπίστωση, προτείνεται μια απλή αλλά αποτελεσματική τροποποίηση: η προσαρμογή της κλίσης (slope) της ReLU για τη διατήρηση της αναπαραστατικής ποικιλομορφίας μεταξύ των επιπέδων. Η μέθοδος αυτή δεν απαιτεί αλλαγές στην αρχιτεκτονική του δικτύου ή την εισαγωγή skip συνδέσεων.

Η εργασία παρέχει την πρώτη μελέτη σχετικά με τον ρόλο της συνάρτησης ενεργοποίησης και του ρυθμού εκμάθησης ανά επίπεδο στην υπερ-εξομάλυνση. Θεωρητικά, αποδεικνύεται η σύνδεση μεταξύ της κλίσης της ReLU και των ιδιαζουσών τιμών των πινάκων βαρών, οι οποίες σχετίζονται με την υπερ-εξομάλυνση.

Επιπλέον, η ανάλυση για την επίδραση της κλίσης της ReLU επεκτείνεται στον ρυθμό εκμάθησης ανά επίπεδο. Τα πειράματα δείχνουν ότι η ρύθμιση των ρυθμών εκμάθησης μπορεί επίσης να μειώσει την υπερ-εξομάλυνση, αν και είναι λιγότερο πρακτική.

Τέλος, πραγματοποιούνται εκτεταμένα πειράματα με δίκτυα έως και 64 επιπέδων, επιβεβαιώνοντας ότι η προτεινόμενη μέθοδος αποτρέπει τη σύγκλιση των αναπαραστάσεων των κόμβων στο ίδιο σημείο, οδηγώντας σε καλύτερες αναπαραστάσεις. Τα οφέλη των βαθιών GNNs με την προτεινόμενη μέθοδο τονίζονται και σε σενάρια “cold start”.

VIII Συμπεράσματα

Στην παρούσα διατριβή παρουσιάζεται ένα ολοκληρωμένο πλαίσιο αντιμετώπισης της υπερ-εξομάλυνσης στα Νευρωνικά Δίκτυα Γράφων (GNNs), το οποίο βασίζεται σε πέντε διακριτές μεθοδολογίες. Κάθε προσέγγιση συμβάλλει σε διαφορετικό σημείο της αλυσίδας επεξεργασίας, από τη διάγνωση και τη θεωρητική τεκμηρίωση έως τις πρακτικές παρεμβάσεις και την αποτελεσματική εκπαίδευση βαθιών μοντέλων.

Τα ευρήματα επιβεβαιώνουν ότι η διατήρηση της διακύμανσης των αναπαραστάσεων των κόμβων, ο έλεγχος της τοπολογικής διάδοσης πληροφορίας, η στοχευμένη εκπαίδευση στρωμάτων, η σωστή αρχικοποίηση των βαρών, και ο έλεγχος μη-γραμμικοτήτων μπορούν να οδηγήσουν στην κατασκευή βαθύτερων, αποδοτικότερων και γενικεύσιμων GNNs. Οι προτεινόμενες μέθοδοι ανοίγουν τον δρόμο για αξιόπιστες εφαρμογές σε πραγματικά σενάρια, υποδηλώνοντας ότι το φαινόμενο της υπερ-εξομάλυνσης είναι διαχειρίσιμο όταν αντιμετωπίζεται με κατάλληλες μεθόδους.

Chapter 1

Introduction

1 Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) have become a cornerstone for analyzing and learning from graph-structured data. Their ability to combine node attributes with the inherent connectivity of graphs has led to remarkable advances in many fields. GNNs are a class of neural architectures specifically designed to process data that can be represented as graphs. In a graph, entities are modeled as nodes and relationships between them as edges. Formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$ be an undirected graph, with $|\mathcal{V}| = N$ nodes $u_i \in \mathcal{V}$, edges $(u_i, u_j) \in \mathcal{E}$ and $X = [x_1, ..., x_N]^T \in \mathbb{R}^{N \times C}$ the initial node features. The edges form an adjacency matrix $A \in \mathbb{R}^{N \times N}$ where edge (u_i, u_j) is associated with element $A_{i,j}$. $A_{i,j}$ can take arbitrary real values indicating the weight (strength) of edge (u_i, u_j). During the training phase, only the labels of a subset $\mathcal{V}_l \subset \mathcal{V}$ are accessible, where $|\mathcal{V}_l| = N_{train}$. The objective is to develop a node classifier that leverages the graph topology and the provided feature vectors to predict the label of each node. GNNs learn a function

$$\mathcal{F} : \mathcal{G} \longrightarrow Z,$$

where $Z \in \mathbb{R}^{N \times m}$ is the matrix of output embeddings. Depending on the downstream task, Z may represent node-level embeddings (for node classification or link prediction), edge-level scores (for edge classification), or a single graph-level embedding $z \in \mathbb{R}^m$ obtained by pooling over nodes (for graph classification or regression).

Message-Passing Framework. The predominant paradigm for GNNs is the message- passing (or neighborhood aggregation) framework [18]. A GNN is composed of L layers, where at layer l each node u maintains a hidden representation $h_u^{(l)} \in \mathbb{R}^{d_l}$ (with $h_u^{(0)} = x_u$, and d_l indicating the hidden size of layer l). The update from layer $l-1$ to layer l involves two steps:

(i) **Aggregation**: Each node collects information from its neighbors. Let $\mathcal{N}(u) = \{v \in \mathcal{V} : (v, u) \in \mathcal{E}\}$ denote the neighbor set of node u. Then

$$m_u^{(l)} = \text{AGGREGATE}^{(l)}\Big(\big\{h_v^{(l-1)} : v \in \mathcal{N}(u)\big\}\Big).$$

Here, $\text{AGGREGATE}^{(l)}$ is a permutation-invariant function (e.g., sum, mean, or an attention-weighted combination) that ensures the output does not depend on the ordering of neighbors.

(ii) **Combination**: The aggregated message is then combined with the node's previous representation to produce the new hidden representation:

$$h_u^{(l)} = \text{COMBINE}^{(l)}\Big(h_u^{(l-1)}, m_u^{(l)}\Big).$$

In practice, $\text{COMBINE}^{(l)}$ often consists of a linear transformation (via a learnable weight matrix) followed by a nonlinear activation function (e.g., ReLU).

After L such message-passing iterations, each node u obtains a final embedding $h_u^{(L)}$. For node-level tasks, one can apply a task-specific prediction layer (e.g., a softmax classifier) to $h_u^{(L)}$. For graph-level tasks, a permutation-invariant pooling operation (such as summation or averaging) over $\{h_u^{(L)} : u \in \mathcal{V}\}$ produces a single vector

$$h_{\mathcal{G}} \;=\; \text{POOL}\Big(\{h_u^{(L)} : u \in \mathcal{V}\}\Big),$$

which is then passed through further layers to yield the final output.

Permutation Invariance and Equivariance. A crucial property of any GNN is that it respects the graph's invariance under node permutations. In particular, if $P \in \{0,1\}^{N\times N}$ is a permutation matrix and we permute the input as $(P\,A\,P^T,\ P\,X)$, a properly defined GNN generates outputs Z satisfying $Z' = P\,Z$ for node-level embeddings, and identical graph-level embeddings. This property ensures that learned representations and predictions do not depend on the arbitrary ordering of nodes.

Expressive Power and Limitations. The representational capacity of message-passing GNNs has been analyzed through the lens of the Weisfeiler–Lehman (WL) graph isomorphism test: if two non-isomorphic graphs cannot be distinguished by 1-WL (color refinement), then no such GNN can assign different representations to them [6]. While more expressive variants have been developed to match the discriminative capability of 1-WL, these models still face inherent limitations and may struggle with certain graph structures. Moreover, as GNNs grow deeper, they can suffer from oversmoothing, where node representations converge to similar values regardless of structural differences, and from oversquashing, which hinders the flow of information across distant nodes. These phenomena, together with optimization difficulties in very deep networks, underscore the need for architectural improvements and regularization methods. In essence, GNNs generalize neural architectures to non-Euclidean domains by iteratively aggregating and transforming node features according to graph topology, allowing the learned representations to respect both feature information and combinatorial structure. However, addressing challenges such as oversmoothing, oversquashing, and training stability remains critical for advancing the expressive power and practical effectiveness of GNNs.

2 Deep Graph Neural Networks

Deep Graph Neural Networks aim to enhance the expressive power of standard architectures by stacking multiple layers of message passing operations. In essence, each GNN layer enables a node to aggregate and transform information from its immediate neighbors, gradually building a representation that captures both the node's features and its local structural context. By extending this process across multiple layers, deep GNNs aim to extract high-level patterns that span increasingly larger neighborhoods of the graph. This enables the model to capture long-range dependencies, which are crucial in many domains (e.g., social network analysis, biological networks, and knowledge graphs), where the underlying semantics or functions depend not only on nodes' immediate neighbors, but also on their multi-hop neighborhoods.

The motivation for deep GNNs arises from the observation that many graph-based tasks require the integration of information across distant parts of the graph. For example, in molecular property prediction, the influence of a functional group may be affected by distant atoms; in recommender systems, the relationship between two users may be inferred only through extended interaction paths; and in citation networks, papers sharing a common research area may only be connected through multi-hop citation paths. Shallow GNNs, typically limited to one or two hops, are insufficient for capturing such complex, non-local dependencies, leading to underfitting and limited predictive performance on tasks where high-order interactions are essential [19, 13]. This limitation, where a model cannot access relevant information due to insufficient depth, is referred to as the under-reach problem. In that particular scenario, GNNs fail to learn because each node does not have access to the appropriate information, which resides more hops away than the depth of the GNN under investigation.

Moreover, from a theoretical perspective, deeper GNNs are more expressive in their ability to distinguish different graph structures. This relates to the concept of the Weisfeiler-Lehman (WL) test of isomorphism: it has been shown that certain classes of GNNs are as powerful as the 1-WL test in distinguishing non-isomorphic graphs, and stacking more layers allows the network to capture patterns that are beyond the representational capacity of shallower counterparts [6]. Consequently, deeper architectures are essential in scenarios that demand finer structural discrimination and higher-order feature interactions.

However, despite their theoretical advantages, deep GNNs suffer from a number of critical limitations that hinder their practical usability. Beyond a certain depth, performance often deteriorates rather than improves. This degradation has been attributed to several interrelated phenomena:

(i) **Oversmoothing**. As the depth increases, node representations tend to converge to a stationary point, leading to a loss of model's ability to discriminate different classes. Although this issue will be discussed in detail in the next section, it is important to note here that it constitutes one of the principal barriers to constructing effective deep GNNs [20, 21, 7].

(ii) **Oversquashing**. As information from exponentially expanding neighborhoods is propagated through limited-capacity pathways, signals from distant nodes can interfere or be lost entirely. This bottleneck effect, referred to as oversquashing, prevents accurate transmission of long-range dependencies even when the network is sufficiently deep [8, 22].

(iii) **Vanishing and Exploding Gradients**. Deep GNNs, like other deep neural architectures, face difficulties during training due to unstable gradients. As messages are propagated over many layers, gradients can vanish or explode, making optimization inefficient and often leading to poor convergence. This problem is exacerbated by nonlinear activation functions and normalization layers, which amplify numerical instability in deeper models [23, 14].

(iv) **Gradient Homogenization**. As GNNs become deeper, the gradients associated with different nodes may become increasingly similar, hindering the model's ability to focus on informative substructures or sparsely represented regions. This gradient homogenization can hinder the optimization dynamics and limit the model's ability to adaptively focus on diverse graph regions [11].

(v) **Loss of Locality**. While deep networks aggregate more global information, they may lose sensitivity to important local features. As messages from distant nodes dominate the representation, subtle but critical neighborhood-specific patterns, such as small motifs or local attribute variations, can be overlooked. This loss of locality can be detrimental in cases where fine-grained distinctions are crucial for the underlying task.

(vi) **Scalability and Resource Constraints**. Deeper GNNs lead to higher computational and memory costs, especially on large-scale graphs. Each additional layer increases the number of neighborhood expansions and intermediate feature maps that must be stored and processed. Without careful design, this can make deep GNNs impractical for large graphs or resource-constrained environments [24].

In light of these challenges, developing architectures and training strategies that enable deep GNNs to effectively learn long-range dependencies while avoiding the aforementioned issues has become a central topic in graph representation learning. The subsequent section will delve deeper into the phenomenon of oversmoothing, one of the most fundamental and well-studied obstacles in scaling GNNs to greater depth.

3 Oversmoothing: Definition and Significance

Oversmoothing refers to a phenomenon in graph neural networks whereby repeated message passing or aggregation operations cause the node feature representations to become increasingly similar or even indistinguishable across the graph. Formally, let $H^{(l)} \in \mathbb{R}^{N \times d_l}$ denote the matrix of node embeddings after l layers of a GNN, where N is the number of nodes and d_l is the embedding dimension of the l-th layer. In many GNN architectures, the update rule can be expressed as:

$$H^{(l)} = \sigma\big(A\, H^{(l-1)}\, W^{(l-1)}\big),$$

where A is a normalized adjacency matrix (or other normalization of the graph structure), $W^{(l-1)}$ is a learnable weight matrix, and $\sigma(\cdot)$ is a nonlinear activation function. As l increases, repeated multiplication by A aggregates information across larger neighborhoods, causing the rows of $H^{(l)}$ to align with the dominant eigenvectors of A. When embeddings converge in this manner, reflecting only graph topology and disregarding input features, distinctive node representations are progressively lost. This collapse of representational variance is precisely termed *oversmoothing* [20, 7, 21].

The phenomenon was first rigorously characterized from a spectral perspective by examining eigenspace contraction under repeated graph convolution: each propagation step functions as a low-pass filter that suppresses high-frequency (local) information, leaving predominantly low-frequency (global) patterns. In the limit, embeddings converge to a subspace spanned by the dominant eigenvectors of the graph Laplacian or normalized adjacency, effectively "averaging out" distinctive node attributes. Each node's new representation becomes a weighted average of its own and its neighbors'. This process is equivalent to a form of Laplacian smoothing, and was initially observed in [25] and later analyzed in [20]. While moderate smoothing can aid semi-supervised learning, stacking multiple convolutional layers intensifies this effect, causing complete representational collapse and

information loss. [7] extended this view by showing that, under ReLU, oversmoothing corresponds to convergence onto a subspace rather than a fixed point, and [21] provided a complementary Dirichlet energy perspective.
Mitigating oversmoothing is critical for several reasons. First, many real-world graphs exhibit complex, multi-scale structure in which local communities or motifs carry essential information; if embeddings collapse, these subtle variations are destroyed, and the GNN cannot distinguish nodes appropriately [20]. Second, oversmoothing undermines the expressivity of deep GNNs: although deeper architectures can in principle capture longer-range dependencies, in practice representational collapse prevents nuanced differentiation, degrading performance on tasks such as link prediction in heterophilous graphs or graph-based regression [13]. Third, from an optimization standpoint, oversmoothing can lead to vanishing gradients and hinder effective training: as representations become uniform, gradients carry diminishing node-specific signals, making it difficult to adapt to fine-grained patterns [7]. Finally, many applications demand interpretable or disentangled node representations (e.g., recommendation systems, knowledge-graph completion), and oversmoothing directly conflicts with this goal by collapsing embeddings into a few dominant modes [13].
Owing to these considerations, preventing or alleviating oversmoothing is a central research direction in GNN design. Approaches including normalization techniques (e.g., PairNorm) [13], architectural and training modifications (e.g., residual connections, slope-modified activations, and partial training) [21, 5], weight initialization methods (e.g., G-Init) [4], and regularization schemes (e.g., graph spectral regularizers, G-Reg) aim to preserve embedding variance across layers while enabling high-order feature aggregation. Consequently, addressing oversmoothing is vital for advancing both theoretical understanding and practical capabilities of deep graph learning models.

4 Central Research Questions

This thesis consists of five papers and investigates oversmoothing in Graph Neural Networks by addressing the following focused questions:

1. **Formal Definition and Detection**:
    - How can oversmoothing be defined precisely in terms of the distance of node embeddings under repeated message passing?
    - Which theoretical criteria indicate that node representations converge towards the oversmoothing subspace?
2. **Quantification**:
    - What metrics or bounds capture the rate and extent of oversmoothing as functions of network depth, propagation steps, and weight-matrix spectral properties?
    - How do factors such as partial training, weight initialization, the smallest singular value of weight matrices, or the choice of activation function quantitatively affect embedding variance across layers?
3. **Mitigation Strategies**:

- Which modifications (partial training, architectural design choices, weight initialization, regularization methods or activation functions), prevent or delay embedding collapse without reducing the effective receptive field?
- How can these strategies be solidified by theoretical analyses to guarantee a nontrivial lower bound on embedding variance?

4. **Empirical Validation**:

   - In benchmark datasets, which methods allow the models to explore their deep variants without suffering from extreme performance degradation?
   - In "cold start" scenarios (unlabeled nodes without features), which methods preserve sufficient variance to propagate label information effectively?
   - In tasks requiring deep aggregation (e.g., newly introduced synthetic dataset with controllable long-range interactions), what is the behavior of models claiming to enable deep GNNs and mitigate oversmoothing?

5 Contributions

The main outcomes of this work are presented in the following papers:

[1] "Analyzing the Effect of Embedding Norms and Singular Values to Oversmoothing in Graph Neural Networks" by Kelesis, D., Fotakis, D., and Paliouras, G., arXiv, 2510.06066 [1].

[2] "Analyzing the Effect of Residual Connections to Oversmoothing in Graph Neural Networks" by Kelesis, D., Fotakis, D., and Paliouras, G., Machine Learning, 114(8):184 [2].

[3] "Partially Trained Graph Convolutional Networks Resist Oversmoothing" by Kelesis, D., Fotakis, D., and Paliouras, G., Machine Learning, 114(10):211 [3].

[4] "Reducing Oversmoothing through Informed Weight Initialization in Graph Neural Networks" by Kelesis, D., Fotakis, D., and Paliouras, G., Applied Intelligence, 55(7):632 [4]

[5] "Reducing oversmoothing in graph neural networks by changing the activation function" by Kelesis, D., Vogiatzis, D., Katsimpras, G., Fotakis, D., and Paliouras, G., ECAI 2023 - 26th European Conference on Artificial Intelligence [5].

The papers comprising this thesis collectively advance our understanding of oversmoothing in Graph Neural Networks by systematically defining, measuring, and mitigating the phenomenon, and by validating proposed methods across diverse settings. Below, we present a unified narrative that shows how each contribution addresses the four central research questions.

5.1 Formal Definition and Detection

A precise characterization of oversmoothing is foundational to any mitigation effort. [1] introduce the Mean Average Squared Distance ($MASED$) metric, which quantifies oversmoothing as the average pairwise distance among node embeddings at each layer. By

framing oversmoothing in terms of embedding distances, this work establishes a clear, interpretable definition: oversmoothing prevails as $MASED$ value drops.

5.2 Quantification

Quantifying the rate and extent of oversmoothing is addressed primarily in [1], with complementary insights from [3, 4, 5]:

- **Depth and Spectral Properties [1]**: By aggregating layer-wise $MASED$ bounds across depth, [1] derive global upper and lower bounds on $MASED$ as functions of network depth and the spectral properties of each weight matrix. These bounds explicitly show that deeper GNNs suffer a faster decay of $MASED$ unless the smallest singular value is maintained above zero.
- **Partial Training [3]**: [3] introduce a theoretical framework that quantifies how freezing a subset of layers (i.e., keeping them at random initialization) introduces stochastic mixing that preserves embedding variance. Under assumptions on model's width, it proves that wider layers amplify variance, and thus reduce oversmoothing quantitatively.
- **Initialization Effects [4]**: The G-Init method sets initial weight variances based on degree statistics so that signal and gradient variances remain balanced across layers. Analytical formulas link these variances to the amount of variance preserved in embeddings.
- **Activation Dynamics [5]**: By modeling the decreasing fraction of active ReLU units with depth, [5] show that standard ReLU amplifies variance decay. The proposed slope-modified ReLU adjusts this decay rate, and theoretical results quantify how the modified slope reduces oversmoothing as depth increases.

These contributions jointly offer a suite of metrics and bounds that capture oversmoothing progression under varying depths, spectral characteristics, training regimes, and architectural choices.

5.3 Mitigation Strategies

Equipped with precise detection and quantification, the thesis proposes and analyzes several methods to reduce oversmoothing without compromising receptive field size:

1. **G-Reg Regularizer [1]**: By penalizing small singular values of weight matrices, G-Reg directly increases the lower bound on $MASED$, ensuring a nontrivial variance among embeddings at each layer.
2. **Decoupled Adjacency Hops [1]**: To avoid coupling depth with the number of trainable weight matrices, a factor that accelerates oversmoothing, the number of distinct weight matrices is kept small while adjacency hops are distributed across them. This preserves the effective receptive field yet slows variance decay.
3. **Analysis of Residual Connections [2]**: Examines widely used residual-connection methods designed to mitigate oversmoothing, demonstrating both theoretically and empirically that, they predominantly aggregate information from local neighborhoods and thus fail to support the long-range propagation required by some tasks.

4. **Partial Training [3]**: Freezing weight matrices mixes features in a way that preserves variance and reduces oversmoothing. Theoretical analysis provides conditions under which a single trained layer suffices, and empirical results demonstrate its efficacy without altering network depth.

5. **Graph-Informed Initialization (G-Init, [4])**: By tailoring initial weight variances to graph degree statistics, G-Init balances forward and backward signal propagation, mitigating early variance collapse. This method does not require architectural modifications.

6. **Slope-Modified ReLU (Slope2GNN, [5])**: A simple adjustment of slope in the ReLU activation reduces oversmoothing as depth increases.

Each strategy is supported by rigorous theoretical analyses that guarantee oversmoothing reduction as depth increases.

5.4 Empirical Validation

The practical utility of these methods is confirmed across multiple settings:

- **Standard Benchmarks**: On common node and graph classification datasets, deep GNNs enhanced with G-Reg [1], G-Init [4], or slope-modified ReLU [5] consistently outperform baselines as depth increases.

- **"Cold Start" Scenarios**: All proposed strategies demonstrate robust performance when unlabeled nodes lack feature information, enabling effective label propagation by preserving embedding variance.

- **Synthetic Long-Range Interaction Tasks**: In a newly introduced dataset with controllable long-range dependencies, residual GNNs [2] underperform due to restricted receptive fields, whereas simpler models, such as GCN, achieve higher accuracy.

- **Trade-Off Analyses**: Experiments validate that distributing adjacency hops across few weight matrices strikes the optimal balance between expressiveness and oversmoothing avoidance [1], and that increased width in partially trained networks amplifies variance without harming generalization [3].

Collectively, these experiments verify the theoretical guarantees and demonstrate that the proposed techniques allow GNNs to exploit depth while avoiding the performance degradation traditionally attributed to oversmoothing.

6 Thesis Outline

Chapter 2: Background & Literature Review

This chapter reviews fundamental concepts in graph representation learning, defines oversmoothing, and surveys existing work on its detection, quantification, and mitigation. Key approaches, such as normalization techniques, residual/skip connections, regularization and initialization methods, and activation-based approaches, are discussed to establish context for the subsequent chapters.

Chapter 3: Oversmoothing Quantification and G-Reg [1]

Chapter 3 presents [1] in detail. It begins by defining $MASED$ metric for measuring oversmoothing and derives layer-wise bounds on $MASED$. The chapter then shows how coupling adjacency powers to the number of the weight matrices accelerates oversmoothing and motivates the decoupling strategy. Finally, it introduces the G-Reg regularizer, which reduces oversmoothing as depth increases.

Chapter 4: Analysis of Residual Connections [2]

Chapter 4 develops [2]'s theoretical and empirical analysis of residual connections in GNNs. It shows that adding residual connections makes a GCN equivalent to a sum of multiple shallow GCNs, each operating at a different neighborhood radius. The chapter concludes by identifying the limitations of residual connections in enabling genuinely deep GNNs when deep aggregation is needed.

Chapter 5: Partial Training [3]

Chapter 5 details [3]'s investigation of partially trained GNNs. It presents theoretical arguments showing that propagation through untrained layers preserves expected embedding variance, especially as network width increases, thereby reducing oversmoothing. Empirical results on benchmark datasets confirm that deep, partially trained GNNs maintain discriminative embeddings.

Chapter 6: Graph Informed Weight Initialization (G-Init) [4]

Chapter 6 covers [4]'s development of G-Init, a graph-specific initialization scheme. The chapter details how G-Init sets initial weight variances according to degree statistics to preserve activation and gradient scales across many layers. Experimental evaluations on node and graph classification benchmarks are then presented, demonstrating that GNNs initialized with G-Init train stably at depths that cause collapse under standard initialization methods.

Chapter 7: Slope-Modified ReLU (Slope2GNN) [5]

Chapter 7 explains [5]'s analysis of how activation functions affect oversmoothing. It introduces a slope-modified ReLU and provides a theoretical comparison of variance decay rates between standard and modified ReLU. Experimental results demonstrate that GNNs using the modified activation maintain higher embedding variance and achieve better performance than those using standard ReLU or other baseline activations.

Chapter 8: Discussion & Future Work

Chapter 8 synthesizes insights from Chapters 3–7, comparing how different approaches, address oversmoothing. It discusses limitations of existing methods, and potential avenues

for future research, including extending analyses to heterogeneous or dynamic (i.e., time-evolving) graphs, and exploring alternative activation or propagation schemes.

Chapter 2

Background & Literature Review

1 Preliminaries and Graph Representation

1.1 Basic Graph Concepts

A graph is a fundamental data structure in mathematics and computer science, defined as an ordered pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of vertices (or nodes), and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges as unordered or ordered pairs of vertices. The number of vertices is denoted by $N = |\mathcal{V}|$.
Graphs can be classified according to edge directionality:

- **Undirected Graphs**. In an undirected graph, each edge $\{u, v\} \in \mathcal{E}$ represents a bidirectional connection between vertices u and v. Consequently, the adjacency matrix of an undirected graph is symmetric, i.e., $A_{ij} = A_{ji}$ for all i, j.
- **Directed Graphs (Digraphs)**. In a directed graph, each edge $(u, v) \in \mathcal{E}$ is an ordered pair indicating a connection from vertex u to vertex v that does not imply a connection from v to u. The adjacency matrix of a directed graph is generally asymmetric.

The adjacency matrix $A \in \{0, 1\}^{N \times N}$ encodes the presence or absence of edges between vertices, defined by

$$A_{ij} = \begin{cases} 1, & \text{if there exists an edge from vertex } i \text{ to vertex } j, \\ 0, & \text{otherwise.} \end{cases}$$

In the special case of undirected graphs, $A_{ij} = A_{ji}$.
The degree matrix $D \in \mathbb{R}^{N \times N}$ is a diagonal matrix whose entries correspond to vertex degrees. For undirected graphs, the degree of vertex i is defined as

$$\deg(i) = \sum_{j=1}^{N} A_{ij},$$

and $D_{ii} = \deg(i)$. In directed graphs, one distinguishes:

$$\deg_{\text{in}}(i) = \sum_{i=1}^{N} A_{ij},$$

$$\deg_{\text{out}}(i) = \sum_{j=1}^{N} A_{ij},$$

so that D_{in} and D_{out} are diagonal matrices with the in-degrees and out-degrees on the diagonal, respectively.

1.2 Node Features and Embeddings

In many applications, vertices in a graph are associated with feature vectors that encode intrinsic or contextual attributes. We collect these vectors into a feature matrix $X \in \mathbb{R}^{N \times C}$, where each row $x_i \in \mathbb{R}^C$ represents the features of vertex i, and C denotes the input feature dimensionality.
An embedding is a transformation that projects these input feature vectors (usually) into a lower-dimensional space while preserving structural and attribute-based relationships. Formally, an embedding function

$$\mathcal{F} : \mathbb{R}^{N \times C} \longrightarrow \mathbb{R}^{N \times d}$$

maps the original feature matrix X to an embedding matrix $H = \mathcal{F}(X) \in \mathbb{R}^{N \times d}$, where $d < C$ is the output dimension of each layer. The goal is that the rows $h_i \in \mathbb{R}^d$ of H capture both the local connectivity patterns and the original features of vertex i. These embeddings serve as inputs for downstream tasks such as node classification [25, 26], link prediction [27, 28], and clustering [29].

1.3 Notation Overview

Throughout this thesis (unless stated otherwise), I use the following notation:

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: Graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$, where $|\mathcal{V}| = N$.
- $A \in \{0, 1\}^{N \times N}$: Adjacency matrix, where $A_{ij} = 1$ iff $(i, j) \in \mathcal{E}$.
- $D \in \mathbb{R}^{N \times N}$: Degree matrix, a diagonal matrix with vertex degrees on the diagonal.
- $X \in \mathbb{R}^{N \times C}$: Node input feature matrix, each row $x_i \in \mathbb{R}^C$ is the feature vector of vertex i.
- $H \in \mathbb{R}^{N \times d}$: Node embedding matrix produced by an embedding function $\mathcal{F}$.

These conventions establish a common framework for describing graph structures, node attributes, and their embedding representations.

2 Geometric learning

Geometric learning (also called geometric deep learning) is the study and development of machine learning models that explicitly exploit the geometric and topological structure inherent in data domains that are not naturally represented in Euclidean space. Traditional deep learning models, such as convolutional neural networks (CNNs) [30, 31, 32, 33] for grid-structured images and recurrent neural networks (RNNs) [34, 35, 36, 37] for sequential data, are designed to process data with regular structures. On the contrary, geometric deep learning extends these concepts (e.g., convolution, pooling, and feature aggregation) to handle data residing on non Euclidean domains, including graphs with irregular connectivity, spaces exhibiting nontrivial group symmetries (e.g., rotations on manifolds), and continuous curved spaces like Riemannian manifolds. At its core, geometric learning aims to identify and impose appropriate inductive biases, typically in the form of invariance or

equivariance, to transformations dictated by the underlying geometry. The desired outcome is learned representations which respect the symmetries and metric relationships of the data domain [10, 38, 39].

Formally, one may state that geometric learning consists of five interrelated categories, sometimes referred to as the "five G's": Grids (regular lattices), Groups (symmetry groups and their actions), Graphs (discrete relational structures), Geodesics (continuous manifolds with Riemannian metrics), and Gauges (local reference frames on manifolds) [10, 39]. In the grid setting, classical convolutional architectures exploit translational invariance by weight-sharing across spatial positions. In the group setting, group convolutions extend this idea by sharing weights across more general transformation groups (e.g., rotations, reflections), thereby ensuring equivariance to these transformations [40]. Graph-based methods further generalize this principle by defining convolutional-like operations on nodes and their neighbors, typically through message passing schemes that aggregate information from local neighborhoods [18, 25, 41, 42, 24]. On continuous manifolds, geometric learning methods define convolutional kernels with respect to geodesic distances, parallel transport, or local coordinate charts, thereby respecting the intrinsic curvature of the space [43, 44, 45, 46]. Finally, gauge-equivariant architectures explicitly model learned representations as fields (scalars, vectors, higher-order tensors) that transform appropriately under changes of local reference frames, guaranteeing that the network's output does not depend on arbitrary choices of coordinates [47, 48].

The emergence of geometric learning can be traced to multiple, diverse but somewhat convergent motivations. First, many real-world applications involve data that are intrinsically non-Euclidean: social networks [49], molecular structures [50, 51, 19], 3D meshes [52, 53], and sensor-network data [54] all exhibit underlying relational or manifold constraints that classical grid-based architectures (e.g., CNNs) fail to fully utilize. Early work on Graph Neural Networks (GNNs) recognized the need to propagate information along arbitrary graph topologies, but these methods often lacked a principled framework for incorporating domain symmetries or continuous geometric priors [55, 56]. Second, the success of group-equivariant CNNs (e.g., for rotation and reflection invariance in image processing) highlighted the importance of respecting known symmetries to improve sample efficiency and generalization [40, 57]. Third, in fields such as computational chemistry and physics, there existed a clear drive to develop architectures capable of handling data defined on manifolds (e.g., molecular surfaces, physical fields on curved domains) while ensuring that learned models satisfy physical constraints (e.g., equivariance to rigid motions). Taken together, these factors motivated a systematic effort to unify diverse aspects of non-Euclidean and symmetry-aware learning under a common theoretical framework, known as geometric deep learning [10].

The rationale for adopting a geometric learning paradigm is multifold. First, encoding known symmetries and invariances directly into the network architecture reduces the effective hypothesis space, thereby improving sample efficiency and reducing overfitting, which is of importance in domains where labeled data are scarce or expensive to obtain (e.g., molecular property prediction). Second, respecting geometric relationships (e.g., graph connectivity, manifold curvature) ensures that learned representations capture the true relational or spatial structure of the data, which often correlates strongly with the target prediction (e.g., functional motifs on protein surfaces, social influence propagation in networks). Third, equivariant architectures guarantee that the network's outputs transform predictably under domain symmetries, which is crucial for tasks requiring physical or semantic consistency (e.g., predicting forces in molecules that must be equivariant to rotations

and translations). As a result, geometric learning methods have demonstrated state-of-the-art performance across a wide range of tasks, including molecular property prediction [58, 59], point-cloud classification and segmentation [60, 61], social network analysis [24], and traffic forecasting on road-network graphs [62, 63, 9], often with substantially fewer learnable parameters than their competitors.

The historical development of geometric learning can be roughly partitioned into three overlapping phases. In the first phase, the primary focus was on early graph-based architectures such as recurrent graph neural networks [55] and spectral-domain methods based on the graph Laplacian [56, 64]. These spectral GNNs leveraged the eigenstructure of the graph Laplacian to define convolutional filters in the graph Fourier domain but suffered from limited scalability and poor transferability across graphs with differing topologies. The second phase witnessed the rise of spatial GNNs, most notably the message-passing neural network paradigm, where information is propagated along graph edges via learnable aggregation functions [18, 25]. This period also saw early attempts to define convolutions on non-Euclidean manifolds via geodesic patches or local charting, enabling deep learning on surfaces and meshes [43, 44]. The third phase has been characterized by efforts to unify these different approaches under a single mathematical framework: by identifying shared notions of symmetry, equivariance, and gauge symmetry, researchers have formulated a general framework that includes grid-based, graph-based, and manifold-based architectures [10, 48]. This unification enabled the design of architectures that can incorporate multiple geometric modalities simultaneously (e.g., graphs embedded on manifolds, or fields defined on meshes) and led to the development of highly expressive equivariant networks (e.g., tensor field networks, E(3)-equivariant GNNs) for applications in computer vision, computational biology, and physical sciences [50, 59].

In summary, geometric learning emerged from the recognition that many modern data analysis problems involve structured domains whose intrinsic geometry cannot be adequately captured by conventional Euclidean architectures. By generalizing convolutional and aggregation operations to graphs, groups, and manifolds, and by enforcing equivariance and invariance properties dictated by domain symmetries, geometric learning provides a principled framework for building deep neural networks that are both expressive and aligned with the underlying data geometry. This unifying perspective has not only led to theoretical insights regarding the expressive power and limitations of GNNs and manifold networks, but has also led to practical advancements in numerous application areas where respecting geometry is essential for obtaining robust and interpretable models [58, 65]. As our work focuses on understanding and mitigating oversmoothing in GNNs, an issue tied to the propagation of information along geometric structures, it is critical to situate our contributions within the broader context of geometric learning, whose foundational principles help illustrate the formulation of oversmoothing and motivate the design of effective mitigation strategies.

3 Spectral Graph Theory Foundations

3.1 Graph Operators

Spectral graph theory analyzes a graph structure through linear operators derived from its adjacency and degree matrices. The central operators are variants of the graph Laplacian, which summarize connectivity and are foundational in tasks such as clustering [66], community detection [67], and signal processing on graphs [68].

Combinatorial Laplacian. For an undirected graph $\mathcal{G}$, the combinatorial Laplacian $L \in \mathbb{R}^{N\times N}$ is defined by

$$L \;=\; D \;-\; A,$$

Since A is symmetric for undirected graphs, L is symmetric positive semi-definite. The entry $L_{ii} = \deg(i)$, and for $i \neq j$, $L_{ij} = -1$ if $(i,j) \in \mathcal{E}$ and 0 otherwise.

Normalized Laplacian. To mitigate the influence of heterogeneous degree distributions, one often employs the normalized Laplacian:

$$\mathcal{L} \;=\; D^{-\frac{1}{2}} L\, D^{-\frac{1}{2}} \;=\; I \;-\; D^{-\frac{1}{2}} A\, D^{-\frac{1}{2}}.$$

Here, I denotes the $N \times N$ identity matrix. The normalized Laplacian $\mathcal{L}$ is symmetric, with eigenvalues in the interval $[0, 2]$. This scaling enables comparison across graphs of different sizes or degree distributions and is widely used in spectral clustering [69, 66, 70, 71].

Random Walk Laplacian. Another useful variant is the random walk Laplacian:

$$L_{\text{rw}} \;=\; D^{-1} L \;=\; I \;-\; D^{-1} A.$$

Unlike $\mathcal{L}$, L_{rw} is generally asymmetric. It is related to the transition probability matrix $P = D^{-1}A$ of a random walk on the graph. Because P governs the dynamics of a Markov chain defined on the vertices, L_{rw} finds applications in the analysis of diffusion processes and PageRank-like algorithms [72, 73].

3.2 Eigendecomposition

The eigendecomposition of the graph Laplacian reveals key structural characteristics. For a symmetric Laplacian L (either combinatorial or normalized), the decomposition is given by

$$L \;=\; U\, \Lambda\, U^{\top},$$

where:

- $U = [u_1, u_2, \ldots, u_N] \in \mathbb{R}^{N\times N}$ is an orthonormal matrix whose columns u_i are eigenvectors of L.
- $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N) \in \mathbb{R}^{N\times N}$ is a diagonal matrix of non-negative eigenvalues sorted as $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$.

Interpretation of Eigenvalues. The spectrum $\{\lambda_i\}$ encodes connectivity properties of the graph. In particular:

- The multiplicity of the eigenvalue $\lambda = 0$ equals the number of connected components in the graph.
- The second smallest eigenvalue, λ_2, known as the algebraic connectivity or Fiedler value, quantifies how well connected the graph is: larger λ_2 indicates stronger overall connectivity [74].

Role in Clustering and Smoothing. The eigenvectors associated with the smallest non-zero eigenvalues (e.g., u_2 corresponding to λ_2) tend to vary slowly over the graph, meaning that adjacent vertices have similar values in these eigenvectors. This property underpins spectral clustering, where one uses the first k eigenvectors (excluding the trivial constant eigenvector) to embed vertices into a lower-dimensional space and then applies clustering algorithms such as k-means to identify communities [70]. Additionally, these eigenvectors serve as smooth basis functions for graph signal processing, enabling tasks like denoising and interpolation [68].

3.3 Spectral Filters

Spectral filtering on graphs manipulates the eigenvalues of the Laplacian to selectively enhance or suppress specific frequency components of signals defined over the vertices. Let $\mathbf{s} \in \mathbb{R}^N$ be a signal on the graph (i.e., s_i is the signal value at vertex i). Its graph Fourier transform is given by

$$\widehat{\mathbf{s}} \;=\; U^\top \mathbf{s},$$

and the inverse transform is $\mathbf{s} = U\,\widehat{\mathbf{s}}$. A spectral filter is defined as a function $g : \mathbb{R} \to \mathbb{R}$ applied to the eigenvalues. The filtered signal $\mathbf{s}'$ is obtained by

$$\mathbf{s}' \;=\; U\, g(\Lambda)\, U^\top \mathbf{s},$$

where $g(\Lambda) = \operatorname{diag}\big(g(\lambda_1), g(\lambda_2), \ldots, g(\lambda_N)\big)$.

Low-Pass Filtering. A low-pass filter $g(\lambda)$ assigns larger weights to smaller eigenvalues (low frequencies) and shrinks larger eigenvalues (high frequencies). Applying a low-pass filter suppresses high-frequency variations in the signal, promoting smoothness across strongly connected regions of the graph. This is beneficial for tasks such as semi-supervised learning, where labels propagate smoothly based on graph connectivity, and for denoising graph signals by removing high-frequency noise [75, 76].

Effect on Information Propagation. The design of a spectral filter directly affects how information spreads on the graph. Low-pass filters encourage information to diffuse along densely interlinked areas, facilitating the discovery of communities and robust label propagation. However, excessive smoothing can eliminate crucial high-frequency features that indicate sharp transitions or boundaries between communities. Thus, selecting or learning an appropriate filter $g(\cdot)$ involves balancing the trade-off between smoothing and preserving important structural information [20].

4 Graph Neural Networks (GNNs)

4.1 Message Passing Framework

Graph Neural Networks (GNNs) have emerged as a powerful paradigm for learning representations of graph-structured data. Central to many GNN architectures is the Message Passing Neural Network (MPNN) framework, introduced by Gilmer et al. [18]. This framework provides a unifying perspective for various GNN models by abstracting the process of information propagation through graphs.

Aggregate and Update Functions. In the MPNN framework, the learning process can be described as a series of message passing iterations, each consisting of two primary steps: message aggregation and node state update. For a graph $\mathcal{G}$ with node features x_u and edge features e_{uw}, the operations at the l-th layer are defined as follows:

$$m_u^{(l)} = \sum_{w \in \mathcal{N}(u)} M_l\big(h_u^{(l)}, h_w^{(l)}, e_{uw}\big),$$
$$h_u^{(l+1)} = U_l\big(h_u^{(l)}, m_u^{(l)}\big),$$

where $h_u^{(l)}$ represents the hidden state of node u at layer l, M_l is the message function that computes messages from neighboring nodes at layer l, and U_l is the update function that integrates the aggregated message $m_u^{(l)}$ with the current state $h_u^{(l)}$. These functions are typically parameterized by neural networks and are designed to be permutation-invariant to ensure consistency across different (but equivalent) graph structures.

Receptive Fields and Depth Implications. The depth of a GNN, determined by the number of message passing layers, directly influences the receptive field of each node. We term as the receptive field of each node the subset of the graph from which it can aggregate information. Specifically, after L layers, a node's representation incorporates information from nodes up to L hops away. While increasing depth allows for capturing broader contextual information, it also introduces challenges such as oversmoothing, where node representations become indistinguishable, and vanishing gradients, which hinder effective training. Consequently, there is a trade-off between depth and the ability to preserve meaningful node representations.

Advancements in Message Passing Frameworks. Since the inception of the MPNN framework, several extensions and modifications have been proposed to address its limitations and enhance its capabilities:

- **Hierarchical Message Passing.** To capture long-range dependencies and high-order neighborhood information, hierarchical message passing frameworks have been developed. These approaches group nodes into multi-level structures, enabling efficient information propagation across different scales of the graph [77, 78].
- **DRew: Dynamically Rewired Message Passing with Delay.** Instead of applying message passing uniformly across the entire graph at every layer, DRew gradually densifies the graph structure and introduces layer-dependent skip connections. This optional, distance-based rewiring allows long-range node interactions to emerge progressively, improving performance and alleviating oversquashing on tasks requiring global context [79].
- **Cooperative Graph Neural Networks (Co-GNNs).** Treats each node as a strategic agent that dynamically chooses whether to listen, broadcast, both, or isolate at each layer. This flexible, adaptive message-passing paradigm enables nodes to regulate information flow based on local structure and learned context, enhancing expressivity, especially for long-range dependencies and heterophilic graphs [80].

These advancements reflect the ongoing evolution of message passing frameworks in GNNs, aiming to address challenges related to scalability, expressiveness, and the effective capture of complex graph structures.

4.2 Prominent GNN Architectures

In the evolution of Graph Neural Networks, several architectures have been proposed to effectively capture both structural and feature information in graph-structured data. Among these, Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), GraphSAGE, Chebyshev Networks (ChebNets), Personalized Propagation of Neural Predictions (PPNP) and its approximation (APPNP), Simplified Graph Convolution (SGC), GCNII, and Graph Transformers have emerged as prominent models. Each architecture introduces distinct mechanisms to aggregate and transform node features, addressing various challenges such as computational efficiency, inductive learning, and preservation of high-frequency information.

Graph Convolutional Networks (GCNs). Introduced by Kipf and Welling [25], GCNs generalize the notion of convolutional filters to graphs by leveraging a first-order approximation of spectral graph convolutions. The layer-wise propagation rule for an L-layer GCN is:

$$H^{(l+1)} = \sigma\Big(\tilde{D}^{-1/2}\,\tilde{A}\,\tilde{D}^{-1/2}\,H^{(l)}\,W^{(l)}\Big), \tag{2.1}$$

where:

- $\tilde{A} = A + I$ is the adjacency matrix after self-loops addition,
- $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$,
- $H^{(0)} = X$ is the input feature matrix, and $H^{(l)} \in \mathbb{R}^{N \times d_l}$ denotes the matrix of node representations at layer l,
- $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ is the trainable weight matrix at layer l,
- $\sigma(\cdot)$ is a pointwise non-linearity (e.g., ReLU).

This formulation can be interpreted as Laplacian smoothing, where each node's representation is updated by a weighted average of its own and its neighbors' features. Although stacking multiple GCN layers allows for capturing information from higher-order neighborhoods, excessive depth leads to oversmoothing, causing node embeddings to become indistinguishable [20].

Residual Graph Convolutional Networks (ResGCNs). ResGCNs [23] enhance standard GCNs by introducing residual connections that facilitate the training of deeper architectures and mitigate issues like oversmoothing. In a typical ResGCN layer, the output is computed as:

$$H^{(l+1)} = \sigma\left(\hat{A}H^{(l)}W^{(l)} + H^{(0)}\right),$$

where:

- $\hat{A}$ is the the augmented symmetrically normalized adjacency matrix after self-loop addition,
- $W^{(l)}$ is the trainable weight matrix.

The addition of $H^{(0)}$ implements a skip connection from the input features, allowing the model to retain original information across layers.

Graph Attention Networks (GATs). To alleviate the limitation of uniform neighbor aggregation in GCNs, Veličković et al. [41] proposed GATs, which utilize self-attention mechanisms on graph neighborhoods. For each node i, GAT computes attention coefficients α_{ij} for each neighbor $j \in \mathcal{N}(i)$ as follows:

$$e_{ij} \;=\; \text{LeakyReLU}\big(\vec{a}^{\top}[\,W\,h_i^{(l)} \,\|\, W\,h_j^{(l)}]\big), \quad \alpha_{ij} \;=\; \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}(i)}\exp(e_{ik})},$$

where:

- $h_i^{(l)} \in \mathbb{R}^{d_l}$ is the input feature of node i at layer l,
- $W \in \mathbb{R}^{d_l \times d_{l+1}}$ is the trainable weight matrix,
- $\vec{a} \in \mathbb{R}^{2d_{l+1}}$ is a learnable weight vector,
- $\|$ denotes the concatenation operation.

The updated feature of node u is computed as:

$$h_u^{(l+1)} \;=\; \sigma\Big(\sum_{j\in\mathcal{N}(u)} \alpha_{ij}\,W\,h_j^{(l)}\Big),$$

often extended to multi-head attention by concatenating or averaging the outputs of K independent attention heads. GATs adaptively weight neighbor contributions according to learned relevance scores, thereby enabling the model to focus on the most informative neighboring nodes. This mechanism also inherently handles graphs with variable node degree distributions without explicit normalization.

GraphSAGE. Hamilton et al. [24] introduced GraphSAGE to address the need for inductive learning on large and evolving graphs. Instead of learning a single embedding for each node in a transductive manner, GraphSAGE learns aggregation functions that generalize to unseen nodes. At each layer l, the embedding of node u is updated via:

$$h_u^{(l+1)} \;=\; \sigma\Big(W^{(l)} \cdot \text{AGGREGATE}^{(l)}\big(\{\,h_u^{(l)}\} \cup \{\,h_v^{(l)} \mid v \in \mathcal{N}(u)\}\big)\Big),$$

where:

- $\text{AGGREGATE}^{(l)}$ is a permutation-invariant aggregator such as mean, max-pooling, or an LSTM-based function,
- $W^{(l)}$ is a trainable weight matrix at layer l,
- $h_u^{(0)} = x_u$ is the initial node feature.

To achieve scalability, GraphSAGE samples a fixed-size set of neighbors for each node at each layer, enabling mini-batch training and inductive inference on nodes not seen during training. This sampling strategy significantly reduces computational cost on large graphs and supports dynamic graph settings.

Chebyshev Networks (ChebNets). Defferrard et al. [64] proposed ChebNets to approximate spectral graph convolutions using Chebyshev polynomials. The K-localized filter at layer l is defined as:

$$x * g_\theta^{(l)} \;\approx\; \sum_{k=0}^{K} \theta_k^{(l)}\, T_k(\tilde{L})\, x,$$

where:

- T_k is the Chebyshev polynomial of order k,
- $\tilde{L} = \frac{2}{\lambda_{\max}} L - I$ is the scaled Laplacian, with $\lambda_{\max}$ being the largest eigenvalue of the combinatorial (or normalized) Laplacian L,
- $\theta_k^{(l)}$ are learnable parameters for layer l,
- $*$ denotes the convolution operation on the graph.

By truncating the polynomial expansion at order K, ChebNets achieve K-hop localized filtering without explicitly computing eigenvectors, reducing the computational complexity to $\mathcal{O}(K|\mathcal{E}|)$. This framework captures high-frequency components more accurately than first-order models like GCNs, improving expressiveness on graphs with complex structural patterns.

Personalized Propagation of Neural Predictions (PPNP) and Approximate PPNP (APPNP). Klicpera et al. [16] introduced PPNP, which integrates Personalized PageRank (PPR) with neural network outputs to enhance semi-supervised node classification. Let $f_\theta(X) = H^{(0)} \in \mathbb{R}^{N \times d_{out}}$ be the initial neural predictions (logits) for d_{out} classes. PPNP computes the final prediction matrix Z as:

$$Z \;=\; \alpha \left(I - (1-\alpha)\, \hat{A}\right)^{-1} H^{(0)},$$

where:

- $\hat{A} = \tilde{D}^{-1/2}\, \tilde{A}\, \tilde{D}^{-1/2}$ is the the augmented symmetrically normalized adjacency matrix after self-loop addition,
- $\alpha \in (0, 1)$ is the teleport (restart) probability controlling the balance between local predictions and propagated values.

Because direct inversion of $(I - (1-\alpha)\, \hat{A})$ is computationally expensive for large N, APPNP approximates this propagation via power iteration:

$$H^{(k+1)} \;=\; (1-\alpha)\, \hat{A}\, H^{(k)} \;+\; \alpha\, H^{(0)}, \qquad H^{(K)} = softmax\big((1-\alpha)\, \hat{A}\, H^{(K-1)} \;+\; \alpha\, H^{(0)}\big),$$

where K is the number of propagation steps. APPNP decouples feature transformation (neural network) and label propagation (PPR), allowing efficient long-range dependency capture without excessive parameterization.

Personalized PageRank Graph Neural Network (PPRGNN). To address the challenges of oversmoothing and limited receptive fields in deep GNNs, Roth and Liebig [12] introduced PPRGNN, which integrates the concept of personalized PageRank into the GNN framework. Specifically, the PPRGNN-like network family [12] is defined as follows:

$$H^{(l+1)} = \sigma\left(\alpha_l \hat{A} H^{(l)} W^{(l)} + f_\theta(X)\right), \tag{2.2}$$

where:

- $f_\theta(\cdot)$ is a neural network (i.e. an MLP), which takes as input the initial nodes' features.

PPRGNN sets $W^{(l)} = W \; \forall l$ and $\alpha_l = \frac{1}{l}$. The model's formulation guarantees convergence to a unique solution without imposing constraints on the architecture or parameters. Moreover, PPRGNN maintains linear time complexity and constant memory usage, making it scalable to large graphs.

Simplified Graph Convolution (SGC). Wu et al. [17] propose SGC by collapsing multiple GCN layers and removing non-linearities to yield:

$$\hat{Y} \;=\; \text{softmax}\big(\hat{A}^L \, X \, W\big),$$

where:

- $\hat{A} = \tilde{D}^{-1/2} \, \tilde{A} \, \tilde{D}^{-1/2}$ is the the augmented symmetrically normalized adjacency matrix after self-loop addition,
- L is the number of propagation steps precomputed before training,
- $W \in \mathbb{R}^{C \times d_{out}}$ maps features to logits for d_{out} classes.

By precomputing $\hat{A}^L X$, SGC reduces the model to a single linear classifier, reducing training complexity while maintaining competitive performance. This simplification is particularly effective when the benefit of non-linear feature transformations is marginal compared to the costs of training deep GCNs.

GCNII. Chen et al. [11] address oversmoothing and vanishing gradients in deep GCNs by introducing residual connections and identity mapping. The layer-wise propagation rule for GCNII is:

$$H^{(l+1)} = \sigma\left(\left((1-\alpha_l)\hat{A}H^{(l)} + \alpha_l H^{(0)}\right)\left((1-\beta_l)I_n + \beta_l W^{(l)}\right)\right),$$

where:

- $\hat{A} = \tilde{D}^{-1/2} \, \tilde{A} \, \tilde{D}^{-1/2}$ is the the augmented symmetrically normalized adjacency matrix after self-loop addition,
- $H^{(0)} = X$ is the input feature matrix,
- α_l and β_l are hyperparameters controlling the strength of residual connection and identity mapping at layer l,
- $W^{(l)}$ is the trainable weight matrix for layer l.

Parameter α_l allows the model to preserve a fraction of the initial features $H^{(0)}$, while β_l controls the contribution of the identity mapping. This dual mechanism allows GCNII to train networks with dozens of layers without significant performance degradation, achieving state-of-the-art results on benchmark datasets.

Graph Transformers. Graph Transformers adapt the self-attention mechanism from Transformer architectures to graph data, enabling the model to capture both local and global dependencies. Unlike traditional GNNs that restrict attention to immediate neighbors, Graph Transformers compute attention scores between all node pairs, defined by:

$$\text{Attention}(Q, K, V) \;=\; \text{softmax}\Big(\frac{Q\,K^\top}{\sqrt{d_k}}\Big)\,V,$$

where:

- $Q = XW_Q$, $K = XW_K$, and $V = XW_V$ are learnable linear projections of the input feature matrix $X \in \mathbb{R}^{N\times C}$,
- d_k is the dimensionality of the key vectors, used for scaling,
- Softmax is applied row-wise over the N keys for each query.

The resulting attention matrix has size $N \times N$, allowing each node to attend to every other node. To incorporate graph structure, Graph Transformers often add positional encodings derived from the Laplacian eigenvectors or shortest-path distances, biasing the attention mechanism toward meaningful graph neighborhoods [81, 82]. Empirical studies show that Graph Transformers excel in tasks requiring modeling of long-range interactions, such as molecular property prediction [83, 84] and computer vision [85]. However, the $\mathcal{O}(N^2)$ complexity of full pairwise attention limits scalability, motivating sparse or approximate attention strategies.

Each of the above architectures extends the core message passing paradigm of GNNs in order to address specific limitations (e.g., computational efficiency, modeling of global dependencies). The choice among these models depends on the downstream task requirements, graph size, and attribute heterogeneity.

4.3 Extensions and Variants

The landscape of GNNs has rapidly expanded, leading to numerous extensions and variants beyond the standard message passing paradigm. In this section, I discuss approaches that enrich the framework by addressing specific structural properties and application requirements. I first distinguish between spatial and message passing GNNs and then present specialized models such as graph recurrent networks, pooling mechanisms, heterogeneous models, and dynamic graph approaches.

Spatial vs. message passing GNNs. Spatial GNNs emphasize the direct modeling of local, coordinate-based relationships among nodes. These methods often define convolution-like operations by imposing a local coordinate system or neighborhood kernel on each node, analogous to image convolution, which can be advantageous when nodes are embedded in Euclidean space with inherent spatial semantics [86, 25]. In contrast, message passing GNNs rely on permutation invariant aggregation functions over a node's neighbors and do not assume any underlying spatial embedding. While spatial models may offer more interpretable local filters, message passing approaches are generally more flexible and applicable to arbitrary graph topologies without requiring explicit spatial regularities [6].

Graph Recurrent Models. For sequential or temporal data on graphs, recurrent architectures extend standard GNNs by incorporating mechanisms that maintain and update temporal hidden states. Graph Recurrent Neural Networks (GRNNs) combine temporal recurrent units (e.g., LSTM or GRU units) with graph propagation, enabling node representations to evolve over time [9, 87]. At each time step t, the hidden state $h_u^{(t)}$ of node u is updated both by aggregating messages from its neighbors at time t and by recurrently integrating its previous state $h_u^{(t-1)}$. Such formulations effectively capture time-varying dependencies and are particularly well-suited for tasks like traffic forecasting [63, 88], event prediction [89], and temporal recommendation systems [62].

Graph Pooling Methods. Pooling in GNNs aims to learn hierarchical, multi-scale representations utilizing the graph structure. Pooling layers reduce the number of nodes by selecting or aggregating a subset of vertices, facilitating global representation learning for tasks such as graph classification. Techniques include:

- **DiffPool** [90]: Learns soft cluster assignments via a differentiable assignment matrix $S^{(l)} \in \mathbb{R}^{N \times k}$, where k is the number of clusters. The coarsened node features and adjacency matrix are computed as $H^{(l+1)} = (S^{(l)})^\top H^{(l)}$ and $A^{(l+1)} = (S^{(l)})^\top A^{(l)} S^{(l)}$, respectively.

- **Top-K Pooling** [91]: Scores each node based on its importance and retains the top k nodes, where k is a learnable fraction of N. The filtered features and adjacency matrix are computed by selecting only the highest-scoring vertices.

These pooling mechanisms enable the extraction of high-level, global features and support multi-scale analysis of graph structures.

Heterogeneous Graph Models. In many real-world applications, graphs contain multiple types of nodes and edges. Heterogeneous Graph Neural Networks (HGNNs) account for this diversity by using distinct aggregation and transformation functions for each relation type. For instance:

- **Relational GCNs (R-GCNs)** [92]: Define separate weight matrices W_r for each relation r and aggregate neighbor messages according to relation type:

$$h_u^{(l+1)} = \sigma\Big(\sum_{r \in \mathcal{R}} \sum_{v \in \mathcal{N}_r(u)} \frac{1}{c_{u,r}} W_r^{(l)} h_v^{(l)} + W_0^{(l)} h_u^{(l)}\Big),$$

  where $\mathcal{R}$ is the set of relation types, $\mathcal{N}_r(u)$ denotes neighbors of u under relation r, and $c_{u,r}$ is a normalization constant.

- **Heterogeneous Graph Transformer (HGT)** [93]: Extends transformer-based attention by incorporating node- and edge-type embeddings into the multi-head attention mechanism, enabling fine-grained interactions between heterogeneous entities.

By distinguishing among node and edge semantics, HGNNs improve performance in tasks such as knowledge graph completion, recommendation systems, and multi-relational data modeling [94].

Dynamic Graph Models. Static GNNs assume a fixed graph structure, which may not capture temporal evolution in applications like social networks or communication systems. Dynamic GNNs incorporate temporal information through one of the following approaches:

- **Snapshot-Based Models** [95]: These models represent a dynamic graph as a sequence of discrete snapshots $\{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots\}$, applying GNNs independently to each snapshot. Temporal dependencies are captured by incorporating recurrent mechanisms or temporal encoding across snapshots.
- **Continuous-Time Dynamic GNNs** [96]: These models treat dynamic graphs as sequences of timestamped events (u, v, t), updating node representations in real-time. EvolveGCN, for instance, evolves GCN parameters over time using recurrent neural networks, enabling the model to adapt to the graph's temporal dynamics.
- **Temporal Message Passing** [97]: This approach extends traditional message passing by integrating temporal information, allowing messages to be weighted based on the time difference between events. Such mechanisms enable the model to capture temporal patterns and recency effects in dynamic graphs.

Dynamic GNNs thereby capture evolving structures and can adapt representations as new nodes or edges appear, making them suitable for tasks such as anomaly detection, link prediction, and dynamic recommendation.

Overall, these extensions and variants demonstrate the broad applicability of GNN architectures across diverse domains. By augmenting the fundamental message passing framework with spatial features, temporal dynamics, hierarchical abstractions, and node- or edge-type heterogeneity, researchers have progressively enhanced the representational capacity and empirical performance of GNNs on complex, real-world problems.

5 Oversmoothing in Graph Neural Networks

5.1 Definition and Intuition

Oversmoothing is a phenomenon observed in Graph Neural Networks wherein, as the number of layers L increases, the node feature representations become increasingly similar, ultimately converging to nearly identical values. This convergence prevents the model from distinguishing between nodes, thereby degrading performance on downstream tasks such as node classification and link prediction. The root cause of oversmoothing resides in the repeated aggregation (or "mixing") of neighboring features, which, over multiple layers, drives the hidden representations toward a constant subspace.
Formally, let $H^{(l)} \in \mathbb{R}^{N \times d_l}$ be the matrix of node features at layer l, where $H^{(0)} = X$ is the input feature matrix, and d_l is the dimensionality at layer l. Denote by $h_i^{(l)} \in \mathbb{R}^{d_l}$ the representation of node i at layer l. Oversmoothing implies that, as $l \to \infty$,

$$\lim_{l \to \infty} \left\| h_i^{(l)} - h_j^{(l)} \right\|_2 \;=\; 0, \quad \forall\, i, j \in V,$$

where $\|\cdot\|_2$ denotes the Euclidean norm. Equivalently, the row vectors of $H^{(l)}$ collapse toward a common subspace, making it difficult to linearly (or nonlinearly) separate nodes belonging to different classes.

An intuitive analogy is to view each GNN layer as applying a low-pass filter on the graph: if $\mathcal{L}$ is a normalized graph Laplacian, then a typical GCN layer multiplies the feature matrix by $\tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2} = I - \mathcal{L}$, which reduces high-frequency components of the graph signal. Empirical investigations have shown that oversmoothing may occur even in networks with as few as two to four layers, depending on factors such as graph topology, degree distribution, and choice of aggregation function. For example, on graphs with high homophily, the mixing is stronger and oversmoothing can appear more rapidly; conversely, on heterophilous graphs, the mixing may be less severe but still leads to signal degradation over many layers [98, 99, 100]. Furthermore, different architectures (e.g., GATs, GraphSAGE) exhibit varying resistance to oversmoothing, often tied to their normalization and attention mechanisms.
In summary, oversmoothing arises because the iterative neighborhood aggregation in GNNs acts analogously to repeatedly applying a low-pass filter, which homogenizes the feature vectors across nodes. Understanding this phenomenon is crucial for designing architectures (e.g., using residual connections, identity mapping, or adaptive filters) that preserve discriminative information while maintaining the benefits of multi-hop aggregation [11, 101].

5.2 Empirical Evidence

Empirical studies have consistently demonstrated that increasing the depth of Graph Neural Networks often leads to performance degradation, a phenomenon primarily attributed to oversmoothing. This section delves into the empirical observations and analyses that shed light on the depth-performance trade-off and the associated variance decay across layers in GNNs.

Depth-Performance Trade-off. The performance of GNNs does not monotonically improve with increased depth. Instead, there exists an optimal number of layers beyond which the model's performance shrinks. This degradation is evident in tasks such as node classification, where deeper GNNs fail to maintain discriminative node representations. For instance, studies have shown that even shallow GCNs exhibit notable performance drops beyond a certain depth, often around eight layers, despite the smoothing rate of graph propagation not being particularly rapid [20, 21, 7]. This observation suggests that factors beyond mere smoothing, such as optimization difficulties, contribute to the performance decline in deeper GNNs.

Variance Decay Across Layers. A critical aspect of oversmoothing is the decay of feature variance across layers. As GNNs propagate information through successive layers, the variance of node features diminishes, leading to homogenized representations. This variance decay has been quantitatively measured using metrics like Dirichlet energy, which assesses the smoothness of node features over the graph. Empirical evaluations have demonstrated that some architectures exhibit an exponential convergence of layer-wise Dirichlet energy to zero, indicating significant oversmoothing, whereas others maintain approximately constant Dirichlet energy across layers, reflecting a slower rate of homogenization [102, 103].

In summary, empirical evidence underscores the challenges posed by oversmoothing in deep GNNs, highlighting the fragile balance between depth and performance. Understanding the variance dynamics across layers is crucial for designing GNN architectures that effectively leverage multiple layers while avoiding feature homogenization.

5.3 Theoretical Explanations

The phenomenon of oversmoothing in GNNs has been extensively studied through the lens of spectral graph theory. This section explores the theoretical foundations of oversmoothing, focusing on how the graph's spectral properties and the depth of the GNN jointly influence the rate at which node embeddings converge, thereby undermining their discriminability.

Spectral Perspective on Oversmoothing. GNNs, particularly those based on message passing mechanisms, can be interpreted as performing iterative smoothing operations on node features. Each layer in a GNN aggregates information from neighboring nodes, effectively applying a form of low-pass filtering that suppresses high-frequency components (i.e., local variations) in the feature space. Repeatedly applying such smoothing operations progressively removes local, high-frequency information from X, leaving only the global, low-frequency components. While moderate smoothing can denoise features and capture global consistency, excessive smoothing causes all node signals to converge toward the principal eigenvector of $\mathcal{L}$ (or a constant vector when the graph is connected), thus eliminating distinctions between nodes [20, 7].

Role of Graph Spectrum and Adjacency Eigenvalues. The spectral properties of the graph, encoded in the eigenvalues and eigenvectors of matrices like the adjacency matrix A or the normalized Laplacian $\mathcal{L}$, play a crucial role in understanding oversmoothing. The eigenvalues of these matrices determine the frequency components of the graph signals. In particular, the largest eigenvalue corresponds to the lowest frequency (global structure), while the smallest eigenvalue corresponds to the highest frequency (local variations).
When a GNN applies a convolution operation, it effectively modulates these frequency components. Repeated applications of such operations tend to diminish high-frequency components, which are crucial for distinguishing nodes from different classes. As evidenced in recent studies [104, 105], this suppression of high-frequency signal reduces the model's ability to separate closely related yet distinct node features, thereby leading to oversmoothing and diminishing classification performance.

Impact of Model Depth. The depth of a GNN, defined by the number of layers, directly influences the extent of smoothing applied to node features. As the number of layers increases, the repeated aggregation operations can cause the node features to converge towards a subspace dominated by the leading eigenvectors of the graph Laplacian. This convergence implies that the node features become increasingly similar, hindering the model's ability to capture discriminative information.
Theoretical analyses have shown that the rate of this convergence is exponential with respect to the number of layers. Specifically, for certain classes of graphs, the difference between node features diminishes exponentially as the depth increases, leading to a rapid onset of oversmoothing even in relatively shallow networks [7].
[7] generalized the idea in [20] considering also, that the ReLU activation function maps to a positive cone. They characterize oversmoothing as convergence to a subspace and provide an estimate of the speed of convergence to this subspace. That speed is expressed as the distance of node representations from the oversmoothing subspace M (details can be found in [7]).

Theorem 1 ([7]). *Let* $s_l = \prod_{h=1}^{H_l} s_{lh}$ *where* s_{lh} *is the largest singular value of weight matrix* W_{lh} *and s = sup*$_{l \in N^+} s_l$*. Then the distance from the oversmoothing subspace* M *is measured as follows:* $d_M(X^{(l)}) = O((s\lambda)^l)$*, where l is the layer number,* λ *is the smallest non-zero eigenvalue of* $I - \hat{A}$*, and if* $s\lambda < 1$ *the distance from the oversmoothing subspace (*d_M*) exponentially approaches zero.*

According to [7], deep GCNs are susceptible to oversmoothing, with the model's only defense against this effect being the product of the largest singular values of the weight matrices. This multiplication arises from the 1-Lipschitzness of ReLU, combined with the propagation scheme of GCN (Equation 2.1). If we remove the activation functions from Equation 2.1, we obtain the final node representations by $H^{(L)} = A^L X \prod_{i=1}^{L} W^{(i)}$. Therefore, the product of the largest singular values of the weight matrices serves as the upper bound of the appearing product, providing additional insight into the findings of [7].
Theorem 1 indicates that increasing the network's depth results in node representations being closer to the subspace M. If the products of maximum singular values and the smallest eigenvalue of the Laplacian of each layer are small, then node representations asymptotically approach M, regardless of the initial values of node features. Extending the aforementioned theorem, the authors conclude to the following estimate about the speed of convergence to the oversmoothing subspace.

Corollary 2 ([7]). *Let* $s = \sup_{l \in N_+} s_l$*. then* $d_M(X^{(l)}) = O((s\lambda)^l)$*, where l is the layer number and if* $s\lambda < 1$ *the distance from oversmoothing subspace exponentially approaches zero. Where* λ *is the smallest non-zero eigenvalue of I -* $\hat{A}$*.*

According to the authors, any sufficiently deep GCN will inevitably suffer from oversmoothing, under some conditions (details can be found in [7]).

In summary, the theoretical exploration of oversmoothing through spectral graph theory provides valuable insights into the limitations of deep GNNs. By analyzing the interplay between graph spectra, adjacency eigenvalues, and model depth, we can devise strategies to design more robust and expressive GNN architectures.

6 Quantifying and Detecting Oversmoothing

6.1 Metrics and Measures

Effectively quantifying oversmoothing in Graph Neural Networks is crucial for understanding and mitigating its impact on model performance. Several metrics have been proposed to measure the extent of oversmoothing, focusing on aspects such as pairwise embedding distances, spectral properties, and information-theoretic measures.

Pairwise Embedding Distances. One intuitive approach to assess oversmoothing involves analyzing the pairwise distances between node embeddings. As oversmoothing progresses, node representations become increasingly similar, leading to a reduction in these distances. Metrics like the Total Pairwise Squared Distance (TPSD) and the row-difference (row-diff) measure have been introduced to quantify this effect. The row-diff metric calculates the

average pairwise distance between node embeddings, providing a direct measure of representation similarity across the graph [16]. Additionally, the Relative Inter-Class Distance (RICD) metric has been proposed to evaluate the difference in similarity between inter-class and intra-class node pairs, offering insights into how oversmoothing affects class separability [13].

Laplacian-Based Indices. Spectral properties of the graph, particularly those derived from the Laplacian matrix, offer another avenue for quantifying oversmoothing. The Dirichlet energy is a prominent metric in this context, measuring the smoothness of node features over the graph structure. It is defined as:

$$\mathcal{E}(X) = \text{tr}(X^\top \mathcal{L} X),$$

A decrease in Dirichlet energy across layers indicates increased smoothness, with values approaching zero signifying complete homogenization of node features [103]. Compared to other metrics like the Mean Average Distance (MAD) [106], Dirichlet energy has been found to be more stable and reliable for detecting oversmoothing.

Mutual Information and Representation Collapse Metrics. Information-theoretic measures, such as mutual information (MI), have been employed to assess the extent of representation collapse due to oversmoothing. MI quantifies the amount of information shared between input features and their corresponding embeddings. A decline in MI across layers suggests that node embeddings are losing informative content, converging towards indistinguishable representations. While MI provides a theoretical framework for understanding oversmoothing, its practical computation can be challenging, and its effectiveness may vary depending on the specific characteristics of the graph and the GNN architecture.

In summary, these metrics provide a comprehensive toolkit for quantifying and detecting oversmoothing in GNNs. Employing these metrics can assist in detecting oversmoothing issues and guiding the development of architectures and training strategies to mitigate its effects.

6.2 Empirical Detection Protocols

Empirical evaluation of oversmoothing in Graph Neural Networks necessitates systematic protocols that can reliably detect and quantify the phenomenon. Two primary approaches have been employed in the literature: the use of synthetic benchmarks and layer-wise variance tracking.

Synthetic Benchmarks. Synthetic datasets provide controlled environments to study oversmoothing by allowing precise manipulation of graph properties such as homophily, degree distribution, and connectivity. For instance, the use of synthetic graphs with adjustable homophily ratios enables the assessment of GNN performance under varying degrees of node similarity. Studies have demonstrated that in low-homophily settings, GNNs are more prone to oversmoothing, as the aggregation of dissimilar neighbor features leads to feature homogenization [107]. Additionally, synthetic benchmarks facilitate the evaluation of oversmoothing mitigation techniques by providing a consistent testing ground across different graph structures.

Layer-Wise Variance Tracking. Monitoring the variance of node representations across layers serves as a practical method for detecting oversmoothing. As GNNs deepen, the variance of node features tends to decrease, indicating a convergence towards similar representations. Metrics such as the Mean Average Distance (MAD) and its variant MADGap have been proposed to quantify this effect. MAD measures the average pairwise distance between node embeddings, while MADGap assesses the difference in MAD between intra-class and inter-class node pairs. A declining MADGap across layers signifies a loss of class discriminability due to oversmoothing [106]. Furthermore, tracking the Dirichlet energy of node features across layers provides insights into the smoothness of the feature space, with decreasing energy values indicating increased oversmoothing [103].

In practice, combining synthetic benchmarks with layer-wise variance tracking offers a comprehensive approach to empirically detect and analyze oversmoothing in GNNs. These protocols enable systematic evaluation of the the impact of architectural choices and training strategies on the preservation of informative node representations.

6.3 Challenges and Gaps in Quantifying and Detecting Oversmoothing

Despite significant advancements in understanding oversmoothing in Graph Neural Networks, several challenges persist in accurately quantifying and detecting this phenomenon. These challenges include scalability issues, the absence of closed-form layer analysis, and a limited theoretical-practical bridging.

Scalability Constraints in Oversmoothing Metrics. Traditional metrics for assessing oversmoothing, such as the Dirichlet energy, often fall short in large-scale or deep GNNs. These metrics may not reliably indicate oversmoothing in practical scenarios, as they can provide meaningful insights only under specific conditions, such as very deep networks or strict assumptions on network weights and feature representations. Consequently, there is a need for more scalable and robust metrics that can effectively capture oversmoothing across diverse GNN architectures and depths.

Absence of Closed-Form Layer Analysis. The absence of closed-form, layer-wise analytical expressions in GNNs hinders the exact quantification of oversmoothing dynamics. Even though some studies have provided asymptotic analyses, they often do not offer insights into the finite-depth behavior of GNNs, which is critical for practical applications. The absence of such analytical tools limits our ability to predict and mitigate oversmoothing effectively.

Limited Bridging Between Theory and Practice. There exists a gap between theoretical models of oversmoothing and their practical implications. Many theoretical frameworks rely on simplified assumptions that may not hold in real-world scenarios, leading to discrepancies between predicted and observed behaviors. For example, spectral and dynamical system frameworks provide insights, but they rely on simplifying assumptions that limit precise characterization of how oversmoothing evolves across network depth [102]. Bridging this gap requires the development of theoretical models that account for the complexities of practical GNN implementations, including various architectures, training regimes, and data characteristics.

In summary, to address these challenges we need to develop scalable metrics, derive closed-form analytical tools, and create theoretical models that align closely with practical observations. Such advancements will enhance our ability to quantify and detect oversmoothing, ultimately leading to more robust and effective GNNs.

7 Mitigating Oversmoothing

7.1 Architectural Innovations

To mitigate oversmoothing in Graph Neural Networks, several architectural innovations have been proposed. These designs aim to preserve discriminative node representations across multiple layers by enhancing information flow, capturing multi-scale neighborhood features, and decoupling feature transformation from propagation. In this section, I discuss three notable architectures: Jumping Knowledge Networks, MixHop, and Decoupled GNNs.

Jumping Knowledge Networks [108]. Jumping Knowledge (JK) Networks introduce a mechanism that aggregates information from multiple layers of a GNN to form the final node representations. Formally, let $H^{(l)} \in \mathbb{R}^{N \times d_l}$ denote the node feature matrix at layer l. Instead of using only $H^{(L)}$ (the output of the last layer), JK Networks compute

$$H_u^{(\mathrm{JK})} \;=\; \mathrm{Combine}\big(H_u^{(1)}, H_u^{(2)}, \ldots, H_u^{(L)}\big),$$

where $\mathrm{Combine}(\cdot)$ is a learnable function (e.g., concatenation followed by a linear projection, max pooling, or attention-based weighting) that adaptively selects and fuses the most informative features from each layer for node u. By allowing nodes to "jump" to representations from earlier layers, JK Networks preserve local and higher-frequency information that might otherwise be vanished in deep layers, thereby mitigating oversmoothing.

MixHop [109]. MixHop is an architecture designed to capture multi-scale neighborhood information within each layer. Instead of stacking many layers to indirectly gather information from distant neighbors, MixHop explicitly computes multiple powers of the normalized adjacency matrix $\hat{A}$. In particular, at layer l, MixHop computes

$$H^{(l+1)} \;=\; \sigma\Big(\Big[\hat{A}^0 H^{(l)} W_0^{(l)} \;\|\; \hat{A}^1 H^{(l)} W_1^{(l)} \;\|\; \cdots \;\|\; \hat{A}^K H^{(l)} W_K^{(l)}\Big]\Big),$$

where $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2}$ is the the augmented symmetrically normalized adjacency matrix after self-loop addition, $W_k^{(l)}$ are trainable weight matrices for the k-hop neighborhood, K is the maximum hop distance considered, and $\|$ denotes concatenation. By simultaneously mixing features from 0- to K-hop neighborhoods, MixHop achieves rich representation power without requiring deep stacking. This multi-horizon aggregation helps maintain diverse node features and reduces the tendency toward homogenization.

Decoupled GNNs [110]. Decoupled GNNs separate feature transformation from neighborhood aggregation, which are traditionally intertwined in standard GNN layers. In a standard GNN layer, the update is of the form

$$H^{(l+1)} \;=\; \sigma\big(\hat{A}\,H^{(l)}\,W^{(l)}\big),$$

entangling feature mixing ($\hat{A}\,H^{(l)}$) with transformation ($\cdot W^{(l)}$). Decoupled GNNs instead perform multiple transformation steps before (or after) a single propagation step. For example, one variant computes:

$$\tilde{H} \;=\; \sigma\big(H^{(l)}\,W_1^{(l)}\big), \quad H^{(l+1)} \;=\; \hat{A}\,\tilde{H},$$

or equivalently

$$H^{(l+1)} \;=\; \sigma\big(\hat{A}\,H^{(l)}\big)\,W_2^{(l)},$$

allowing separate control over transformation depth and message passing. By decoupling these operations, the network can apply multiple nonlinear transformations without repeated neighborhood mixing, thereby limiting cumulative smoothing effects. This design enables deeper architectures with reduced oversmoothing while still leveraging the benefits of feature transformation.

In summary, architectural innovations such as Jumping Knowledge Networks, MixHop, and Decoupled GNNs offer effective strategies to reduce oversmoothing in GNNs. By aggregating multi-layer features, capturing multi-scale neighborhood information within a single layer, and decoupling transformation from propagation, these designs preserve the discriminative power of node representations. Consequently, they enable the construction of deeper and more expressive GNN models while controlling feature homogenization across layers.

7.2 Skip and Residual Connections

Skip and residual connections have emerged as pivotal architectural strategies for mitigating the oversmoothing phenomenon in GNNs. Drawing inspiration from the success of Residual Networks (ResNets) [33] in deep learning, these connections preserve initial node features across layers, thereby enhancing the expressive capacity of deep GNNs.

Mechanism and Implementation. In conventional GNNs, each layer updates node representations by aggregating information from neighboring nodes. As the number of layers L increases, repeated aggregation can cause node features to converge to similar values, diminishing the model's discriminative power. Residual (or skip) connections address this issue by allowing the direct propagation of input features to subsequent layers. Formally, let $H^{(l)} \in \mathbb{R}^{N\times d_l}$ denote the matrix of node features at layer l. A residual connection is given by

$$H^{(l)} \;=\; \mathcal{F}\big(H^{(l-1)}\big) \;+\; H^{(l-1)},$$

where $\mathcal{F}\colon \mathbb{R}^{N\times d_{l-1}} \to \mathbb{R}^{N\times d_l}$ denotes the transformation (e.g., aggregation followed by a linear map and nonlinearity) applied at layer l. By learning the residual function $\mathcal{F}(H^{(l-1)})$ instead of the full mapping, the model preserves essential information from $H^{(l-1)}$, which helps maintain feature diversity across deep architectures.

Impact on Receptive Fields and Gradient Flow. Incorporating residual connections influences both receptive fields and gradient propagation. Because $H^{(l-1)}$ is added directly to $\mathcal{F}(H^{(l-1)})$, the node representation $H^{(l)}$ retains information from all previous hops, ensuring that representations capture both local and global structural information. This multi-scale representation is crucial for tasks requiring knowledge of immediate neighbors and distant nodes.

Moreover, residual connections create alternative gradient pathways during backpropagation. In deep networks, gradients may diminish as they pass through multiple layers, leading to vanishing gradient problems. The additive term $H^{(l-1)}$ provides a shortcut for gradients, thereby improving gradient flow and enabling stable training of deeper GNNs [23, 11].

Theoretical Insights and Empirical Evidence. Theoretical analyses confirm the efficacy of residual connections in preventing oversmoothing. For example, under linear propagation, the addition of $H^{(l-1)}$ slows down the exponential decay of feature variance, ensuring that node embeddings do not collapse into a constant vector. Specifically, let $\tilde{A} = \tilde{D}^{-1/2}(A+I)\tilde{D}^{-1/2}$ be the the augmented symmetrically normalized adjacency matrix after self-loop addition. Without residuals, repeated multiplication by $\tilde{A}$ drives $H^{(l)}$ toward the principal eigenspace of $\tilde{A}$. With residual connections, the update

$$H^{(l)} = \tilde{A}\, H^{(l-1)} + H^{(l-1)} = (I + \tilde{A})\, H^{(l-1)}$$

modifies the eigenvalues of the propagation operator, reducing the spectral gap and slowing homogenization.
Empirical evaluations confirm these theoretical results. For instance, the GCNII architecture introduces an initial residual connection that combines $H^{(0)}$ with each layer's output. This design maintains the influence of $H^{(0)}$ across all layers, effectively mitigating oversmoothing and enabling deeper GNNs to achieve higher accuracy on node classification benchmarks [11].

Variants and Extensions. Beyond standard residual connections, several variants have been proposed to further enhance GNN performance:

- **Initial Residual Connections.** These incorporate the initial node features $H^{(0)}$ into each layer's computation, ensuring that original information remains influential throughout the network depth. For example, in ResGCN, $H^{(l+1)}$ depends on both $\tilde{A}\, H^{(l)}$ and $H^{(0)}$, preserving high-frequency details [23].

- **Jumping Knowledge Networks (JK-Nets).** JK-Nets aggregate outputs from multiple layers. This allows each node u to adaptively select features from different neighborhood ranges, which is particularly beneficial in heterogeneous graphs [108].

- **Deep Adaptive Graph Neural Networks (DAGNNs).** DAGNNs dynamically adjust the number of propagation steps for each node based on learned attention weights. Specifically,

$$H^{(l+1)} = \sum_{k=0}^{K} \alpha_k^{(u)}\, \tilde{A}^k\, H^{(0)},$$

  where $\{\alpha_k^{(u)}\}_{k=0}^{K}$ are adaptive weights for node u. This flexibility balances local and global information, reducing unnecessary smoothing [111].

In summary, skip and residual connections are fundamental to enhancing the depth and expressiveness of GNNs. By preserving initial node features and facilitating robust gradient flow, these architectural strategies mitigate oversmoothing, enabling the design of deeper and more powerful GNN models. Variants such as initial residual connections, JK-Nets, and DAGNNs further extend these ideas, offering tailored mechanisms to balance feature preservation and neighborhood aggregation for complex graph-structured data.

7.3 Normalization Techniques

Normalization techniques have emerged as a method to mitigate the oversmoothing phenomenon in Graph Neural Networks. By standardizing feature distributions across various dimensions, these methods aim to preserve the discriminative power of node representations, especially in deeper network architectures. This section analyzes several prominent normalization techniques, focusing on their underlying mechanisms and their role in mitigating oversmoothing.

Batch Normalization (BatchNorm). Originally introduced to stabilize and accelerate training in deep neural networks [112], BatchNorm normalizes layer inputs to have zero mean and unit variance across a mini-batch. In GNNs, it is typically adapted to normalize node features within individual graphs. Theoretical analyses have shown that BatchNorm rescaling prevents node embeddings from collapsing into a one-dimensional subspace, effectively preserving their variance and expressivity across layers [113]. However, the centering operation may distort the underlying graph signal, as highlighted in recent work [113].

Layer Normalization (LayerNorm). LayerNorm normalizes across the feature dimension of each data sample rather than across the batch [114]. This makes it particularly effective for GNNs dealing with variable batch sizes or unstable batch statistics. In GNNs, LayerNorm stabilizes node-wise feature distributions and mitigates oversmoothing by ensuring consistent scales of features across layers, without relying on global batch statistics [115].

PairNorm. PairNorm is a normalization technique specifically designed to address oversmoothing in GNNs. It operates by maintaining a constant pairwise distance between node representations, effectively preventing them from converging to similar values across layers [13]. By re-centering and re-scaling node features after each message passing step, PairNorm preserves the relative distances among nodes, which is essential for tasks requiring discriminative embeddings. However, it is noteworthy that while PairNorm reduces oversmoothing, it may increase intra-class variance, potentially affecting classification performance.

Differentiable Group Normalization (DGN). Differentiable Group Normalization (DGN) introduces a group-wise normalization strategy that considers the community structures within graphs [116]. By normalizing node features within the same group or class, DGN enhances intra-class compactness while promoting inter-class separability. This approach effectively balances the need for smoothness within communities and distinctiveness across different groups, thereby addressing oversmoothing without compromising the model's discriminative capabilities.

GraphNorm. GraphNorm is tailored for graph-level tasks and normalizes node features by considering the entire graph's statistics. It performs a centering operation followed by scaling, akin to BatchNorm, but specifically adapted for graph structures [115]. By accounting for graph-specific characteristics, GraphNorm ensures that the normalization process preserves the inherent structural information, thereby mitigating oversmoothing in graph-level representations.

PowerEmbed. PowerEmbed introduces a layer-wise normalization approach that emphasizes the top-k eigenvectors of the graph, capturing essential global spectral information [117]. This technique facilitates the learning of comprehensive node representations by integrating both local and global information. By focusing on the spectral properties of the graph, PowerEmbed offers a principled method to mitigate oversmoothing, especially in scenarios where capturing long-range dependencies is crucial.

In summary, normalization techniques play a critical role in enhancing the depth and expressiveness of GNNs by addressing oversmoothing. While methods like BatchNorm and LayerNorm provide foundational normalization strategies, specialized techniques such as PairNorm, DGN, GraphNorm, and PowerEmbed offer tailored solutions that consider the unique aspects of graph-structured data. The choice of normalization method should align with the specific requirements of the task and the characteristics of the underlying graph topology.

7.4 Regularization

Regularization techniques serve as essential mechanisms for mitigating oversmoothing in Graph Neural Networks. By introducing controlled perturbations or constraints during training, these methods preserve informative node representations across multiple layers. This section describes several prominent regularization strategies, detailing their principles and contributions to alleviating oversmoothing.

DropEdge. DropEdge randomly removes a subset of edges from the input graph during each training iteration [14]. By stochastically dropping connections, this technique reduces the influence of any single neighbor, thereby limiting the extent of feature homogenization that occurs through repeated message passing. As a form of data augmentation, DropEdge encourages the model to learn more robust representations that do not rely excessively on specific edge patterns. Empirical results demonstrate that applying DropEdge to architectures such as GCN and GraphSAGE improves both classification accuracy and robustness to adversarial perturbations.

Critical DropEdge (C-DropEdge). Critical DropEdge extends the standard DropEdge approach by automatically determining an optimal, graph-dependent edge preservation rate, rather than relying on manually tuned, fixed drop probabilities. At each layer, C-DropEdge selects a critical fraction of edges, preserving a topology that balances connectivity and trainability. This edge retention rate is computed once per graph based on its structure, minimizing oversmoothing by theoretically aligning with the GNTK convergence limits. As a result, C-DropEdge maintains sufficient signal propagation even in deep GNNs, while avoiding over-pruning that could hinder learning. Empirically, this method offers more stable and robust performance across different depths, outperforming both vanilla GCNs and fixed-ratio DropEdge, especially in deeper architectures, and using a much smaller hyperparameter search space, since the edge-preservation rate is determined from the graph itself [118].

Spectral Regularization. Spectral regularization leverages the eigenvalues and eigenvectors of the graph Laplacian to impose smoothness constraints on node features. By penalizing high-frequency components (often quantified via Dirichlet energy) these methods

impose a balance between smoothing within local neighborhoods and preserving differences between distinct classes [21, 119, 106]. For example, adding a regularization term proportional to the Dirichlet energy discourages rapid convergence of node features, thus preventing overly uniform representations. Such spectral penalties are grounded in graph-theoretic principles and offer a mathematically principled approach to limiting oversmoothing.

Edge Perturbation. Edge perturbation techniques explicitly modify graph's topology by adding or removing edges according to a predefined distribution [120]. By perturbing edges, either randomly or based on criteria such as edge betweenness, these methods increase the diversity of node neighborhoods and prevent uniformity induced by repeated aggregation. Edge perturbation also acts as data augmentation, exposing the model to multiple graph variants during training. As a result, node representations become less prone to collapsing into similar vectors, and generalization performance improves.

Consistency Regularization. Consistency regularization enforces that model predictions remain stable under different perturbations of the input graph. Techniques such as Graph Random Neural Networks (GRAND) generate multiple augmented versions of the same graph (through methods like random propagation, feature masking, or edge dropout), and require that the model's outputs for these variants remain consistent [121]. This constraint encourages the network to learn representations that are invariant to small graph modifications, thereby reducing oversmoothing and enhancing resilience to noise.

In summary, regularization techniques are beneficial for addressing oversmoothing in GNNs. By disrupting the graph structure (DropEdge, A-DropEdge, edge perturbation), imposing spectral constraints (spectral regularization), or promoting prediction stability (consistency regularization), these methods preserve feature diversity and discriminative power across layers. Integrating regularization into GNN training enables deeper, more expressive models capable of handling complex graph-structured tasks while avoiding excessive feature homogenization.

7.5 Weight Initialization

Weight initialization plays a pivotal role in the training dynamics and performance of Graph Neural Networks. Inappropriate initialization can amplify oversmoothing, causing node representations to become indistinguishable faster, as network depth increases. This section first discusses the limitations of conventional initialization methods when applied to GNNs and then introduces graph-aware initialization techniques designed to preserve feature variance and mitigate oversmoothing.

Limitations of Conventional Initialization Methods. Traditional weight initialization strategies, such as Xavier (Glorot) [122] and Kaiming (He) [15], were developed for feedforward and convolutional neural networks. These methods aim to maintain the variance of activations and gradients across layers, thereby preventing vanishing or exploding gradients during training. However, when applied directly to GNNs, they ignore the unique characteristics of graph-structured data, particularly the repeated message passing operations.
In GNNs, each layer aggregates information from neighboring nodes. As layers are stacked, this aggregation effect compounds, causing the variance of node features to rapidly dimin-

ish. This reduction in variance is the primary mechanism behind oversmoothing. Because Xavier and Kaiming initializations do not take into account graph topology or the spectral properties of the graph Laplacian, they often lead to suboptimal feature variance preservation in deep GNN architectures. Consequently, deeper GNNs initialized with these conventional methods can exhibit accelerated homogenization of node embeddings, resulting in degraded performance on tasks such as node classification and link prediction [123].

Graph-Aware Initialization Techniques. To address these shortcomings, several graph-aware initialization strategies have been proposed. These methods explicitly incorporate structural information about the graph into the initialization of weights, with the goal of preserving feature variance across layers and mitigating the onset of oversmoothing.

VIRGO (Variance Instability Reduction in Graph Optimization). VIRGO [123] is grounded in a theoretical analysis of how variance propagates throughout GNNs layers, both in the forward pass (node representations) and the backward pass (gradients), while factoring in activation functions, hidden dimensions, message passing, and graph structure. Based on these derivations, it identifies initialization methods that stabilize variance across layers, effectively balancing information flow and preventing both explosion and collapse of variance. Practically, VIRGO determines optimal scaling of weight matrices by considering properties of the normalized adjacency matrix (including the spectral radius), and selects initial weight distributions (e.g., Gaussian or uniform) accordingly.

Isometry-Aware Initialization with Gradient-Guided Rewiring. Jaiswal et al. [124] highlight the limitations of applying standard Glorot initialization to deep GCNs, which often suffer from vanishing gradients and oversmoothing. They introduce a topology-aware isometric initialization tailored to GCNs, derived from a gradient-flow analysis that ensures each layer preserves the magnitude of activations and gradients. Additionally, rather than relying on fixed skip-connections, they propose adaptive rewiring: during training, on-demand skip links are introduced between layers based on observed gradient flow metrics. This dynamic combination significantly improves training stability and enables deeper GCNs. However, in that method, it is not yet clear whether the performance of deep models stems from the initialization scheme itself or the adaptively introduced skip-connections.

MLPInit: Peer-MLP-Driven Weight Initialization. Han et al. [125] propose a straightforward yet powerful method called MLPInit, which uses weight matrices pretrained on a Peer MLP (an MLP with the same architecture as the target GNN) to initialize GNN training. After training the MLP solely on node features (ignoring graph topology) the GNN is initialized with the resulting weights and then fine-tuned. This strategy not only accelerates convergence, but also often improves predictive performance compared to random initialization methods.

Empirical Evidence and Practical Implications. Empirical studies validate the effectiveness of graph-aware initialization methods. For instance, experiments using VIRGO on citation network datasets (e.g., Cora, Citeseer) and large-scale social graphs show improved convergence speed and enhanced classification accuracy relative to Xavier or Kaiming initialization. These findings underscore the importance of aligning weight initialization with

graph-specific factors, such as degree distribution and spectral gap, to preserve the expressiveness of node embeddings in GNNs.

In summary, weight initialization is a critical factor in the design and training of GNNs. Although traditional methods like Xavier and Kaiming provide a foundational approach for variance preservation in Euclidean domains, they do not account for the iterative aggregation and spectral structure inherent in graphs. Graph-aware initialization methods, such as VIRGO, incorporate topological and spectral information to maintain feature variance across layers, thereby enhancing overall model performance. As GNN architectures continue to grow in depth and complexity, the development and adoption of specialized (and oversmoothing resistant) initialization methods will remain essential for advancing state-of-the-art performance on graph-structured tasks.

7.6 Activation Functions

Activation functions can heavily influence the expressiveness and training dynamics of Graph Neural Networks. They introduce non-linearity into the model, enabling the learning of complex patterns. However, the choice of activation function significantly influences the appearance of oversmoothing as network depth increases. This section examines the impact of various activation functions on oversmoothing and explores strategies to mitigate this issue.

ReLU and Its Variants. The Rectified Linear Unit (ReLU) is widely adopted due to its simplicity and effectiveness. ReLU outputs zero for any negative input and passes through positive inputs unchanged. While this behavior promotes sparse activations, it can amplify oversmoothing by eliminating negative activation values and reducing information flow through deeper layers. To address these limitations, several variants have been proposed:

- **Leaky ReLU**: Unlike standard ReLU, Leaky ReLU allows a small, non-zero output for negative inputs. This modification helps prevent units from becoming inactive ("dying ReLUs") [126] and maintains some gradient flow for negative inputs, which in turn helps preserve variance in node features and reduce oversmoothing [127].
- **Exponential Linear Unit (ELU)**: ELU applies an exponential transformation to negative inputs that smoothly approaches a negative constant, while passing through positive inputs linearly. By shifting the mean activation closer to zero and providing a smooth gradient for negative inputs, ELU improves learning dynamics and reduces the tendency for node features to collapse [128].
- **Scaled Exponential Linear Unit (SELU)**: SELU is designed to self-normalize activations by automatically maintaining zero mean and unit variance across layers. This normalization effect helps stabilize deep network training and prevents rapid variance decay in node features [129].

Slope Effects and Variance Preservation. The slope of the activation function for negative inputs is crucial in preserving the variance of node features across layers. A zero slope (as in standard ReLU) can lead to rapid variance reduction and oversmoothing. Introducing a small positive slope for negative values (as in SeLU) allows some negative information to propagate, thereby maintaining feature diversity deeper in the network [129].

Deep GCNs with tanh. An initial approach to reduce oversmoothing was proposed in [130]. The paper demonstrates that using ReLU in deep GCNs leads to a rapid collapse in the column-rank of feature matrices. This rank collapse effectively reduces the dimensionality of node representations, reducing expressivity. Their experiments show that replacing ReLU with tanh stabilizes this collapse: the feature-matrix rank remains nearly constant across deep layers, in contrast to the observed drop with ReLU. Based on these insights, they proposed tanh as a superior activation function in deep GCNs capable of maintaining richer feature representations.

In summary, the choice of activation function greatly influences GNN depth scalability and performance. While ReLU remains popular for its simplicity and efficiency, its tendency to amplify oversmoothing limits deep GNN expressivity. Variants like Leaky ReLU, ELU, and SELU help address dying neurons, maintain feature variance, and improve gradient flow, but they lack theoretical foundations directly addressing oversmoothing. Tanh has been proposed as an alternative, but like all saturating functions, it flattens (saturates) for large-magnitude inputs, which, in turn, can still lead to similar node representations as depth increases.

8 Synthesis and Research Gaps

8.1 Summary of Observations

Recent years have witnessed extensive study of the oversmoothing phenomenon in Graph Neural Networks, resulting in numerous empirical methods to mitigate its negative effects. Techniques such as architectural innovations, skip and residual connections, normalization methods, and regularization strategies consistently demonstrate their ability to preserve node representation diversity and enhance model performance across different tasks. For example, residual connections have been both theoretically and empirically proven to counteract oversmoothing by "short-circuiting" initial node features to subsequent layers. In addition, architectures such as Jumping Knowledge Networks and MixHop capture multi-scale neighborhood information in order to reduce the homogenization of node representations.

Despite these practical successes, there remains a need for a theoretical framework that quantifies oversmoothing, identifies its primary causes, and systematically proposes methods for its mitigation. Existing theoretical analyses, such as those addressing the trade-off between denoising and mixing effects in graph convolutions, often rely on assumptions tied to specific GNN variants or graph structures. Furthermore, recent findings suggest that oversmoothing can be reduced through carefully chosen weight initialization methods or architectural constraints, enabling GNNs to retain expressive power even at larger depths. This indicates that our current understanding is still fragmented, lacking a unified theory capable of predicting oversmoothing behavior across a broad range of models and datasets.

In summary, while empirical strategies provide effective means of mitigating oversmoothing, the theoretical foundations underlying their success, and the conditions under which they are most effective, remain underdeveloped. Bridging this gap between empirical practice and theoretical insight is essential for the principled design of deeper and more robust GNN architectures.

8.2 Open Problems

Several critical challenges hinder the development of a unified theory and resilient GNN architectures that avoid oversmoothing while still leveraging the advantages of greater depth in GNNs:

Layer-wise Quantification of Oversmoothing. Current research lacks precise metrics that capture how oversmoothing evolves at each layer. While global measures such as Dirichlet energy or mean average distance (MAD) can assess overall feature smoothness, they fail to reveal layer-specific dynamics. Developing fine-grained, layer-level metrics would enable more targeted diagnosis and intervention, shedding light on exactly where and how oversmoothing emerges.

Interactions Between Residual Connections and Graph Topology. Although residual connections are widely used to alleviate oversmoothing by preserving feature variance, their effectiveness varies depending on graph structure. Factors such as degree distribution, clustering coefficients, and connectivity patterns influence how residual connections interact with message passing. A rigorous theoretical investigation into how residual mechanisms behave under varying graph topologies is essential to fully assess their benefits and limitations.

Impact of GNN Training. Training GNNs has dual effects: while parameter optimization enables deep GNNs to learn mechanisms that counteract oversmoothing and develop more informative node representations, GNNs are also subject to backward oversmoothing, where gradients themselves become homogenized (creating stationary points and hindering convergence). Additionally, issues like vanishing or exploding gradients early in training can dominate performance degradation, implying that trainability often limits deep GNNs performance. Exploring the effect of training in GNNs and oversmoothing remains an underexplored territory.

Informed Weight Initialization Method. Weight initialization substantially affects training dynamics in GNNs. Standard methods like Xavier and Kaiming do not account for graph topology, leading to rapid variance decay and exacerbated oversmoothing. While graph-aware initialization methods like VIRGO demonstrate promise, no universally applicable, theoretically grounded method exists that consistently mitigates oversmoothing as depth increases and reliably enhances performance across diverse GNN architectures and datasets. Developing such a method remains an open challenge.

Impact of Activation Functions. The choice of activation function influences gradient propagation and feature variance, but its specific role in oversmoothing is not fully understood. Clarifying how activation functions influence feature diversity and gradient flow is essential for designing activation functions that enhance depth scalability.

8.3 Positioning of This Work

In response to these open problems, this work makes the following key contributions:

Layer-wise Oversmoothing Metrics. We propose a novel metric to quantify oversmoothing at each layer of a GNN. By capturing the evolution of node representations distance layer by layer, the proposed metric enables precise diagnosis and targeted architectural adjustments to prevent representation collapse.

Theoretical Analysis of Residual-Topology Interactions. We develop a formal framework to analyze how residual connections interact with various graph topologies. By deriving conditions under which residual fail to enable deep architectures, we highlight their limitations under specific tasks which require deep GNNs.

Exploration of Partial Training. We systematically investigate how partial training can mitigate oversmoothing. By analyzing the impact on feature diversity, we identify propagation schemes that maintain expressiveness in deep GNNs.

Informed Weight Initialization Method. Building on recent advances, we introduce a weight initialization method that integrates key graph statistics, such as node degree distribution, into the initialization process. Our approach aims to stabilize training and delay the onset of oversmoothing, and we validate its effectiveness across multiple GNN variants and benchmark datasets.

Impact of Activation Functions. We systematically investigate how the use of activation functions with controlled slope can mitigate oversmoothing. By analyzing their impact on feature diversity and gradient flow, we identify design principles for activation functions that maintain expressiveness in deep GNNs.

Through these contributions, this work aims to bridge the gap between empirical success and theoretical understanding, advancing the principled design of GNN architectures that remain robust against oversmoothing across a wide range of graph-related tasks.

9 Conclusion of the Literature Review

Graph Neural Networks have advanced substantially, providing powerful mechanisms for learning from graph-structured data. Nevertheless, the oversmoothing phenomenon, where node embeddings become indistinguishable as network depth increases, persists as a critical barrier to the development of deeper and more expressive models. Extensive empirical work has produced numerous mitigation techniques, including residual connections, normalization methods, architectural innovations, and regularization strategies, all of which have demonstrated effectiveness in preserving feature diversity and improving performance on a variety of tasks.
Despite these empirical successes, a unifying theoretical framework that quantifies oversmoothing, identifies its primary drivers, and systematically proposes mitigation strategies remains absent. Existing metrics, such as Dirichlet energy and Mean Average Distance (MAD), offer partial insight into feature homogenization, and do not fully capture how oversmoothing progresses through individual layers. Moreover, the interplay between residual connections, graph topology, and weight initialization lacks rigorous exploration. Consequently, our understanding of how to design deep GNNs that consistently avoid oversmoothing across diverse graph structures remains incomplete.

Building upon the insights from the literature, the next chapters present theoretically solidified and empirically verified methods to quantify and reduce oversmoothing in deep GNNs. Specifically:

- Develop layer-specific metrics that accurately quantify the onset and progression of oversmoothing.
- Theoretically analyze how residual connections interact with various graph topologies, and, in particular, when deep GNNs are required.
- Investigate the role of partial training in maintaining feature diversity.
- Propose informed weight initialization method for GNNs that aligns with spectral and structural properties of input graphs.
- Investigate the role of the slope of the activation functions in maintaining feature diversity and promoting stable gradient flow.

By addressing these points, this work aims to bridge the gap between empirical practice and theoretical understanding, ultimately enabling the principled design of robust, deep GNN architectures.

Chapter 3

Analyzing the Effect of Embedding Norms and Singular Values to Oversmoothing in Graph Neural Networks

Chapter 3 introduces a principled approach to measuring and reducing oversmoothing in deep GNNs. Oversmoothing poses a fundamental barrier to designing expressive, deep GNN architectures. To better understand and address this challenge, it is essential to develop a quantitative measure that captures the extent of oversmoothing in a given model. In this chapter, building on prior studies [106], we propose a novel distance measure, namely $MASED$, and conduct an in-depth investigation into its properties.
We derive upper and lower bounds on its value and examine how the lower bound can be increased, in order to maintain node embedding variance and reduce oversmoothing. In this direction, we highlight the key role of the node embedding norms and of the smallest singular values of weight matrices, which have been largely overlooked. Additionally, we shed light on the relationship between the number of adjacency hops and weight matrices in GNNs. Our analysis provides insights into the characteristics of a deep model that lead to oversmoothing and how this, in turn, affects model performance. These findings can motivate the design of more effective strategies for mitigating the impact of oversmoothing in deep GNNs. We also propose one such mitigation strategy, in the form of a novel regularization method, named *G-Reg*, which aims to reduce co-linearity between the rows of the weight matrices and, in turn, increase their smallest singular values.
In summary, the main contributions of the work presented in this chapter are as follows:

- **Contribution on Oversmoothing Quantification**: We introduce the Mean Average Squared Euclidean Distance ($MASED$) to quantify oversmoothing in GNNs. We derive the upper and lower bounds of $MASED$ and reveal the relationship between the distances among node representations and the structure of the weight matrix, highlighting how the independence of its rows affects oversmoothing. Moreover, we show that $MASED$ values can predict oversmoothing early in the learning process. A rapid decline, accompanied by a reduction in the weight matrix's singular values, leads to poor model performance.

- **The effect of embedding norms and angles on oversmoothing**: We measure the average angle between the centroids (i.e., average embedding of nodes belonging to the same class) of the training nodes and the average norms of node embeddings. We observe that if norms are close to zero then the model is incapable of learning. However, if the norms are large enough and the model has the needed capacity, small angles suffice to maintain competitive performance.

• **New weight regularization method (*G-Reg*)**: We propose a regularization of the values of the weight matrix in order to maintain higher values of $MASED$ and higher values of the embedding norms. We experimentally confirm the benefits of *G-Reg*, utilizing deep GCNs, residual GCNs, and SGCs, with up to 32 layers across 7 node classification tasks. We show that the proposed regularization reduces oversmoothing, and demonstrate its benefits, in the presence of the "cold start" situation, where node features are available only for the labeled nodes.

• **Reducing weight redundancy in multi-hop aggregation**: We propose using fewer weight matrices than the number of neighborhood hops, improving learnability and reducing oversmoothing.

1 Theoretical Analysis

1.1 Mean Average Squared Euclidean Distance (MASED)

The Mean Average Squared Euclidean Distance ($MASED$) of node representations can act as a surrogate to measure the extent of oversmoothing in node representations, while allowing a rigorous analysis of its properties and a derivation of the connection with weight matrix properties. $MASED$ is also highly valuable for capturing the dynamic behavior of a GNN throughout its training process. During the early training epochs, slight variations in the model's learning dynamics can be detected through changes in the average distances among node embeddings. A rapid decrease in $MASED$ indicates that the model quickly begins to enforce uniformity on the node representations, serving as an early alarm for the onset of oversmoothing even before the overall performance declines. This sensitivity makes $MASED$ a robust proxy that reflects subtle shifts in the network, often correlating with other indicators, such as shrinking singular values of the weight matrix. In essence, by monitoring $MASED$, we can detect early signs of information degradation and modify training strategies to maintain the discriminative power of node embeddings.
The $MASED$ metric is defined over a graph G with N nodes as follows:

$$MASED(G) = \frac{1}{N}\sum_{i=1}^{i=N}\frac{1}{N}\sum_{j=1}^{j=N}\left(d_{i,j}^{euc}\right)^2 = \frac{1}{N^2}\sum_{i=1}^{i=N}\sum_{j=1}^{j=N}\left(d_{i,j}^{euc}\right)^2, \tag{3.1}$$

where $d_{i,j}^{euc}$ is the Euclidean distance between the representations/embeddings of node i ($h_i^{(l)}$) and node j ($h_j^{(l)}$).
Note that $MASED$ can be calculated over the output of each layer of the model under investigation. In order to simplify our analysis, we focus on the output of the l-th layer of a GCN, i.e., $ReLU(H^{(l)}) = ReLU(\hat{A}H^{(l-1)}W^{(l)})$.
In the rest of our analysis we calculate the Euclidean distance using the Gramian matrix (G^{gram}) of the embedding matrix $H^{(l)}$, i.e. $G^{gram} = H^{(l)} \cdot (H^{(l)})^T$. The Gramian takes that form because the rows of $H^{(l)}$ correspond to node embeddings. The value of the squared Euclidean distance between node embeddings (i.e., rows of matrix $H^{(l)}$) can be calculated using elements of G^{gram}, utilizing the following relationship:

$$\left(d_{i,j}^{euc}\right)^2 = g_{i,i} - 2g_{i,j} + g_{j,j}, \tag{3.2}$$

where $g_{i,j}$ is the (i,j)-element of G^{gram} matrix.
For each node pair we calculate the squared Euclidean distance between its nodes using

Equation 3.2. Summing these values yields the total Euclidean squared distance as follows:

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\left(d_{i,j}^{euc}\right)^2 = N\sum_{i=1}^{N} g_{i,i} + \sum_{i=1}^{N}\sum_{j=1}^{N} -2g_{i,j} + N\sum_{j=1}^{N} g_{j,j} =$$

$$N \cdot tr(G^{gram}) - 2 \cdot \mathbf{1}^T G^{gram}\mathbf{1} + N \cdot tr(G^{gram}) =$$

$$2N \cdot tr(G^{gram}) - 2 \cdot \mathbf{1}^T G^{gram}\mathbf{1}, \tag{3.3}$$

where $\mathbf{1}$ is the all-one column vector. Equation 3.3 holds because $g_{i,i}$ and $g_{j,j}$ do not depend on j and i, respectively, and the double summation over $g_{i,j}$ is equivalent to the summation of every element of G^{gram}. Additionally, summing either of $g_{i,i}$ or $g_{j,j}$ yields the trace of G^{gram}.

Combining Equations 3.3 and 3.1 leads to the following expression of $MASED$:

$$MASED^{(l)}(G) = \frac{2 \cdot tr(G^{gram})}{N} - \frac{2 \cdot \mathbf{1}^T G^{gram}\mathbf{1}}{N^2} = \frac{2}{N}\left(tr(G^{gram}) - \frac{1}{N}\mathbf{1}^T G^{gram}\mathbf{1}\right). \tag{3.4}$$

This alternative formulation of $MASED$ is particularly useful in deriving its upper and lower bounds, which are, in turn, important for controlling oversmoothing.

1.2 Bounds on MASED

In this subsection we derive the upper and lower bounds of $MASED$. To simplify the notation we drop the superscripts (i.e., we denote $H^{(l)}$ as H, $W^{(l)}$ as W and $MASED^{(l)}(G)$ as $MASED(G)$), and introduce $\hat{H}^{(l-1)} = \hat{A}H^{(l-1)}$, the smoothed node embeddings after the step of averaging across neighboring nodes. These lead to the following expressions about $H^{(l)}$ and G^{gram}:

$$H^{(l)} = ReLU(\hat{A}H^{(l-1)}W^{(l)}) \equiv H = ReLU(\hat{H}W) \in \mathbb{R}^{N\times d}.$$

$$G^{gram} = HH^T \in \mathbb{R}^{N\times N},$$

whose diagonal elements can be written as $G_{i,i}^{gram} = ||H_{i,*}||_2^2 = r_i^2$, i.e., the diagonal elements of the Gramian are the squared norms of the node embeddings, where $H_{i,*}$ indicates the i-th row of H.

Considering the form of G^{gram} we also derive the following:

$$\mathbf{1}^T G^{gram}\mathbf{1} = \mathbf{1}^T HH^T\mathbf{1} = \left(\mathbf{1}^T H\right)\left(\mathbf{1}^T H\right)^T = ||\mathbf{1}^T H||_2^2.$$

Hence, we can rewrite the terms of Equation 3.4 as:

- $tr(G^{gram}) = \sum_{i=1}^{N} G_{ii}^{gram} = \sum_{i=1}^{N} r_i^2$.
- $\frac{1}{N}\,\mathbf{1}^T G^{gram}\,\mathbf{1} = \sum_{j=1}^{d}\left(\frac{1}{\sqrt{N}}\sum_{i=1}^{N} H_{i,j}\right)^2 = \sum_{j=1}^{d} z_j^2$.

Where

$$r_i = \|H_{i,*}\|_2, \quad z_j = \frac{1}{\sqrt{N}}\sum_{i=1}^{N} H_{i,j}, \tag{3.5}$$

we recover the equivalent form of Equation 3.4

$$MASED(G) \;=\; \frac{2}{N}\Big(\sum_{i=1}^{N} r_i^2 \;-\; \sum_{j=1}^{d} z_j^2\Big). \tag{3.6}$$

1.2.1 Upper Bound

In many cases, our objective is to drive $MASED$ as high as possible, since larger values of $MASED$ correspond to more diverse node embeddings, which, in turn, reduce oversmoothing. However, it is equally important to understand the factors that limit $MASED$. An explicit upper bound acts as a guide to reveal which aspects of the model affect $MASED$ the most.
Considering the form of $MASED$ in Equation 3.6 we observe that it is the result of the subtraction between positive quantities. Hence, the upper bound can be the upper bound of the first term of Equation 3.6. Using the Cauchy–Schwarz inequality and the upper bound of the norm of a product we conclude to the following Lemma.

Lemma 3.

$$MASED(G) = \frac{2}{N}\Big(\sum_{i=1}^{N} r_i^2 \; - \; \sum_{j=1}^{d} z_j^2\Big) \leq \frac{2}{N}\sum_{i=1}^{N} r_i^2 \leq \frac{2}{N} N \max_i r_i^2 \implies$$

$$MASED(G) \leq 2\sigma_{\max}^2(W) \cdot M_{\hat{H}}^2,$$

where $\sigma_{\max}(\cdot)$ denotes the largest singular value of the respective weight matrix, and $M_{\hat{H}} = \max_i ||\hat{H}_{i,*}||_2$. In the above result we have used the fact that $r_i = \|\hat{H}_{i,*}W\|_2 \leq \sigma_{\max}(W)\|\hat{H}_{i,*}\|_2$, which holds for every row of H (i.e., each node embedding).

By examining this bound, we observe how changes in each parameter, such the largest singular value of the weight matrix, or embedding norms, affects the maximum possible value of $MASED$. This analysis provides both a theoretical ceiling for $MASED$ and practical insight into which strategies are most effective for approaching that ceiling. Our result is aligned with the current line of research presented in Theorem 1. Additionally to that conclusion, Lemma 3 highlights the importance of the norms of the embeddings in increasing the upper bound of $MASED$. Hence, by increasing the largest singular value of the weight matrix and increasing embedding norms, we may allow the GNN to avoid oversmoothing. Conversely, when the largest singular values of the weight matrices or the norms of the node embeddings are small, $MASED$ is suppressed leading to smaller distances between node embeddings, and oversmoothing.

1.2.2 Lower Bound

While an upper bound on $MASED$ is important, especially when certain constraints or model parameters inherently limit how large $MASED$ can become, it does not provide a complete picture. In some cases, the upper bound may be loose or rarely attained in practice, offering limited insight into typical or guaranteed behavior.
In contrast, a meaningful lower bound is often more informative in contexts where the goal is to drive $MASED$ to higher values. A strong lower bound ensures that $MASED$ remains above a certain threshold under all valid conditions, providing a reliable performance guarantee. It reflects the worst-case scenario we can expect and helps us understand how much improvement is possible.
By focusing on the lower bound, we can identify which parameters or conditions most effectively raise this minimum and force the model to avoid oversmoothing. Note that trivially, $MASED$ is lower bounded by zero, when all node embeddings coincide, hence leading to zero distance between every pair of embeddings. We term that particular case as

extreme oversmoothing, in which there is no variance in node embeddings and no classifier can properly distinguish them. Imposing the spectral-alignment condition, we conclude to the following Lemma.

Lemma 4. *The following lower bound on $MASED$ holds:*

$$MASED(G) \geq 2\varepsilon\mathbb{E}[r^2] \geq 2\varepsilon \cdot r_{\text{min}}^2 \geq 2\varepsilon \cdot \sigma_{\text{min}}^2(W) \cdot m_{\hat{H}}^2,$$

where r_{min} denotes the smallest value of r_i's, $\sigma_{\text{min}}(W)$ denotes the smallest singular value of W, and $m_{\hat{H}} = \min_i ||\hat{H}_{i,*}||_2$.

In order to prove Lemma 4, we need to derive a more actionable lower bound of Equation 3.6. Hence, we define the following quantities:

$$\mathbb{E}[r^2] = \frac{1}{N}\sum_{i=1}^{N} r_i^2 = \frac{1}{N} tr(HH^\top) \qquad \mathbb{E}[z^2] = \frac{1}{d}\sum_{j=1}^{d} z_j^2 = \frac{1}{dN}\mathbf{1}^\top HH^\top \mathbf{1},$$

where $\mathbb{E}[\cdot]$ denotes the expected value, and r_i and z_j are defined in Equation 3.5. Specifically, $r_i \in \mathbb{R}^N$ measures the Euclidean norm of each row of H, and $z_j \in \mathbb{R}^d$ measures the normalized column-wise sum.
As a result, Equation 3.5 transforms to:

$$MASED(G) = 2\left(\mathbb{E}[r^2] - \frac{d}{N}\mathbb{E}[z^2]\right). \tag{3.7}$$

The above formula indicates the need of determining an upper bound of $\mathbb{E}[z^2]$, which, in turn, will lead to a lower bound of $MASED$.
For this, we exploit the spectral decomposition of the positive semi-definite matrix $G^{gram} = HH^\top$, writing

$$G^{gram} = \sum_{\ell=1}^{N} \lambda_\ell \, v_\ell v_\ell^\top, \quad \mathbf{1}^\top G^{gram}\,\mathbf{1} = \mathbf{1}^\top HH^T\,\mathbf{1} = \sum_{\ell=1}^{N} \lambda_\ell \, (v_\ell^\top \mathbf{1})^2,$$

where each $\lambda_\ell \geq 0$ (i.e., eigenvalues of G^{gram}) and $\{v_\ell\}$ are orthonormal vectors (i.e., eigenvectors of G^{gram}). We isolate the contribution of the uniform direction $\mathbf{1}$ relative to the total trace $\sum_\ell \lambda_\ell$. Here, $(v_\ell^\top \mathbf{1})^2$ measures how closely each eigenvector aligns with the all-ones pattern, and weighting by λ_ℓ shows how strongly that direction influences the quadratic form $\mathbf{1}^\top G^{gram}\,\mathbf{1}$, which, in turn, determines $\mathbb{E}[z^2]$. Intuitively, if a large fraction of the total "energy" (trace) of G^{gram} lies in the uniform direction, then the column averages z_j behave almost like constant multiples of the per-row norms r_i. In that case, the lower bound of $MASED$ is very close or equal to zero.
To avoid this case, we impose a spectral-alignment condition. The spectral-alignment condition suggests that node embeddings have not collapsed to the same representation: there exists $\varepsilon \in (0, 1)$ such that

$$\sum_{\ell=1}^{N} \lambda_\ell \, (v_\ell^\top \mathbf{1})^2 \;\leq\; (1-\varepsilon)\, N \sum_{\ell=1}^{N} \lambda_\ell.$$

Note that since G^{gram} is symmetric, the spectral theorem guarantees that its eigenvectors can be chosen to form an orthonormal basis. Hence, they are mutually orthogonal and each

has unit norm (i.e., $\|v_\ell\|_2 = 1$), so $(v_\ell^\top \mathbf{1})^2 \leq \|\mathbf{1}\|^2 = N$. The left-hand side is exactly the energy of the uniform direction with respect to G^{gram}. If $H_{i,*} = \beta_i\, u$ for some fixed u, then $v_1 = \mathbf{1}/\sqrt{N}$, $(v_1^\top \mathbf{1})^2 = N$, and $\sum_\ell \lambda_\ell (v_\ell^\top \mathbf{1})^2 = N \sum_\ell \lambda_\ell$. Our bound $\leq (1-\varepsilon) N \sum \lambda_\ell$ thus strictly excludes this degenerate case. The usage of, "$\leq$" caps uniform alignment and guarantees genuine row-diversity.

The spectral-gap assumption is critical for preventing feature collapse in graph neural networks and for ensuring meaningful variability in node representations. If every row of H were a scalar multiple of a single vector, then all node features would lie on a single line in $\mathbb{R}^d$, and any aggregation or comparison based on dot-products or learned linear mappings would render all representations indistinguishable up to scale. By requiring a nonzero spectral gap ($\varepsilon > 0$), we guarantee that at least an ε-fraction of the total row-energy of H resides outside this degenerate direction, thereby preserving genuine discriminative structure. Equivalently, since the all-ones vector $\mathbf{1}$ encodes the uniform pattern "all rows identical", bounding its squared projection onto the top eigenspace of $HH^\top$ by $(1-\varepsilon)N$ ensures that no more than a $(1-\varepsilon)$-fraction of the total variance in H can be explained by uniform alignment.

In turn, at least an ε-fraction of variance must lie in directions orthogonal to $\mathbf{1}$, so that node features are not all "pointing the same way". This requirement is mild and generally satisfied in practical settings. Only pathological rank-one configurations (all rows exactly proportional, corresponding to $\varepsilon = 0$) violate the gap. Therefore, a strictly positive spectral gap is a realistic and broadly applicable condition.

We therefore focus on $\varepsilon > 0$, noting that $\varepsilon = 0$ characterizes the pathological feature-collapse scenario, where $MASED$ equals zero. The assumption of a positive ε allows us to derive a positive lower bound, which highlights the contributing factors that can help increase the values of $MASED$. Since

$$\mathbb{E}[z^2] = \frac{1}{dN}\,\mathbf{1}^\top HH^\top \mathbf{1} = \frac{1}{dN}\sum_{\ell=1}^{N} \lambda_\ell\,(v_\ell^\top \mathbf{1})^2 \leq \frac{1-\varepsilon}{dN}\,N \sum_{\ell=1}^{N} \lambda_\ell \implies$$

$$\mathbb{E}[z^2] \leq \frac{1-\varepsilon}{dN} N\, tr(HH^\top) = (1-\varepsilon)\,\frac{N}{d}\,\mathbb{E}[r^2],$$

the alignment condition yields

$$\mathbb{E}[z^2] \;\leq\; (1-\varepsilon)\,\frac{N}{d}\,\mathbb{E}[r^2] \;\implies\; \mathbb{E}[r^2] - \tfrac{d}{N}\,\mathbb{E}[z^2] \;\geq\; \varepsilon\,\mathbb{E}[r^2].$$

Substituting into the definition of $MASED$ in Equation 3.7 gives

$$MASED(G) = 2\Big(\mathbb{E}[r^2] - \tfrac{d}{N}\,\mathbb{E}[z^2]\Big) \;\geq\; 2\varepsilon\,\mathbb{E}[r^2] \;\geq\; 0. \tag{3.8}$$

This bound confirms that, under our spectral-alignment assumption, the lower bound of $MASED$ is non-negative. Only in the degenerate case $\varepsilon = 0$, when the all-ones vector is the top eigenvector and all embeddings collapse to one direction, does this bound collapse to the trivial lower bound, i.e., zero.

Using Equation 3.8, we, finally, derive the lower bound presented in Lemma 4. Lemma 4 shows the effect of the smallest singular value and the smallest norm of the node embeddings on the lower bound of $MASED$, which is related to oversmoothing appearance. Additionally, it highlights the tools available to reduce oversmoothing.

1.3 Network-Level Analysis of MASED

Having established the upper (Lemma 3) and lower (Lemma 4) bounds on $MASED$ at each individual layer, we now extend these results to the entire network. By tracking the evolution of $MASED$ from layer to layer, we derive global guarantees on how it changes from input to output. This extension is crucial as it reveals whether $MASED$ increases or decreases over multiple layers, and helps to identify the key layer-wise parameters that influence the network's overall behavior. In what follows, we show how the per-layer bounds combine and discuss the conditions under which the network preserves, amplifies, or shrinks the value of $MASED$.

In order to estimate the bounds of $MASED$ at the final layer of the model, we reintroduce the superscripts in our notation, i.e., we will bound $MASED^{(L)}(G)$ with L being the model's depth. We also utilize Lemma 3 and Lemma 4 along with the inequality:

$$\sigma_{min}(B)||u||_2 \leq ||Bu||_2 \leq \sigma_{max}(B)||u||_2,$$

which holds for every matrix B and vector u.

The lower bound (Lemma 4) depends on $m_{\hat{H}}^{(L)}$, which can be bounded as follows:

$$m_{\hat{H}}^{(L)} = \min_i ||\hat{H}_{i,*}^{(L)}||_2 = \min_i ||\hat{A}H_{i,*}^{(L-1)}||_2 \geq \sigma_{min}(\hat{A}) \min_i ||H_{i,*}^{(L-1)}||_2 = \sigma_{min}(\hat{A}) \min_i ||ReLU(\hat{A}H^{(L-2)}W^{(L-1)})||_2.$$

Substituting $H^{(l)}$ telescopically leads to:

$$m_{\hat{H}}^{(L)} \geq \sigma_{min}^{L-1}(\hat{A}) \cdot \min_i ||X_{i,*}||_2 \cdot \prod_{i=1}^{L-1} \sigma_{min}(W^{(i)}).$$

Similarly for $M_{\hat{H}}^{(L)}$ we get:

$$M_{\hat{H}}^{(L)} \leq \sigma_{max}^{L-1}(\hat{A}) \cdot \max_i ||X_{i,*}||_2 \cdot \prod_{i=1}^{L-1} \sigma_{max}(W^{(i)}).$$

Combining the above results with Lemma 3 and Lemma 4 we arrive at the following conclusion about the bounds of $MASED^{(L)}$, i.e., the mean average squared distance of node representations at the final layer of the model:

$$2\varepsilon\left(\sigma_{min}^{L-1}(\hat{A}) \cdot m_X \cdot \prod_{i=1}^{L} \sigma_{min}(W^{(i)})\right)^2 \leq MASED^{(L)}(G) \leq 2\left(\sigma_{max}^{L-1}(\hat{A}) \cdot M_X \cdot \prod_{i=1}^{L} \sigma_{max}(W^{(i)})\right)^2, \tag{3.9}$$

where $m_X = \min_i ||X_{i,*}||_2$, $M_X = \max_i ||X_{i,*}||_2$, and X denotes the initial node features.

In Equation 3.9, the term $\sigma_{\min}(\hat{A})$, i.e., the smallest singular value of $\hat{A}$ is often zero in real-world graphs, leading the global lower bound to zero. For this reason, the per-layer $MASED^{(l)}(G)$ value is more useful.

Ensuring that each $MASED^{(l)}(G)$ remains not only strictly positive but substantially above zero, particularly in the lower layers, helps preserve the variance of the input features and avoids the pitfalls of relying solely on the global $MASED^{(L)}(G)$ lower bound. While some reduction of $MASED^{(l)}(G)$ in the upper layers may be expected or even beneficial for classification, maintaining substantially non-zero values in early layers is critical for robust feature propagation.

2 Implications of the Theoretical Analysis

Equation 3.9 highlights that $MASED$ is primarily influenced by two factors: the singular values of the weight matrices and their total number. To address the former, we propose a regularization method; namely *G-Reg*, as a means of increasing the singular values, while for the latter we provide further intuition and suggest reducing the number of weight matrices as a practical remedy.

2.1 *G-Reg*: Regularization of the Standard Deviation of the Weight Matrix

An idea for reducing oversmoothing, born from the results of our theoretical analysis, is to introduce a novel regularization approach tailored to graph-based models. Existing regularizers (e.g., standard weight decay or dropout) do not explicitly address the reduction of the singular values of the weight matrices, which have been shown to play an important role in oversmoothing. If the rows of $W^{(l)}$ tend to become linearly dependent, the smallest singular value $\sigma_{\min}\left(W^{(l)}\right)$ will monotonically decrease toward zero, which, in turn, collapses the lower bound of Equation 3.9 to zero.
To address this issue, we propose *G-Reg*, which aims to reduce the linear dependence of the rows of the weight matrices by rewarding large standard deviation among the elements of each row. Increasing the standard deviation can thus increase the directional diversity of the matrix rows. This results is achieved under the assumption of row-wise independent perturbations, which means that each row is randomly perturbed independently of the others. Since linear independence of the rows implies that the matrix has full (or almost full) row rank, it follows that the smallest singular value $\sigma_{\min}(W^{(l)}) > 0$. Positive $\sigma_{\min}(W^{(l)})$ values allow the lower bound of Equation 3.9 to remain positive as well, pushing the average node distances to higher values.
Formally, let $W^{(l)}$ denote the learnable weight matrix at layer l, we define the *G-Reg* penalty as

$$\mathcal{L}_{\text{G-Reg}} = \lambda_w \sum_{l=1}^{L} \frac{1}{d} \sum_{i=1}^{d} std(W_{i,*}^{(l)}), \tag{3.10}$$

where std denotes the standard deviation, $W_{i,*}^{(l)}$ is the i-th row of matrix $W^{(l)}$, and $\lambda_w > 0$ is a tunable strength parameter.

2.2 Effect of multiple weight matrices on MASED

Given the importance of weight values, as mentioned above, we study further the effect of the number of weight matrices through which the input signal passes. Each weight matrix can be regarded as a transformation from one embedding space to another. Hence, the number of weight matrices expresses the number of embedding spaces through which the input signal passes.
Increasing the depth of GCNs introduces a significant risk of losing critical information even before training begins. This phenomenon arises from the probabilistic properties of random weight matrices at initialization. Typically, these matrices are initialized with values drawn from a Gaussian distribution, resulting in eigenvalues within the unit circle and singular values centered around 2.
Suppose that the top-k singular values of each weight matrix exceed a threshold (e.g., 0.5).

Then the corresponding input feature dimensions are only mildly weakened as they pass through each layer. Conversely, feature dimensions tied to smaller singular values shrink quickly in deep networks. As the network depth L increases, the probability that a useful feature direction retains sufficient strength across all layers diminishes exponentially, limiting its influence on the final output. Specifically, if the weight matrices have a width $d > k$ (k once again denoting the number of singular values that exceed a threshold), the probability $Pr[Q]$ that an informative direction survives through all layers is given by:

$$Pr[Q] = \left(\frac{k}{d}\right)^L = \beta^L, \quad \beta < 1.$$

This exponential decay implies that deeper networks are increasingly likely to suppress informative features before any learning occurs. Consequently, the model may be "doomed to fail", as essential information is lost during the initial forward passes, leading to suboptimal performance that cannot be improved through subsequent training. The underlying issue is that the initial random weight matrices, combined with the depth of the network, effectively filter out significant components of the input features. As a result, gradient signals weaken or vanish entirely, preventing the model from updating its parameters effectively. Consequently, increasing the number of weight matrices can degrade overall performance. This is also related to the value of $MASED$, as shown in Equation 3.9. As the number of weight matrices increases, the number of factors in the products of the smallest and largest singular values also increases. To keep $MASED$ large, each weight matrix should maintain both a large minimum and maximum singular value. This becomes increasingly difficult as more layers are added. Therefore, while using multiple weight matrices allows the model to capture complex input relationships, too many weight matrices might lead to performance deterioration.

2.3 Decoupling Weight Matrices from Adjacency Powers

Equation 3.9 additionally shows that the distance between node embeddings after graph convolution depends on both the number of weight matrices and the adjacency power used for aggregation. Conventionally, each layer l has a different weight matrix $W^{(l)}$, coupling receptive field size to parameter count. Yet, successive adjacency multiplications already capture high-order structure, making the use of a separate $W^{(l)}$ for each layer not always beneficial. This was discussed in Subsection 2.2.

Based on this observation, we explore the possibility of aggregating an extended multi-hop neighborhood via a single weight matrix, thereby reducing the number of matrices in the network. This principle resembles the APPNP [16] single-matrix propagation scheme. According to our analysis, reducing the number of matrices can also limit the extent of oversmoothing.

In practice, we distribute the number of hops L that we want to capture, i.e., the distance from a single node, across K stacked blocks of SGC, each aggregating up to L/K hops before applying its own transformation. A visual representation of this mechanism is provided in Figure 3.1. For example, if $L = 10$ (i.e., we wish to aggregate information up to 10 hops away), using two stacked layers of SGC implies that each layer aggregates information up to $L/2 = 5$ hops away. This reduces parameter redundancy and enables multi-hop feature mixing.

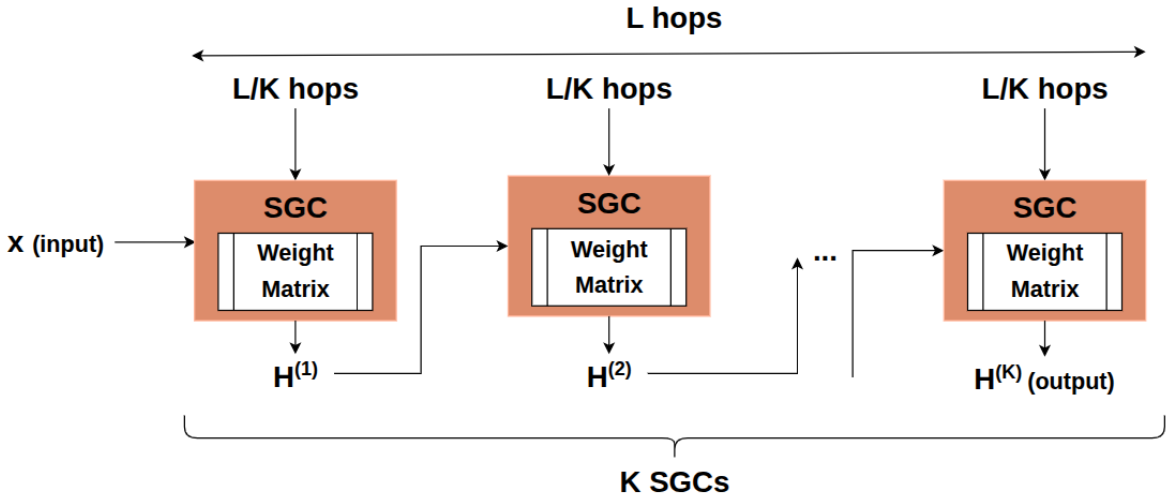

Figure 3.1: Each SGC layer uses the adjacency matrix raised to the power of L/K, and takes as input either the output of the previous layer or the initial node features if it is the first layer.

3 Experiments

In this section we perform a series of experiments, inspired by the theoretical results presented above. The experiments aimed to confirm the power of $MASED$ in quantifying oversmoothing and the reduction of the problem by the proposed changes in the learning process.

3.1 Experimental Setup

Datasets: Aligned to most of the literature, we focused on seven well-known benchmarks: *Cora, CiteSeer, Pubmed, Photo, Computers, Physics* and *CS*. For the first three co-citation datasets we used the same data splits as in [25], where all the nodes except the ones used for training and validation are used for testing. For the *Photo, Computers, Physics* and *CS* datasets we followed the same splits as in [131]. Dataset statistics can be found in Appendix 24.
Models: We experimented with the architectures of GCN [25], ResGCN [23], and SGC [17].
Hyperparameters: We performed a hyperparameter sweep (details in Appendix 6) to determine the optimal hyperparameter values, based on their performance on the validation set. For GCN and ResGCN, we set the number of hidden units for each layer to 128 across all benchmark datasets. For SGC when using a single layer, the input dimension equals the number of features while the output dimension is the number of classes. When using multiple layers of SGC the number of hidden units is also 128. L_2 regularization was applied with a penalty of $5 \cdot 10^{-4}$, and the learning rate was set to 10^{-3} for GCN and ResGCN, while for SGC the optimal value was $6 \cdot 10^{-3}$. Depth varied between 2 and 40 layers depending on the setting under investigation. Finally, $\lambda_w \in \{0, 2, 3, 4, 6, 8\}$ for GCN and ResGCN, while for SGC $\lambda_w \in \{0, 0.01, 0.5, 1, 2\}$.
Configuration: Each experiment was run 10 times and we report the average performance over these runs. We trained all models within 200 epochs using Cross Entropy as a loss function.

3.2 Experimental Results

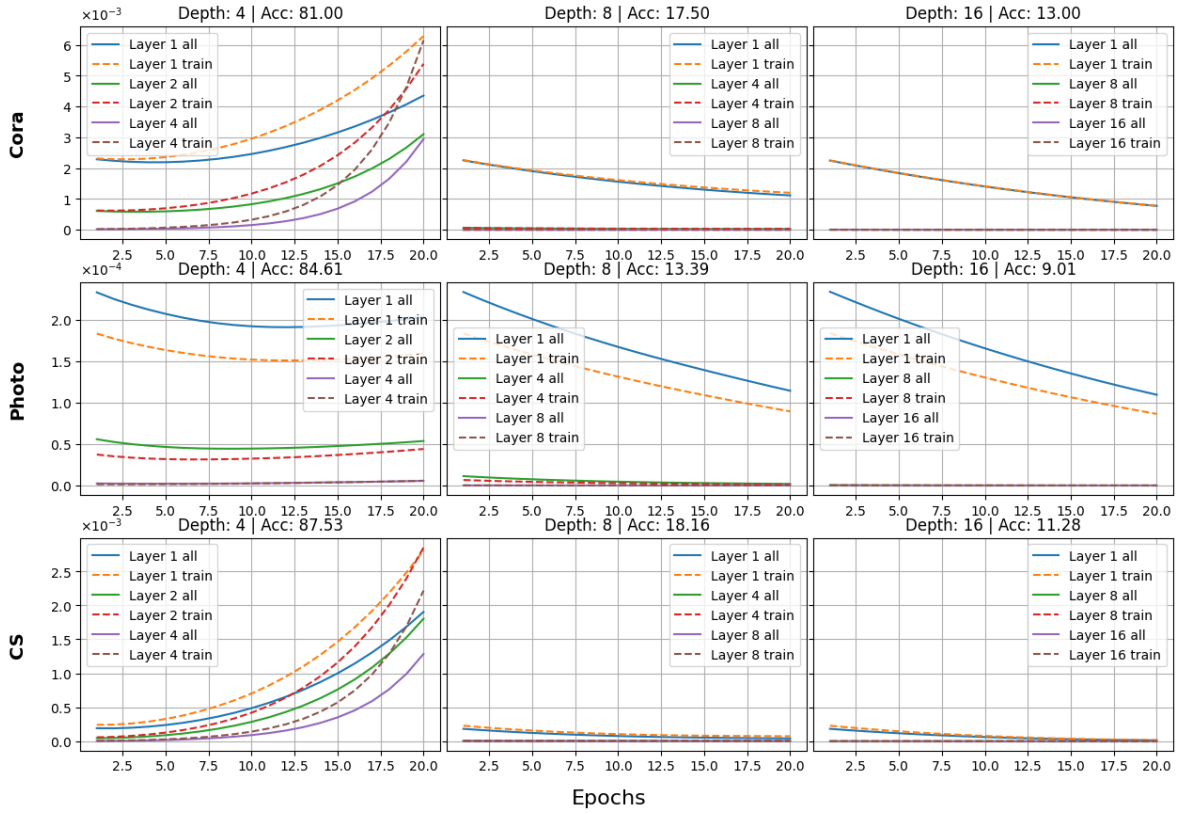

Figure 3.2: Epoch evolution of the Mean Average Squared Euclidean Distance ($MASED$) value of the embeddings of all nodes and training nodes separately. We show results for 3 different depths of a GCN model, illustrating how $MASED$ changes in the first, the middle and the last layer of the model. We also include the accuracy achieved by each model.

MASED evolution at different network depths:
Based on Equation 3.9, we investigate the extent to which $MASED$ exhibits the predicted scaling, with larger values in early layers and smaller ones in deeper layers. Figure 3.2 presents the evolution of $MASED$ across training epochs for a plain GCN at depths 4, 8, and 16. Separate curves are shown for training nodes and for all nodes for each layer on *Cora, Photo*, and *CS* datasets. At depth 4, $MASED$ increases over the 20 epochs, driven most strongly by the first layer while at deeper layers it rises more slowly. This upward trend indicates that feature values are diverging before being mixed in later layers. At depths 8 and 16, all layers show a steady decrease in $MASED$. In subsequent experiments we investigate whether this behavior aligns with embedding norms as Equation 3.9 suggests.
These findings show that $MASED$ is most informative at the first layer, its rise or fall signaling whether the network can expand or shrink embedding differences. Similar plots are presented for the rest of the datasets along with similar plots for ResGCN and SGC in Appendix 2. Note that Figure 3.2 shows only the first 20 epochs; the complete results over 200 epochs can be found in Appendix 2.
Furthermore, these observations align with the mathematical bounds derived from Equation 3.9: both the upper and lower bounds scale as β^L, where L is the layer index and β is

an expression depending on weight matrix properties and the underlying graph topology. Hence the first layer is generally allowed to attain larger values, while deeper layers are more prone to smaller values.

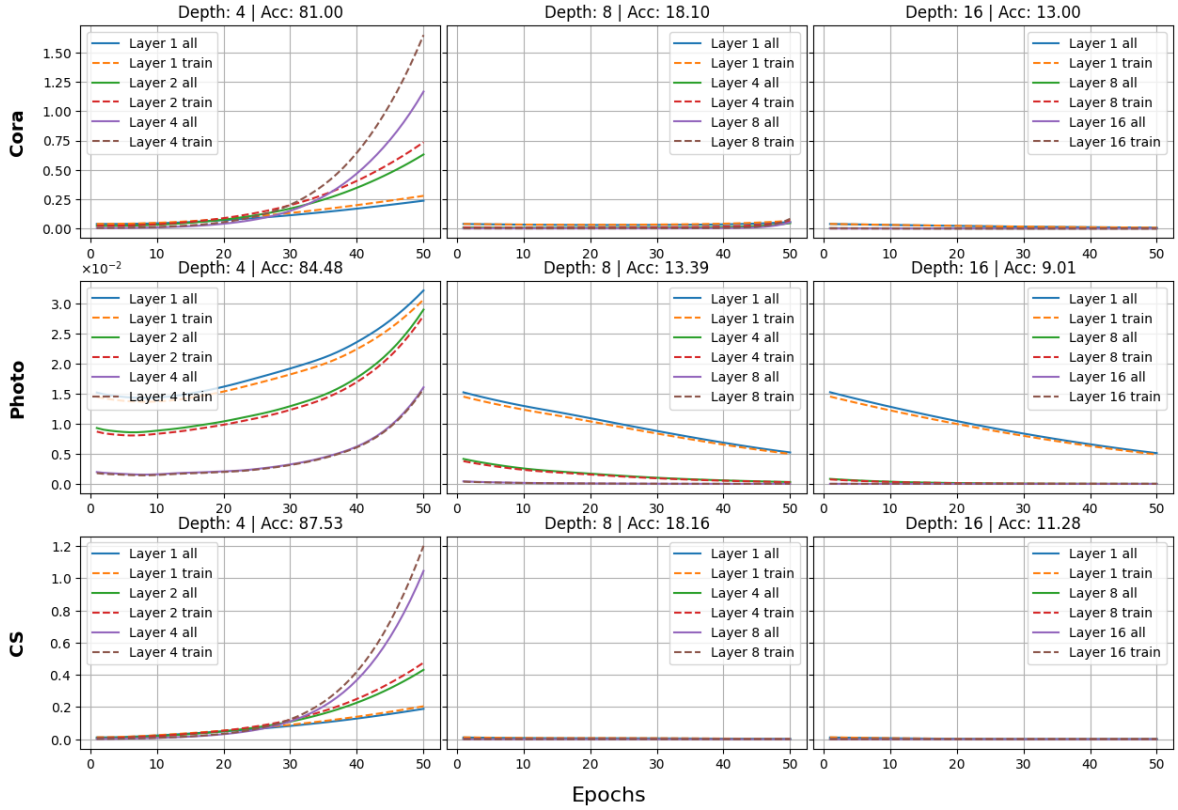

Figure 3.3: Epoch evolution of the average value of the norms of the embeddings of all nodes and of the training nodes separately. We show results for 3 different depths of a GCN model and average norm values in different layers within the model. We show how norms evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

Norms and angles of embeddings at different network depths:
Building on the previous experimental results for $MASED$, we now investigate how norms influence its values and examine whether this behavior aligns with the proposed theoretical predictions. Figure 3.3 plots the average norm of node embeddings for a GCN at depths 4, 8 and 16 on the *Cora, Photo* and *CS* datasets. At depth 4 (high accuracy), the average norm of node embeddings steadily rises over epochs, reflecting growing feature norms and healthy propagation at moderate depth; at depth 8 (smaller accuracy), the increase is far more subdued, its curve remaining nearly flat and indicating early onset of oversmoothing that limits further norm growth; at depth 16 (smallest accuracy), the average norm of node embeddings is essentially constant, signifying collapse of all node representations and complete oversmoothing. The gap between the trajectories for shallow and deep models underscores the sensitivity of norm dynamics to depth, with deeper networks rapidly losing the capacity to amplify node signals effectively. Similar plots for the rest of the datasets, along with the residual GCN (ResGCN) and SGC variants are shown in Appendix 3. Note that Figure 3.3 shows only the first 50 epochs; the complete results over 200 epochs can be found in Appendix 3.

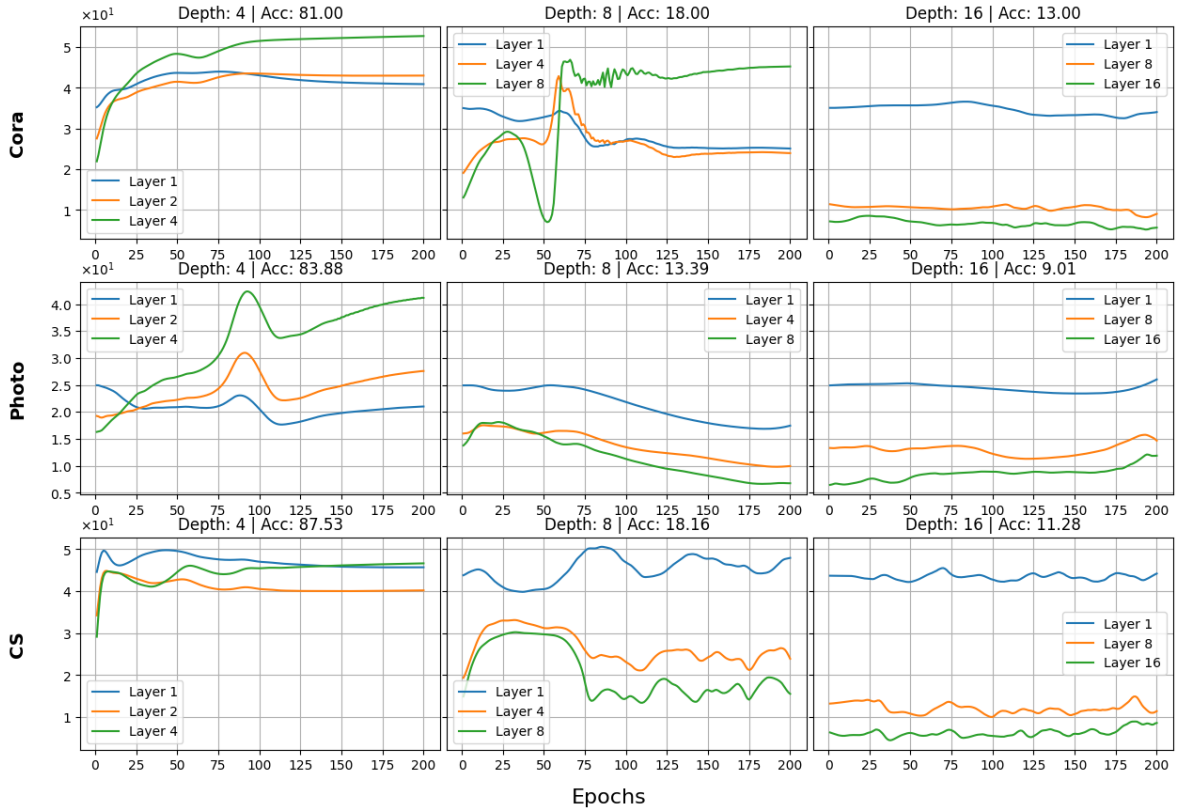

Figure 3.4: Epoch evolution of the average value of the angles between the class centroids of the embeddings of the training nodes. We show results for 3 different depths of a GCN model and average norm values in different layers within the model. We show how angles evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

Figure 3.4 presents the average angle between the class centroids of the training nodes, i.e., the centroids of each class of the training nodes on *Cora*, *Photo* and *CS* datasets. At depth 4 fluctuations start at moderate amplitude and then almost disappear, indicating stable gradient flow and smooth convergence as feature norms increase; at depth 8 the curves have larger early spikes and a slower decay, reflecting instability from deeper aggregation and a more unstable training process; at depth 16 fluctuations drop to near zero almost immediately, mirroring the flat average norm of node embeddings and showing that extreme oversmoothing not only suppresses norm growth but also prevents meaningful parameter updates. These small fluctuations at larger depths, combined with the flat (and almost zero) average norm highlight the key role of the embedding norms in reducing oversmoothing (in agreement with Equation 3.9). One would expect that if the angles between embeddings remain non-zero then the model would be capable of solving the underlying task. However, we observe that if the norms of the embeddings become very small, then the input signal information is lost and the angles between node embeddings do not suffice to capture the differences between node classes. Similar plots for the rest of the datasets and for ResGCN and SGC are provided in Appendix 3.

Reducing oversmoothing through regularization:
Figure 3.5 shows node classification accuracy of a GCN on each dataset. Model depth varies on the horizontal axis and regularization strength λ_w of the proposed *G-Reg* is encoded by

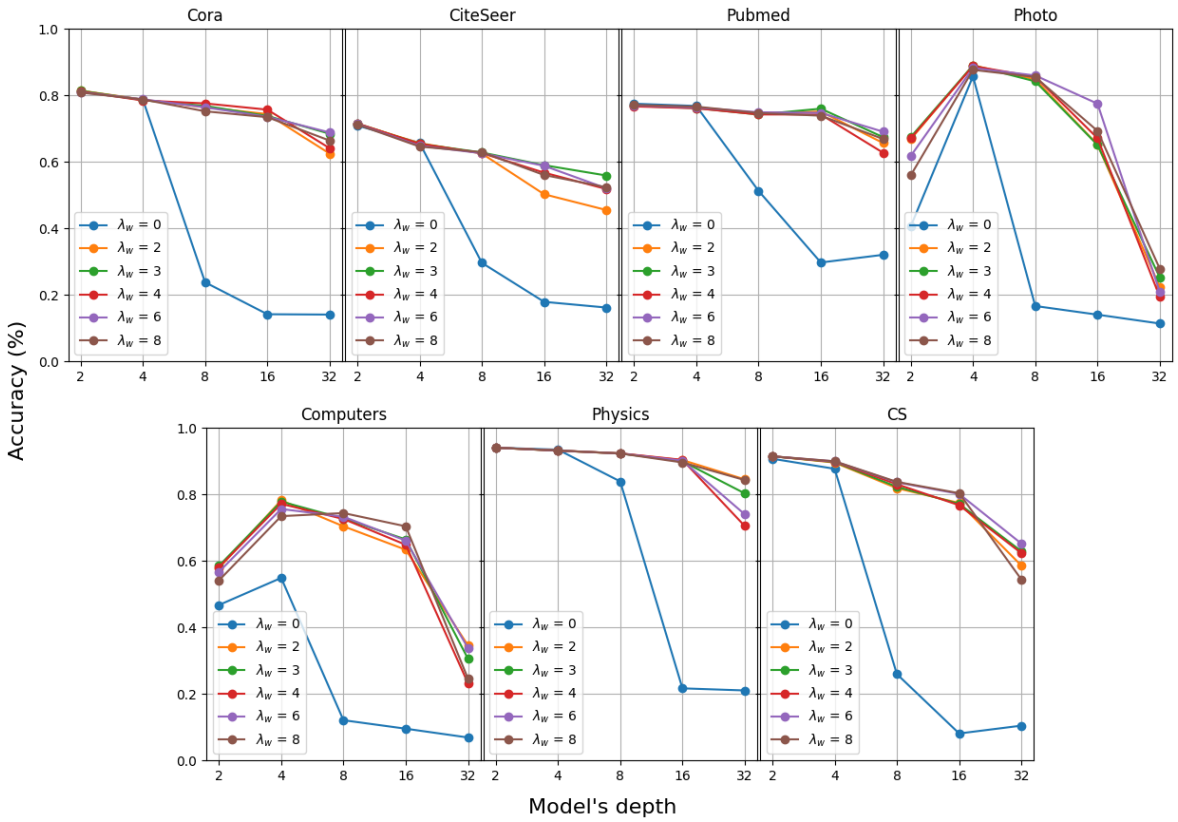

Figure 3.5: GCN with and without the proposed *G-Reg* regularization across 7 datasets for varying depth. We include results for different values of λ_w.

curve color. In all seven subplots the unregularized baseline ($\lambda_w = 0$) peaks at shallow depths and then declines sharply. On the contrary, GCNs with $\lambda_w > 0$ resist oversmoothing and achieve much higher accuracy than their unregularized counterpart. These results confirm that by rewarding larger standard deviation of the weight rows, through *G-Reg*, oversmoothing can be reduced, permitting effective propagation at depths where the unregularized models fail. In particular, the proposed method enables deep architectures that resist oversmoothing and remain capable of solving node classification tasks at large depths. Similar plots for ResGCN and SGC appear in Appendix 4.

Our empirical observations align with the theoretical bounds derived from Equation 3.9, which predict that as depth L increases, both the upper and lower bounds on $MASED$ shrink, thereby inducing oversmoothing. The proposed regularization reduces the co- linearity of weight matrix rows, which, in turn, increases the smallest singular value. As a consequence, the lower bound will increase, leading to larger $MASED$ values and variance of node embeddings in deeper layers. In this way, the regularization counteracts the depth- induced tightening of representational limits and enables the model to reduce oversmoothing.

Performance under the "cold start" scenario:

Table 3.1 reports the best accuracy achieved by each of the three models (GCN, ResGCN, and SGC) on each dataset, under the "cold start" setup, where only the labeled nodes have features initially. The results are presented together with the corresponding λ_w value, and the depth $\#L$ at which each model attains that performance. We observe that nonzero λ_w consistently achieves better performance in larger depths. In particular, regularized

Table 3.1: Comparison of different methods with and without the proposed regularization in the "cold start" scenario. Only the features of the nodes in the training set are available to the model. We present the best accuracy (i.e., Acc.) of the model and the depth (i.e., # Layers) at which this accuracy was achieved, for GCN, ResGCN and SGC.

Dataset	GCN			ResGCN			SGC		
	λ_w	Acc.(%) & std	#L	λ_w	Acc.(%) & std	#L	λ_w	Acc.(%) & std	#L
Cora	0	60.50 ± 4.4	4	0	69.13 ± 0.9	6	0	61.16 ± 0.4	5
	8	**73.26** ± 0.9	19	3	**73.88** ± 0.8	29	1	**65.68** ± 1.6	8
CiteSeer	0	41.95 ± 0.2	4	0	45.85 ± 1.2	7	0	38.86 ± 0.1	7
	4	**48.08** ± 1.3	25	2	**47.97** ± 1.2	23	1	**49.79** ± 0.1	17
Pubmed	0	60.81 ± 3.9	4	0	69.23 ± 0.5	6	0	63.57 ± 0.1	6
	4	**72.15** ± 0.7	25	2	**71.22** ± 1.1	23	0.01	**64.51** ± 0.1	6
Physics	0	51.12 ± 7.9	3	0	82.45 ± 0.6	6	0	**74.54** ± 1.3	5
	8	**89.98** ± 0.7	23	8	**89.98** ± 0.7	32	0.01	74.17 ± 3.6	5
CS	0	14.11 ± 8.2	4	0	47.63 ± 9.8	6	0	71.43 ± 1.5	7
	8	**77.48** ± 2.4	18	8	**79.51** ± 1.4	19	0.5	**73.80** ± 0.1	7
Photo	0	20.99 ± 9.8	2	0	27.14 ± 7.9	17	0	45.06 ± 4.3	2
	2	**81.33** ± 2.8	7	2	**84.16** ± 0.9	6	0.5	**48.91** ± 6.6	2
Computers	0	18.85 ± 9.9	18	0	15.84 ± 9.4	2	0	7.97 ± 3.5	2
	4	**69.74** ± 6.3	10	3	**71.92** ± 3.0	8	0.5	**9.6** ± 2.8	2

models often attain peak performance at depths two to five times greater than the unregularized baselines and consistently increase accuracy by a statistically significant amount. This pattern is consistent across all three models, including the ones that are considered tolerant towards oversmoothing. In cold-start experiments, where unlabeled node features are zeroed out, unregularized models are restricted to very shallow architectures, whereas models with optimized λ_w achieve their best results at much deeper configurations. These findings demonstrate that the proposed regularization not only improves overall accuracy but also enables deeper GNNs and effectively leverages additional propagation steps under both standard and feature-scarce conditions.

Varying the number of SGC layers at different network depths:
Figure 3.6 compares SGC accuracy on *Cora, Photo* and *CS* datasets as the number of trainable weight matrices (SGC layers), but also the overall depth of the model (i.e., number of hops of each node's neighborhood) change. Across all datasets, the 2-layer configuration consistently delivers the highest accuracy, followed by the 1-layer, then the 4-layer, and finally the 8-layer, which performs the worst. This ordering reflects the need for sufficient trainable layers (i.e., more than one) to perform the necessary feature transformations, while avoiding too many trainable weight matrices which can have negative effect. In addition to achieving the highest accuracy, the 2-layer configuration exhibits higher resistance to oversmoothing, as the depth of the model increases, as compared to other models, e.g. the 1-layer one. Results for the remaining datasets appear in Appendix 5.
These results reinforce the argument of subsection 2.2 that too many layers can harm performance by introducing redundant weight matrices. In our experiments, two layers strike the optimal balance, demonstrating that flexible assignment of the total number of hops to

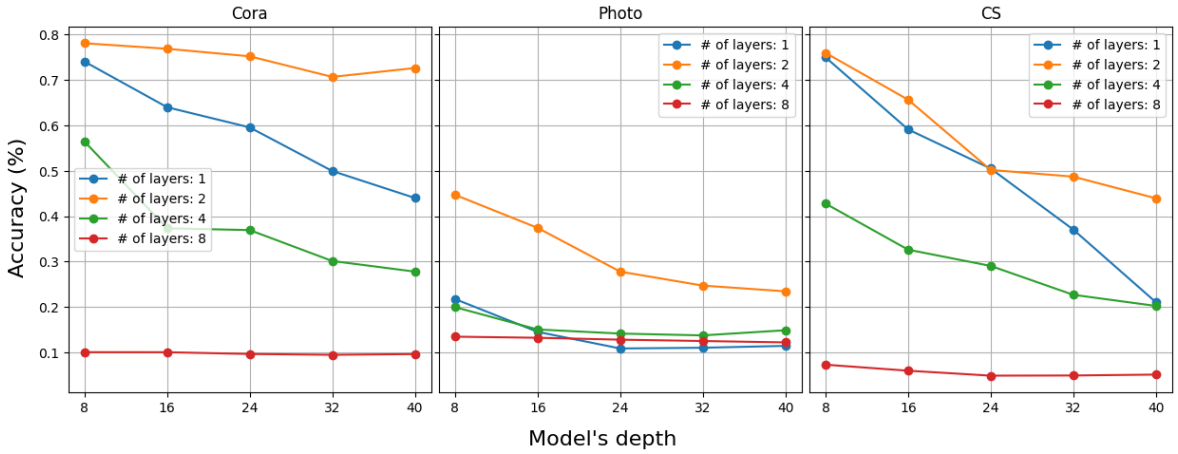

Figure 3.6: Comparison of SGC models with varying number of stacked SGC layers, across 3 different datasets for varying depth. At every depth all models have access to the same information. We only vary the number of trainable weight matrices (i.e., the number of SGC layers).

a small number of weight matrices is essential for deep graph models.

4 Conclusion

In this chapter, we have proposed the use of Mean Average Squared Euclidean Distance ($MASED$) between node embeddings, as a way to quantify the extend of oversmoothing in GNNs. We further derived layer-wise bounds on $MASED$ and shown how they combine across depth to derive the global upper and lower bounds. Based on those bounds, we have highlighted the importance of the norms of the node embeddings and the key role of both the largest and the smallest singular values of the weight matrices. A nonzero smallest singular value can prevent feature collapse and ensures a meaningful lower bound on $MASED$, which, in turn, preserves variance among node representations and gradient flow.

Furthermore, we have shown that tying the number of trainable weight matrices directly to the total number of hops causes redundancy and oversmoothing in deep GNNs. Our theoretical bounds from Equation 3.9 explain this effect and motivate reducing the number of trainable weight matrices to a number that is much lower than the total number of hops. We have also introduced *G-Reg*, a regularization method, which penalizes the small standard deviation between the rows of the weight matrices, hence leading to larger smallest singular values, which, in turn, increase the bounds of Equation 3.9. We have conducted an extensive set of experiments which showed that these strategies improve accuracy and robustness, even when combined with methods that resist oversmoothing in different ways.

The theoretical analysis presented in this chapter opens up a multitude of possible research options to address the problem of oversmoothing. One such direction that we consider important is the interaction between different existing approaches against oversmoothing. $MASED$ highlights the influence of weight matrix singular values and norms, providing a principled way to quantify the problem. Leveraging $MASED$ as a common evaluation tool will enable a systematic exploration of how architectural changes, normalization techniques, and activation adjustments interact, and whether they can be combined in a

complementary manner to enable deeper GNNs.

Chapter 4

Analyzing the Effect of Residual Connections to Oversmoothing in Graph Neural Networks

Chapter 4 explores the role of residual connections in alleviating oversmoothing and enabling the use of deep GNNs for tasks requiring long-range interactions. While a variety of normalization and spectral methods have been proposed [103, 13], residual-style skip connections offer a simple yet powerful mechanism to preserve feature diversity across many layers [16]. The main aim of this chapter is to investigate whether using residual connections allows deep GNNs to create meaningful representations, in problems where long interactions between nodes are needed.
In particular, we study a family of models that employ residual connections, namely APPNP [16], GCNII [11] and PPRGNN [12]. These models attempt to mitigate oversmoothing and enable the construction of deep GNN architectures. We reformulate the aggregation mechanism of the APPNP model and show that it corresponds to a power iteration process. Subsequently, we establish a connection between the residual strength parameter (i.e., α) of the APPNP model and the convergence rate of the model. Furthermore, we analyze GNNs that can be characterized by equations similar to those defining the PPRGNN model, and we investigate the impact of identity mapping (i.e., adding the identity matrix to each weight matrix of the model) proposed in [11].
Specifically, the main contributions of the work presented in this chapter are as follows:

- **Residual Connections**: We analyze separately the effect of adding residual connections in every layer of a GNN, which makes the model equivalent to the weighted sum of the representations of varying depth models with exponentially decreasing weights.

- **Identity Mapping**: We prove that identity mapping creates a weighted powerset sum of the weight matrices of the model. Our results shed light on the relationship between identity mapping and oversmoothing.

- **APPNP and Power Iteration**: We also prove that APPNP boils down to a power iteration with convergence rate dependent on the residual strength parameter α. We derive a formula connecting α, the depth and tolerance of the model, where tolerance measures the proximity of representations created by consecutive layers of the model.

- **Residuals in long-range interactions**: We have conducted experiments under conditions of reduced information, such as in the "cold start" scenario, where node features are available only for the labeled nodes. We further extend this scenario by introducing a new synthetic dataset with controllable long interactions. Our experiments reveal the limitations of methods that use residual connections to deal with such tasks, due to the emphasis

they give to initial node features, hence emulating shallow architectures.

1 Notations

Definition 5 (Lipschitz Continuity). *A function $f : \mathbb{R}^N \to \mathbb{R}^M$ is called Lipschitz continuous if there exists a constant L so that*

$$\forall a, b \in \mathbb{R}^N : ||f(a) - f(b)||_2 \leq L||a - b||_2.$$

For the ReLU activation function, the constant L equals 1, hence ReLU is 1-Lipschitz. We further utilize the powerset of a set of objects, needed in the analysis of the effect of identity mapping.

Theorem 6 ([132]). *Let $A, B \in M_n$ be Hermitian matrices and $\lambda_1(\cdot) \geq \lambda_2(\cdot) \geq ... \geq \lambda_n(\cdot)$ denote the eigenvalues of a matrix in descending order. Then for the eigenvalues of the sum of two matrices it holds:*

$$\lambda_k(A) + \lambda_n(B) \leq \lambda_k(A + B) \leq \lambda_k(A) + \lambda_1(B) \qquad \forall k. \tag{4.1}$$

Weyl's theorem provides both lower and upper bounds on the eigenvalues of a matrix sum. The proof of this theorem is based on the Courant-Fischer Min-max principle [133]. This, in turn, allows us to extend Theorem 6 to the singular values of a matrix sum. By following a similar proof technique to that of Weyl's theorem, we obtain the following Corollary.

Corollary 7. *Let $A, B \in M_n$ be Hermitian matrices and $s_1(\cdot) \geq s_2(\cdot) \geq ... \geq s_n(\cdot)$ denote the singular values of a matrix in descending order. Then for the singular values of the sum of two matrices holds that:*

$$s_k(A) + s_n(B) \leq s_k(A + B) \leq s_k(A) + s_1(B) \qquad \forall k. \tag{4.2}$$

2 Theoretical Analysis

In this section we analyze the effect of residual connections and identity mapping. We first explore the architecture of APPNP, demonstrating its reduction to a power iteration and deriving a formula that describes the similarity of node representations between consecutive layers. Following, we show that models utilizing residual connections with weight matrices (i.e., PPRGNN, GCNII) are equivalent to the sum of shallow models. Finally, we study the case where only identity mapping is being used and prove that it results in a weighted powerset sum of the weight matrices of the model.

2.1 Residuals without Learning Parameters - APPNP

We start by recalling the aggregation scheme of APPNP [16] and demonstrating its equivalence to a power iteration through a transformation that preserves a one-to-one relationship between the original and transformed representations. We also prove theoretically that the eigenvalues of the matrix used for the power iteration are at most equal to 1 and that the residual strength parameter α of APPNP controls the second largest eigenvalue of the matrix. Finally, we derive the formula connecting α with the model's depth and the average distance between node representations of consecutive layers. Following these steps, we

prove that increasing the depth of APPNP beyond a certain threshold has no effect on the final node representations.
The APPNP [16] aggregation scheme is given by:

$$
\begin{aligned}
H^{(0)} &= f_\theta(X), \\
H^{(k+1)} &= (1-\alpha)\hat{A}H^{(k)} + \alpha H^{(0)}, \\
H^{(K)} &= \text{softmax}\left((1-\alpha)\hat{A}H^{(K-1)} + \alpha H^{(0)}\right),
\end{aligned}
\tag{4.3}
$$

where $f_\theta(\cdot)$ is the output of a neural network (i.e., an MLP) operating on the features of each node, i.e., each row of the feature matrix X.
A different way to approach the transformation happening at each layer in APPNP, is by modifying the node representation matrix $H^{(k)}$ and concatenating it vertically with $H^{(0)}$. In that new representation matrix, the upper $N \times C$ block contains node representations created by the respective layer (if the standard APPNP method is used) and the lower $N \times C$ block contains the initial node representations, with C being the number of features of each node. The new matrix $\hat{H}^{(k)} = \begin{bmatrix} H^{(k)} \\ H^{(0)} \end{bmatrix}$ will be a block matrix allowing us to rewrite Equation 4.3 as follows:

$$
\begin{aligned}
\hat{H}^{(k+1)} &= \begin{bmatrix} (1-\alpha)\hat{A} & \alpha I_N \\ 0_N & I_N \end{bmatrix} \hat{H}^{(k)}, \\
H^{(K)} &= \text{softmax}\left(\hat{H}^{(K)}[:N, :]\right),
\end{aligned}
\tag{4.4}
$$

where $I_N, 0_N$ are the identity and zero matrix of size N respectively. The notation $\hat{H}^{(K)}[:N, :]$ denotes that we only keep the first N rows of $\hat{H}^{(K)}$. To simplify our notation we introduce matrix $B = \begin{bmatrix} (1-\alpha)\hat{A} & \alpha I_N \\ 0_N & I_N \end{bmatrix}$, transforming equation 4.4 as follows: $\hat{H}^{(k+1)} = B\hat{H}^{(k)}$, which can also be expressed in terms of the initial representation matrix, as follows:

$$
\hat{H}^{(k+1)} = B^{k+1}\hat{H}^{(0)}. \tag{4.5}
$$

Equation 4.5 illustrates that APPNP is a power iteration with transition matrix B. Since B is a block upper triangular matrix, its eigenvalues are the union of the eigenvalues of the matrices lying on the diagonal. Denoting the set of eigenvalues of a matrix (including multiplicities) as $eig(\cdot)$, we have:

$$
eig(B) = \left((1-\alpha)eig(\hat{A})\right) \bigcup eig(I_N).
$$

The identity matrix I_N has 1 as its only eigenvalue with a geometric multiplicity of N. The augmented normalized adjacency matrix $\hat{A}$ has eigenvalues in the range of $(-1, 1]$. Since the largest eigenvalue of $\hat{A}$ is 1, we conclude that the second largest (excluding multiplicities this time) eigenvalue of B equals $(1-\alpha)$ i.e., $\lambda_2(B) = 1-\alpha$.
Since not every eigenvalue of B is less than 1, a power iteration with B as its transition matrix does not converge to an all-zero matrix, but rather approaches a limiting matrix:

$$
\lim_{k \to \infty} B^k = B_{lim} \neq 0_{2N},
$$

which proves that Equation 4.5 converges to fixed node representations, after a critical number of repetitions (i.e., layers), disregarding subsequent layer addition. These final node representations depend solely on the initial node representations and B_{lim}, disregarding the model's depth beyond the critical depth required for convergence.
To further explore the limits of APPNP, we investigate the number of layers, beyond which the distance of node representations remains the same, up to a fault tolerance error. Let this error be $tol = ||\hat{H}^{(k+1)} - \hat{H}^{(k)}||_2^2 = ||H^{(k+1)} - H^{(k)}||_2^2$, since $\hat{H}^{(k)}$ is a block matrix having the same lower block for every k (i.e., $H^{(0)}$). In a power iteration method, the convergence rate (cr) is determined by the ratio of the second largest to the largest eigenvalue of the transition matrix, i.e. $cr = \frac{\lambda 2(\cdot)}{\lambda_1(\cdot)}$. When B is the transition matrix, $cr = 1 - \alpha < 1$, because α is positive, and the exact value of cr is determined by α.

Lemma 8. *In a power iteration method with a convergence rate equal to $(1 - \alpha)$, in order to achieve a tolerance error of at most tol, the number of iterations needed (L) is given by:*

$$L \sim \frac{log_{10}(tol)}{log_{10}(1 - \alpha)}. \tag{4.6}$$

The proof of Lemma 8 can be found in Appendix 7. Equation 4.6 completes our analysis of APPNP and provides the relationship between α, the model's depth, and the tolerance error between node representations of consecutive layers of APPNP. Equation 4.6 indicates that increasing the number of iterations, which is equivalent to stacking more layers, results in smaller differences between node representations of consecutive layers, which will eventually converge to a predefined set of representations.
Figure 4.1 illustrates the effect of Lemma 8 and its practical implications. As α increases, the required number of layers for convergence decreases. Generally, a model with approximately 20 layers reaches its final representations, with additional layers providing minimal improvement. Figure 4.1 depicts Equation 4.6 for two different values of the constant multiplier of the ratio, emphasizing the trend rather than exact values. The precise constant cannot be determined, as it depends on the eigendecomposition of the initial state in the eigenbasis of the transition matrix (Appendix 7). Consequently, this decomposition is influenced by the learning task, the MLP output, and the structure of B, making an exact analysis infeasible.
Although, APPNP seems to avoid oversmoothing, it actually forces the model to a set of representations produced by the product of B_{lim} with the output of an MLP, i.e. $f_\theta(\cdot)$, resembling the combination of MLP methods with a smart aggregation. APPNP's aggregation is not adaptable to the model's depth, it emphasizes local neighborhood aggregation, and restricts the model from capturing long-range interactions, which is the scenario where deep GNNs are needed.

2.2 Residuals with Learning Parameters

2.2.1 The case of PPRGNN

Instead of decoupling aggregation and propagation as in APPNP, methods that use residual connections augment the output of each layer with initial node representations. Specifically, the PPRGNN-like network family [12] is defined as follows:

$$H^{(l+1)} = \sigma\left(\alpha_l \hat{A} H^{(l)} W^{(l)} + f_\theta(X)\right), \tag{4.7}$$

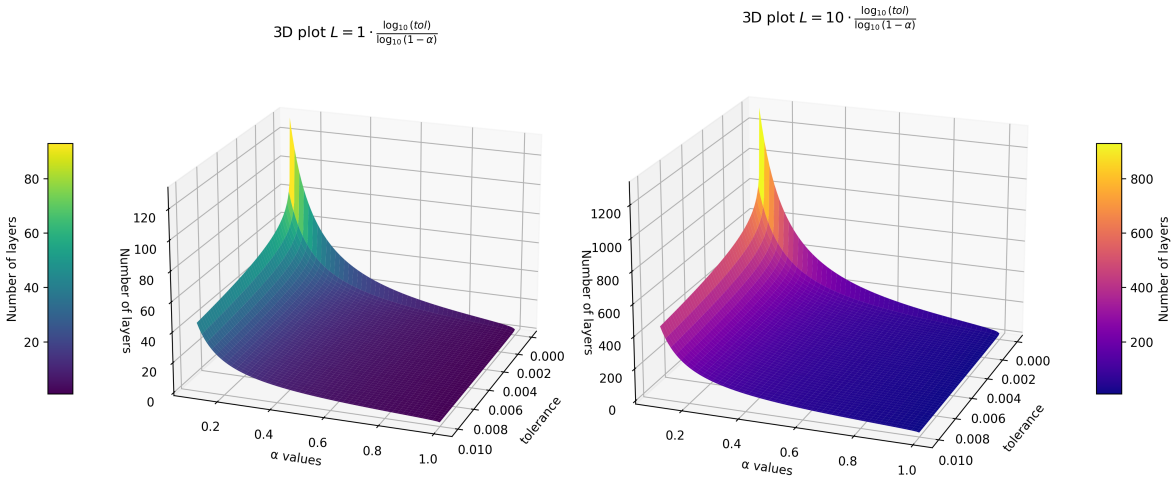

Figure 4.1: Illustration of Equation 4.6 showing the relationship between the model's parameter α, the number of layers and the convergence of node representations (tolerance error). The two plots show the effect of changing the constant multiplier of the ratio, by an order of magnitude (left: 1, and right: 10).

where $H^{(l)}$ denotes nodes representations at layer l, $f_\theta(\cdot)$ is a neural network (i.e. an MLP), which takes as input the initial nodes' features and $\sigma(\cdot)$ is usually an element-wise ReLU function. PPRGNN sets $W^{(l)} = W \ \forall l$ and $\alpha_l = \frac{1}{l}$, but we will analyze the general case of arbitrary $W^{(l)}$ and $a_l \leq 1$.
Using the 1-Lipschitzness of ReLU and expanding Equation 4.7, we will show that such models mainly focus on the local neighborhood, in order to extract information and generate node representations. The aggregation method primarily emphasizes on the output of the upper layers of the network, and gives less consideration to the contributions from the lower layers when forming the final node representations.
Specifically, due to the 1-Lipschitz property of ReLU, excluding it from subsequent analysis is reasonable, as it does not significantly alter the scale of node representations. Expanding Equation 4.7 leads to:

$$H^{(l+1)} = \hat{A}^l f_\theta(X) \prod_{i=1}^{l} \alpha_i W^{(i)} + \hat{A}^{l-1} f_\theta(X) \prod_{i=2}^{l} \alpha_i W^{(i)} + \ldots + \hat{A} f_\theta(X) \alpha_l W^{(l)} + f_\theta(X). \quad (4.8)$$

Note that if $\alpha_l = \frac{1}{l}$ (as proposed in PPRGNN), the product of α_l factors equals $1/l!$, which converges to zero very fast. Let us consider only the limiting case where $\alpha_l = 1$ for every l, because adopting the proposed values by PPRGNN will vanish many of the terms of Equation 4.8 quickly. The vanishing terms are those containing higher powers of the adjacency matrix, representing information flow from distant neighbors and enabling long-range interactions. As these terms diminish, the model's output converges to a sum of shallow model outputs. Consequently, after training, the lower layers (corresponding to the vanishing terms in Equation 4.8) can be removed, allowing the input to pass through the remaining upper layers while producing representations that are nearly the same as those of the full PPRGNN.
Setting $\alpha_l = 1$ for every l, transforms Equation 4.8 to the sum of intermediate outputs (i.e., output of each layer) of a deep GCN model. This architecture bears resemblance to the JK-Networks [108] architecture, with the sum operation as a pooling method atop the

network, a technique not proposed in [108].
In order to compute the node representations at layer L, using Equation 4.8, we need to create L GCNs with depths ranging from 1 to L and sum the resulting node representations. These GCNs will share their weight matrices in reverse order, meaning that a GCN with depth $K < L$ will have the K upper weight matrices of the GCN with depth L. For example, a GCN with two layers will have the weight matrices of the two upper layers of the GCN with L layers. This leads to the observation that Equation 4.8 is simply the sum of the outputs of varying depth GCNs.
This observation, in turn, indicates that PPRGNN-like networks transform deep architectures into the sum of varying depth GCNs. A large fraction of these GCNs (the deep ones) will be oversmoothed as theory suggests [7, 21] and their contribution will be negligible, since they will produce similar representations for all nodes (probably close zero). Even in the cases where the deep GCNs manage to avoid oversmoothing, their contribution diminishes exponentially fast, depending on the layer index, as indicated by Equation 4.8. This results in a model relying primarily on the representations of the shallow GCNs. Consequently, such a deep model can be studied as two disjoint components:
i) Most lower layers have a relatively limited impact on the final node representations. Their weight matrices could, in some cases, retain initial random values with minimal effect on the network's predictions.
ii) The upper layers, in contrast, have a more significant role in shaping node representations. Adjustments in these layers' weight matrices are crucial for the model, as they influence the final node representations and are the primary sites for learning.

2.2.2 The case of GCNII

A prominent method utilizing residual connections in GNNs was introduced by [11], namely GCNII, and defined as:

$$H^{(l+1)} = \sigma\left(\left(\left(1-a_l\right)\hat{A}H^{(l)} + a_l H^{(0)}\right)\left(\left(1-\beta_l\right)I_n + \beta_l W^{(l)}\right)\right), \tag{4.9}$$

where $\sigma(\cdot)$ is an element-wise ReLU function, a_l denotes the residual connection strength and $(1-\beta_l)$ is the strength of identity mapping. In the following analysis we set $\beta_l = 1$, for every l, in order to switch off identity mapping and study the effect of the residual connections. Expanding Equation 4.9 after setting $\beta_l = 1$ and using once again the 1-Lipschitzness of ReLU, we get similar results to Equation 4.8:

$$H^{(l+1)} = \hat{A}^{l+1}H^{(0)}\prod_{k=0}^{l}(1-a_k)W^{(k)} + a_0 W^{(0)}\hat{A}^{l}H^{(0)}\prod_{k=1}^{l}(1-a_k)W^{(k)} + \tag{4.10}$$

$$+a_1 W^{(1)}\hat{A}^{l-1}H^{(0)}\prod_{k=2}^{l}(1-a_k)W^{(k)} + \ldots + a_l\hat{A}^{l-l}H^{(0)}\prod_{k=l}^{l}W^{(k)}.$$

Considering that $0 \leq a_l < 1$ for every layer, we observe that the residual connections utilized in GCNII result in a sum of node representations of shallow models. Hence, the analysis provided in 2.2.1 for PPRGNN-like networks applies also to GCNII. The only difference between Equation 4.8 and Equation 4.10 lies in the form of the product of α_i (i.e., the decay factor) which governs the contribution of deeper models. A short proof of Equation 4.10 can be found in Appendix 8.

2.3 Identity Mapping

In this section we investigate the effect of identity mapping on GNNs and show that the resulting node representations depend on the powerset sum of the weight matrices of the model. In contrast, node representations produced by models that do no use identity mapping depend on the product of the weight matrices. Finally, we show that the powerset sum leads to better guarantees regarding the singular values of the final model, which are directly connected to oversmoothing [7].
A GNN model utilizing identity mapping is defined as:

$$H^{(k+1)} = \sigma\left(\hat{A}H^{(k)}\left(W^{(k)} + I\right)\right), \tag{4.11}$$

where $\sigma(\cdot)$ is a ReLU activation function. Disregarding the activation functions due to ReLU's 1-Lipschitzness transforms Equation 4.11 to the sum of the powerset of all weight matrices of the model. The above statement can be verified by performing an algebraic expansion of Equation 4.11, with k set to $L-1$. To compare identity mapping in GNNs against standard GCN, we present the final node representations of both models (ignoring the activation functions).

$$\begin{aligned} &\text{GCN:} && H^{(L)} = \hat{A}^L H^{(0)} \prod_{i=1}^{L} W^{(i)}. \\ &\text{Identity Mapping:} && H^{(L)} = \hat{A}^L H^{(0)} \sum_{S\subseteq\{1,2,\ldots,L\}} \prod_{i\in S} W^{(i)}, \end{aligned} \tag{4.12}$$

where L is the model's depth.
From Equation 4.12, we observe that GCN uses the product of the weight matrices, which is upper bounded by the product of the largest singular values of these matrices, while a GNN defined as in Equation 4.11 uses the sum of the powerset of the weight matrices. We now prove that the powerset sum of the weight matrices yields a matrix with a bigger largest singular value than the one produced by the product of the weight matrices, indicating that identity mapping is more effective against oversmoothing than GCN. Let us denote with PS_{sum} the sum of the powerset of model's weight matrices, i.e. $PS_{sum} = \sum_{S\subseteq\{1,2,\ldots,L\}} \prod_{i\in S} W^{(i)}$. Using Corollary 7 about the singular values of the sum of matrices we get the following:

$$s_1\left(\prod_{i=1}^{L} W^{(i)}\right) + \sum_{i\in D} s_n(i) \leq s_1(PS_{sum}) \leq \sum_{i\in D'} s_1(i), \tag{4.13}$$

$$D = \left\{\prod_{i\in S} W^{(i)} : S \subseteq \{1,2,\ldots,L\}, S \neq \{1,2,\ldots,L\}\right\},$$

$$D' = \left\{\prod_{i\in S} W^{(i)} : S \subseteq \{1,2,\ldots,L\}\right\}.$$

Note that in the lower bound of Equation 4.13 the first term is the same term that appears in a GCN and controls the convergence speed to the oversmoothing region. Powerset sum produces largest singular values that are strictly larger than the ones of GCN, by a gain

factor defined as $Gain = \sum_{i \in D} s_n(i)$. Note also that $Gain > 1$, because D contains combinations of the weight matrices and the identity matrix I, which has a smallest singular value equal to 1. Thus, it follows that the largest singular value of PS_{sum} is strictly greater than 1 and greater than the largest singular value of the product of the weight matrices, used in GCN. Since the $Gain$ factor depends on the smallest singular values of the powerset of the weight matrices, it can take large values leading the model to instabilities, if identity mapping is used in every layer. In order to avoid this, identity mapping could be used in some layers of the model, e.g. the lower ones.
Recalling the results of Theorem 1 about oversmoothing and its connection to the singular values of weight matrices, we observe that powerset sum provably yields slower convergence speed to the oversmoothing region. This is due to the fact that the sum of the powerset of weight matrices has greater largest singular value than the product of the largest singular values of these matrices. This observation connects identity mapping with existing literature about oversmoothing and shows why it enables deep architectures without 'short-circuiting' initial information to intermediate layers of the model, which is the case when it comes to GNNs using residual connections.
Setting $a_l = 0$ in Equation 4.9 creates a weighted powerset sum of the weight matrices:

$$H^{(k+1)} = \left(\prod_{l=1}^{k} (1 - \beta_l) \right) A^k \cdot X \cdot \sum_{i=1}^{k+1} PS \left(\frac{\beta_l}{1 - \beta_l} W^{(k)} \right). \tag{4.14}$$

This observation sheds more light to the effect of weighted identity addition in GCNII. [11] define $\beta_l = \lambda / l$, where λ is a hyperparameter, causing β_l to decrease geometrically with the layer index. The use of β_l leads to a diminishing contribution of the weight matrices in the upper layers of the model, thereby reducing the significance of the network's depth.

2.4 Deep GNNs and Long Interactions

2.5 Deep GNNs and the Cold Start Problem

The primary challenge in creating deep GNNs is the oversmoothing effect, which is more prevalent as the depth of the network increases. A common debate is whether we really need deep GNN architectures and mainly when do we need them. Most existing datasets consist of homophilic graphs, where valuable information is typically within close proximity to each node (usually within 2 or 3 hops). To explore the potential of deep architectures, we need to examine tasks that require associating nodes that are further apart in the graph. One such task is the "cold start" problem, which is akin to scenarios in recommender systems, where new users or products lack prior feature information. The "cold start" problem can be emulated by removing feature vectors (to be precise, replacing them with all-zero vectors) from all nodes except the labeled ones used for training. This process can be applied to any of the existing benchmark datasets, as proposed by [13]. In such a scenario, the hope is that deep GNNs could recover features from distant nodes and create informative representations. In the context of our work, results from such experiments can shed light on the inherent limitations of residual connections and highlight the potential of deep GNNs to outperform shallow architectures in addressing long-range problems, like the "cold start" problem.

2.6 Synthetic Dataset

In addition to the cold start task, in this work we introduce a new synthetic dataset, named $Star$. The new dataset is a collection of star graphs (see Figure 4.2), where a star graph consists of a central node and a number of path-graphs attached to it. The underlying task is to label the central node using information residing in the peripheral nodes (i.e., nodes at the end of each path). All intermediate nodes in each path do not contain useful information for the classification of the central node. The use of shallow GNNs in this dataset faces what is known as the under-reach problem. The under-reach problem occurs when GNNs lack the required depth to access the information that is needed to solve the underlying task. Similar to the synthetic dataset $NeighborsMatch$ introduced in [8], in the $Star$ dataset we follow a simplistic approach about the underlying task and node features. All nodes have a single-element feature vector as follows: i) peripheral nodes features consist of a positive number, ii) intermediate node features are zero, and iii) all central nodes have the same feature which is different from the features of peripheral nodes. Each central node is classified based on the majority of the peripheral nodes in that particular star graph. If the number of odd-featured peripheral nodes is larger than the even-featured peripheral nodes, then 1 is given as label to the central node, otherwise the label of the central node is 0. Only central nodes are the targets of classification, hence we randomly split them into train, validation and test set.

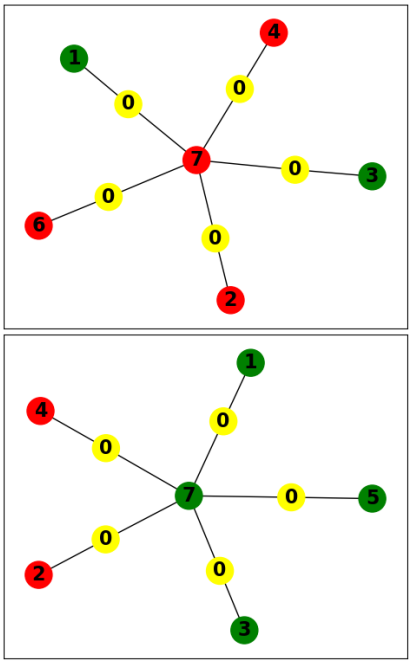

Figure 4.2: Example of the $Star$ dataset with two stars, each having 5 paths of length 2 attached to the central node.

A model that manages to learn the task will be able to propagate information from peripheral nodes to the central one and based on the number of odd-featured peripheral nodes predict the correct label. Changing the length of the paths allows us to select how long the interactions among nodes need to be. A small path length allows shallow models to perform well, while longer paths require deeper GNNs. Consequently, the synthetic dataset can effectively test the ability of GNNs to capture long interactions.

In the two stars of Figure 4.2, the corresponding central nodes have different classes, although they have the same feature value. Deep GNNs are needed in order to solve the underlying task of majority voting, where peripheral nodes vote either odd (green) or even (red) about the label of the central node. Intermediate nodes (yellow) do not affect the classification of the central nodes and only act as buffer nodes necessitating long interactions.

3 Experiments

In this section we experiment extensively with various deep architectures. In particular, we conduct three sets of experiments of increasing difficulty and need for deep networks **i)** We

experiment on benchmark datasets, where the need for long interactions is unknown. **ii)** We experiment on benchmark datasets under the presence of the "cold start" problem, where long interactions among nodes are needed, but we do not know how far the interacting nodes are. **iii)** We experiment on the $Star$ dataset, where both the existence and the length of long interactions are controllable.

3.1 Experimental Setup

Datasets: Aligned to most of the literature, we focus on six well-known benchmarks: *Cora, CiteSeer, Pubmed, Photo, Computers* and *Arxiv*. For the co-citation datasets we use the same data splits as in [25], where all the nodes except the ones used for training and validation are used for testing. For the *Photo* and *Computers* datasets we follow the same splits as in [131], while for the *Arxiv* dataset we utilize the OGB suite presented in [134]. Dataset statistics can be found in Appendix 24. Moreover, as stated above, we use the new synthetic $Star$ dataset to explore controllable long interactions.
Models: We experiment with the proposed architectures of GCN [25], APPNP [16], PPRGNN [12] and GCNII [11].
Hyperparameters: We performed a hyperparameter sweep (see Appendix 11), to determine the optimal hyperparameter values, based on their performance on the validation set. For GCN and PPRGNN, we set the number of hidden units for each layer to 128 across all real datasets. L_2 regularization was applied with a penalty of $5 \cdot 10^{-4}$, and the learning rate was set to 10^{-3}. As for GCNII and APPNP, we used the settings suggested by the authors of the corresponding papers. Dropout was not utilized in any of the models. Depth varied between 2 and 64 layers.
Configuration: Each experiment was run 10 times and we report the average performance over these runs. We train all models within 1200 epochs using Cross Entropy as a loss function. Details on the evolution of the training loss and the computational requirements of the models are provided in Appendix 9.

3.2 Experimental Results

Reducing oversmoothing:
Figure 4.3 presents the classification performance of different architectures on all six benchmark datasets. Models utilizing residual connections seem to avoid oversmoothing and maintain high performance in deep architectures. Note that the accuracy of these models in co-citation networks seem to be independent of their depth.
As explained in section 2.1, APPNP converges to fixed node representations and increasing its depth has negligible or no effect. The performance of PPRGNN seems to degrade slowly as the depth of the network increases. This is because Equation 4.8 gets increasingly populated with oversmoothed terms. These terms are themselves deeper GCNs, as we have proved in sub-section 2.2.1, hence having a negative impact on the final node representations of the model. On the other hand, GNNs that employ residual connections either converge to fixed node representations (APPNP) or act as a summation of shallower networks (PPRGNN). In both cases, performance indicates that oversmoothing could be avoided and infinite depth GNNs are feasible. Additionally, the results of GCNII are also aligned to our analysis, as it manages to maintain high performance in deeper architectures, due to the usage of residual connection and identity mapping. Finally, the accuracy of the simple GCN increases up to some depth, beyond which it fails to create meaningful node

representations, due to oversmoothing.
Among the methods utilizing residual connections, there is no clear "winner" in terms of accuracy. However, it is worth noting that the best performance of each method is achieved at a relatively small depth.

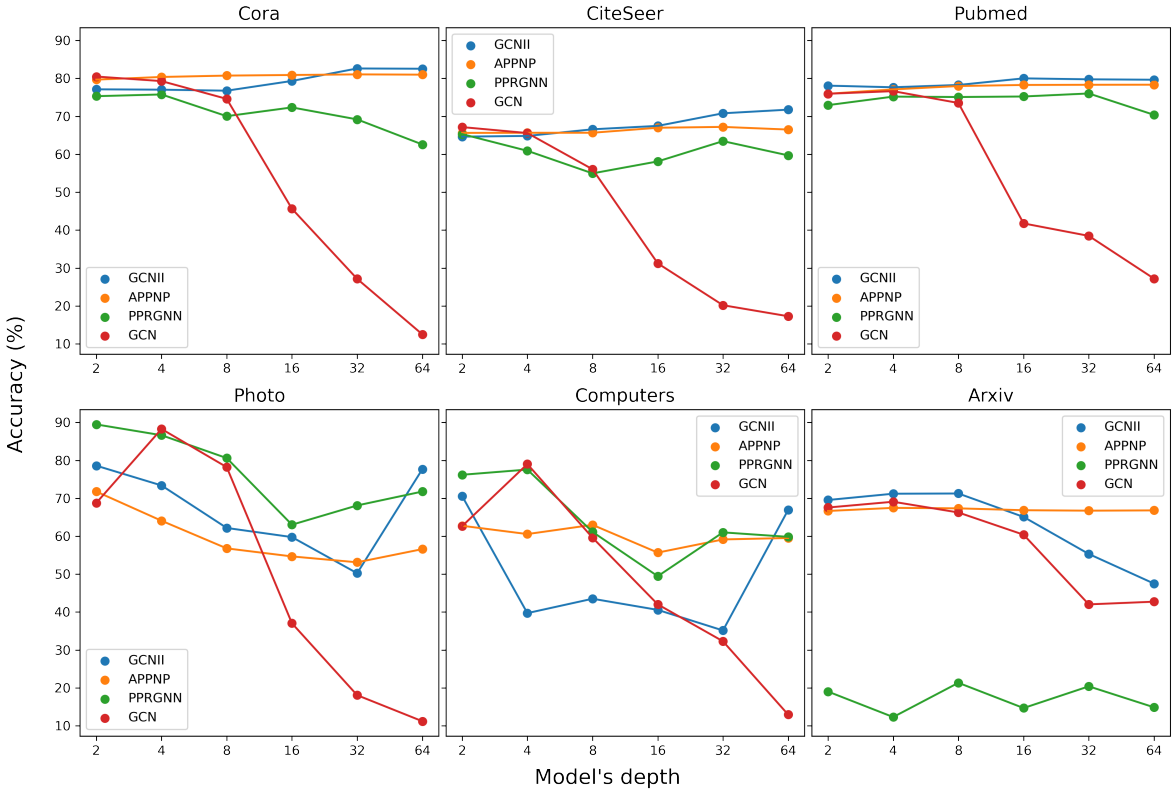

Figure 4.3: Average test node classification accuracy (%) of the four models under investigation in the six benchmark datasets.

Need for deep GNNs:
Given the good performance of all methods, using shallow networks, it is safe to assume that the benchmark datasets used above do not require long interactions among nodes. Hence, we move to a set of experiments that aim to highlight the value of deep architectures. To achieve that we switch our focus to problems involving long-range dependencies. We use the term long-range dependencies in problems, where the close neighborhood of each node does not contain the necessary information to create meaningful representations.
For this purpose, we perform experiments using the "cold start" problem in six datasets. The results are shown in Table 4.1. The table reports the average test node classification accuracy (Acc., %) and the depth (#L) at which each model achieves its best performance. A model that effectively addresses the "cold start" problem should achieve high accuracy, indicating the ability to capture long-range dependencies.
In order to confirm the significance of the results, we used the Friedman test per dataset and overall. In particular, we apply the Friedman test [135] on the ten accuracy measurements obtained per dataset and per method. The null hypothesis is that the performance differences are not significant ($p<0.05$). The null hypothesis is rejected in all datasets, confirming that the differences between methods are significant per dataset (full statistics can be found in Appendix 10). Furthermore, Conover's post-hoc analysis was used to confirm pairwise significant differences.

Generally, one would expect methods designed for deep architectures, such as APPNP and GCNII, to outperform simpler models like GCN. The figures in Table 4.1 do not confirm this hypothesis. This result was further confirmed by a global Friedman test on the per-dataset mean accuracies of the classifiers (full statistics in Appendix 10). Looking closer at the results of the different methods, APPNP achieves the best performance in two datasets (Cora and CiteSeer), PPRGNN in two datasets (Pubmed and Photo), and GCN and GCNII each perform best in one dataset each (Computers and Arxiv, respectively). In all but one of those cases (Pubmed), Conover's post-hoc analysis confirmed the significant difference between the first and the second best method. The results in Table 4.1 suggest that deeper architectures are sometimes necessary, but they must be coupled with effective mechanisms for addressing oversmoothing and aggregation challenges. For instance, APPNP benefits from its propagation mechanism, which ensures nodes access relevant information before converging to fixed representations. Similarly, PPRGNN leverages a combination of shallow GCNs to achieve competitive results in some datasets.
On the other hand, GCNII, despite its depth, fails to achieve high accuracy across most datasets, with its performance often falling below that of simpler models like GCN. For example, GCN outperforms GCNII consistently and with statistical significance (according to Conover's analysis), in most datasets.
Hence, our analysis on the "cold start" problem shows that longer interactions are not captured sufficiently by the methods that we tested, especially the ones using residual connections.

Table 4.1: Average test node classification accuracy (%) and standard deviation on the "cold start" problem in *Cora, CiteSeer, Pubmed, Photo, Computers* and *Arxiv* datasets. With **bold** and underline is the best and second best performing model for each dataset. We also show at what depth (i.e., # Layers) each model achieves its best performance.

Model / Dataset	GCNII		APPNP		PPRGNN		GCN	
	Acc.(%)	#L	Acc.(%)	#L	Acc.(%)	#L	Acc.(%)	#L
Cora	66.19 ± 2.6	12	**70.77** ± 0.6	29	66.31 ± 1.4	6	67.51 ± 1.2	5
CiteSeer	36.17 ± 2.9	18	**49.73** ± 2.0	31	45.14 ± 0.9	7	44.66 ± 0.4	5
Pubmed	48.38 ± 5.2	6	71.89 ± 0.1	24	**72.00** ± 1.0	28	69.82 ± 0.9	7
Photo	20.42 ± 0.5	13	76.50 ± 4.1	7	**83.43** ± 1.2	6	80.81 ± 2.0	5
Computers	16.85 ± 4.5	2	40.72 ± 6.9	4	41.52 ± 9.9	5	**65.09** ± 9.7	7
Arxiv	**67.61** ± 0.5	7	62.11 ± 0.4	4	16.96 ± 7.2	4	66.67 ± 0.2	7

Controllable long interactions:
In order to further shed light to the ability of the different methods to explore long interactions, we experiment with the $Star$ dataset, introduced in section 2.6. Figure 4.4 shows the performance of the models in three different instances of the $Star$ dataset. The proposed dataset allows us to control the number of paths and their length, which, in turn, control the length of the interactions. One may regard the dataset as an extreme case of the "cold start" problem, because each node has no short paths to reach the necessary information, which in turn imposes the need to capture long interactions. The resulting dataset is somewhat imbalanced, with the majority class being represented by roughly 65% of the data instances (i.e., stars). This means that the accuracy of a random classifier would be around 0.65, corresponding to the flat lines observed in the figures for many methods. In other words,

these methods cannot learn meaningful classifiers, indicate models incapable to learn, because they cannot connect the necessary information, in order to create meaningful node representations. In particular, all variants of GCNII and APPNP have an accuracy at or below chance, while PPRGNN achieves higher accuracy for only the simplest of the three datasets. Among all methods, simple GCN performs best, because it does not rely on residual connections and allows the central node to utilize some distant information. However, increasing the network depth seems to hurt the GCN, rather than allowing it to capture longer interactions. In particular, as the length of the long interactions and the number of paths increases, the performance of GCN also drops close to that of random choice. This set of experiments shows the inability of methods utilizing residual connections to access distant information, which is necessary in cases like the one presented in the $Star$ dataset. Consequently, there is a need for a GNN model capable to harness the benefits of its depth, in order to access distant information, while avoiding oversmoothing.

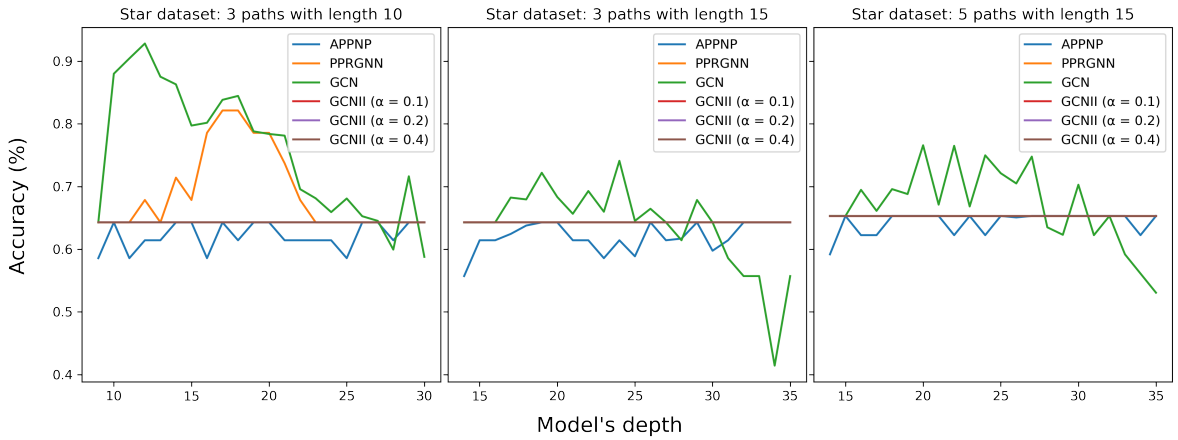

Figure 4.4: Average test classification accuracy (%) of the methods under investigation and two additional variants of GCNII, in the $Star$ dataset. We vary the number of paths and the path length (pl) of the $Star$ dataset and experiment with models of depth in the range $[pl, pl + 20]$. All variants of GCNII, overlap in the graph, making them non-visible.

The convergence of APPNP:
In section 2.1, Equation 4.5, we showed that APPNP performs a power iteration, which converges to a set of predefined node representations. Therefore, APPNP can be regarded as a graph-steered aggregation of the output of an MLP. This means that the model combines node representations (i.e., the output of the MLP) according to the relationships encoded in the graph. The aggregation scheme proposed, utilizing PPR, seems to be extendable to arbitrary depth, leading to the assumption that infinite depth GNNs can be modeled by APPNP. Through PPR, APPNP limits the model's aggregation to the local neighborhood of each node and does not enable deep learning. Figure 4.5 shows that accuracy on the co-citation datasets remains almost constant beyond a specific depth. This behavior is explained by Equation 4.6 as follows: beyond a specific depth, APPNP's power iteration has converged, node representations from consecutive layers are almost identical and yield the same classification results, hence the accuracy remains almost the same.

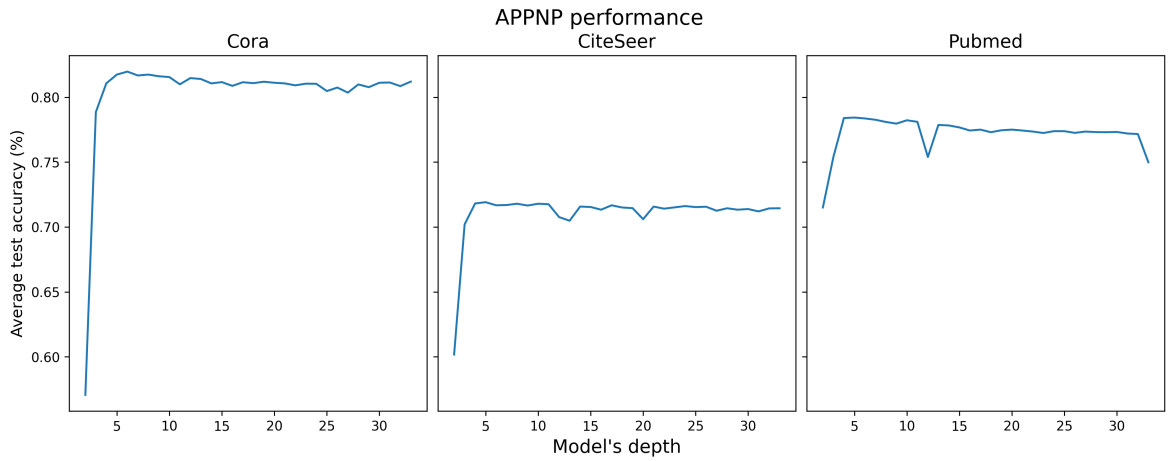

Figure 4.5: Average test classification accuracy (%) of varying depth APPNP models in *Cora, CiteSeer* and *Pubmed* datasets.

The relationship between PPRGNN and GCNs:
According to Equation 4.8 in sub-subection 2.2.1, PPRGNN is equivalent to the sum of shallow GCNs. Following suggestion of the authors of the original paper, we set $\alpha_l = \frac{1}{l}$ and in order to reduce bias, we set $f_\theta(X) = X$, in Equation 4.7. We calculate the sum of the resulting representations of shallow GCNs, by mapping the $W^{(i)}$ matrices of Equation 4.8 to the weight matrices of the GCNs in the following way:

- GCN with 1 layer will have $W^{(L)}$ as its weight matrix
- GCN with 2 layers will have $W^{(L)}$ and $W^{(L-1)}$ as its weight matrices
- ...
- GCN with k layers will have $W^{(L)}, ..., W^{(L-k+1)}$ as its weight matrices

Ignoring the activation functions in Equation 2.1, the output of the GCNs described above can be expressed as:

- GCN with 1 layer: $H^1 = \hat{A}XW^{(L)}$
- GCN with 2 layers: $H^2 = \hat{A}^2XW^{(L)}W^{(L-1)}$
- ...
- GCN with k layers: $H^k = \hat{A}^kX \prod_{i=L-k+1}^{L} W^{(i)}$

A closer look at Equation 4.8 indicates that the above mapping recreates PPRGNN through the summation of the GCNs. To be consistent with the results of Equation 4.8, we need to add the initial feature vectors to the sum of the GCNs as well.
Figure 4.6 shows the number of GCNs of varying depth we need to sum, in order to get node representations close to the full PPRGNN. Note that the number of GCNs we sum indicates the depth of the deepest of them, e.g., if we add 10 GCNs each of them will have a depth of 1,2,..,10 respectively. We present the results over the three co-citation datasets with a fixed depth for the PPRGNN model, namely 20 layers. We vary the number of GCNs and show the respective distance between representations of the last layer of the full PPRGNN

model and the summation of GCNs. The average distance between full PPRGNN and its approximations is shown in logarithmic scale on the y-axis. For each node, we compute the distance between its embeddings generated by the full PPRGNN and its approximation. We then average these distances and take the logarithm of the resulting value.
From Figure 4.6, we observe that the full PPRGNN model can be approximated with as few as 6 relatively shallow GCNs. Similar results hold even if the full PPRGNN is deeper than 20 layers. These results verify the theoretical analysis, and explain that PPRGNN generates shallow models, which cannot capture long interactions.

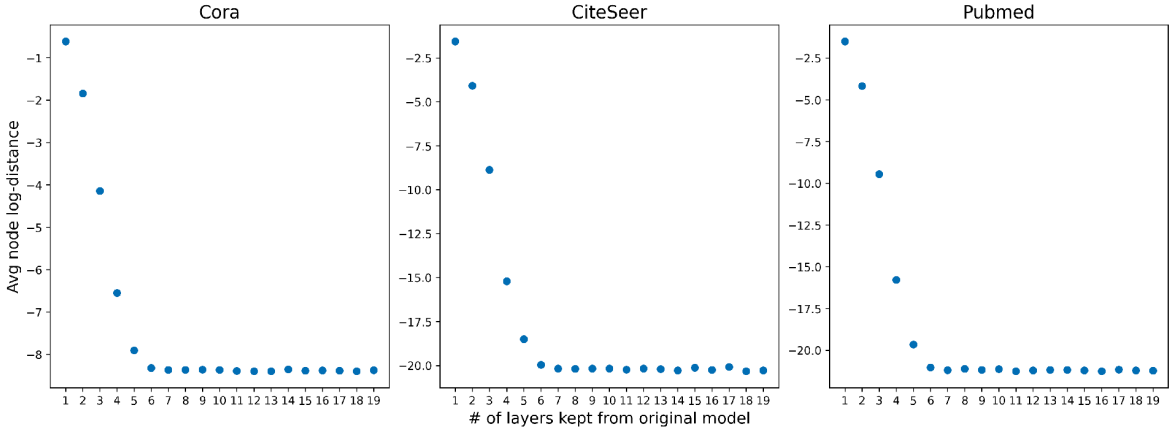

Figure 4.6: Logarithmic average distance of node representations at the last layer of a fully-trained PPRGNN (i.e., with all 20 layers) and an approximation of it using a varying number of GCNs, as per Equation 4.8. X-axis shows the number of shallow GCNs summed in order to approximate the full PPRGNN. GCNs use the weight matrices of only the corresponding upper layers of the original model. Results are shown for the *Cora, CiteSeer* and *Pubmed* datasets.

4 Conclusion

In this chapter, we have analyzed theoretically the effect of residual connections in GNNs and we have studied the aggregation scheme of prominent models. We have shown, that GNNs using residual connections converge to node representations that capture information in the close neighborhood of each node. Hence, they cannot capture long interactions, which require deep models. We have verified our results in a series of experiments, including simpler benchmarks, as well as more demanding synthetic tasks. Despite the ability of deep GNNs that use residual connections to maintain high performance on benchmark datasets, they fail to capture long-range dependencies as needed in the "cold start" problem. To further confirm this observation, we have introduced a new synthetic dataset of controllable long-interactions and have demonstrated the performance of the models on it. These results can steer the development of a mechanism that not only avoids oversmoothing but also leverages the graph information provided by additional layers to enhance model performance.

Chapter 5

Partially Trained Graph Convolutional Networks Resist Oversmoothing

This chapter examines the representational capacity of partially trained graph neural networks, models in which only a single layer is trained while the remaining layers are left at their initial random state. [25] demonstrated that even an untrained GCN can generate informative node embeddings suitable for node classification tasks, as exemplified by their experimentation with the Zachary's karate club network [49]. We systematically explore how such partially trained architectures behave as their width changes, particularly with respect to oversmoothing and embedding quality.
Our findings indicate that increasing the width of partially trained GCNs enhances their performance, making them resistant to oversmoothing. Our analysis provides both experimental results and theoretical insights, in order to construct a comprehensive picture of the behavior of partially trained GCNs. Additionally, we have extended our experiments to both GAT [41] and GCNII [11] architectures, which further verified our claims.
In summary, the main contributions of the work presented in this chapter are as follows:

- **Partially trained GCNs**: We investigate scenarios where only a single layer of a GCN model is permitted to receive updates during training. Our analysis reveals that as the width increases, the model is capable of obtaining high accuracy. We further validate our results through experiments with a Graph Attention Network (GAT).

- **The power of deep partially trained GCNs**: Deep partially trained GCNs are shown experimentally to resist oversmoothing. Additionally, we highlight the advantages of these deep GNNs in scenarios with limited information, such as the "cold start" situation, where node features are only available for labeled nodes in a node classification task. These results were also confirmed for GAT models.

- **Position and type of the trainable layer**: We conducted experiments to explore the effect of placing the trainable layer at different positions within the network. Additionally, we examined different options for the trainable layer and found that utilizing a simple GCN layer is a good choice.

1 Preliminaries

1.1 Initialization and Largest Singular Value

In our analysis, a significant portion of the model remains untrained, retaining its initial weight elements. Therefore, we briefly discuss the common weight initialization method for GNNs, i.e. Glorot initialization [122]. Glorot initialization proposes drawing each element of the weight matrix independently from the same zero-mean distribution, which can be either Uniform or Gaussian. For simplicity in notation and analysis, we adopt Glorot initialization using a zero-mean Gaussian distribution with variance equal to $1/d$, where d represents the width of the layer.

Theorem 9 ((Bai-Yin's law [136]). *Let $Z = Z_{N,n}$ be an $N \times n$ random matrix with elements that are independent and identically distributed random variables having zero mean, unit variance, and finite fourth moment. Suppose that the dimensions N and n tend to infinity while the aspect ratio n/N converges to a constant in [0, 1]. Then, almost surely, we have:*

$$s_{max}(Z) = \sqrt{N} + \sqrt{n} + o(\sqrt{n}). \tag{5.1}$$

Here, $s_{max}(Z)$ denotes the largest singular value of matrix Z, and o(·) is the little–o.

Applying Bai-Yin's law to a Glorot-initialized square matrix leads to the following Corollary.

Corollary 10. *Let $Z = Z_{N,N}$ be a Glorot-initialized random matrix. As N tends to infinity, $s_{max}(Z) \to 2 + o(1)$.*
This is true because Z is initialized with a zero-mean Gaussian distribution having variance equal to 1 / N, instead of unit variance (as Theorem 9 suggests). Therefore, the largest singular value in Equation 5.1 is scaled by a factor of $1 / \sqrt{N}$.

Corollary 10 shows that a randomly Glorot-initialized weight matrix has a largest singular value greater than 2 as its width increases. We leverage this observation to establish a connection between our analysis and the findings in the existing literature about oversmoothing.

1.2 Partially Trained Neural Networks

While fully untrained models are expected to perform poorly, limited attention has been paid to partially trained networks. This approach resembles the fine-tuning [137] of pretrained models, which is common in large neural networks. In classical pretraining methods, a large model is initially trained on generic data and then fine-tuned on a downstream task. During fine-tuning, only a subset of its upper layers are permitted to receive updates and modify the weight values.
As initially noted by [25], even an untrained GCN can generate node features suitable for node classification. In our analysis, we permit only a single layer of the model to receive updates during training, while maintaining the remaining layers in their initial random state.

2 Theoretical Analysis

2.1 Partially Trained GCNs

We propose partially training of a GCN, where the training process is constrained to a single layer of the network. The model architecture remains the same as the one presented in Equation 2.1, but all weight matrices except one remain constant during training. The trainable layer can be placed anywhere within the network. The proposed architecture is presented in Figure 5.1. With training limited to a single layer (i.e., j-th layer) of the network, we can partition the weight matrices into three sets: i) The lower $j-1$ untrained matrices, ii) The j-th trainable matrix, and iii) The upper $L-j$ untrained matrices, where L is the model depth. Since learning is a dynamic process, which cannot be fully controlled, we investigate the properties of the two sets containing the untrained matrices and their effect on the model's behavior.

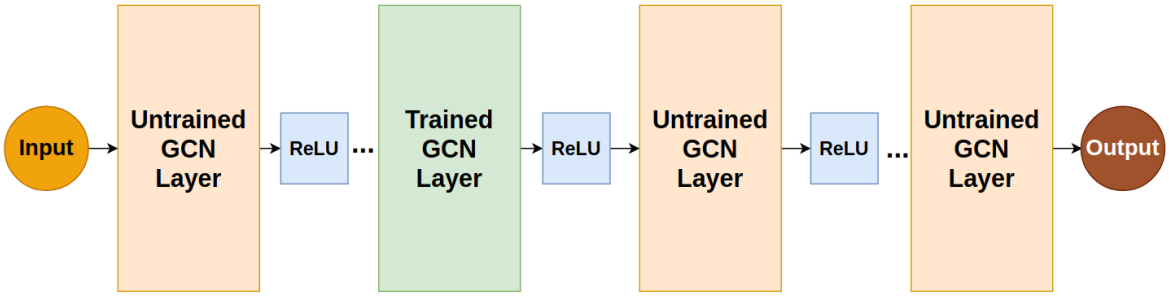

Figure 5.1: Architectural diagram of a partially trained GCN.

To simplify the notation, we omit the influence of the ReLU activation functions in our analysis. This simplification also provides an upper bound for the final node representations of the model, due to the fact that ReLU is 1-Lipschitz. That particular property restricts the extent to which the output of the function can change in response to changes in its input. Therefore, by disregarding the impact of ReLU activation functions, we establish a theoretical ceiling on the improvement achievable in the model's node representations.
Considering the partition of matrices into three sets and disregarding the ReLUs, we rewrite Equation 2.1 to represent the final node embeddings, as follows:

$$H^{(L)} = \hat{A}^{L-1} \cdot X \left(\cdot \prod_{i=1}^{j-1} W^{(i)} \right) \cdot W^{(j)} \cdot \left(\prod_{i=j+1}^{L-1} W^{(i)} \right) =$$

$$\hat{A}^{L-1} \cdot X \cdot W^{left} \cdot W^{(j)} \cdot W^{right}, \tag{5.2}$$

where L is model's depth, $\hat{A}$ is the augmented symmetrically normalized adjacency matrix, X are the initial node features, $W^{(j)}$ is the weight matrix of the trainable layer of the model, and W^{left}, W^{right} denote the products of the untrained weight matrices.
Depending on the value of j, representing the position of the trainable layer, the behavior of the network changes: i) $j = 1$: Learning occurs in the first layer, with subsequent layers propagating and aggregating information using random weights. ii) $j \in [2, L-2]$: Initial node features undergo random transformations, followed by learning in layer j. The learned features are then randomly aggregated by subsequent layers. iii) $j = L-1$: Initial features pass through the random network, where information is propagated and aggregated. The final layer is responsible for generating meaningful representations.

In the following analysis, we study the formulation of W^{left} and W^{right}, the distribution of their elements, and how they affect the information flow within the network. The analysis holds also for Graph Attention Networks (GAT), which can be viewed as a GCN applied to a weighted graph.

2.2 Product of Untrained Weight Matrices

We now take a closer look at the properties of W^{left} and W^{right}. Our analysis focuses on the product of two untrained matrices, denoted as $W^{(1)}$ and $W^{(2)}$, of size $d \times d$. The product of these two matrices is expressed as:

$$W^{(1)} \cdot W^{(2)} = \begin{bmatrix} w^1_{1,1} & \dots & w^1_{1,d} \\ \vdots & \ddots & \vdots \\ w^1_{d,1} & \dots & w^1_{d,d} \end{bmatrix} \cdot \begin{bmatrix} w^2_{1,1} & \dots & w^2_{1,d} \\ \vdots & \ddots & \vdots \\ w^2_{d,1} & \dots & w^2_{d,d} \end{bmatrix} = \\ = \begin{bmatrix} w^{1,2}_{1,1} & \dots & w^{1,2}_{1,d} \\ \vdots & \ddots & \vdots \\ w^{1,2}_{d,1} & \dots & w^{1,2}_{d,d} \end{bmatrix}.$$

We focus on $w^{1,2}_{1,1}$ of the resulting matrix, as the same analysis extends to all elements. Since both $W^{(1)}$ and $W^{(2)}$ remain constant during training, and all their elements are drawn from the same distribution, it suffices to determine the distribution of $w^{1,2}_{1,1}$. We are interested in the distribution of $w^{1,2}_{\vdots,\vdots}$, because they remain fixed throughout the training process, thereby influencing the flow of the initial node information within the model. By characterizing their distribution, we aim to gain insights into how this information evolves and study the properties of the resulting embeddings.
The value of $w^{1,2}_{1,1}$ is computed as:

$$w^{1,2}_{1,1} = \sum_{i=1}^{d} w^1_{1,i} w^2_{i,1}. \tag{5.3}$$

Each term of the sum in Equation 5.3 is a product of two independent and identically distributed Gaussian Random Variables (R.V.). The following Lemma shows the distribution of the product of two such R.V.s.

Lemma 11. *Let Y_1, Y_2 be two independent R.V.s which follow the same Gaussian distribution. Their product can be written as:*

$$Y_1 Y_2 = \frac{(Y_1 + Y_2)^2}{4} - \frac{(Y_1 - Y_2)^2}{4}.$$

If $(Y_1 \pm Y_2)$ follow a Normal distribution then $(Y_1 \pm Y_2)^2$ will follow a Chi-square (χ^2) distribution. Therefore,

$$Y_1 Y_2 \sim \frac{Var(Y_1 + Y_2)}{4} Q - \frac{Var(Y_1 - Y_2)}{4} R = \\ \frac{Var(Y_1) + Var(Y_2)}{4} (Q - R),$$

where $Q, R \sim \chi^2_1$ and are independent of each other.

Lemma 11 shows that each term in the sum of Equation 5.3 follows a χ_1^2 distribution with one degree of freedom. Based on that result, we focus on determining the distribution of the difference of two R.V.s, when each of them follows a χ_1^2 distribution.

Lemma 12. *Let $F = Q - R$, where $Q, R \sim \chi_1^2$. Then F follows a symmetric around zero Variance-Gamma ($V.G._{(\alpha,\beta,\lambda,\mu)}$) distribution with parameters: $\alpha = \lambda = 1/2$ and $\beta = \mu = 0$.*

The proof is based on the moment-generating function of the Chi-square distribution with one degree of freedom.

$$M_Q(t) = M_R(t) = (1 - 2t)^{-1/2}.$$

$$M_F(t) = \left(\frac{1/4}{1/4 - t^2}\right)^{1/2}. \tag{5.4}$$

Equation 5.4 shows that the moment of F matches the moment of a variance-gamma distribution with the aforementioned parameters.
Using the parameter set defined in Lemma 12 to the equations of the Variance-Gamma distribution we get: i) $E[F] = 0$, and ii) $Var(F) = 4$.
Combining the results of Lemma 11 and Lemma 12, we deduce that the product of two independent R.V.s, Y_1, and Y_2, each following a Gaussian distribution, follows a Variance-Gamma distribution. Specifically, the parameter set of the distribution is outlined in Lemma 12, i.e. $Y_1 Y_2 \sim J_{Y_1,Y_2} \cdot F$, where $J_{Y_1,Y_2} = \big(Var(Y_1) + Var(Y_2)\big)/4$, and $F \sim V.G._{(1/2,0,1/2,0)}$.
Applying this result to Equation 5.3 leads to:

$$w_{1,1}^{1,2} \sim \frac{Var(Init(W^{(1)})) + Var(Init(W^{(2)}))}{4} \sum_{i=1}^{d} F_i,$$

where $Var(Init(W^{(k)}))$ is the variance of the distribution used for the initialization of the weight matrix of the k-th layer, and $F_i \sim V.G._{(1/2,0,1/2,0)}$.
Both $W^{(1)}$ and $W^{(1)}$ are initialized by a zero-mean Gaussian distribution with variance equal to $1/d$ (Glorot initialization), which in turn leads to the following result about the distribution of each element of the $W^{(1)} \cdot W^{(2)}$ matrix:

$$w_{1,1}^{1,2} \sim \frac{1/d + 1/d}{4} \sum_{i=1}^{d} F_i = \frac{1}{2d} \sum_{i=1}^{d} F_i = \frac{1}{2} \bar{F}_d,$$

with $\bar{F}_d$ being the average of F_i's.
For the last step of our analysis of the product of the weight matrices, we utilize the Central Limit Theorem (CLT). CLT states that the distribution of $\sqrt{d}(\bar{F}_d - E[F_i]) \xrightarrow{approx.} \mathcal{N}(0, Var(F_i))$ as d increases.
Considering that $E[F_i] = 0$ and $Var(F_i) = 4$, we conclude that the distribution of $\bar{F}_d \xrightarrow{approx.} \mathcal{N}\left(0, \frac{4}{d}\right)$.
Hence, we arrive at the distribution of $w_{1,1}^{1,2} \sim \frac{1}{2}\bar{F}_d \xrightarrow{approx.} \mathcal{N}\left(0, \frac{1}{d}\right)$, which is the same distribution as the one used in Glorot initialization. Consequently, the elements of the product of two weight matrices follow the same distribution as that followed by the elements of the initial matrices. Using induction we can also prove that the product of an arbitrary number of untrained weight matrices is a Gaussian matrix, with elements following a $\mathcal{N}\left(0, \frac{1}{d}\right)$ distribution, as model's width (i.e., d) increases.

2.3 Effect of W^{left}

We have proved that both W^{left} and W^{right} are Gaussian matrices, with random elements following the distribution of the Glorot initialization method. Looking back at Equation 5.2, since both the graph topology and the initial node features are fixed, their product ($B = \hat{A}^{L-1}X$) remains constant during training. Therefore, we investigate the effect of multiplying a Gaussian matrix with a constant matrix, i.e. matrix B. In order to keep the notation simple, we analyze the case of $j = L-1$ in Equation 5.2, which means that W^{right} disappears. Equation 5.2 is transformed as follows:

$$H^{(L)} = \left(B \cdot W^{left}\right) \cdot W^{(j)}. \tag{5.5}$$

Lemma 13. *Let Y be a random vector with $Y \sim \mathcal{N}(\mu, \Sigma)$, and O a non-singular matrix, the distribution of $O \cdot Y$ is given as:*

$$O \cdot Y \sim \mathcal{N}(O\mu, O\Sigma O^T).$$

According to Lemma 13, every column of the product of a Gaussian matrix and a non-singular matrix follows the same Gaussian distribution. Hence the resulting matrix also follows that particular Gaussian distribution.
The proof of Lemma 13 comes from the definition of a multivariate Gaussian random vector and the impact of an affine transformation on its distribution. Lemma 13 indicates that $B \cdot W^{left} \sim \mathcal{N}(0, \frac{1}{d}BB^T)$, because $W^{left} \sim \mathcal{N}(0, \frac{1}{d})$ and B is non-singular.
Let us take a closer look at matrix B, ignoring temporarily W^{left}. Matrix B comprises node features that are averaged multiple times, based on the underlying graph topology. As the depth increases, the number of copies of $\hat{A}$ in B also increases, intensifying the averaging process. This, in turn, increases the similarity and the correlation between the feature vectors (i.e., rows of B). Highly correlated features become too similar, causing oversmoothing. This problem can also be viewed as a reduction in the standard deviation of each feature column in B.
We focus on the training subset, as it drives representations (via weight updates) either towards or away from the oversmoothing subspace. Let B_{train} be the observed dataset, i.e., a submatrix of B containing only the respective rows of the training (i.e., labeled) nodes. We assume that the initial node features are independent and identically distributed (i.i.d.), drawn from an unknown distribution. Consequently, the row vectors of B_{train}, representing node features averaged multiple times, also follow an unknown distribution. Additionally, we assume that these row vectors B_i, follow a continuous and irreducible p-dimensional distribution with mean μ and a positive-definite covariance matrix Σ, e.g., $B_i \sim N_p(\mu, \Sigma)$. Lastly, we make the mild assumption that μ = 0.
Next, we focus on estimating the sample correlation matrix. Let $Y = (Y_{ij})_{N_{train} \times p}$ be the matrix resulting from the original dataset B_{train}, normalized by column, i.e. $Y_{ij} = (B_{ij} - \hat{\mu}_j)/\hat{\sigma}_j$, where $\hat{\mu}_j = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} B_{ij}$ and $\hat{\sigma_j}^2 = \frac{1}{N_{train}-1} \sum_{i=1}^{N_{train}} (B_{ij} - \hat{\mu}_j)^2$. Then the sample correlation matrix R is defined as:

$$R = \frac{1}{N_{train}} Y^T Y.$$

Since we assumed that $\hat{\mu}_j = 0 \; \forall j$, we get that $Y_{ij} = B_{ij}/\hat{\sigma}_j$, which in turn leads to $Y = B_{train}\hat{\Sigma}$, where $\hat{\Sigma} = diag(1/\sigma_1, 1/\sigma_2, ..., 1/\sigma_C)$.

In this formula of $\hat{\Sigma}$, σ_i represents the standard deviation of each feature column in B, computed column-wise. That is, σ_i corresponds to the standard deviation of the i-th feature across all nodes, assuming each node has C initial features. This, in turn, transforms R as follows:

$$R = \frac{1}{N_{train}} \hat{\Sigma} B_{train}^T B_{train} \hat{\Sigma} = \hat{\Sigma} \cdot S \cdot \hat{\Sigma},$$

where $S = \frac{1}{N_{train}} B_{train}^T B_{train}$ is the sample covariance matrix, because $\hat{\mu}_j = 0 \; \forall j$.
Leveraging the properties of the diagonal matrix $\hat{\Sigma}$, we can derive bounds for R as follows:

$$C_{lower} \cdot S = \left(\min_j \frac{1}{\sigma_j} \right)^2 S \leq R \leq \left(\max_j \frac{1}{\sigma_j} \right)^2 S = C_{upper} \cdot S, \tag{5.6}$$

where $C_{lower} = \left(\min_j \frac{1}{\sigma_j} \right)^2$, and $C_{upper} = \left(\max_j \frac{1}{\sigma_j} \right)^2$.

The addition of W^{left}, alters the distribution from which the samples B'_i are drawn. The product $B \cdot W^{left}$ follows a Gaussian distribution with a predefined variance. We proved that $B'_i \sim N\left(0, \frac{N_{train}}{d} \left(\frac{1}{N_{train}} B_{train}^T B_{train} \right) \right) = N\left(0, \frac{N_{train}}{d} S\right)$. As a consequence, the correlation matrix R' after the addition of W^{left} is bounded as follows:

$$\frac{N_{train}}{d} C_{lower} \cdot S \leq R' \leq \frac{N_{train}}{d} C_{upper} \cdot S, \tag{5.7}$$

where d the width of the trainable layer.
Therefore, the presence of W^{left} in Equation 5.5 modifies the upper and lower bounds of the correlation matrix.
We observe that both C_{lower} and C_{upper} depend on the inverse of the smallest and largest standard deviations among the columns of B_{train}, respectively. As depth increases and oversmoothing occurs, the largest standard deviation shrinks. This shrinkage increases the value of C_{lower}, which, in turn, increases the lower bound of R, leading to higher correlation values.
On the contrary, when W^{left} is introduced, the denominator d keeps the upper bound of R' small. Comparing Equation 5.6 with Equation 5.7 leads to the following conclusion: *Increasing the depth reduces the smallest standard deviation among feature elements, thereby increasing the lower bound and overall values of the correlation matrix. The introduction of W^{left} adjusts the upper bound of the correlation matrix and reduces its values as d increases.*
Since the rows of B are highly correlated, as the depth increases, we expect that the covariance matrix (and its estimator S) will contain large elements. This, in turn, highlights the role of the term $\frac{N_{train}}{d}$, which tones down these large elements. Therefore, we would like to adjust the elements of the correlation matrix, in order to: i) prevent similar embeddings for all nodes, and ii) avoid highly dissimilar embeddings between nodes of the same class. Therefore, the term $\frac{N_{train}}{d}$ plays a crucial role in reducing the correlation between the rows of matrix B. Given that N_{train} is fixed, we can reduce the correlation, by increasing d.
Additionally, since we assume zero-mean distribution of the row values, the cosine similarity of vectors equals their correlation coefficient. Hence, lower values in the correlation matrix correspond to lower cosine similarity, indicating less similar node representations. This observation highlights the effect of W^{left}, which can increase the dissimilarity between node embeddings, counteracting the oversmoothing caused by repeated multiplications with the adjacency matrix. Finally, we observe that d serves a dual purpose, as

its increase (a) strengthens the conditions of CLT; namely the more random variables we sum, the stronger is the convergence of their average to a Gaussian distribution, and (b) reduces the correlation between node features (as explained above). However, excessively increasing d might lead to very dissimilar node embeddings and poor training performance.

2.4 Untrained Weight Matrices and Oversmoothing

Conceptually, as discussed in section 2.3, untrained weight matrices act as reducers of the correlation between node embeddings before learning occurs. Their presence helps the model to preserve the informative aspects of the initial node embeddings, while also aggregating information from distant nodes in the multi-hop neighborhood, due to the powers of the adjacency matrix. Consequently, the trainable layer is applied to node embeddings that contain information from distant nodes, thereby enabling effective learning.
We now shift our focus to the singular values of the weight matrices, which are known to influence oversmoothing. The untrained matrices are all initialized using Glorot method and remain constant during training, which means that their singular values do not change. Leveraging Corollary 10 about these matrices, we conclude that their largest singular value is greater than 2, as their width (i.e., d) tends to infinity. Therefore, the product of the largest singular values of the untrained weight matrices maintains a high value. According to Corollary 2, a large value of the product of the largest singular values of the weight matrices allows the model to reduce oversmoothing, which is what we expect from the partially trained models.

3 Experiments

3.1 Experimental Setup

Datasets: We conduct experiments on well-known benchmark datasets for node classification: *Cora, CiteSeer, Pubmed*, utilizing the same data splits as in [25], where all nodes except the ones used for training and validation are used for testing. Furthermore, we use the *Photo, Computer, Physics* and *CS* datasets, adopting the splitting method presented in [138]. To assess the real world applicability of our partial training paradigm, we extended our evaluation to the *Arxiv* dataset [134] as well. Dataset statistics can be found in Appendix 24.
Models: The focus of our work is on the effect of the model's width, under the partially trained setting. Therefore, our base model is the simple and conventional GCN architecture [25]. To further validate the proposed method, we also experiment with the GAT model [41], which leverages an attention mechanism for node feature aggregation, i.e., the features of each node are derived by the weighted sum of the features of its neighbors. Additionally, we experimented with a more advanced method that addresses the problem of oversmoothing, through the use of residual connections and identity mapping, namely GCNII [11]. The reason for including such a method was to verify that the proposed partial training approach does not harm its performance. All partially trained models update a single layer, and keep the rest of the model weights frozen to the initial random values.
Hyperparameters: We vary the width of GCNs and GATs between 64 and 8192. The fully trained models are subject to L_2 regularization with a penalty of $5 \cdot 10^{-4}$, while the partially trained models do not use regularization. The learning rate for the fully trained models is set to 10^{-3}, whereas for the partially trained, it is set to 0.1. For GAT we use a single atten-

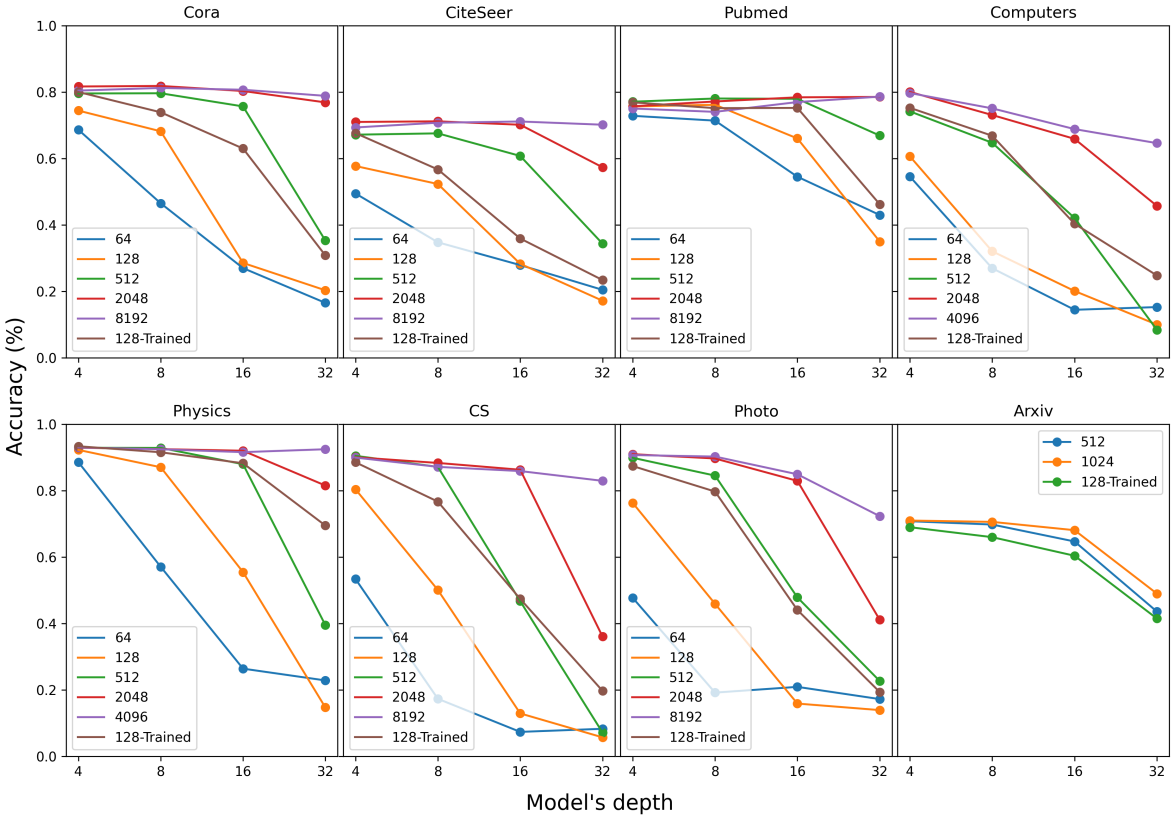

Figure 5.2: Comparison between a fully trained GCN and different configurations (in terms of width) of partially trained GCNs across 8 datasets for varying depth. The trainable layer is always the second.

tion head. The above values are determined, based on the performance of the models on the validation set.

Configuration: Each experiment is run 10 times and we report the average accuracy and standard deviation over these runs. We train all models for 200 epochs using Cross Entropy as the loss function. For the *Arxiv* dataset training was run for 10.000 epochs to ensure convergence.

3.2 Experimental Results

One trainable layer for different widths and depths:

In our first experiment, we investigate the effect of width and depth in partially trained GCNs and GATs, of varying depth. Specifically, in Figure 5.2 and Figure 5.3 we observe that wider networks tend to be more resistant to oversmoothing, as the depth of the network increases. This aligns with our theoretical findings, as presented in section 5.2. The trainable layer of the networks is the second, a choice which was based on experimentation. Additionally, partially trained GCNs and GATs outperform fully trained ones given a wide enough trainable layer. Therefore, a wide single layer can effectively learn informative node representations, when node information is appropriately aggregated by untrained layers.

To further explore the capabilities of the partial training paradigm and assess real-world applicability, we evaluated GCN with and without partial training on the large-scale *Arxiv* dataset. Across all depths, both partially trained variants yield consistent accuracy im-

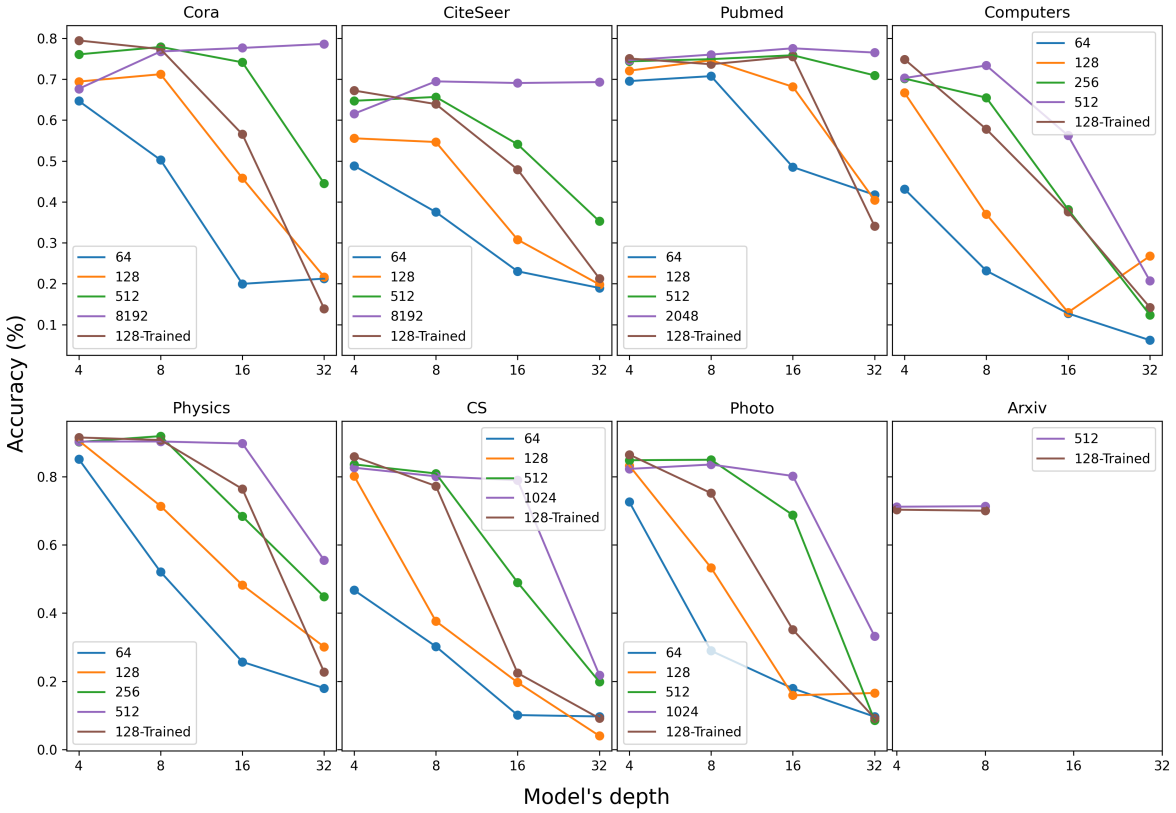

Figure 5.3: Comparison between a fully trained GAT and different configurations (in terms of width) of partially trained GATs across 8 datasets for varying depth. The trainable layer is always the second. Arxiv experiments were limited to 8 layers due to hardware constraints.

provements over the fully-trained baseline. At shallow depths (4 and 8 layers), the gains are modest, but increase at deeper settings (16 and 32 layers). For the GAT model, experiments on the Arxiv dataset were limited to 8 layers due to hardware constraints. The above result highlights again that partially trained models are more effective under conditions where oversmoothing risk is higher. Additionally, the results with the *Arxiv* dataset demonstrate that the partial training setup can scale to million-edge graphs, with its benefits becoming more evident at greater depths where the risk of oversmoothing is higher.

It is also noteworthy, that increasing the width of fully trained GCNs and GATs does not improve their performance, while, in contrast, we observe significant improvements for the partially trained models as their width increases. In particular, Figure 5.4 and Figure 5.5 illustrate that fully trained models perform better with smaller widths. Wider fully trained models achieve lower accuracy, likely due to the increased number of trainable parameters that hinder the optimization process.

As mentioned above, we also experimented with the more advanced GCNII across varying network depths. The results of those experiments, which can be found in Appendix 12, show that the proposed approach does not harm the performance of GCNII and in some cases may even improve it slightly.

Partially trained GNNs can solve the "cold start" problem:

The second set of experiments aimed at demonstrating the advantages of using deep architectures that are resistant to oversmoothing. For this purpose, we simulate the "cold start" scenario, which resembles the situation where a new item enters a recommender system

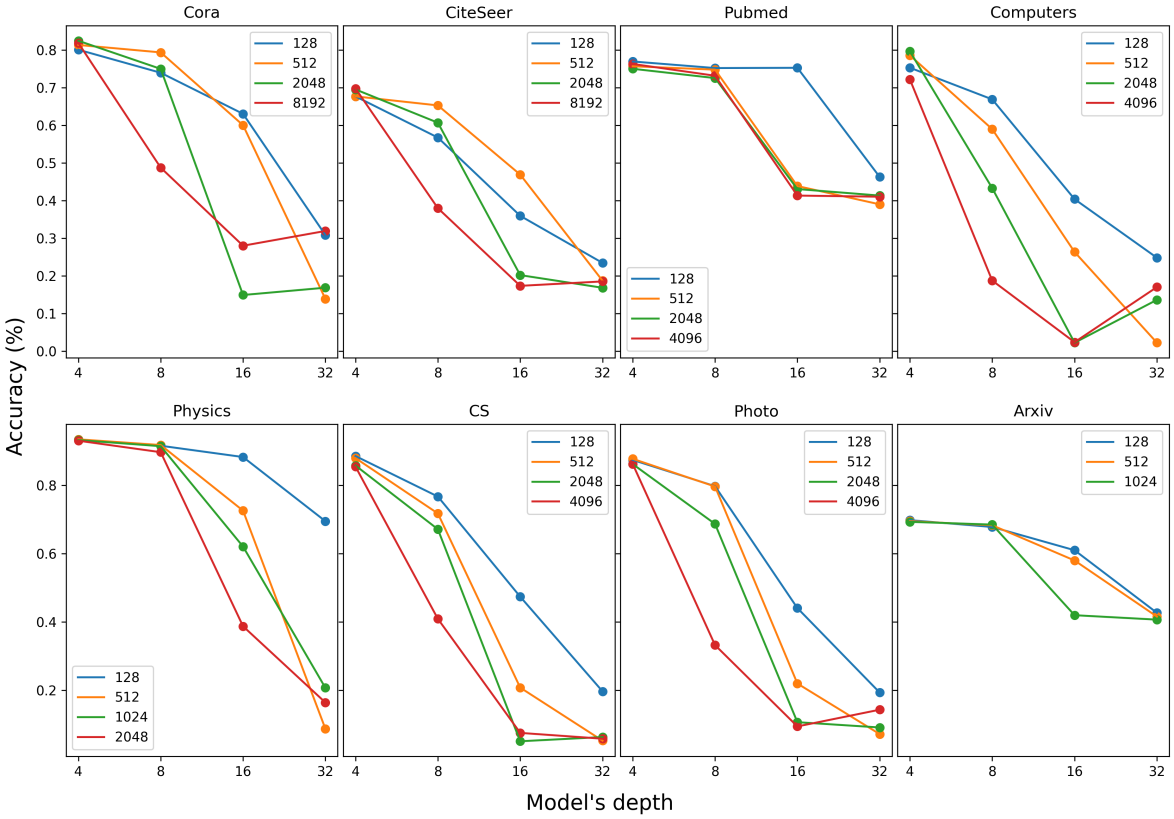

Figure 5.4: Comparison between different configurations (in terms of width) of fully trained GCNs across 8 datasets for varying depth.

without any features. This setup, initially proposed in [13], requires leveraging information from distant nodes to create meaningful embeddings for the new item. In order to simulate it, we create the cold-start variants of the datasets, by removing feature vectors from the unlabeled nodes and replacing them with all-zero vectors. Classification accuracies under this "cold start" scenario (see Table 5.1) are lower than in the standard setting, as the model is trained without feature information for the unlabeled nodes. These modified datasets enable us to assess the effectiveness of deep architectures in handling scenarios with limited feature information.

For different widths of partially trained GCNs and GATs, we present the best performance achieved and the depth at which the model attains that performance in Table 5.1.

In this context, we observed that partially trained models perform better if both of their last two layers are allowed to receive updates. This is due to the increased difficulty of the problem, compared to standard node classification without missing features. From Table 5.1, we observe that the deeper models consistently outperform shallower ones in almost every dataset, underscoring the necessity of relatively deep architectures, which are less prone to oversmoothing. Additionally, Table 5.1 shows that as width increases, the performance of the models improves, and the depth at which optimal performance is attainable also increases. It is worth noting that the fully trained GCNs and GATs generally exhibit inferior performance, which is also attained by shallower architectures. This is due to the high-degree of oversmoothing of such models, which makes deeper architectures unusable.

Role of the position of the trainable layer:

Having demonstrated the advantages of partially trained GCNs and GATs in both standard node classification tasks and the "cold start" problem, we now conduct a set of ablation

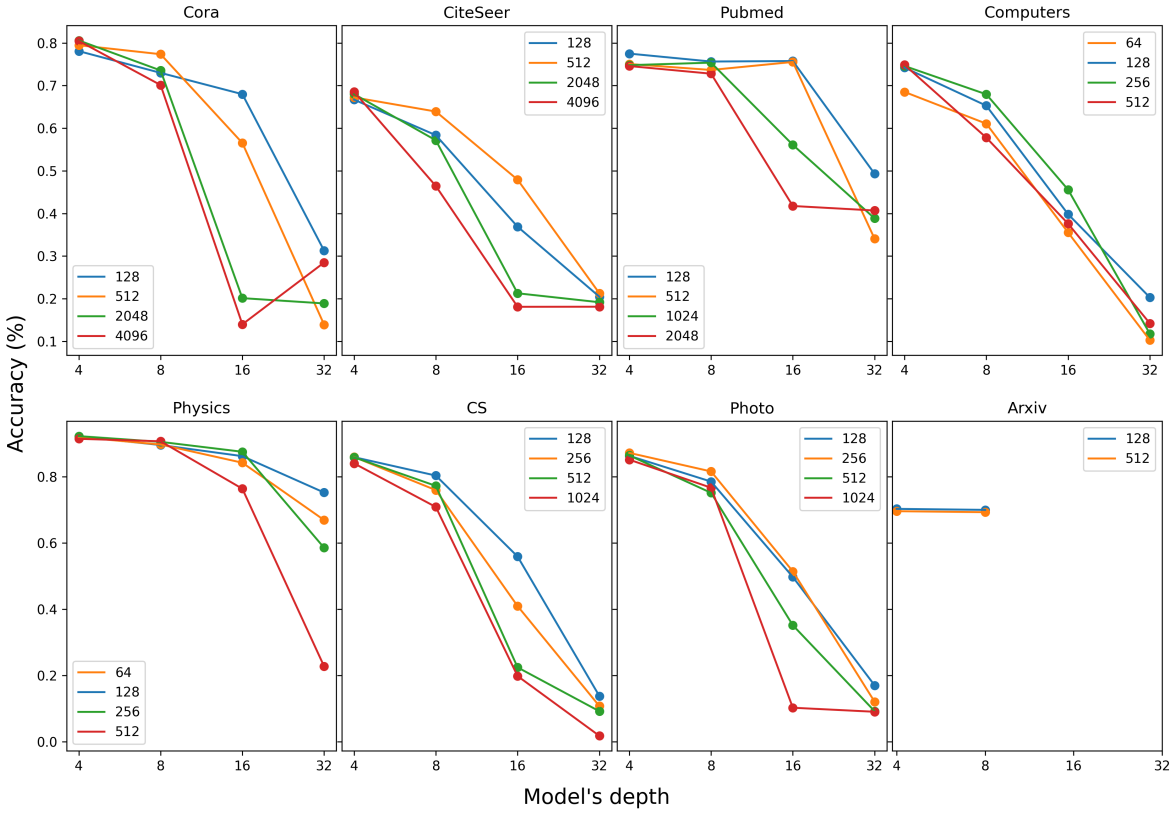

Figure 5.5: Comparison between different configurations (in terms of width) of fully trained GATs across 8 datasets for varying depth. Arxiv experiments were limited to 8 layers due to hardware constraints.

experiments. According to Equation 5.2, the trainable layer can be placed at any position within the network. Our theoretical analysis has primarily focused on the scenario where the final layer is trainable.

Table 5.2 shows the impact of varying the position of the trainable layer across different combinations of widths and depths. We observe that, in general, the optimal position for the trainable layer is the second. Placing the trainable layer at a relatively early stage of the model, facilitates training on the basis of a very local aggregation of information, i.e. 2-hop. Subsequent convolutional layers then distribute the learned embeddings, while also preventing over-correlation, as explained theoretically. The performance gap between different positions of the trainable layer diminishes with increasing width, showing that a large width makes the model more robust to the placement of the trainable layer. Additionally, the optimal position is different for different datasets. Table 5.2 presents the results for three datasets chosen to highlight this observation. For the rest of the datasets we have observed similar behavior and the respective results appear in the Appendix 13.

MLP vs GCN for trainable layer:

In this experiment, we aimed to answer the question of whether the trainable layer has to be a graph convolutional layer, or if it could be replaced by a Multi-Layer Perceptron (MLP). The underlying hypothesis was that if sufficient information has already been effectively propagated through the network by the untrained graph convolution layers, then an MLP could potentially generate informative node representations.

Table 5.3 presents the performance achieved by models employing either a single GCN or an MLP as their trainable layers. We focus on a rather challenging configuration of a 16-

Table 5.1: Comparison of different model widths of partially trained GCNs and GATs with fully trained ones, in the "cold start" scenario. Only the features of the nodes in the training set are available to the model. We present the best accuracy of the model and the depth (i.e. # Layers) this accuracy is achieved.

Dataset	Width	GCN		GAT	
		Accuracy (%) & std	#L	Accuracy (%) & std	#L
Cora	128	63.31 ± 2.5	6	58.41 ± 2.3	4
	512	70.60 ± 1.5	15	66.66 ± 1.8	22
	8192	**73.24 ± 0.6**	17	**72.09 ± 1.6**	28
	Trained	69.35 ± 0.9	6	66.23 ± 1.3	5
CiteSeer	128	42.09 ± 0.7	4	40.75 ± 2.0	5
	512	48.17 ± 1.2	14	45.26 ± 2.6	19
	8192	**51.32 ± 0.6**	31	**50.29 ± 0.9**	31
	Trained	46.62 ± 0.7	7	44.38 ± 0.8	5
Pubmed	128	64.68 ± 3.2	7	63.47 ± 3.5	5
	512	69.80 ± 0.8	7	67.22 ± 2.7	28
	8192	**72.73 ± 0.5**	16	**69.83 ± 0.8**	7
	Trained	69.48 ± 0.6	7	66.50 ± 1.3	5
Physics	128	38.61 ± 0.0	2	54.99 ± 7.5	3
	512	75.82 ± 0.6	4	72.11 ± 1.5	4
	4096	**88.11 ± 0.5**	14	**75.29 ± 5.7**	16
	Trained	82.50 ± 0.1	5	74.11 ± 2.6	4
CS	128	40.52 ± 0.2	2	43.42 ± 1.4	3
	512	63.94 ± 0.8	4	56.12 ± 0.7	4
	8192	**77.49 ± 0.4**	14	**72.43 ± 3.0**	17
	Trained	66.11 ± 1.2	6	61.32 ± 3.1	7
Photo	128	73.87 ± 0.5	2	67.14 ± 2.0	2
	512	76.72 ± 0.0	2	68.85 ± 2.5	2
	8192	**86.11 ± 0.2**	5	**75.90 ± 3.5**	12
	Trained	82.53 ± 1.2	5	74.08 ± 1.1	3
Computers	128	61.07 ± 0.7	2	56.46 ± 2.6	2
	512	67.07 ± 0.0	2	57.72 ± 2.1	2
	4096	**76.58 ± 0.4**	5	**63.92 ± 4.3**	13
	Trained	74.03 ± 0.8	6	57.55 ± 3.3	3

layer model, which has access to distant information but needs to avoid oversmoothing. Results indicate that GCN slightly outperforms MLP in terms of accuracy and stability, as evidenced by achieving better scores with smaller standard deviations. However, the performance improvement is marginal, suggesting that both approaches can generate informative node representations, provided the necessary information has been appropriately received and not excessively mixed by graph convolution.

Varying width within the network:
In our final experiment, we investigate the impact of changing the width of the untrained layers. Table 5.4 shows the performance of a 16-layer partially trained GCN with varying width across all of its untrained layers. Each row corresponds to a model with a different

Table 5.2: Performance comparison of GCN models with different width and depth, as well as different placement of the trainable layer.

Accuracy (%) & std						
Dataset	(Width, Depth)	Position				
		2	4	8	16	32
CiteSeer	(512, 8)	**67.58** ± 1.25	62.13 ± 1.18	53.47 ± 1.85	-	-
	(2048, 8)	**71.20** ± 0.49	67.20 ± 1.18	61.14 ± 1.11	-	-
	(8192, 8)	**70.78** ± 1.45	69.52 ± 1.08	66.39 ± 0.74	-	-
	(512, 32)	34.35 ± 12.25	32.40 ± 9.81	29.01 ± 9.86	27.72 ± 11.78	**30.12** ± 10.94
	(2048, 32)	57.36 ± 6.46	55.76 ± 5.72	57.22 ± 2.87	56.72 ± 3.84	**58.69** ± 3.80
	(8192, 32)	**70.18** ± 0.64	68.21 ± 0.87	65.45 ± 1.06	65.45 ± 0.99	65.99 ± 1.09
Photo	(512, 8)	**84.53** ± 1.32	83.54 ± 0.74	66.85 ± 5.44	-	-
	(2048, 8)	**89.68** ± 0.12	89.41 ± 0.18	82.59 ± 0.36	-	-
	(8192, 8)	**90.26** ± 0.21	89.82 ± 0.22	87.73 ± 0.18	-	-
	(512, 32)	22.69 ± 15.00	22.15 ± 11.70	24.42 ± 15.12	**25.68** ± 15.29	21.61 ± 12.61
	(2048, 32)	41.12 ± 17.04	44.96 ± 15.92	50.78 ± 13.71	**51.76** ± 13.70	50.19 ± 13.14
	(8192, 32)	**72.32** ± 2.93	71.39 ± 2.04	70.58 ± 1.27	70.66 ± 1.95	71.62 ± 1.80
Computers	(512, 8)	**64.74** ± 2.14	63.81 ± 1.57	59.70 ± 1.03	-	-
	(2048, 8)	**73.10** ± 0.26	72.22 ± 0.22	64.77 ± 0.71	-	-
	(4096, 8)	**74.80** ± 0.00	73.15 ± 0.00	66.52 ± 0.00	-	-
	(512, 32)	8.43 ± 5.18	8.83 ± 5.24	**12.71** ± 11.89	10.82 ± 5.48	8.33 ± 5.21
	(2048, 32)	45.74 ± 19.44	48.15 ± 19.88	49.38 ± 19.14	50.85 ± 17.11	**50.97** ± 17.32
	(4096, 32)	65.00 ± 0.00	**67.62** ± 0.00	65.17 ± 0.00	66.28 ± 0.00	64.84 ± 0.00

Table 5.3: Performance comparison between an MLP and a GCN, used as trainable layers, in a 16-layer GCN model, for *Cora, Citeseer* and *Pubmed* datasets.

Accuracy (%) & std		
Model / Dataset	MLP	GCN
Cora	80.83 ± 0.8	**81.09** ± 0.4
CiteSeer	70.31 ± 1.2	**71.15** ± 1.1
Pubmed	78.32 ± 0.8	**78.55** ± 0.3

configuration. In our experiments, wide layers are placed lower in the model, but changing their position does not yield improvements.
Results indicate that models having the majority of their layers wide, perform better. This observation is aligned with our theoretical analysis, in which we have assumed wide square weight matrices (i.e., maintaining the same width throughout the network).

Table 5.4: Performance comparison on the *Cora* dataset, when we vary the width of a 16-layer GCN model, in which only the final layer is trainable. For each configuration each width column indicates the number of the untrained layers which have that particular width. Wider layers are placed at the lowest positions.

Width			Accuracy (%) & std
2048	4096	8192	
12	0	3	76.54 ± 1.0
8	0	7	76.82 ± 1.0
4	0	11	77.50 ± 1.0
0	0	15	**79.45 ± 1.0**
0	12	3	78.37 ± 0.7
0	8	7	78.62 ± 1.0
0	4	11	79.06 ± 0.8

Memory and Convergence:
Table 5.5 reports peak GPU memory and convergence metrics for GCN and GCNII on the *Arxiv* dataset, for varying depth and width [1]. As expected, wider and deeper networks demand more memory, but this cost depends solely on the architecture size. Partially-trained models consistently converge faster than the fully trained ones (at 50% - 75% of the epochs), while achieving higher accuracy as shown in subsection 4.2. Per-epoch runtime increases with width, due to larger number of parameters, but partially-trained models require fewer epochs and achieve superior performance, reducing oversmoothing in deeper architectures.

Table 5.5: Peak memory and convergence metrics. Each entry shows memory (GB) and time per epoch×epochs. Width is annotated with (Full) or (Partial), depending on the type of training adopted.

Depth	Width	Model	Memory	Time×Epochs
4	128 (Full)	GCN	1.6	0.1×8000
	1024 (Partial)	GCN	6.0	0.4×4500
	256 (Full)	GCNII	9.1	0.4×800
	512 (Partial)	GCNII	17.6	0.8×400
8	128 (Full)	GCN	2.5	0.2×7000
	1024 (Partial)	GCN	11.2	1.0×4000
	256 (Full)	GCNII	14.0	0.8×800
	512 (Partial)	GCNII	22.5	1.6×600
16	128 (Full)	GCN	4.5	0.3×10000
	1024 (Partial)	GCN	23.0	2.4×8000
	256 (Full)	GCNII	19.0	1.7×1000
	512 (Partial)	GCNII	32.2	3.3×800

[1] GAT was excluded from this study, due to its memory-handling problems.

4 Conclusion

In this chapter we investigated the use of partially trained GCNs for node classification tasks. We have shown theoretically that training only a single layer in wide GCNs and GATs is sufficient to achieve comparable or superior performance to fully trained models. Additionally, we linked our proposed method to existing literature on oversmoothing and demonstrated that partially trained models can perform well even as their depth increases. We emphasized the dual role of width in our analysis, both in terms of Central Limit Theorem convergence and for reducing the correlation of node embeddings, due to repeated weight averaging. We have assessed our theoretical results through a variety of experiments that confirmed the potential of partially trained GCNs and GATs. Finally, we applied our method to the state-of-the-art GCNII architecture. In these experiments we observe that the proposed method yields consistently at least as high an accuracy as the fully trained GCNII, even though GCNII's architecture already provides strong resistance to oversmoothing. Importantly, our integration required no modification to GCNII's core design, underscoring that partial training can complement diverse GNN models with minimal effort.

Chapter 6

Reducing Oversmoothing through Informed Weight Initialization in Graph Neural Networks

This chapter addresses the problem of oversmoothing in deep Graph Neural Networks through the lens of weight initialization. Most existing GNNs adopt initialization methods originally developed for fully connected or convolutional neural networks, such as Kaiming [15] or Xavier [122] initialization. These methods, however, neglect the structural properties of graph data and the unique signal propagation behavior inherent in GNN architectures.
Several methods have been proposed to alleviate oversmoothing and enable deep GNNs [103, 13], but none of them considers the effect of weight initialization. The aim of this chapter is to propose a weight initialization that is suitable for GNNs and investigate the effect weight initialization can have to oversmoothing. In particular, we generalize the analysis of [15] to GNNs and present results about the variance of signals and gradients flowing inside the network. We analyze the case of a general GNN model, which may combine graph convolution, residual connections and skip connections, aiming to cover the majority of existing graph convolutional models. Leveraging these theoretical results, we then focus on the simpler case of a GCN and we propose a novel weight initialization method (G-Init), that stabilizes variance and reduces oversmoothing. We attribute this effect to the initial largest singular values of the weight matrices, which may influence the largest singular values after convergence. These, in turn, are known to be related to oversmoothing [7]. Finally, experiments on a diverse range of datasets verify our theoretical results.
Hence, the main contributions of the work presented in this chapter are as follows:

• **Theoretical analysis of weight initialization for GNNs**: We generalize the analysis of [15] to convolutional GNNs. We investigate a parametric GNN encompassing variations with and without residual connections, skip connections, identity addition to the weight matrix and controlled effects of graph convolution and weight matrix. We derive the respective formulas about the variance of the forward signals and the backward gradients within the model.

• **A new weight initialization method (G-Init)**: We propose a new way to initialize GNN weight matrices, focusing on the initialization of GCN. Our method aims to stabilize the variance and capitalize on its relationship with oversmoothing reduction.

• **Experimental confirmation of the benefits of G-Init**: We conduct experiments utilizing deep GCNs and GATs with up to 64 layers across 8 datasets for node classification tasks and show that the proposed initialization reduces oversmoothing. Additionally, we demonstrate

the benefits of G-Init, in the presence of the "cold start" situation, where node features are available only for the labeled nodes. Additionally, we extend our experiments to graph classification tasks, in which G-Init outperforms standard initialization methods.

1 Notations

Extending the GCN formula to a parametric and more generic GNN we arrive at:

$$H^{(l+1)} = \sigma\big(\big(\alpha\hat{A}H^{(l)} + \beta H^{(0)} + \gamma H^{(l-1)}\big) \times \big(\delta W^{(l)} + \epsilon I\big)\big), \tag{6.1}$$

where $\alpha, \beta, \gamma, \delta$ and ϵ are preselected parameters that determine the convolutional GNN architecture (e.g., if $\alpha = \delta = 1, \beta = \gamma = \epsilon = 0$ we have a GCN, while if $\alpha = 0.9, \beta = 0.1, \gamma = 0, \delta = \lambda/l, \epsilon = 1-\delta$ we arrive at GCNII [11]). We observe that, α and δ control the importance of graph convolution and the effect of the weight matrix respectively. Furthermore, β, γ and ϵ dictate the existence or absence of residual connections, skip connections and identity addition respectively.
GNNs are also commonly used for graph classification tasks. Such tasks involve multiple graphs of arbitrary sizes, each associated with a label that needs to be predicted. To achieve this, GNNs are slightly extended with the addition of a $READOUT$ function. That function takes as input the final node embeddings and aggregates them to produce a graph representation. It is common practice to use average pooling as a $READOUT$ function. We will follow the same approach in our experiments throughout this work.

1.1 Weight Initialization

Before training, all entries of the weight matrices are sampled from a probability distribution. Selecting the appropriate distribution is very important. The most commonly used initialization methods (i.e., [122] and [15]) focus on the stabilization of the variance across layers. They aim to stabilize both the variance of the signals flowing forward and of the gradients which flow backwards. The aforementioned methods either use uniform or zero-mean Gaussian distributions. When Gaussian distributions are used, their variance plays a crucial role. In particular, [122] proposed to initialize the weights, using a zero-mean Gaussian with variance equal to $1/n_l$, where n_l is the dimensionality of the l-th layer. Moving one step further [15] proposed an initialization with a zero-mean Gaussian with variance equal to $2/n_l$, taking into account the characteristics of the ReLU activation function.
In this work, we establish a connection between the weight initialization and the initial singular values of the weight matrices. These values have the potential to influence the final singular values and, consequently, oversmoothing. For this, we will use the circular law conjecture, which was proven with strong convergence by [139].

Theorem 14 (Circular Law Conjecture [139]). *Let Z_n be a random matrix of order n, whose entries are i.i.d. samples of a random variable of zero-mean and bounded variance σ_{std}^2. Also let $\lambda_1, ..., \lambda_n$ be the eigenvalues of $\frac{1}{\sigma_{std}\sqrt{n}} Z_n$. The circular law states that the distribution of λ_i converges to a uniform distribution over the unit disk as n tends to infinity.*

The circular law conjecture dictates the relationship between the standard deviation (σ_{std}) of the random variable and the radius of the disk. In fact, if we increase σ_{std} there is a proportional increase of the disk radius, which in turn increases the largest eigenvalue of Z_n.

2 Theoretical Analysis

Despite their success in CNNs and FFNs, existing weight initialization methods fail to capture the effect of the graph topology, which is of high importance in GNNs. Therefore, we generalize the method developed in [15] to provide new formulas about the variance flowing within the network, which takes into account the underlying graph topology. Subsequently, we simplify the formulas specifically for the case of a GCN and introduce a novel weight initialization method (G-Init) tailored to this particular model. Building on the theoretical foundation established by [15], we extend their analysis to the graph domain, by incorporating the adjacency matrix, a fundamental component of the aggregation scheme in GNNs. Additionally, we examine the forward propagation of signals and the backward flow of gradients within the model. The key distinction of the proposed approach, and the essence of our generalization, lies in the explicit role of the adjacency matrix in our formulation. Notably, when this matrix is omitted, our analysis naturally reduces to that of [15], reinforcing the validity and continuity of our approach as an extension of their work.

2.1 Forward Propagation

In order to simplify the notation, we will use the augmented normalized adjacency matrix (i.e., $\hat{A} = \hat{D}^{-1}(A + I)$) instead of the symmetrically normalized augmented adjacency matrix in Equation 6.1. This simplification results in an equivalent analysis, differing only in the use of the factor $\frac{1}{d_i}$ instead of $\frac{1}{\sqrt{d_i d_j}}$, in the formula of node representations. Hence, the representation of a node i in layer l becomes:

$$y_l^{(i)} = \left(\frac{\alpha}{d_i} \sum_{j \in \hat{N}(i)} x_l^{(j)T} + \beta \cdot x_{(0)}^{(i)} + \gamma \cdot x_{(l-1)}^{(i)} \right) \cdot (\delta W_l + \epsilon I) + b_l, \tag{6.2}$$

where $N(i)$ is the neighborhood of node i and $\hat{N}(i)$ is the augmented neighborhood, after self-loop addition, and b_l is the bias. Here $x_l^{(j)}$ is an $n_l \times 1$ vector that contains the representation of node j at layer l and $W^{(l)}$ is an $n_l \times n_l$ matrix, where n_l is the dimensionality of the layer l. Finally, $x_l^{(j)} = \sigma\left(y_{l-1}^{(j)}\right)$, according to Equation 6.1. Setting $x_l^{(i)'} = \frac{\alpha}{d_i} \sum_{j \in \hat{N}(i)} x_l^{(j)T} + \beta \cdot x_{(0)}^{(i)} + \gamma \cdot x_{(l-1)}^{(i)}$ and $W_l' = \delta W_l + \epsilon I$ aligns Equation 6.2 above with Equation 5 in [15].

We let the initial elements of $W^{(l)}$ be drawn independently from the same distribution. Aligned with [15], we assume that the elements of $x_l^{(j)}$ are also mutually independent and drawn from the same distribution. Finally, we assume that $x_l^{(j)}$ and $W^{(l)}$ are independent of each other. Hence, we get:

$$Var[y_l^{(i)}] = n_l Var[\delta w_l x_l^{(i)'} + \epsilon x_l^{(i)'}], \tag{6.3}$$

where $y_l^{(i)}, x_l^{(i)'}$ and w_l represent the random variables of each element in the respective matrices. Utilizing the fact that the variance of the sum of two variables is equal to the sum of their individual variances, plus twice the covariance between them, we let w_l have zero-mean, leading the variance to be:

$$Var[y_l^{(i)}] = n_l \left(Var\left[\delta w_l x_l^{(i)'}\right] + Var\left[\epsilon x_l^{(i)'}\right] + 2Cov(\delta w_l x_l^{(i)'}, \epsilon x_l^{(i)'}) \right) =$$

$$n_l\left(\delta^2 Var[w_l] E\left[\left(x_l^{(i)'}\right)^2\right] + \epsilon^2 Var\left[x_l^{(i)'}\right]\right) \implies$$

$$Var[y_l^{(i)}] \leq n_l \cdot E\left[\left(x_l^{(i)'}\right)^2\right]\left(\delta^2 Var[w_l] + \epsilon^2\right). \tag{6.4}$$

In Equation 6.4, $x_l^{(i)'}$ aggregates information from the neighborhood of each node, its previous representation and its initial representation. These components are subsequently combined to generate the new node representation. Considering that $x_l^{(i)} = \sigma\left(y_{l-1}^{(i)}\right)$, it is necessary to decompose $x_l^{(i)'}$ into three components: the first containing $x_l^{(i)}$, the second containing $x_{l-1}^{(i)}$, and the third containing the remaining information of $x_l^{(i)'}$. In order to achieve that, we will employ the Cauchy–Bunyakovsky–Schwarz inequality (CBS), because it allows to transform a squared sum of elements into a sum of squares of these elements.

Lemma 15.

$$E\left[\left(x_l^{(i)'}\right)^2\right] = E\left[\left(\frac{\alpha}{d_i}\sum_{j\in\hat{N}(i)} x_l^{(j)T} + \beta\cdot x_{(0)}^{(i)} + \gamma\cdot x_{(l-1)}^{(i)}\right)^2\right] \overset{CSB}{\leq}$$

$$(d_i + \mathbb{1}(\beta\neq 0) + \mathbb{1}(\gamma\neq 0)) \times \left(\frac{\alpha^2}{d_i^2} E\left[\left(x_l^{(i)}\right)^2\right] + \gamma^2\cdot E\left[\left(x_{l-1}^{(i)}\right)^2\right] + j(\alpha,\beta)\right),$$

where $j(\alpha,\beta) = \frac{\alpha^2\cdot k_l^{(i)}}{d_i^2} + \beta^2\cdot E\left[(\,x_0^{(i)}\,)^2\right]$, $k_l^{(i)} = \sum_{j\in N(i)}\left(x_l^{(j)T}\right)^2$, *i.e., the sum of neighbor representations excluding the self representation and* $\mathbb{1}(\cdot)$ *is the indicator function.*

If we let w_{l-1} have a symmetric distribution around zero and $b_{l-1} = 0$ then y_{l-1} has zero-mean and symmetric distribution around its mean. This leads to $E\left[\left(x_l^{(i)}\right)^2\right] = \frac{1}{2}Var[y_{l-1}]$, when the activation function is ReLU. Applying that result and Lemma 15 to Inequality 6.4 leads to the first main theorem of our work.

Theorem 16. *The upper bound of the variance of the signals flowing forward in a generic GNN defined by Equation 6.1 is:*

$$Var[y_l^{(i)}] \leq n_l\cdot(d_i + \mathbb{1}(\beta\neq 0) + \mathbb{1}(\gamma\neq 0)) \times$$

$$\left(\frac{\alpha^2}{2d_i^2} Var[y_{l-1}^{(i)}] + \frac{\gamma^2}{2}\cdot Var[y_{l-2}^{(i)}] + j(\alpha,\beta)\right) \times \left(\delta^2 Var[w_l] + \epsilon^2\right), \tag{6.5}$$

where n_l *is the weight matrix dimension,* d_i *is the degree of node* i, $\mathbb{1}(\cdot)$ *is the indicator function,* $\alpha, \beta, \gamma, \delta$ *and* ϵ *are parameters, depending on the underlying architecture of the model, and* $j(\alpha,\beta)$ *is defined in Lemma 15.*

The extended proof of Theorem 16, along with a lower bound of $Var[y_l^{(i)}]$, is available in Appendix 14.

2.2 Backward Propagation

For back-propagation, the gradient for node i is computed by:

$$\Delta x_l^{(i)} = W_l' \Delta y_l^{(i)}. \tag{6.6}$$

We follow the same notation as in [15], where $\Delta x_l^{(i)}$ and $\Delta y_l^{(i)}$ denote gradients $\left(\frac{\partial E}{\partial x_l^{(i)}}, \frac{\partial E}{\partial y_l^{(i)}}\right.$ and E is the los and have dimensionality of $n_l \times 1$. Equation 6.6 represents the result of applying the derivative to the loss function of the model's predictions. Considering the common practice in GNNs to maintain a constant hidden dimension across layers, we proceed our analysis with W_l being an $n_l \times n_l$ matrix. We note that, in the general case proposed in [15], W_l' might be substituted by a matrix $\hat{W}'_l$, with $\hat{n}_l \times \hat{n}_l$ dimensions, which can be formed by W_l' through reshaping. That modification does not affect our analysis, except in the appearance of the factor $\hat{n}_l$ in the results instead of n_l. The reshaped matrix, $\hat{W}'_l$, retains the structural properties of W_l', allowing us to use the latter for our analysis. More details about the use of $\hat{W}'_l$ can be found in [15].

We also set $\Delta x_l^{(i)'} = \frac{\alpha}{d_i} \sum_{j \in \hat{N}(i)} \left(\Delta x_l^{(j)T}\right) + \gamma \cdot \Delta x_{(l-1)}^{(i)T}$, the average gradient reaching node i derived from the forward pass and the interaction with its neighbors (message passing), plus the gradient of its representation in the previous layer. The presence of residual connections does not impact the gradient, as it adds a constant value to the node representation (i.e., $x_0^{(i)}$), which has a derivative equal to zero. This ensures that the gradient for each node is computed by aggregating the gradients of neighboring nodes, adjusted by the degree of the node, along with the gradient of its own representation from the previous layer, preserving the effect of residual connections.

If we assume that w_l (a random variable representing the weight matrix elements) and $\Delta y_l^{(i)}$ are independent of each other and w_l is initialized with a symmetric distribution around zero, then $\Delta x_l^{(i)}$ has zero mean for all l. Therefore, $\Delta x_l^{(i)'}$ also has a zero mean, as it results from the summation of zero-mean variables.

In back-propagation we have $\Delta y_l^{(i)} = \sigma'(y_l^{(i)}) \Delta x_{l+1}^{(i)'}$, where $\sigma'(\cdot)$ is the derivative of the activation function (i.e., ReLU). The derivative of the activation function, $\sigma'(\cdot)$, determines how the error is scaled at each node. Since for ReLU, $\sigma'(\cdot)$ is either one or zero, with equal probabilities, the back-propagated error either passes through (when the neuron is active) or gets blocked (when the neuron is inactive). Additionally, we assume independence between $\sigma'(y_l^{(i)})$ and $\Delta x_{l+1}^{(i)'}$. Hence, we arrive at the conclusion that $E\left[\Delta y_l^{(i)}\right] = E[\sigma'(y_l^{(i)})] \cdot E\left[\Delta x_{l+1}^{(i)'}\right] = E\left[\Delta x_{l+1}^{(i)'}\right]/2 = 0$, due to the two branches of the ReLU derivative, the independence between $\sigma'(y_l^{(i)})$ and $\Delta x_{l+1}^{(i)'}$, and the zero-mean of $\Delta x_l^{(i)'}$. Consequently we get:

$$\begin{aligned} E\left[\left(\Delta y_l^{(i)}\right)^2\right] = E\left[\left(\Delta y_l^{(i)} - 0\right)^2\right] = Var\left[\Delta y_l^{(i)}\right] = Var\left[\sigma'(y_l^{(i)})\right] \cdot Var\left[\Delta x_{l+1}^{(i)'}\right] + \\ E\left[\sigma'(y_l^{(i)})\right]^2 \cdot Var\left[\Delta x_{l+1}^{(i)'}\right] + Var\left[\sigma'(y_l^{(i)})\right] \cdot E\left[\Delta x_{l+1}^{(i)'}\right]^2 = \\ \frac{1}{4} Var\left[\Delta x_{l+1}^{(i)'}\right] + \frac{1}{4} Var\left[\Delta x_{l+1}^{(i)'}\right] = \frac{1}{2} Var\left[\Delta x_{l+1}^{(i)'}\right]. \end{aligned}$$

Finally, we compute the variance of the gradient in Equation 6.6 as follows:

$$Var\left[\Delta x_l\right] = n_l Var[w_l'] Var\left[\Delta y_l^{(i)}\right] = \frac{1}{2} n_l \left(\delta^2 Var[w_l] + \epsilon^2\right) Var\left[\Delta x_{l+1}^{(i)'}\right]. \tag{6.7}$$

Following a similar approach as in the forward pass and in Lemma 15 we get:

Lemma 17.

$$E\left[\left(\Delta x_{l+1}^{(i)'}\right)^2\right] = E\left[\left(\frac{\alpha}{d_i}\sum_{j\in\hat{N}(i)}\Delta x_{l+1}^{(j)T} + \gamma\cdot\Delta x_{(l)}^{(i)T}\right)^2\right]$$

$$\overset{CSB}{\leq} (d_i + \mathbb{1}(\gamma\neq 0))\cdot\left(\frac{\alpha^2}{d_i^2}E\left[\left(\Delta x_{l+1}^{(i)}\right)^2\right] + \gamma^2 E\left[\left(\Delta x_{l}^{(i)}\right)^2\right] + q(\alpha)\right),$$

where $q(\alpha) = \frac{\alpha^2}{d_i^2}\cdot o_{l+1}^{(i)}$, *and* $o_{l+1}^{(i)} = \sum_{j\in N(i)}\left(\Delta x_{l+1}^{(j)T}\right)^2$, *i.e., the sum of gradients originating from the neighbors of the node, excluding self-originating gradient and* $\mathbb{1}(\cdot)$ *is the indicator function.*

Taking into consideration that $Var\left[\Delta x_{l+1}^{(i)'}\right] = E\left[\left(\Delta x_{l+1}^{(i)'}\right)^2\right]$ and applying Lemma 17 to Equation 6.7 gives rise to the second main theorem of this work.

Theorem 18. *The upper bound of the variance of the gradients flowing backward in a generic GNN defined by Equation 6.1 is:*

$$Var\left[\Delta x_l^{(i)}\right] \leq \frac{m_w}{1-\gamma^2 m_w}\cdot\left(\frac{\alpha^2}{d_i^2}Var\left[\Delta x_{l+1}^{(i)}\right] + q(\alpha)\right), \tag{6.8}$$

with

$$m_w = \frac{1}{2}n_l\left(d_i + \mathbb{1}(\gamma\neq 0)\right)\cdot\left(\delta^2 Var[w_l] + \epsilon^2\right),$$

where n_l *is the weight matrix dimension,* d_i *is the degree of node* i, $\mathbb{1}(\cdot)$ *is the indicator function* α, γ, δ *and* ϵ *are parameters, depending on the underlying architecture of the model, and* $q(\alpha)$ *is defined in Lemma 17.*

The extended proof of Theorem 18, along with a lower bound of $Var[\Delta x_l^{(i)}]$, are provided in Appendix 15.

2.3 G-Init

Having analyzed the generic case of convolutional GNNs, we focus on the simpler yet prominent GCN model. The GCN is defined by Equation 2.1, a special case of Equation 6.1 with $\alpha = \delta = 1, \beta = \gamma = \epsilon = 0$. Leveraging Theorem 16 and Theorem 18, we aim to control the variance of the input signal at the final layer (L) of the model and the variance of the gradients flowing backward within it.
Regarding the variance at the final layer, we utilize Equation 6.5 and telescopically replace the factor $Var\left[y_{l-1}^{(i)}\right]$, until we arrive at $Var\left[y_1^{(i)}\right]$, which is the variance of the first layer of the model.

$$Var\left[y_L^{(i)}\right] \leq \frac{n_L}{d_i}Var[w_L]\left(\frac{1}{2}Var\left[y_{L-1}^{(i)}\right] + k_L^{(i)}\right) \implies$$

$$Var\left[y_L^{(i)}\right] \leq Var\left[y_1^{(i)}\right] \left(\prod_{l=2}^{L} \frac{n_l}{2d_i} Var[w_l]\right) + \sum_{l=2}^{L} \left(\prod_{j=l+1}^{L} \frac{n_j}{2d_i} Var[w_j]\right) \frac{n_l}{d_i} k_l^{(i)} Var[w_l]. \tag{6.9}$$

A proper initialization method should avoid reducing or magnifying the magnitudes of input signals exponentially. We aim to control the upper bound of the variance of the final layer of the model, which in turn requires tuning the products of Equation 6.9, in order to obtain a proper scalar, e.g. 1. A sufficient condition to achieve this is the following:

$$\frac{n_l}{2d_i} Var[w_l] = 1, \qquad \forall l. \tag{6.10}$$

This leads to a zero-mean Gaussian distribution of the weights, with standard deviation (std) equal to $\sqrt{2d_i/n_l}$.
Similar results can be obtained for the weight initialization, based on the gradients flowing backward in the network, if we use the respective equations.
Using the initialization generated by either of the two directions (forward or backward) is sufficient, as both methods prevent the exponential reduction or increase of magnitudes in both the input signals (flowing forward) and the gradients (flowing backward). Hence, for the new initialization method (G-Init) we choose a zero-mean Gaussian, whose standard deviation is $\sqrt{2d_i/n_l}$.

3 Experiments

We conducted extensive experiments to thoroughly evaluate the performance of G-Init. We tested node classification on 8 benchmark datasets and explored "cold start" variants on them. Additionally, we performed graph classification on 10 datasets, showcasing the versatility of G-Init. To ensure robustness, we varied the model depth and compared G-Init's performance against several state-of-the-art initialization methods, consistently achieving superior results. This comprehensive evaluation highlights the effectiveness of G-Init, not only in "cold start" scenarios, but also in a wide range of graph learning applications.

3.1 Experimental Setup

Datasets: Aligned to most of the literature about node classification tasks, we focus on some well-known benchmarks in the GNN domain: *Cora, CiteSeer, Pubmed*, adopting the data splits of [25], where all the nodes except the ones used for training and validation are used for testing. Additionally, we use the *Arxiv* dataset [134] from the OGB suite. Further, we experiment with the *Photo, Computers, Physics* and *CS* datasets following the splitting method presented in [138]. Finally, we test the method on graph classification tasks, utilizing various datasets from two collections: TUDataset [140] and MoleculeNet [141]. To ensure a fair comparison between all methods, we use the same train, validation, and test splits that are publicly available. Dataset statistics can be found in Appendix 24.
Methods: The focus of our experiments is on the effect of different initialization methods. For this reason, we combine the basic GCN architecture [25], with different weight initialization methods. To further explore the capabilities of G-Init, we also experiment with the GAT model [41], which leverages an attention mechanism for node feature aggregation, i.e.,

the features of each node are derived by the weighted sum of the features of its neighbors. Specifically, we compare our method (G-Init) against Xavier initialization [122] and Kaiming initialization [15]. For each of the two competing methods, we explore two variants; namely drawing samples from a uniform distribution with predefined limits and drawing samples from a zero-mean Gaussian distribution of predefined standard deviation. We use the notation $Uniform$ and $Normal$ to denote these two variants. We also compare against a recently proposed initialization method for GNNs, i.e., VIRGO initialization [123]. Thus, our comparison includes all well-established initialization methods for GNNs, which serve as standard baselines in the literature.

Hyperparameters: We performed a hyperparameter sweep, details can be found in Appendix 18, to determine the optimal hyperparameter values, based on their performance on the validation set. For both models, the number of hidden units of each layer is set to 128 across all datasets, except for the *Arxiv* dataset, where it is set to 256. For GAT we use a single attention head. For both models L_2 regularization is applied with a penalty of $5{\cdot}10^{-4}$, and the learning rate is set to 10^{-3}.

Configuration: Each experiment is run 10 times and we report the average performance and standard deviations over these runs. We train all models for 200 epochs (1200 for *Arxiv* dataset) using Cross Entropy as the loss function. We evaluate each model based on its accuracy in both the node and the graph classification tasks.

3.2 Experimental Results

The choice of an appropriate value of d_i in G-Init is important. Choosing $d_i = 1$ would neglect the structure of the graph, while opting for a larger value could result in increased magnitudes of weight elements, leading to training instability. Consequently, we needed to identify a proper value for d_i that balances these two factors. The smallest possible degree for a node in a graph, with self loops included, is equal to 2 (indicating a single neighbor). Empirically, we have found that values of d_i in the range $(1, 2]$ achieve high performance. In our experiments, we set $d_i = 2$, except for the *Arxiv* dataset where $d_i = 1.6$ further improves the performance. These values are in accordance with the proposed theoretical analysis and improve the performance of the models. In summary, we propose initializing the weights of the models with a zero-mean Gaussian distribution, with a standard deviation (std) of $\sqrt{4/n_l}$ (in the *Arxiv* dataset, the std is equal to $\sqrt{3.2/n_l}$).

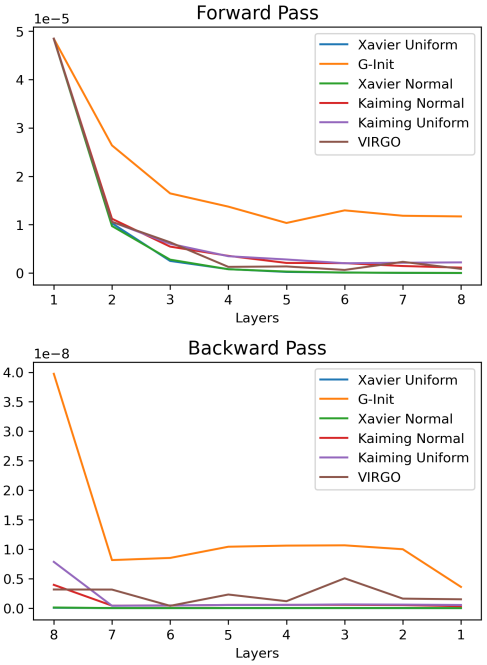

Figure 6.1: Variance plots on the *Cora* dataset.

Variance stability:

The first goal of our experimentation was to validate our theoretical results regarding the effect of G-Init on the variance within the model, before training. Specifically, we measured the average node variance as the signal flows forward in an 8-layer GCN and the average variance of the gradients flowing backward. Figure 6.1 presents the results for the *Cora* dataset. Both in the forward and in the backward pass, G-Init maintains a larger variance than the other methods.

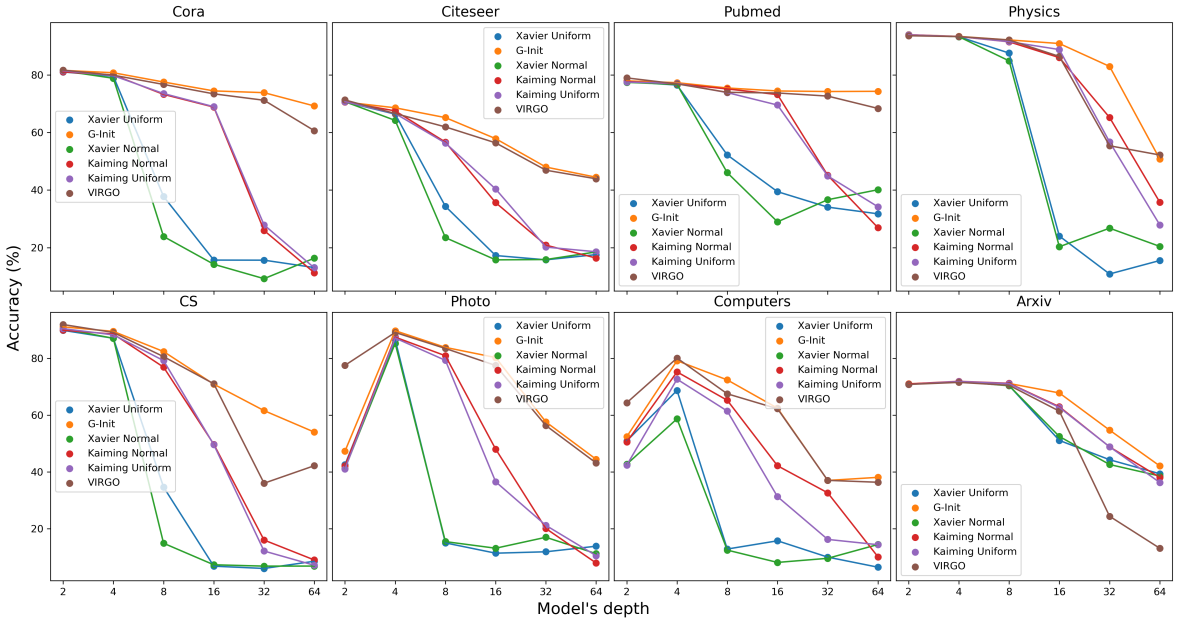

Figure 6.2: Comparison between 6 weight initialization methods across 8 datasets for varying GCN model depth.

Node classification and oversmoothing reduction:
Given this increased variance in the signal and the gradients, the next goal of our experiments was to assess the effect on classification performance. Figure 6.2 and Figure 6.3 illustrate the performance of a GCN and a GAT model, respectively, with varying depth on 8 different datasets. G-Init enables the model to achieve superior performance across all combinations of datasets and depths. In fact, a GNN model initialized with G-Init outperforms its counterparts initialized with any of the other methods. The experimental results verify the benefit of using a graph-informed weight intialization method, compared to the standard initialization methods that were devised for CNNs and FFNs.
GCN, GAT, and models employing graph convolution, inherently reduce the variance of node representations (information signal), due to the repetitive application of the Laplacian operator. G-Init enables the model to maintain a higher variance in the lower layers, preventing the collapse of node representations to a subspace, where they would become indistinguishable.
Our experimentation also confirms the relationship between weight initialization and oversmoothing. GCN and GAT cannot avoid oversmoothing using classical initialization methods, even at moderate depths. On the contrary, G-Init significantly reduces this effect, facilitating the use of deeper architectures. This assertion is validated not only by the model's accuracy but also by the resulting t-SNE [142] plots, which show that node representations have not been mixed to the extent of becoming indistinguishable. T-SNE plots of the GCN

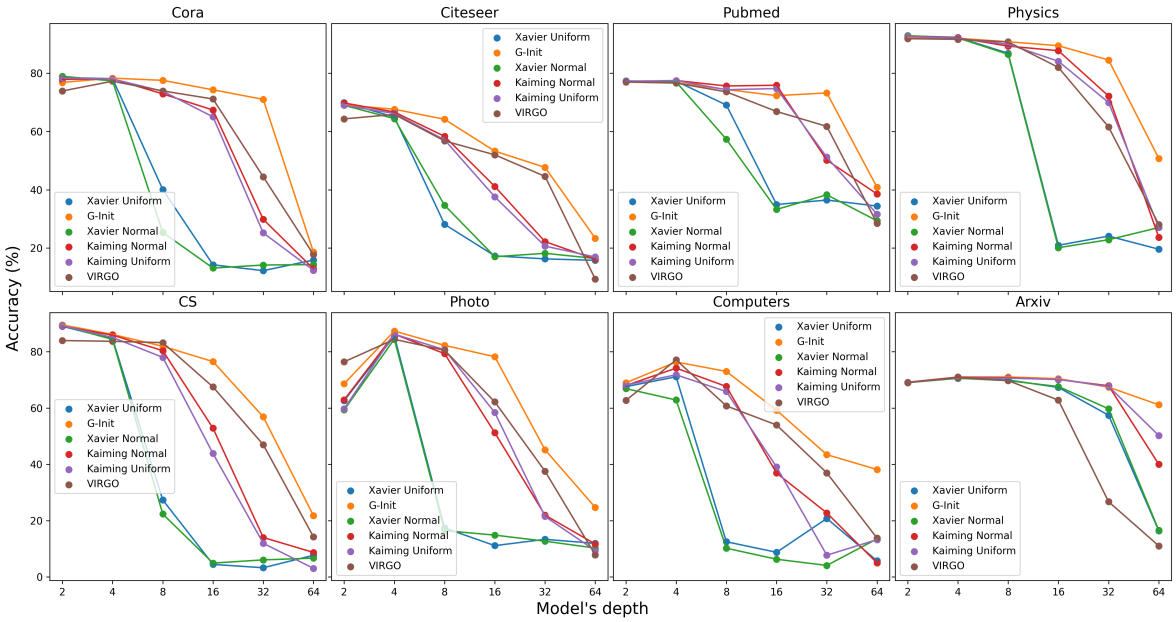

Figure 6.3: Comparison between 6 weight initialization methods across 8 datasets for varying GAT model depth.

model for the *Cora* dataset are presented in Figure 6.4, while for other datasets and initialization methods t-SNE plots are available in Appendix 17.

The oversmoothing reduction achieved by G-Init can be attributed to different initial singular values of the weight matrices, compared to the classical initialization methods. In particular, [15] propose an initialization using a zero-mean Gaussian with a standard deviation equal to $\sqrt{2}/\sqrt{n_l}$. Based on Theorem 14, we conclude that that approach creates weight matrices, whose eigenvalues lie on a disk with a radius equal to $\sqrt{2}$, while G-Init does the same on a disk of a greater radius (i.e., $\sqrt{4}$). Hence, G-Init initializes the model with weight matrices having larger maximum singular values than those produced by Kaiming initialization. These initial values may influence the largest singular values of the weight matrices after convergence, which, in turn, affect the extent to which oversmoothing can be reduced, based on Theorem 1.

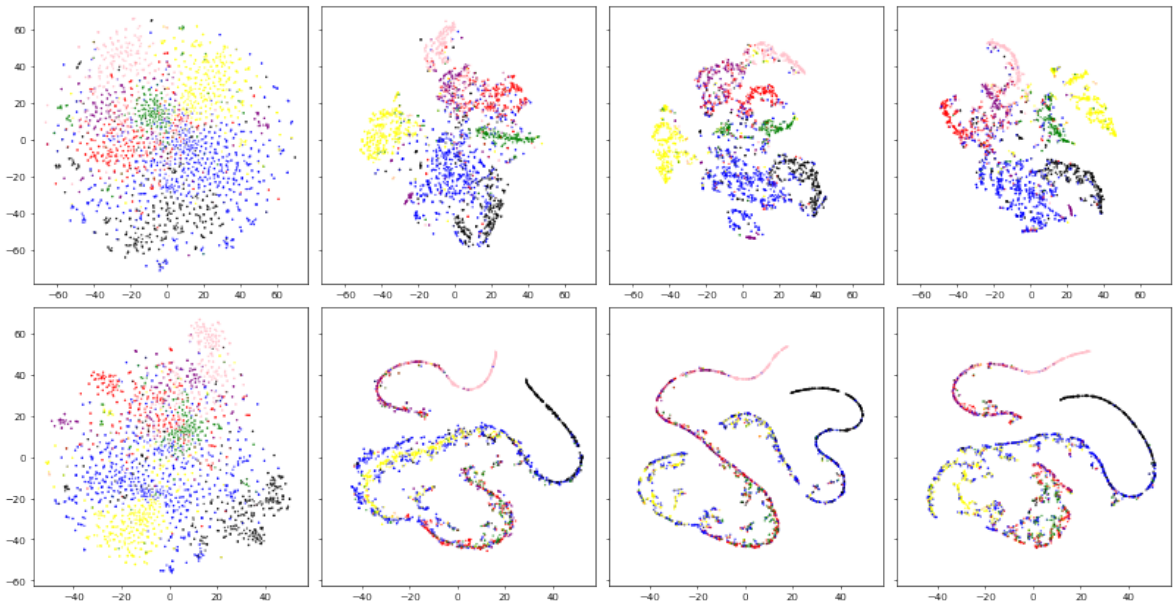

Figure 6.4: T-SNE plot of *Cora* dataset. The upper row presents results for a G-Init initialized 32-layer GCN, while the lower row showcases results for a Kaiming Normal initialized 32-layer GCN.

Performance under the "cold start" scenario:
In order to further explore the impact of G-Init on oversmoothing, we conducted a set of experiments aiming to highlight the value of using deep architectures that are not prone to oversmoothing. Specifically, we introduced a "cold start" situation in the datasets (details about "cold start" are available in Appendix 16), by replacing the feature vectors of the unlabeled nodes with all-zero vectors. For each dataset, Table 6.1 presents the best performance achieved and the depth at which the model attains that performance, for G-Init, VIRGO, and Kaiming Normal initialization, because these three are the top-performing methods. An extended version of Table 6.1 with results for every combination of initialization method and dataset is available in Appendix 16. We observe that, G-Init consistently outperforms the other methods across almost all datasets.

Table 6.1: Comparison of different initialization methods in the "cold start" scenario. Only the features of the nodes in the training set are available to the model. We present the best accuracy of the model and the depth (i.e. # Layers) this accuracy is achieved, for both GCN and GAT.

Dataset	Method	GCN		GAT	
		Accuracy (%) & std	#L	Accuracy (%) & std	#L
Cora	Kaiming Normal	68.35 ± 1.9	6	60.73 ± 4.5	4
	VIRGO	73.01 ± 1.0	26	70.68 ± 1.5	18
	G-Init	**74.04** ± 1.7	25	**72.34** ± 2.0	27
CiteSeer	Kaiming Normal	44.62 ± 1.8	6	37.09 ± 9.1	5
	VIRGO	49.18 ± 1.4	18	41.01 ± 9.5	13
	G-Init	**49.75** ± 0.7	27	**49.31** ± 0.4	30
Pubmed	Kaiming Normal	68.48 ± 1.5	6	60.25 ± 2.7	4
	VIRGO	71.55 ± 1.5	14	63.06 ± 9.8	14
	G-Init	**71.65** ± 1.8	23	**72.24** ± 1.1	32
Physics	Kaiming Normal	**94.00** ± 0.0	2	93.45 ± 0.0	1
	VIRGO	82.34 ± 6.1	8	85.58 ± 4.3	5
	G-Init	93.99 ± 0.0	1	**93.62** ± 0.1	2
CS	Kaiming Normal	90.13 ± 0.3	3	90.77 ± 0.1	1
	VIRGO	71.28 ± 1.9	6	76.85 ± 3.7	6
	G-Init	**90.28** ± 0.2	3	**90.82** ± 0.1	3
Photo	Kaiming Normal	86.53 ± 0.6	5	**86.35** ± 0.9	4
	VIRGO	83.00 ± 3.5	6	77.60 ± 4.0	6
	G-Init	**87.56** ± 1.2	4	86.07 ± 0.9	4
Computers	Kaiming Normal	75.18 ± 3.0	4	74.35 ± 3.5	4
	VIRGO	75.17 ± 2.7	6	69.91 ± 3.6	7
	G-Init	**78.03** ± 1.0	5	**76.28** ± 1.7	5

Graph classification:
We also conducted experiments with various initialization methods on the task of graph classification, utilizing a GCN and a GAT model with average pooling on top. As shown in Table 6.2 and Table 6.3, G-Init outperforms the other initialization methods in 8 out of 10 datasets, closely trailing as the second-best in the remaining two. These results verify that G-Init enhances the performance of both GCN and GAT also in graph classification tasks.

Table 6.2: Comparison between 6 weight initialization methods across 10 datasets for graph classification for the GCN model.

Methods	Dataset				
	BACE	BBBP	Clintox	Sider	Tox21
Xavier Normal	81.07 ± 1.9	69.27 ± 1.4	91.51 ± 1.1	59.75 ± 1.2	75.33 ± 0.6
Xavier Uniform	80.43 ± 2.5	69.68 ± 1.2	91.69 ± 1.1	59.77 ± 1.6	75.34 ± 0.8
Kaiming Normal	80.78 ± 1.9	69.66 ± 1.2	91.59 ± 1.6	59.50 ± 1.5	75.52 ± 0.6
Kaiming Uniform	79.36 ± 3.3	69.37 ± 1.0	91.15 ± 1.5	59.55 ± 1.8	75.47 ± 1.0
VIRGO	81.16 ± 2.1	69.93 ± 1.2	92.26 ± 1.3	58.74 ± 1.0	74.97 ± 0.8
G-Init	**82.04** ± 1.1	**70.03** ± 1.3	**93.05** ± 1.4	**60.01** ± 1.1	**75.81** ± 1.0

Methods	Dataset				
	Toxcast	MUTAG	ENZYMES	Proteins	$IMDB_{bin}$
Xavier Normal	64.18 ± 0.5	90.53 ± 3.2	52.41 ± 2.6	68.84 ± 1.6	72.11 ± 1.0
Xavier Uniform	64.20 ± 0.8	88.42 ± 3.2	51.25 ± 2.0	68.48 ± 1.3	72.67 ± 1.3
Kaiming Normal	64.45 ± 0.6	90.00 ± 1.8	52.41 ± 1.8	69.02 ± 1.5	72.33 ± 0.9
Kaiming Uniform	64.37 ± 0.3	89.47 ± 2.4	**53.44** ± 2.6	68.93 ± 1.6	72.00 ± 1.5
VIRGO	64.40 ± 0.6	90.53 ± 3.2	49.67 ± 3.6	**69.73** ± 1.7	72.00 ± 1.3
G-Init	**64.96** ± 0.5	**92.63** ± 3.1	52.71 ± 2.5	68.84 ± 1.2	**72.89** ± 0.7

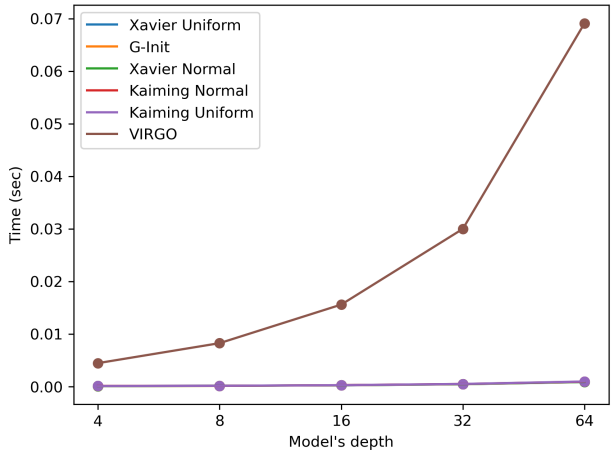

Figure 6.5: Comparison between 6 weight initialization methods for the needed time to initialize a GCN as its depth increases. All methods, except for VIRGO, overlap in the graph, making them appear one on top of the other.

Comparison against VIRGO:
The above results showed that VIRGO approached the performance of G-Init in some datasets. Since both have been specifically devised for GNNs, we compared them further. In particular, we examined the computational cost of the different methods, since VIRGO needs to calculate intermediate factors, which are depth dependent. Figure 6.5 presents the time needed to initialize a model as its depth increases. We observe that time required for initialization by VIRGO increases exponentially with the depth of the model, in contrast to G-Init. This behavior provides further reasons for using G-Init for GNN weight initialization. Note that the computational cost of G-Init remains independent of the graph size, as it depends solely on the number of model parameters. This is analogous to standard

Table 6.3: Comparison between 6 weight initialization methods across 10 datasets for graph classification for the GAT model.

Methods	Dataset				
	BACE	BBBP	Clintox	Sider	Tox21
Xavier Normal	76.89 ± 2.3	**67.89** ± 1.9	89.31 ± 1.1	58.46 ± 1.5	75.13 ± 0.9
Xavier Uniform	76.84 ± 3.5	67.62 ± 2.4	89.68 ± 1.7	58.17 ± 0.9	75.25 ± 0.6
Kaiming Normal	76.34 ± 3.1	67.13 ± 2.2	89.00 ± 2.7	58.65 ± 1.5	75.20 ± 0.5
Kaiming Uniform	76.30 ± 2.8	67.12 ± 2.0	89.05 ± 2.2	58.02 ± 1.3	75.13 ± 0.8
VIRGO	76.32 ± 2.2	66.36 ± 2.7	89.12 ± 3.5	58.51 ± 1.0	74.25 ± 0.7
G-Init	**77.04** ± 2.1	66.79 ± 1.1	**90.01** ± 1.8	**59.21** ± 1.5	**75.34** ± 0.5

Methods	Dataset				
	Toxcast	MUTAG	ENZYMES	Proteins	$IMDB_{bin}$
Xavier Normal	64.32 ± 0.6	90.51 ± 3.2	**48.26** ± 3.6	68.29 ± 2.1	72.79 ± 3.4
Xavier Uniform	64.56 ± 0.4	91.05 ± 3.4	46.48 ± 4.3	68.11 ± 2.0	71.50 ± 2.2
Kaiming Normal	64.57 ± 0.5	90.53 ± 3.9	46.67 ± 4.0	64.64 ± 8.0	71.89 ± 2.9
Kaiming Uniform	64.53 ± 0.6	91.05 ± 3.4	47.41 ± 5.0	68.29 ± 2.8	73.97 ± 1.9
VIRGO	63.77 ± 1.3	91.87 ± 5.2	47.86 ± 4.7	66.07 ± 3.6	73.00 ± 2.9
G-Init	**64.64** ± 0.3	**93.16** ± 3.4	46.73 ± 3.7	**68.73** ± 2.0	**74.50** ± 2.1

initialization schemes, such as Kaiming and Xavier, ensuring scalability regardless of the complexity of the graph.

4 Conclusion

In this chapter, we conducted a theoretical analysis of the variance within a generic convolutional GNN and introduced a novel weight initialization method, extending the approach presented by [15] to GNNs. In particular, the proposed method takes into account the graph topology and avoids exponentially large or small variance values. We assessed the behavior of the method through a series of experiments across a diverse set of benchmark datasets, encompassing both node and graph classification tasks. Our results have confirmed the relationship between weight initialization and oversmoothing reduction, which allowed us to use deep networks, without modifying either the model's architecture or the graph topology. Additional experiments under the "cold start" scenario, where features are limited to labeled nodes, demonstrated the superior performance of G-Init.

Chapter 7

Reducing Oversmoothing in Graph Neural Networks by Changing the Activation Function

This chapter investigates oversmoothing in deep Graph Neural Networks from a new perspective, without focusing on modifying the model structure to mitigate the issue. In contrast, we analyze the impact of two fundamental yet often overlooked components: the activation function and the learning rate, as applied layer-wise throughout the network.
We demonstrate, both theoretically and experimentally, that the choice and configuration of activation functions play a central role in the smoothing dynamics of node embeddings. In particular, we study how the standard ReLU activation contributes to representation collapse in deep GNNs. Building on this insight, we propose a simple yet effective modification: adjusting the slope of ReLU to preserve representational diversity across layers. Crucially, our method does not require changes to the network's architecture or the introduction of skip connections.
We provide what is to the best of our knowledge the first study regarding the role of the activation function and the learning rate per layer of the model to oversmoothing and propose a new method to address the problem. We confirm our hypotheses both experimentally and theoretically. The new method is shown to prevent node embeddings from converging to the same point, thus leading to better node representations. We summarize our main contributions as follows:

- **Role of Activation Function in Oversmoothing**: We prove theoretically the connection between the activation function and oversmoothing. In fact, we show the relation between the slope of ReLU and the singular values of weight matrices, which are known to be associated with oversmoothing [7, 21]. We have also verified our theoretical results experimentally.

- **Role of Learning Rate in Oversmoothing**: Our analysis on the effect of the slope of ReLU to oversmoothing has a direct extension to the learning rates used per layer of the network. We conducted further experiments to study the effect of tuning the learning rates, showing that this approach could also reduce oversmoothing, but it is less practical.

- **The power of Deep GNNs**: We have performed extensive experiments using up to 64-layer networks, tackling oversmoothing with the proposed method. We further show the benefits that such deep GNNs can provide in the presence of reduced information, such as in a "cold start" situation, where node features are available only for the labeled nodes in a node classification setting.

1 Preliminaries

1.1 Normalization in deep neural networks

Of relevance to oversmoothing are also methods that perform normalization for deep neural networks [143, 144]. In such methods, the output of each neuron is normalized, in order to keep a portion of the initial feature variance. Related work proposed Self Normalized Networks (SNN), that use a different activation function (SeLU) [129]. These models perform self normalization inside each neuron, in order to keep the variance between consecutive layers stable. They have managed to enable deep Fully Connected models and achieve good performance.
A recent work regarding normalization in GNNs is the Pairnorm method [13], which aims to keep constant the total pairwise distance between node representations. Compared to SNNs, Pairnorm performs normalization needing a constant value as hyper-parameter determined per dataset, while SNNs normalize the output utilizing their activation function, which can be used in a family of neural networks. The SeLU activation function has a saturating region to reduce data variance and a slope slightly greater than 1 in order to increase data variance, when needed.

2 Understanding and dealing with oversmoothing

Based on the mathematical definition of oversmoothing by [7], in this section we establish a connection with the training process of GNNs. In particular, we analyse the role of the activation function and the learning rate and we propose modified versions of GNNs to address the issue.

2.1 Theoretical Analysis

We start by establishing the connection between oversmoothing and variance reduction of node representations. Consider oversmoothing as the convergence to a subspace of the feature space, where node representations are almost the same. In that particular subspace, the initial variance of feature vectors has been massively reduced. Using ReLU, this unwanted oversmoothing effect is "harsh", due to the mapping of negative input to zero. We use the term "harsh" to indicate the smoothing case, where representations are mapped to a single point (i.e. zero), instead of converging to a subspace.
We will now show how the slope of the ReLU activation function affects oversmoothing, starting with two important assumptions: (a) the non-exploding gradients, and (b) independence of ReLU's probability not to output zero. We use these assumptions to extract bounds on the weights of the network and then use these bounds to determine the relationship between the largest singular value and the elements of each weight matrix. Given that relationship, we connect our analysis with existing literature regarding oversmoothing through Theorem 1.

Assumption 19. *For each layer l there exists a number G_l which is the upper bound to the gradients of the output of the subsequent layer (l+1) with respect to the weight elements of $W^{(l)}$, i.e.*

$$\frac{dO^{(l+1)}}{dw^{(l)}_{old_{i,j}}} \leq G_l, \quad \forall w^{(l)}_{old_{i,j}} \tag{7.1}$$

where $O^{(l+1)}$ is the output of (l+1)-th layer.

This assumption needs to hold in all cases, in order to avoid the exploding gradient case, that would not allow the learning process to converge.

Assumption 20. *The probability, that the output of the ReLU function does not equal zero at layer l is independent of the outputs of the previous layers, in a feed-forward ReLU neural network, i.e.*

$$P(ReLU(h^{(l)}) \neq \vec{0} | \forall h^{(j)}, j < l) \leq p, \quad p \in [0, 1) \tag{7.2}$$

where $h^{(l)}$ is the node representation at layer l, ReLU is applied piece-wise on that representation and $\vec{0}$ is the all zero vector.

Following the results of [126] we get that $P(ReLU(h^{(l)}) \neq \vec{0} | \forall h^{(j)}, j < l) = 1/2$ for feed-forward networks (FFs), on the condition that the output of the previous layers is positive. That condition does not necessary hold in GNNs, due to the role of the adjacency matrix in the aggregation scheme. In FFs, when the output of a layer is zero, so are the outputs of all subsequent layers. In GNNs there might be cases, where the output of layer l (a node representation) is zero but the aggregation using the adjacency matrix produces a non-zero representation in layer (l+1). Apart from this difference, GNNs can be considered a special case of FFs, with the proper wiring due to the adjacency matrix aggregation. Therefore, the above assumption holds also for GNNs.

Lemma 21. *For a network of depth dep the total gradient reaching the l-th layer* $\left(\text{i.e., } \frac{dJ}{dw^{(l)}_{old_{i,j}}} \right)$ *in order to update $W^{(l)}$ is bounded by:*

$$\frac{dJ}{dw^{(l)}_{old_{i,j}}} \leq \alpha^{(dep-l)} \cdot G_l \tag{7.3}$$

Where $\mathcal{J}$ is the model's loss function (i.e. Cross Entropy), α stands for the ReLU slope and G_l is the upper bound of gradients of the output of the subsequent layer (l+1) with respect to the weight elements of $W^{(l)}$.

The proof of Lemma 21 is shown in Appendix 19 and is based on the chain rule for backpropagation. Given the derivative of the ReLU function and the upper bound G_l of Assumption 19, we can derive the bound of Equation 7.3. To compute the gradient with respect to weight element $w^{(l)}_{i,j}$ of layer l, we repeatedly differentiate nested ReLU functions, leading to a product of their slopes (or zero). In the final differentiation step, we get the gradient of the output of (l+1)-th layer with respect to $w^{(l)}_{i,j}$, which is bounded by G_l.

Lemma 22. *While model's loss, through gradients, flows backwards, some weight elements do not receive updates, because we have dying ReLUs (i.e. ReLUs that output zero) [126]. The total number of weight elements getting updated at layer l is bounded by:*

$$\#\{w^{(l)}_{i,j}\} \leq p^{(dep-l)} \cdot d^2 \tag{7.4}$$

where d is the largest of the two dimensions of $W^{(l)}$, i.e. there are at most d^2 elements in $W^{(l)}$, if it is a square matrix.

The proof of Lemma 22 appears in Appendix 20. Given Lemma 21, the gradient flowing backwards, with respect to a weight element $w_{i,j}^{(l)}$, contains a product of ReLU derivatives. In order for $w_{i,j}^{(l)}$ to get an update, all ReLU derivatives need to be non-zero, otherwise the gradient will be zeroed and $w_{i,j}^{(l)}$ will not be updated. Additionally, due to Assumption 20, weight elements located at the lower weight matrices tend to receive fewer updates. This is due to the fact, that the further the gradient flows backwards the more ReLU derivative factors appear in it. Based on Assumption 20, the probability of all of them to be non-zero decreases.
Regarding the singular values (denoted by $s_l(\cdot)$) of a weight matrix $W^{(l)}$ at the l-th layer, the largest of them is given by: $max(s_l(W^{(l)})) = ||W^{(l)}||_2 \leq ||W^{(l)}||_F = \sqrt{\sum |w_{i,j}^{(l)}|^2}$, where $||\cdot||_F$ is the Frobenius norm and $||\cdot||_2$ is the spectral norm. Using the general weight updating rule $\left(\text{i.e., } w_{new_{i,j}}^{(l)} = w_{old_{i,j}}^{(l)} + \eta \cdot \frac{dJ}{dw_{old_{i,j}}^{(l)}}\right)$, where J is the model's loss and η is the learning rate, we arrive at the main theorem of this work.

Theorem 23. *The upper bound of the largest singular value of the weight matrix $W^{(l)}$ at layer l for a GNN model, utilizing a ReLU activation function, depends on the slope of the function. That bound is given per layer and shows the effect of each iteration of updates on the weight matrix. We denote with W_{old} and W_{new} the weight matrices before and after the update during an iteration of the training process respectively.*

$$max(s_l(W_{new}^{(l)})) \leq ||W_{old}^{(l)}||_F + \sqrt{3} \cdot p^{\left(\frac{dep-l}{2}\right)} \cdot d \cdot \alpha^{(dep-l)} \cdot B_l \tag{7.5}$$

where $B_l = \eta \cdot G_l$, dep is the network's depth, d is the largest dimension of the $W^{(l)}$ matrix, p is the upper bound of the probability of ReLU not to output zero and α is ReLU's slope.

The proof of Theorem 23 appears in Appendix 21 and uses the upper bound of the largest singular value by the Frobenius norm, expanded according to the weight update rule. Separating weight elements into two sets (updated and not updated) we identify the gradient values responsible for the weight updates. These values are bounded (Lemma 21), as is the number of updated elements (Lemma 22), leading to Equation 7.5. The resulting upper bound shows the connection of the slope of ReLU and the upper bound of the probability of ReLU not to output zero with the largest singular value of the weight matrix. That largest singular value is connected with the oversmoothing problem and as we mentioned above could act as a resource to reduce it.

Theorem 24. *The lower bound of the largest singular value of the weight matrix $W^{(l)}$ at layer l for a GNN model, utilizing the ReLU activation function, depends on the slope of the function. That bound is defined per layer and shows the effect of each iteration of updates on the weight matrix. We denote with W_{old} and W_{new} the weight matrices before and after the update that happens during an iteration of the training process respectively.*

$$max(s_l(W_{new}^{(l)})) \geq \sqrt{2\sqrt{3} \cdot max(w_{old_{i,j}}^{(l)}) \cdot B_l'} \cdot \alpha^{\frac{dep-l}{2}} \tag{7.6}$$

where $B_l' = \eta \cdot L_l$, L_l is the lower bound to the gradient with respect to the output of the subsequent layer $\left(\frac{dO^{(l+1)}}{w_{oldi,j}^{(l)}} \geq L_l\right)$, dep is the network's depth and α is ReLU's slope.

The proof of Theorem 24 appears in Appendix 21 and is analogous to the one used in Theorem 23. Theorem 24 indicates the effect of ReLU's slope to the lower bound of the largest

singular value of the weight matrix. From Equation 7.6 we observe that, increasing the slope of ReLU results in an increment of the lower bound of the maximum singular value of the weight matrix. Therefore, a larger ReLU slope yields greater largest singular value, which in turn reduces the oversmoothing effect.

2.2 Alleviating Oversmoothing

We transfer the idea of SeLU to GNNs, focusing on the part that increases the variance, because the repetition of the Laplacian operator acts as a variance reducer. In order to avoid oversmoothing in deep GNNs using this approach, we need to identify a 'sweet' spot, where variance reduction from graph convolution is counteracted as needed by the slope of the activation function. Typically in GNNs, ReLU is used with a slope value $\alpha = 1$. As a result, the second exponential factor in Equation 7.5 can be ignored (it is always equal to 1). This pushes the bound for the largest singular value of weight matrices towards a fixed low value, when the architecture of the network gets deeper (dep increases), and especially at the lower layers (small l) of the network. This is due to the first exponential factor of Equation (7.5) converging to zero for large values of dep.
The restriction of the largest singular values to low values, increases the speed of convergence to the oversmoothng subspace, as stated in Corollary 2 from [7]. According to the corollary, the speed depends on λ and s. The former parameter (λ) is a property of the data (largest eigenvalue of the adjacency matrix), while the latter (s) is the product of the largest singular values of the weight matrices.
Realizing the importance of the slope of ReLU in the training process, we move on to propose a simple modification of the activation function that reduces oversmoothing. Following [126], we proceed our analysis with $p = 1/2$ (see Assumption 20), which simplifies calculations. In particular, we observe that a slope of 2 makes the second exponential factor prevail over the first one, leading to a new combined factor of $2^{\left(\frac{dep-l}{2}\right)}$. This in turn increases the upper bound for the largest singular value, restricting the influence of the layer index (l) and depth (dep) in Equation (7.5). It is worth noting, that the upper bound should not be constant across all layers (which could be achieved by setting a corresponding value for the slope), because different layers of the network have different roles and the goal of a GNN is to bring intra-class representations close, while keeping inter-class representations apart. Moreover, an increased value of the slope pushes the lower bound of the largest singular value, as shown in Equation 7.6.
Using the results of [7] we can further connect the choice of slope for the activation function to the speed of convergence to the oversmoothing subspace. The proposed method increases the bound for each largest singular value, which in turn decreases the speed of convergence to the oversmoothing subspace, according to Corollary 2.

2.3 Modifying the slope of ReLU: Limitations

Oono & Suzuki [7] have proved that any deep enough GCN-like GNN will eventually reach the oversmoothing subspace. In theory, this also holds for our method, when the GNN gets very deep. Could this be avoided by increasing the slope of ReLU further? Unfortunately not, as we cannot make the slope too large. It is known, that by increasing the slope of ReLU too much, one may face the problem of exploding gradients [145], that impedes the learning procedure. In fact, if we increase the slope of ReLU too much, the lower bound in Equation 7.6 over increases, the upper bound in Equation 7.5 becomes too loose and the

performance degrades. Our proof suggests, that a slope equal to 2 avoids oversmoothing and our experiments have shown that it also avoids the exploding gradients problem. In fact, the choice of 2 is not exact, but that particular value seems to work well as depth increases. Our experiments have shown, that the proposed method is not sensitive on the exact slope and that values around 2 also manage to reduce oversmoothing.

2.4 Modifying the learning rate, instead of the slope of ReLU

Instead of changing the slope of ReLU, we could opt to modify the Learning Rate (LR) per layer in order to increase the lower bound in Equation 7.6 and the upper bound in Equation 7.5. In particular, a different learning rate (η) per layer leads to a different B_l ($= \eta \cdot G_l$) per layer, counteracting the problematic first exponential factor of Equation 7.5.
However, determining the right LR per layer is a non-trivial task. Intuitively, LR should be larger in the lower layers and smaller in the upper ones. Keeping the slope equal to 1, the first problematic exponential term is smaller (note that $p < 1$) in the lower layers, reducing the value of the upper bound. To avoid this, one needs to tune LR carefully. Additionally, LR affects heavily the learning process and large LR values might lead to oscillations and poor learning performance. Modifying the slope and the learning rate combined led to negligible improvements. Preliminary results with modified LR values are shown in Appendix 22.

3 Experiments

3.1 Experimental Setup

Datasets: Aligned to most of the literature, we use four well-known benchmark datasets: *Cora, CiteSeer, Pubmed* and the less homophilic dataset *Texas*. The statistics of the datasets are reported in the Appendix 24. For *Cora, CiteSeer* and *Pubmed* we use the same data splits as in [25], where all nodes except the ones used for training and validation are used for testing. For *Texas* we use the same splits as in [146].
Models: We utilize two different GNNs as our base models for the proposed methodology; namely GCN [25] and GAT [41]. We compare the results of these models with and without the modification of the slope of ReLU for varying number of layers. We do not compare against methods utilizing residual connections, because the aim of this work is to show that oversmoothing could be avoided without "short-circuiting" initial information to latter layers. We use residual GNNs as a baseline in the Extended Experiments section, Appendix 23.
Hyperparameters: We set the number of hidden units (of each layer) of GCN and GAT to 128 for both models across all datasets. The L_2 regularization is set with penalty $5 \cdot 10^{-4}$ for both models and learning rate of 10^{-3}. We vary the depth between 2 and 64 layers. The number of attention heads for GAT is set to 1.
Configuration: Each experiment is run 10 times and we report the average performance over these runs. We train all models for 200 epochs using Cross Entropy as a loss function. For our experiments we used an RTX 3080ti GPU.

3.2 Experimental Results

Reducing oversmoothing:
Table 7.1 presents the classification performance of the two base models (GCN and GAT)

Table 7.1: Performance comparison of vanilla GCN and GAT, against SeLU and Slope2GNN enhanced versions of the same models, in *Cora, CiteSeer, Pubmed, Texas*. Average test node classification accuracy (%) for networks of different depth. With **bold** is the best performing model for each depth and each dataset.

Accuracy (%)									
# Layers	Method	Cora		CiteSeer		Pubmed		Texas	
		GCN	GAT	GCN	GAT	GCN	GAT	GCN	GAT
2	Original	81.38	77.79	70.52	**69.04**	77.65	**77.33**	**59.01**	**59.54**
	Slope2GNN	**81.84**	**77.81**	**70.55**	68.24	**78.34**	77.02	57.93	56.67
	SeLU	81.74	76.59	69.91	66.81	77.80	76.83	59.01	57.21
4	Original	78.09	78.06	65.21	63.98	**76.75**	**76.34**	**59.55**	58.28
	Slope2GNN	**80.39**	**79.54**	**67.57**	**67.27**	76.46	75.59	57.66	57.39
	SeLU	79.85	79.33	67.53	67.24	74.09	72.62	58.02	**58.65**
8	Original	23.56	33.11	32.54	31.33	49.08	55.86	**57.48**	**57.65**
	Slope2GNN	76.54	**78.33**	65.80	65.55	**75.00**	**74.20**	57.18	56.85
	SeLU	**79.41**	78.19	**67.27**	**67.58**	73.43	71.93	56.85	57.47
16	Original	14.92	15.53	17.73	17.80	28.95	29.88	39.91	41.71
	Slope2GNN	**76.81**	**75.00**	57.10	56.13	**76.35**	73.81	**57.21**	**57.53**
	SeLU	76.38	74.92	**61.99**	**62.23**	76.11	**74.27**	52.88	57.44
32	Original	14.02	14.67	16.69	18.67	38.49	29.34	25.50	18.64
	Slope2GNN	**73.21**	**69.62**	47.20	49.16	**75.78**	**74.63**	**55.79**	54.14
	SeLU	68.77	69.36	**47.90**	**52.97**	71.80	73.60	55.77	**57.75**
64	Original	12.48	15.18	17.65	13.35	31.73	28.62	27.39	06.76
	Slope2GNN	**61.59**	24.17	**46.46**	**26.24**	**71.68**	38.01	46.31	41.44
	SeLU	26.85	**30.55**	27.10	20.42	40.90	**41.00**	**58.38**	**56.30**

on all four datasets, with and without the modified slope of ReLU (called Slope2GNN) for varying number of layers. Additionally, it presents results using SeLU, instead of ReLU, since it also modifies the slope of the function, while additionally normalizing node representations, as explained in section 1.1. To the best of our knowledge, this is the first time SeLU is used with GNNs.

Table 7.1 shows that methods with a modified activation function reduce oversmoothing significantly. As expected, baselines maintain a high performance for shallow architectures in homophilic datasets, where oversmoothing is not a problem. The modified activation functions (Slope2GNN and SeLU) consistently improve the test accuracy as the number of layers (#L) increases. Note that even the proposed method suffers performance degradation for very deep GNNs, due to the unavoidable nature of oversmoothing [7]. GCN and GAT average over information in the close neighborhood of the node, assuming homophily, which is not the case in the *Texas* dataset. Interestingly, low homophily in this dataset makes oversmoothing appear at a larger depth, compared to other datasets. This is because connected nodes do not have similar representations and the model needs more Laplacian operations to oversmooth them.

The effect of deeper networks on different activation functions:

Focusing on the use of SeLU, although it seems to make the models resistant to oversmoothing, the effect seems to be lost for GCNs with 64 layers. This may be due to the sensitivity of the function on the choice of slope value, which is lower than 2. Moreover, the saturating

region of SeLU, used to reduce data variance, might act in favor of oversmoothing. Finally, SNNs aim to keep the variance of the model stable, ignoring the effect of the Laplacian smoothing in GNNs.
To further examine the behavior of each method, as the number of layers increases, in Figure 7.1, we present a more detailed progression of the accuracy of GCNs, as the networks get deeper on the *Cora* dataset. The effect of oversmoothing and its reduction by the two methods that modify ReLU is very apparent here. Furthermore, the simpler Slope2GNN approach seems to be more resistant to the increase in the depth of the network.
The need for deep networks:
Having shown the benefit of using the modified activation functions in deep GNNs, we move to a set of experiments that aims to highlight also the value of using such deep architectures. In particular, we report our experimental results in the "cold start" scenario. The cold-start datasets that we use in these experiments are generated by removing feature vectors from unlabeled nodes and replacing them with all-zero vectors. For each combination of GNN and activation function, we present in Table 7.2 the best performance achieved and the depth at which the model attains that performance. The main observation in these results is the improvement in terms of accuracy with the use of deeper GNNs. This improvement is only attainable with the modified activation functions that reduce the effect of oversmoothing. The vanilla versions of GCN and GAT cannot go further than the performance of their shallow versions. Worth-mentioning is also the fact that the simple Slope2GNN approach manages to benefit from the deeper architectures even in the hard *Texas* dataset. This is not the case for SeLU, which achieves its best scores with very shallow networks. This may indicate the need to tune the slope of the activation function, as discussed above.

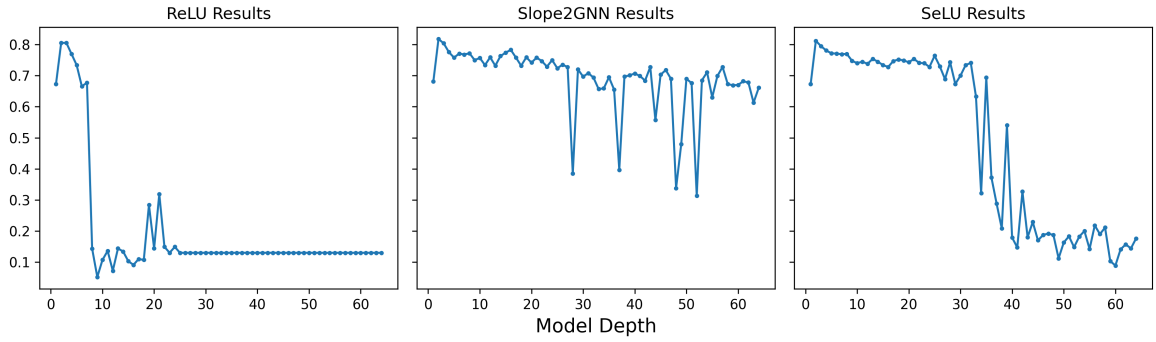

Figure 7.1: Comparison between three different activation functions for a GCN on *Cora* dataset. Y-axis shows model accuracy on test nodes, while varying the model's depth (shown on the x-axis).

Slope sensitivity:
Finally, we performed an experiment to understand the effect of different ReLU slope values on oversmoothing. Figure 7.2 presents results with GCN on the *Cora* dataset, showing that the proposed method is not sensitive to the exact slope value. Slope values around 2 seem also capable of reducing oversmoothing. Experiments on other datasets are included in Appendix 23.

Table 7.2: Comparison of different models and activation functions on the "cold start" problem. We show accuracy (%) on the test set, using GCN and GAT as backbone GNN models. Only the features of the nodes in the training set are available to the model. We also show at what depth (i.e. # Layers) each model achieves its best accuracy.

Model		GCN		GAT	
Dataset	Method	Accuracy (%)	#L	Accuracy (%)	#L
Cora	Original	64.85	4	59.74	3
	Slope2GNN	**73.47**	19	**72.78**	20
	SeLU	73.00	22	72.65	23
CiteSeer	Original	41.94	4	38.50	4
	Slope2GNN	**49.88**	21	**49.63**	26
	SeLU	49.30	19	49.38	20
Pubmed	Original	60.16	4	50.45	4
	Slope2GNN	**72.61**	32	71.14	32
	SeLU	72.50	23	**71.41**	26
Texas	Original	32.40	4	30.10	2
	Slope2GNN	**33.33**	6	**31.00**	4
	SeLU	31.62	3	30.81	2

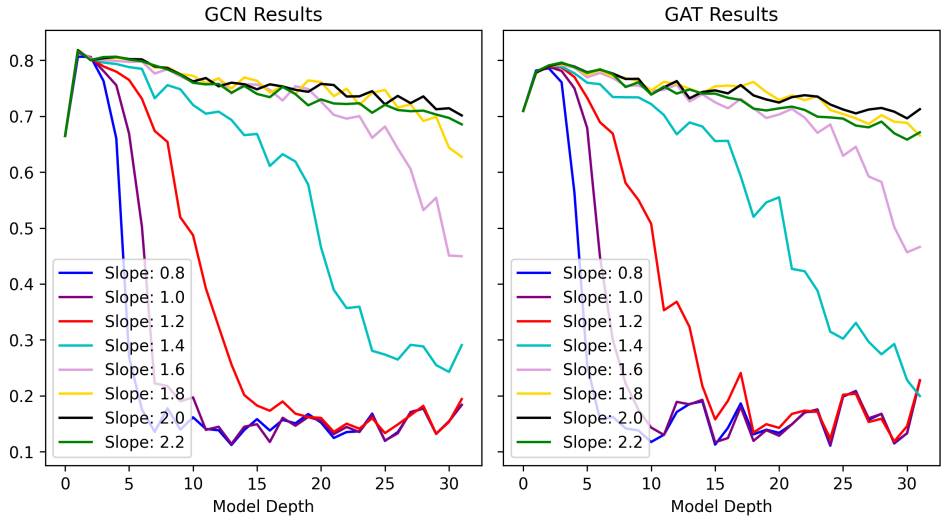

Figure 7.2: Varying ReLU slope results on *Cora* dataset.

4 Conclusion

In this chapter we have shown the important role that the slope of the ReLU activation function plays in the oversmoothing problem in GNNs. We have further proposed a simple modification that reduces drastically the problem. We have illustrated the benefits of the approach in a set of experiments with different datasets and GNNs of different depth. Additionally, we showed the improvement in accuracy one can achieve through deep architectures that do not suffer from oversmoothing. This was evident in a set of experiments that simulated the "cold start" problem of missing node features in GNNs.

Chapter 8

Discussion and Future Work

This section synthesizes the principal outcomes from Chapters 3-7, critically examines the limitations shared across our methods, and outlines promising directions for future research in oversmoothing mitigation for Graph Neural Networks.

1 Unified Insights

Across five approaches to detecting and reducing oversmoothing, several core principles emerge:
First, variance preservation is fundamental. Chapter 3's layer-wise $MASED$ metric and G-Reg regularizer explicitly monitor and penalize the collapse of node-embedding distances, while Chapter 6's G-Init method adjusts weight variances to maintain signal spread at initialization. Both strategies aim to ensure a nonzero smallest singular value of each layer's weight matrix, thereby preserving representational diversity and effective gradient flow.
Second, the interaction between architecture and topology critically shapes smoothing dynamics. In Chapter 3, we showed that coupling adjacency depth to trainable transformations accelerates collapse, motivating decoupling strategies. Chapter 4's analysis of residual architectures (APPNP, GCNII, PPRGNN) further reveals that skip connections alone may fail to capture long-range dependencies when topology and residual strength are mismatched to task requirements.
Third, training regime can act as a powerful orthogonal mitigation. Chapter 5 demonstrated that partially training only one layer in a wide GCN or GAT preserves feature diversity and rival full-training accuracy, highlighting network width as an underappreciated resource for expressivity and oversmoothing resistance.
Fourth, activation dynamics influence global depth behavior. Chapter 7 showed that a simple adjustment to the ReLU slope dramatically slows representation collapse, underscoring that local nonlinearities serve as systemic levers for smoothing control.

Together, these findings form an integrated toolkit: diagnostic metrics ($MASED$), propagation re-design (decoupling, residual analysis), selective optimization (partial training), variance-aware initialization (G-Init), and activation-based smoothing control. Each contributes a distinct but complementary mechanism for enabling deeper, more expressive GNNs.

2 Limitations

Despite these advances, our work operates under several important constraints:

Theoretical Assumptions. To obtain tractable analyses, we often assume linear message-passing, omit activation functions in spectral bounds, or treat node features as independent. In practice, modern GNNs employ attention, nonlinear aggregators, and highly correlated feature spaces, which may violate these simplifying assumptions and lead to looser bounds.

Dataset and Scenario Gaps. Most public benchmarks (e.g., citation or social-network datasets) are shallow, rarely requiring more than 4-6 GNN layers; thus, empirical validation of very deep models remains limited. All experiments so far focus on static, homogeneous graphs, leaving dynamic, heterogeneous, and multilayer networks unexplored. While "cold start" scenarios test feature sparsity, systematic evaluation across varying degrees of node-feature availability is yet to be performed.

Training Dynamics. Although Chapter 7 hints at benefits from layer-wise learning rate schedules, we lack a principled framework for selecting optimal rates per layer or for jointly tuning initialization, regularization, activation, and optimizer settings. The interplay among these components remains underexplored.

3 Future Directions

Building on these insights and acknowledging the above limitations, we identify several promising directions for future research:

Dynamic and Heterogeneous Graphs. Investigate how oversmoothing emerges in temporal networks, where adjacency evolves over time, and adapt G-Reg, G-Init, and activation slope methods to cope with the changing topology. Extend residual-topology analyses to heterogeneous graphs with multiple relation types (e.g., knowledge graphs, recommender systems) to derive propagation bounds that account for varying edge semantics.

Continual and Online Learning. Generalize the partially trained paradigm to streaming settings by developing algorithms that selectively update only the most recent layers or most informative nodes. Design adaptive mechanisms to freeze or unfreeze layers in real time, enabling models to dynamically balance smoothness and expressivity in nonstationary environments.

Layer-Wise Optimization Strategies. Develop theoretical guidelines for assigning distinct learning rates or optimizer hyperparameters per layer, informed by spectral analysis or gradient-flow metrics. Explore second-order or natural-gradient optimizers that adaptively balance variance across layers, potentially mitigating oversmoothing without explicit singular value penalties.

Real-World Deployment and Robustness. Benchmark our oversmoothing mitigation methods on large-scale industrial graphs, such as user-item networks in recommender systems or infrastructure grids, to evaluate scalability, runtime performance, and memory footprint. Assess robustness to adversarial perturbations, noisy features, and missing edges. Tailor methods to domain-specific applications in chemistry (molecular graphs), biology (protein

interaction networks), and traffic modeling, where deep GNNs can capture long-range dependencies critical for accurate predictions.

By pursuing these directions, we aim to broaden the applicability of our oversmoothing detection and mitigation strategies to establish a unified framework for training deep, expressive GNNs across diverse, real-world scenarios.

Acknowledgments

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 4th Call for HFRI PhD Fellowships (Fellowship Number: 10860).

Chapter 9

Appendix

1 Lower Bound on Δ_r

Let

$$r_i = \|H_{i,*}\|_2 = \|\hat{H}_{i,*}W\|_2, \quad M_{\hat{H}} = \max_{1\le i\le N} \|\hat{H}_{i,*}\|_2, \quad m_{\hat{H}} = \min_{1\le i\le N} \|\hat{H}_{i,*}\|_2,$$

and define the row–norm spread $\Delta_r = r_{\max} - r_{\min}$.

Step 1: Bounds on $r_{\max}$ and $r_{\min}$

From the singular-value inequalities,

$$\|\hat{H}_{i,*}W\|_2 \;\ge\; \sigma_{\min}(W)\,\|\hat{H}_{i,*}\|_2, \quad \|\hat{H}_{i,*}W\|_2 \;\le\; \sigma_{\max}(W)\,\|\hat{H}_{i,*}\|_2.$$

Choose i^* such that $\|\hat{H}_{i^*,*}\|_2 = M_{\hat{H}}$. Then

$$r_{\max} = \max_i r_i \;\ge\; r_{i^*} = \|\hat{H}_{i^*,*}W\|_2 \;\ge\; \sigma_{\min}(W)\,M_{\hat{H}}.$$

Similarly, pick j^* with $\|\hat{H}_{j^*,*}\|_2 = m_{\hat{H}}$. Then

$$r_{j^*} = \|\hat{H}_{j^*,*}W\|_2 \;\le\; \sigma_{\max}(W)\,m_{\hat{H}},$$

and since $r_{\min} = \min_i r_i \le r_{j^*}$, we have

$$r_{\min} \;\le\; \sigma_{\max}(W)\,m_{\hat{H}}.$$

Step 2: Subtract to obtain Δ_r

Subtracting the two bounds (lower bound on $r_{\max}$ minus upper bound on $r_{\min}$) gives

$$\Delta_r = r_{\max} - r_{\min} \;\ge\; \sigma_{\min}(W)\,M_{\hat{H}} \;-\; \sigma_{\max}(W)\,m_{\hat{H}}.$$

Step 3: Express via the condition number

Define the condition number $\kappa(W) = \sigma_{\max}(W)/\sigma_{\min}(W)$. Then $\sigma_{\max}(W) = \kappa(W)\,\sigma_{\min}(W)$, and the bound becomes

$$\Delta_r \;\ge\; \sigma_{\min}(W)\,\big(M_{\hat{H}} - \kappa(W)\,m_{\hat{H}}\big).$$

This is nonnegative (and thus meaningful) whenever $M_{\hat{H}} \ge \kappa(W)\,m_{\hat{H}}$, i.e. when the inherent spread in $\hat{H}$ exceeds the distortion introduced by W. We make that assumption in order to further proceed our analysis.

2 MASED Evolution plots

Figures 9.1, and 9.2 present the $MASED$ evolution for ResGCN and SGC models. Additionally, Figures 9.3, 9.4, and 9.5 present the $MASED$ evolution on the *CiteSeer, Pubmed, Computers,* and *Physics* datasets for all models.

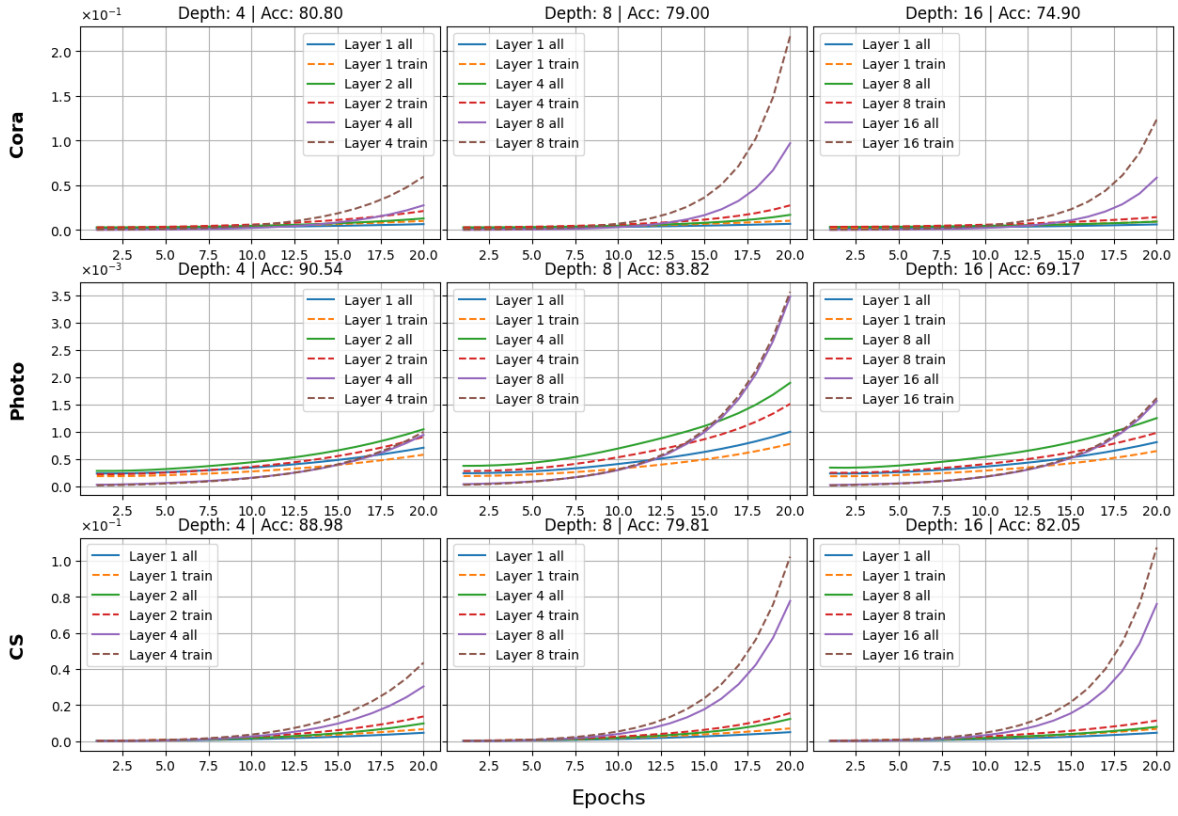

Figure 9.1: Epoch evolution of the Mean Average Squared Euclidean Distance ($MASED$) value of the embeddings of all nodes and training nodes. We show results for 3 different depths of a ResGCN model and $MASED$ values in different layers within the model. We show how $MASED$ evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

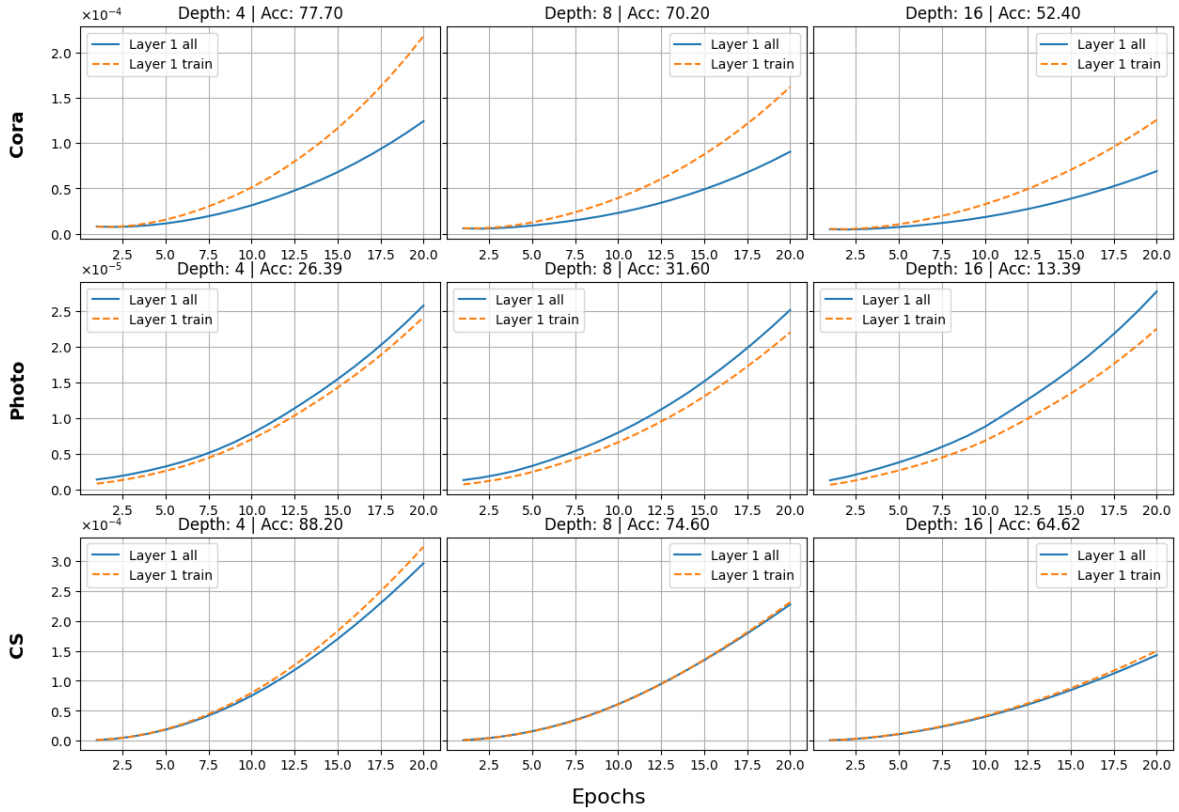

Figure 9.2: Epoch evolution of the Mean Average Squared Euclidean Distance ($MASED$) value of the embeddings of all nodes and training nodes. We show results for 3 different depths of a SGC model and $MASED$ values in different layers within the model. We show how $MASED$ evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

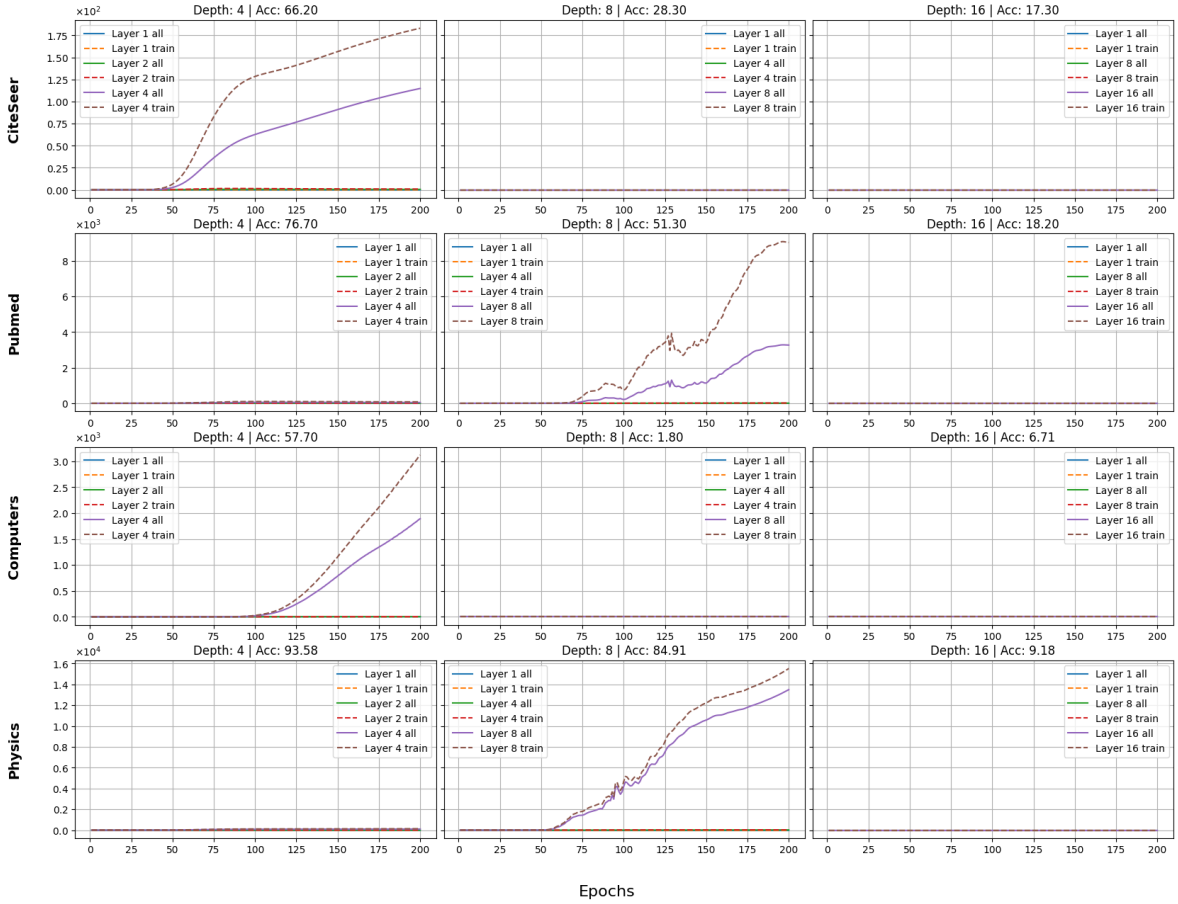

Figure 9.3: Epoch evolution of the Mean Average Squared Euclidean Distance ($MASED$) value of the embeddings of all nodes and training nodes. We show results for 3 different depths of a GCN model and $MASED$ values in different layers within the model. We show how $MASED$ evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

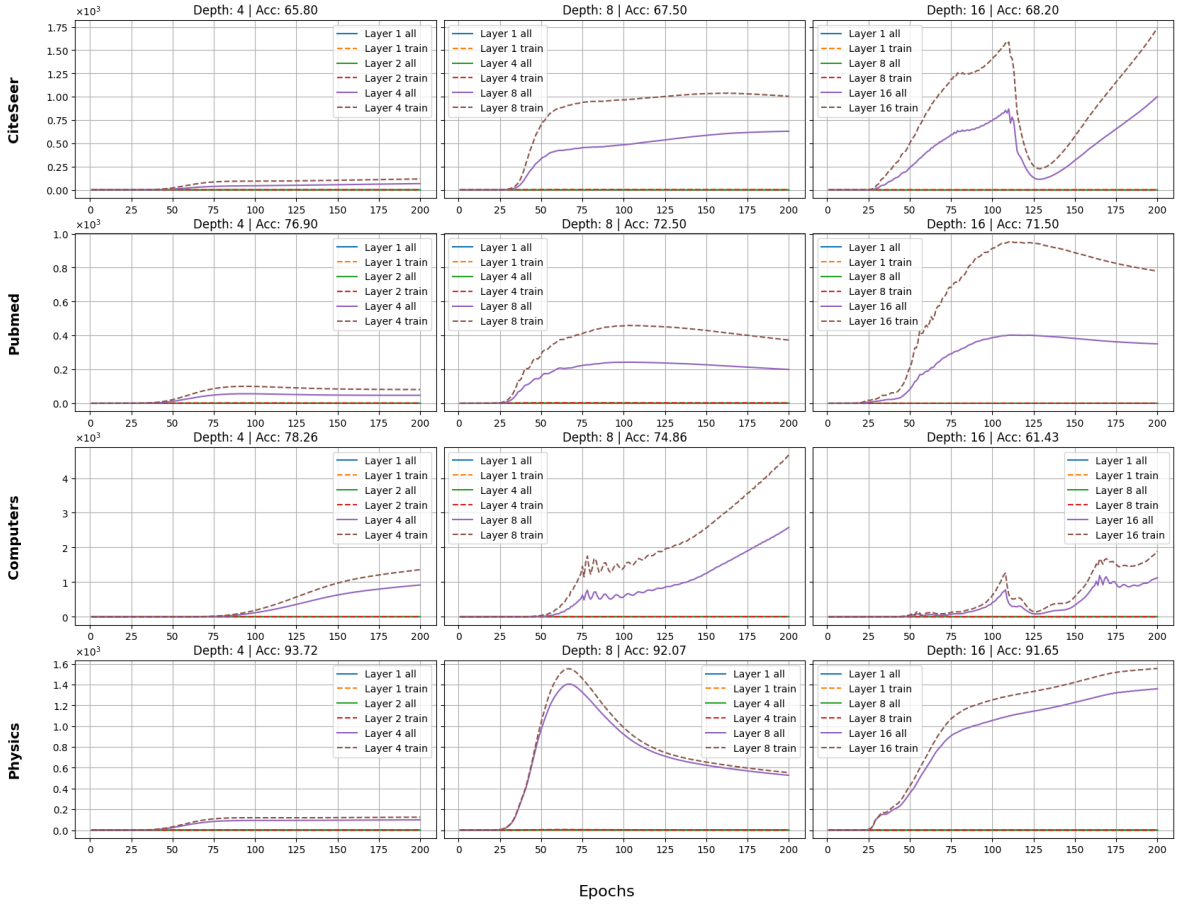

Figure 9.4: Epoch evolution of the Mean Average Squared Euclidean Distance ($MASED$) value of the embeddings of all nodes and training nodes. We show results for 3 different depths of a ResGCN model and $MASED$ values in different layers within the model. We show how $MASED$ evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

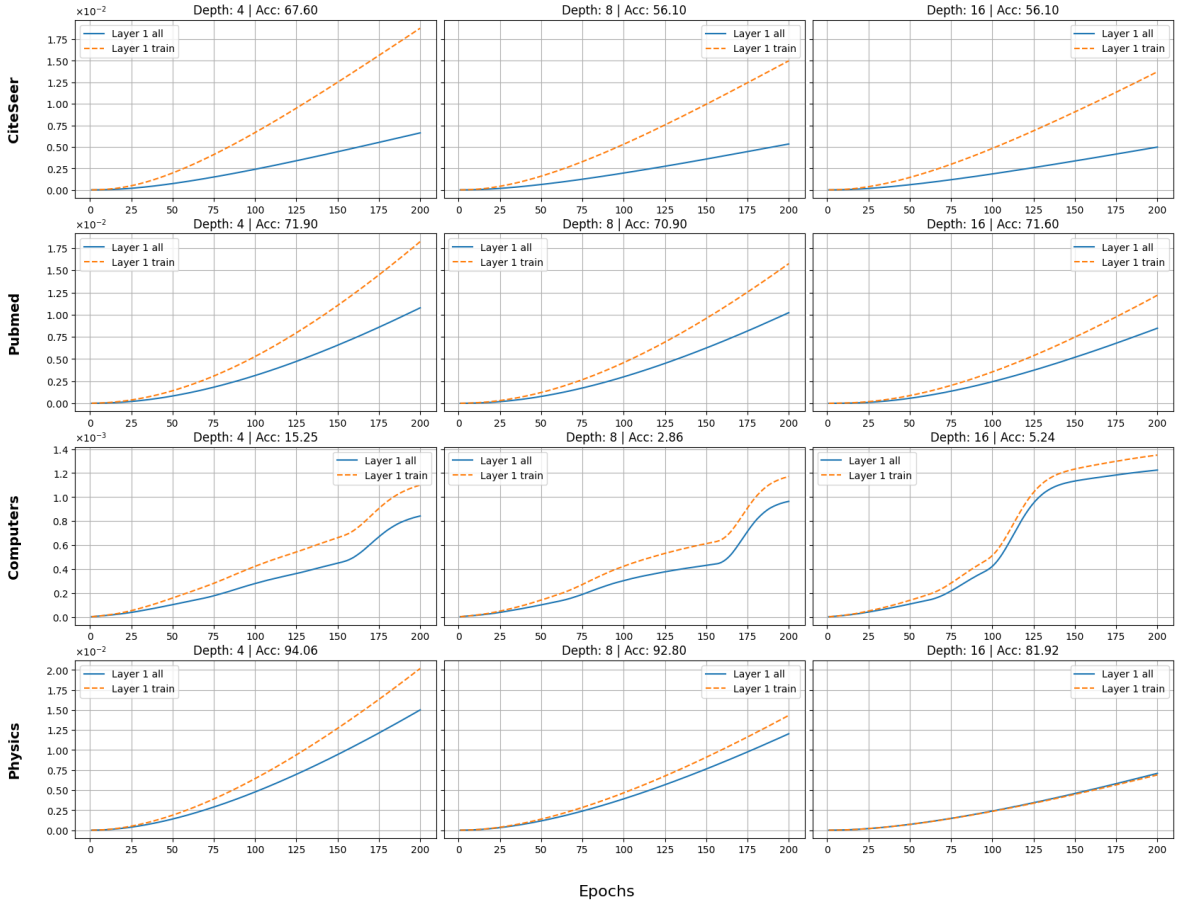

Figure 9.5: Epoch evolution of the Mean Average Squared Euclidean Distance ($MASED$) value of the embeddings of all nodes and training nodes. We show results for 3 different depths of a SGC model and $MASED$ in different layers within the model. We show how $MASED$ evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

3 Embedding Norms & Centroids Angle Evolution plots

Figure 9.6 and Figure 9.7 show the evolution of the norms during 50 epochs for ResGCN and SGC models.

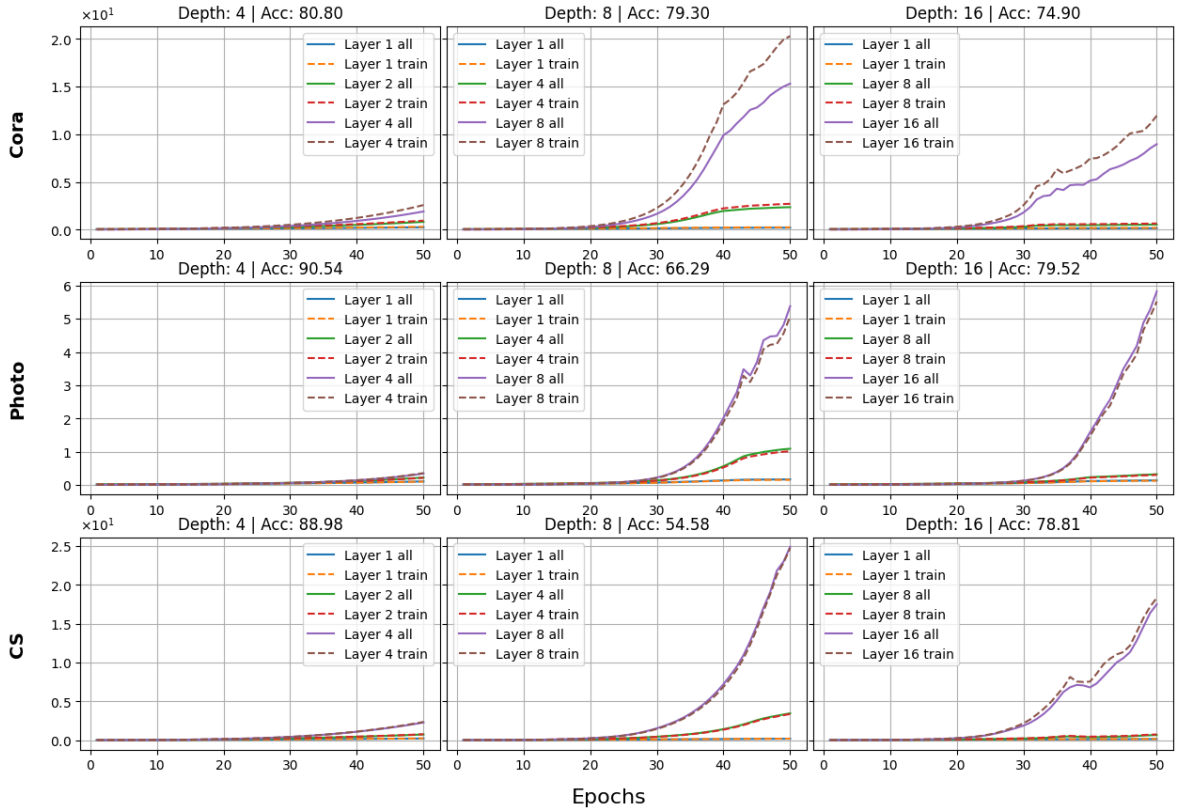

Figure 9.6: Epoch evolution of the average value of the norms of the embeddings of all nodes and training nodes separately. We show results for 3 different depths of a ResGCN model and average norm values in different layers within the model. We show how norms evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

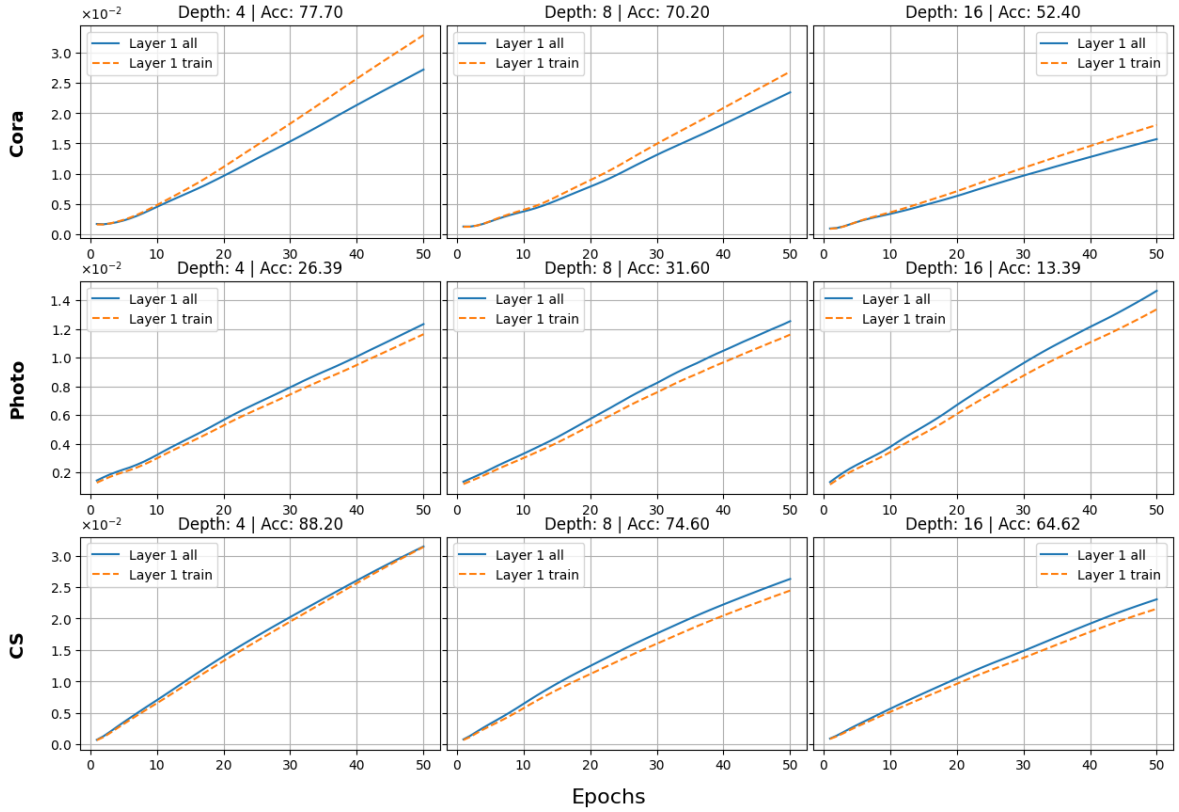

Figure 9.7: Epoch evolution of the average value of the norms of the embeddings of all nodes and training nodes separately. We show results for 3 different depths of a SGC model and average norm values in different layers within the model. We show how norms evolve in the first, the middle and the last layer of each model.

Figures 9.8, 9.9, and 9.10 present the evolution of norms for 200 epochs for GCN, ResGCN, and SGC models.

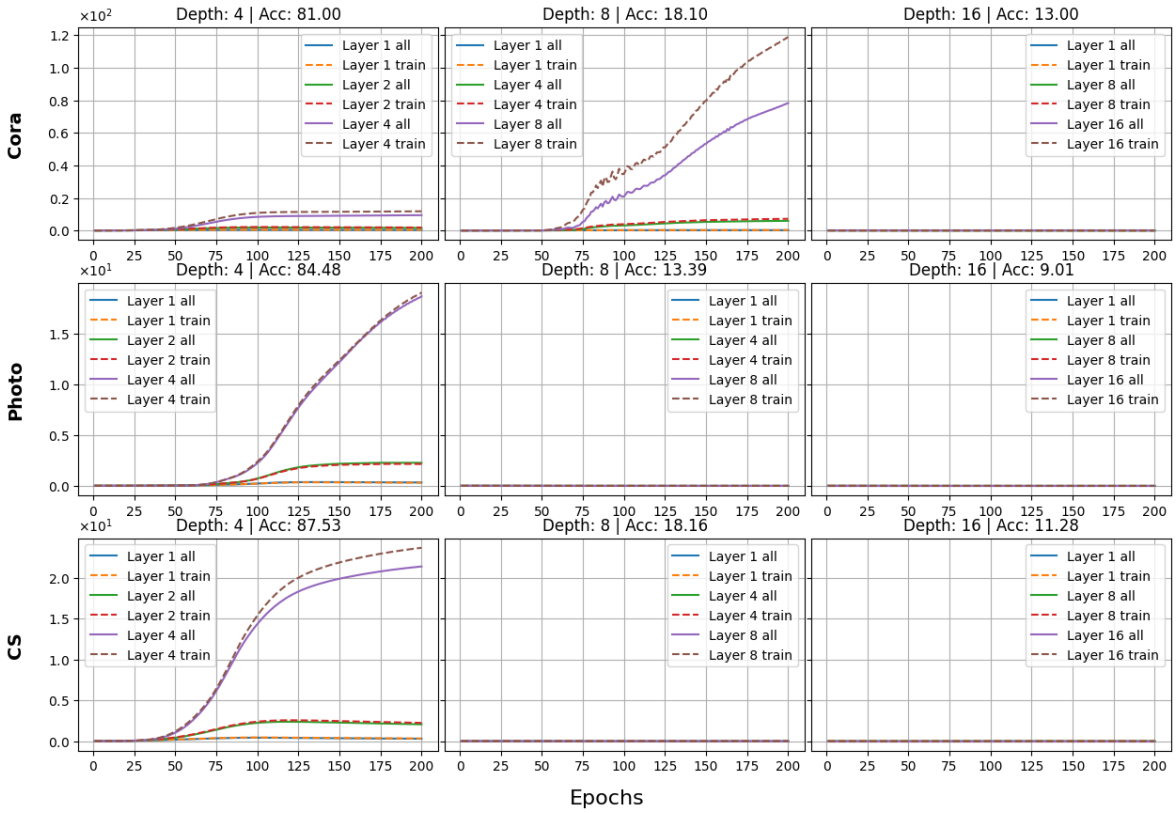

Figure 9.8: Epoch evolution of the average value of the norms of the embeddings of all nodes and training nodes separately. We show results for 3 different depths of a GCN model and average norm values in different layers within the model. We show how norms evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

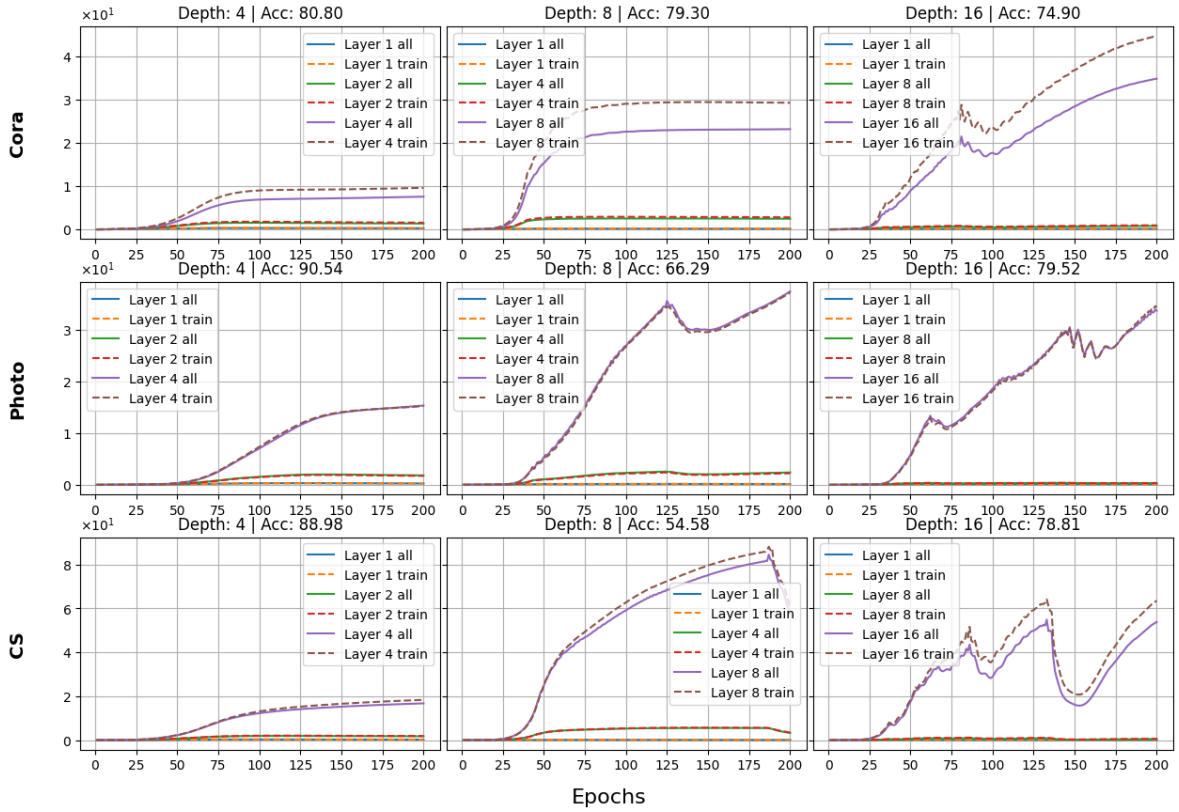

Figure 9.9: Epoch evolution of the average value of the norms of the embeddings of all nodes and training nodes separately. We show results for 3 different depths of a ResGCN model and average norm values in different layers within the model. We show how norms evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

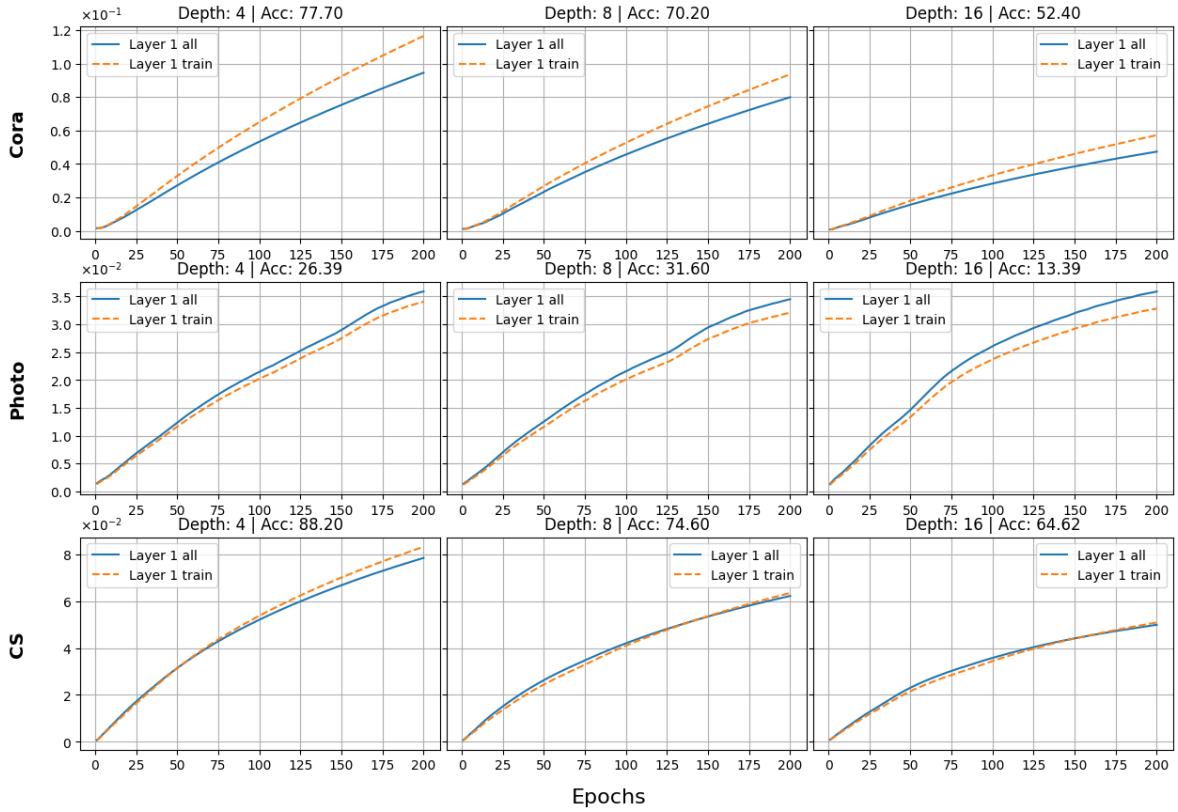

Figure 9.10: Epoch evolution of the average value of the norms of the embeddings of all nodes and training nodes separately. We show results for 3 different depths of a SGC model and average norm values in different layers within the model. We show how norms evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

Figures 9.11, 9.12, and 9.13 show the norm evolution on the *CiteSeer, Pubmed, Computers,* and *Physics* datasets for all models under investigation.

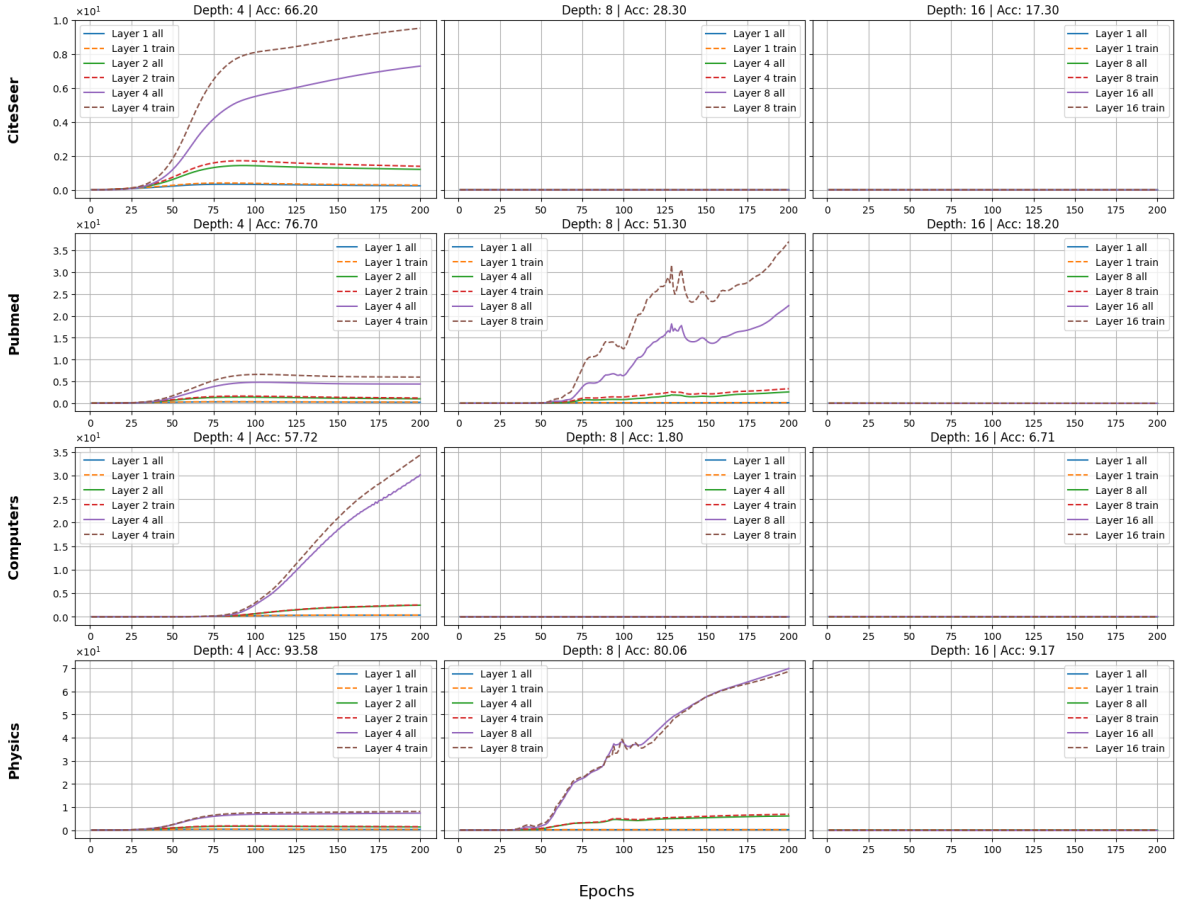

Figure 9.11: Epoch evolution of the average value of the norms of the embeddings of all nodes and training nodes separately. We show results for 3 different depths of a GCN model and average norm values in different layers within the model. We show how norms evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

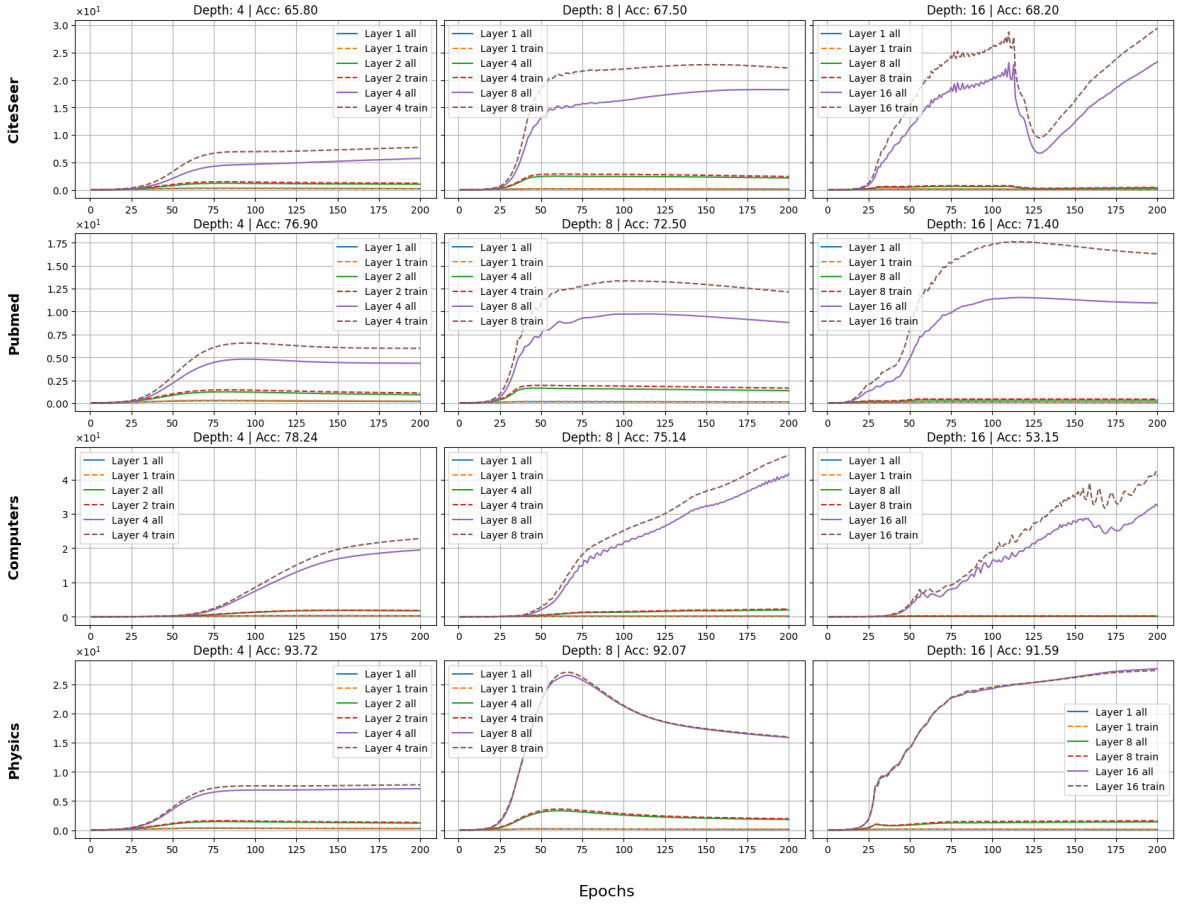

Figure 9.12: Epoch evolution of the average value of the norms of the embeddings of all nodes and training nodes separately. We show results for 3 different depths of a ResGCN model and average norm values in different layers within the model. We show how norms evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

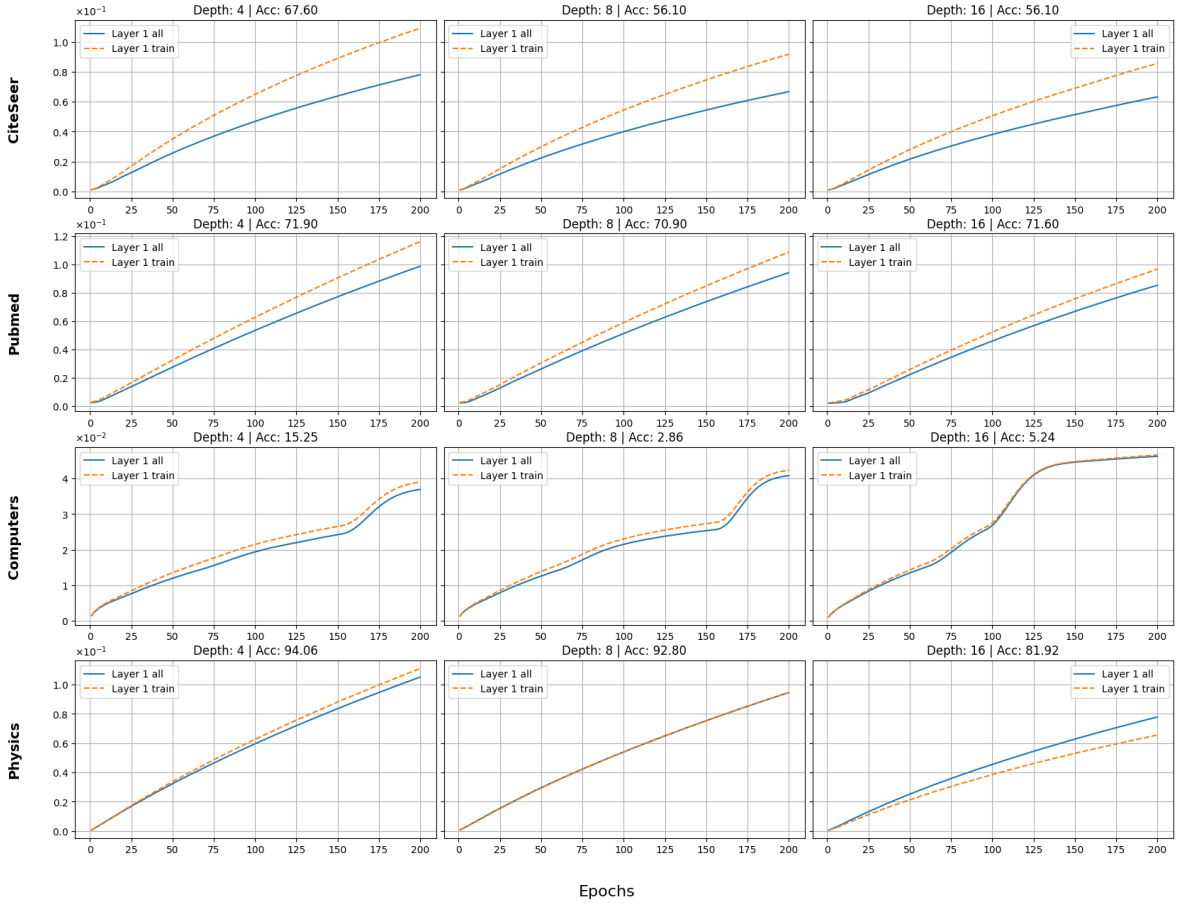

Figure 9.13: Epoch evolution of the average value of the norms of the embeddings of all nodes and training nodes separately. We show results for 3 different depths of a SGC model and average norm values in different layers within the model. We show how norms evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

Figures 9.14, and 9.15 show the evolution of the average angle between the centroids of the embeddings of the training nodes for ResGCN and SGC. Figures 9.16, 9.17, and 9.18 present the angle evolution on the *CiteSeer, Pubmed, Computers,* and *Physics* datasets.

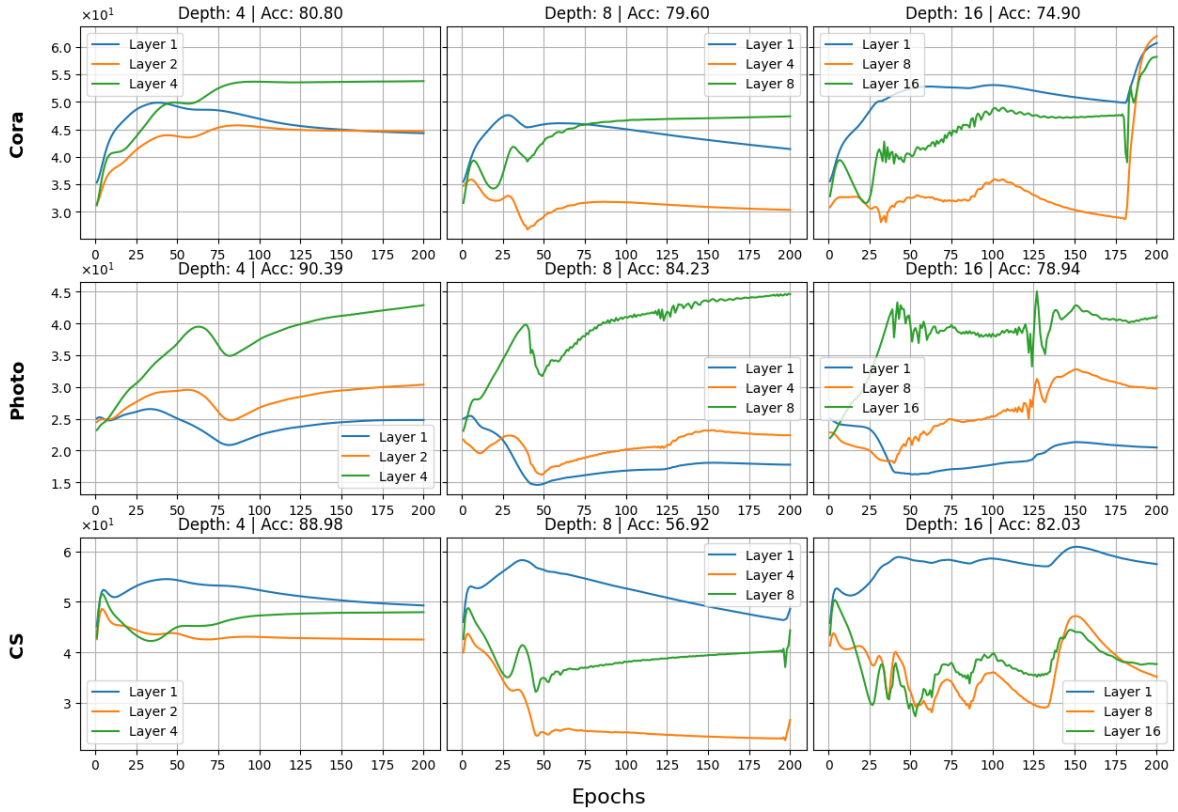

Figure 9.14: Epoch evolution of the average value of the angles between the centroids of the embeddings of the training nodes. We show results for 3 different depths of a ResGCN model and average norm values in different layers within the model. We show how angles evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

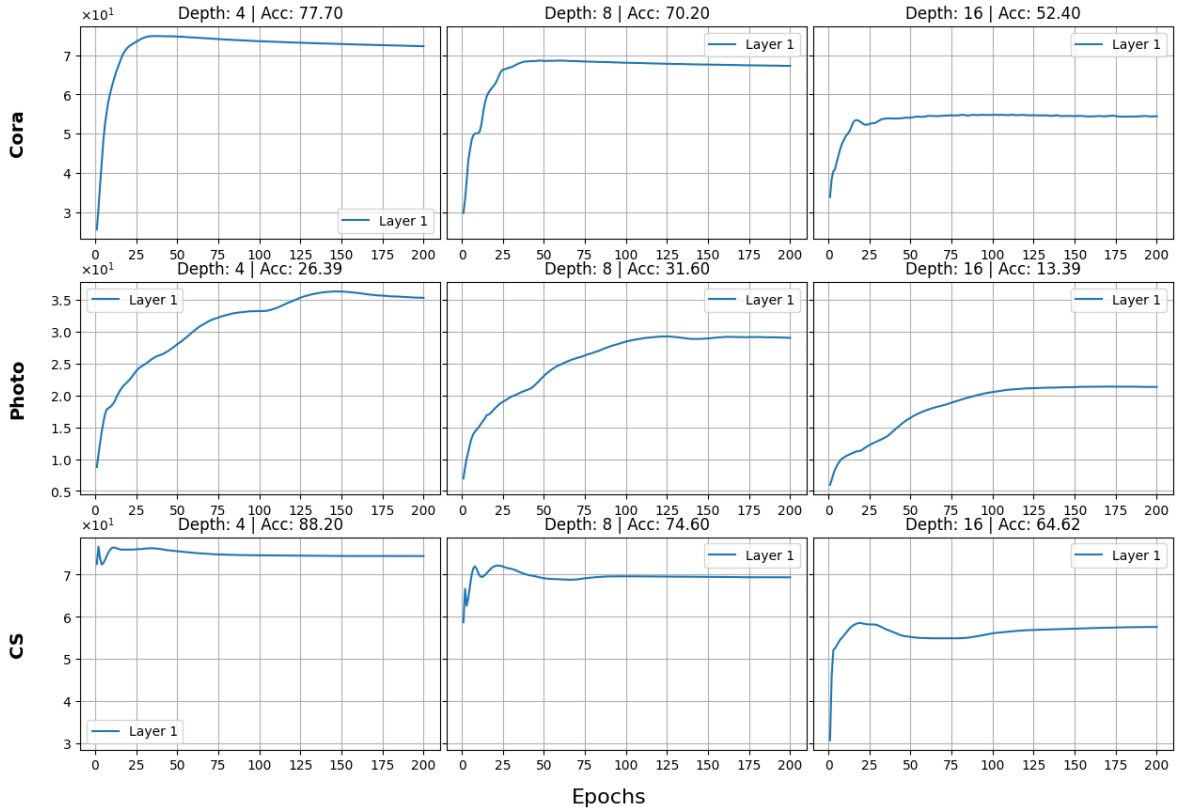

Figure 9.15: Epoch evolution of the average value of the angles between the centroids of the embeddings of the training nodes. We show results for 3 different depths of a SGC model and average norm values in different layers within the model. We show how angles evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

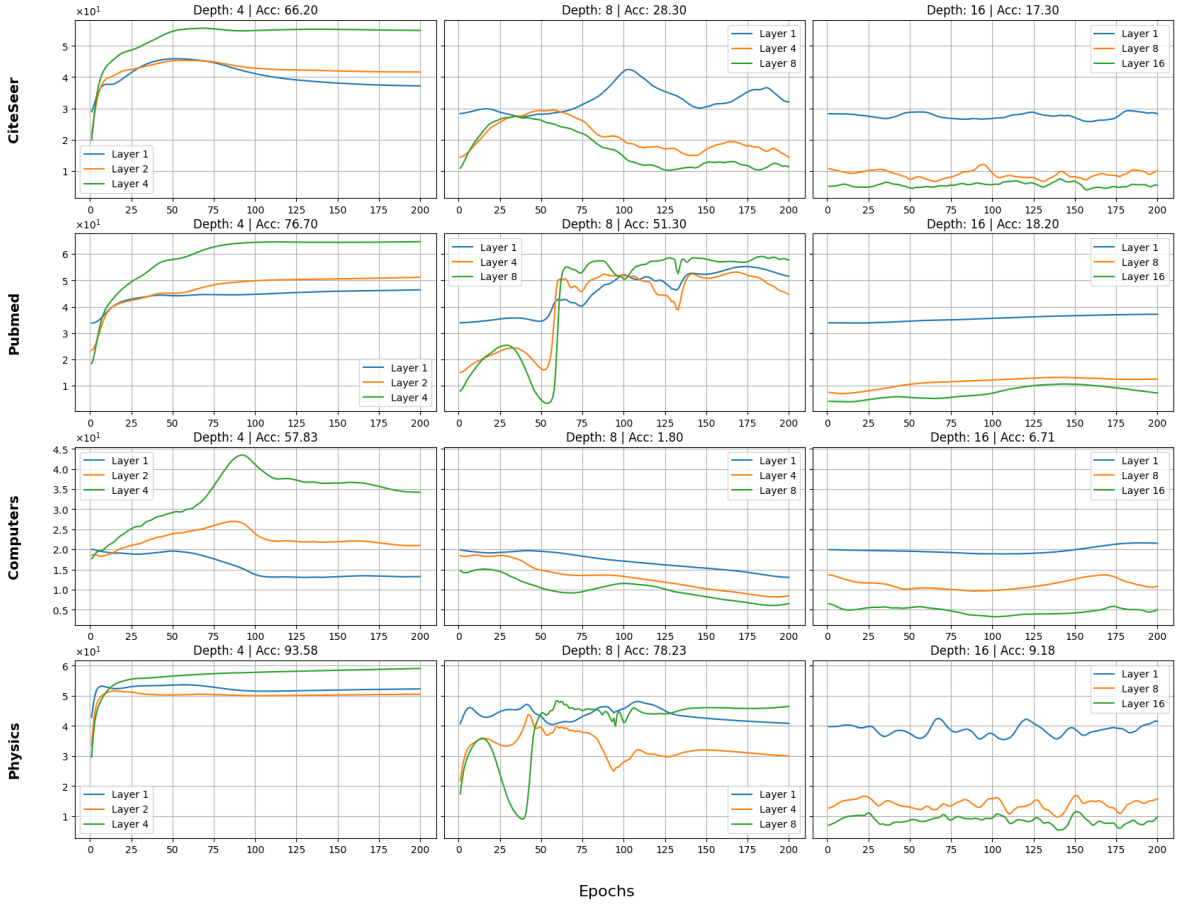

Figure 9.16: Epoch evolution of the average value of the angles between the centroids of the embeddings of the training nodes. We show results for 3 different depths of a GCN model and average norm values in different layers within the model. We show how angles evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

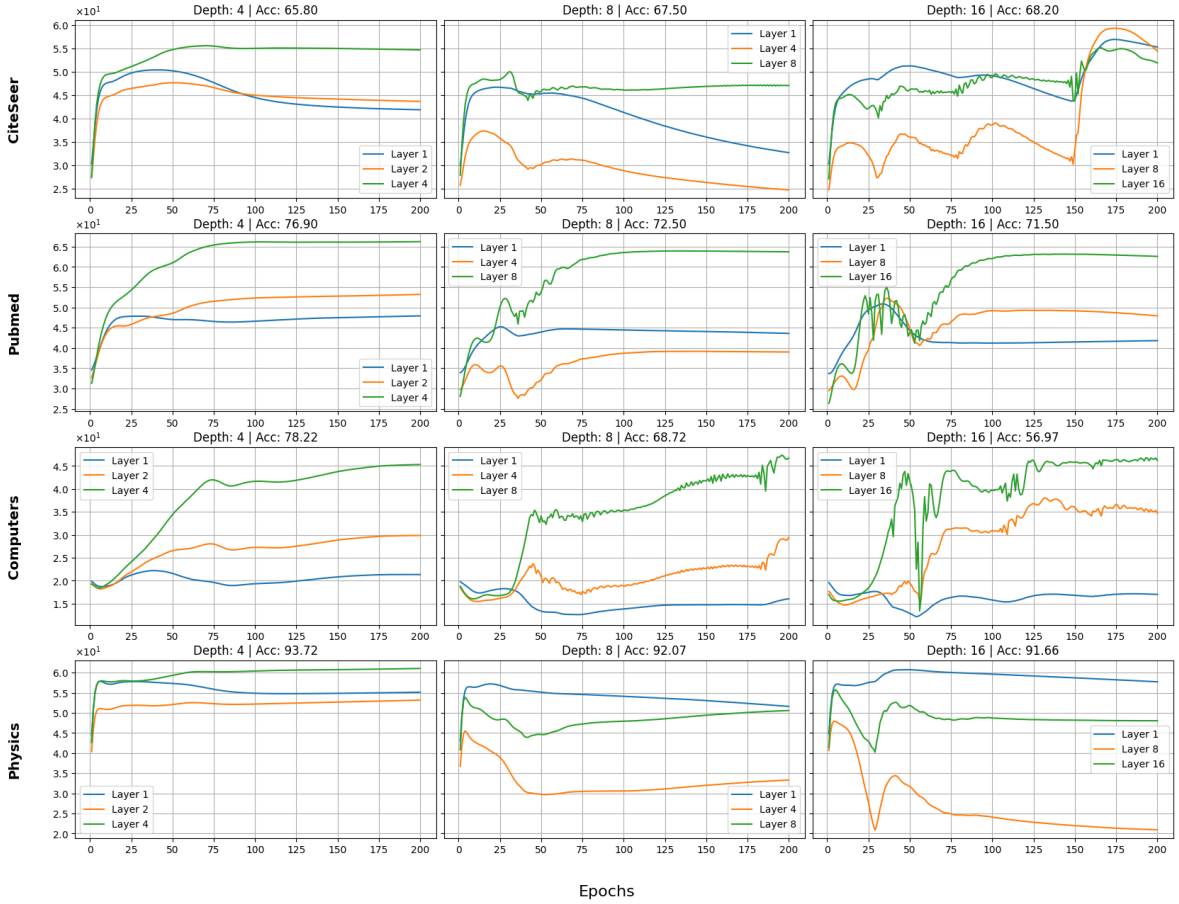

Figure 9.17: Epoch evolution of the average value of the angles between the centroids of the embeddings of the training nodes. We show results for 3 different depths of a ResGCN model and average norm values in different layers within the model. We show how angles evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

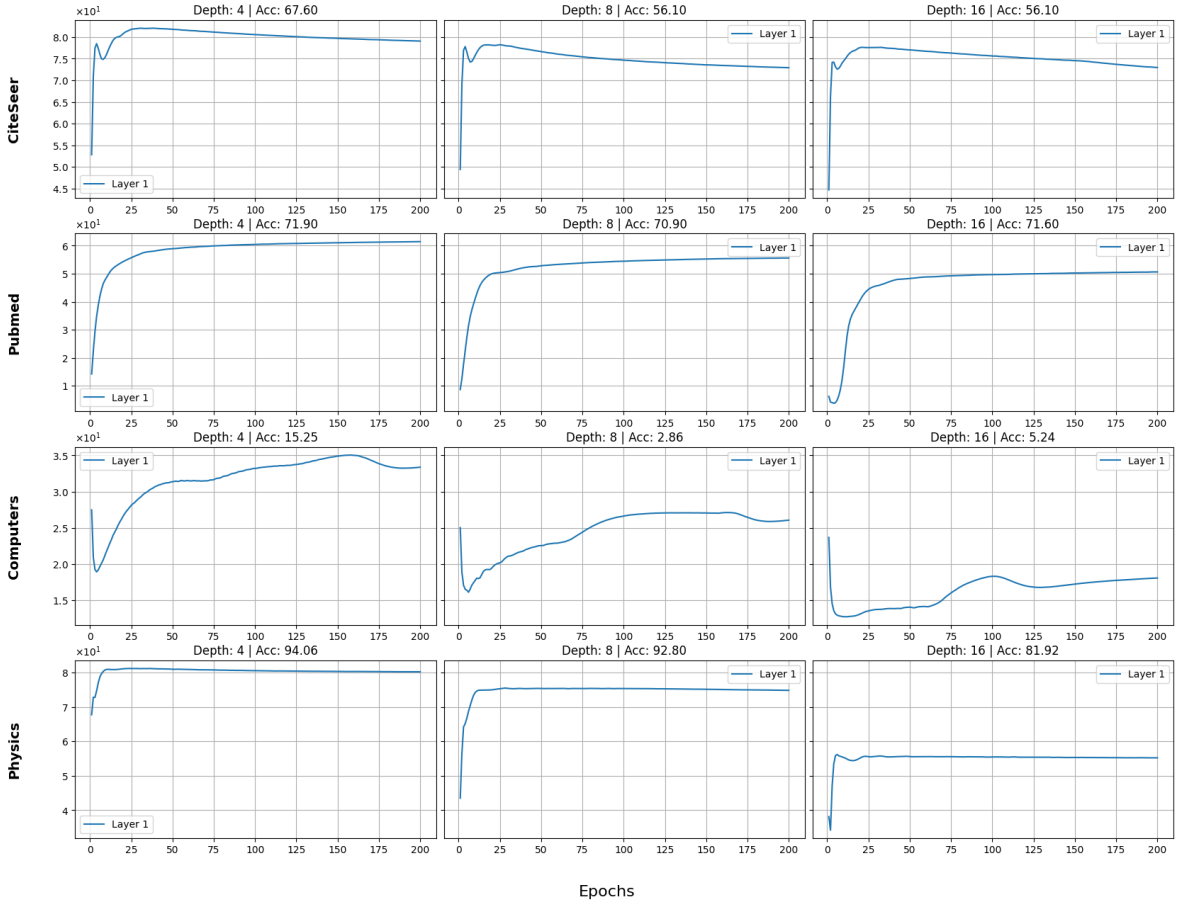

Figure 9.18: Epoch evolution of the average value of the angles between the centroids of the embeddings of the training nodes. We show results for 3 different depths of a SGC model and average norm values in different layers within the model. We show how angles evolve in the first, the middle and the last layer of each model. We also include the accuracy achieved by each model.

4 Regularization plots

Figures 9.19, and 9.20 present the results of ResGCN and SGC on every dataset with and without the proposed regularization.

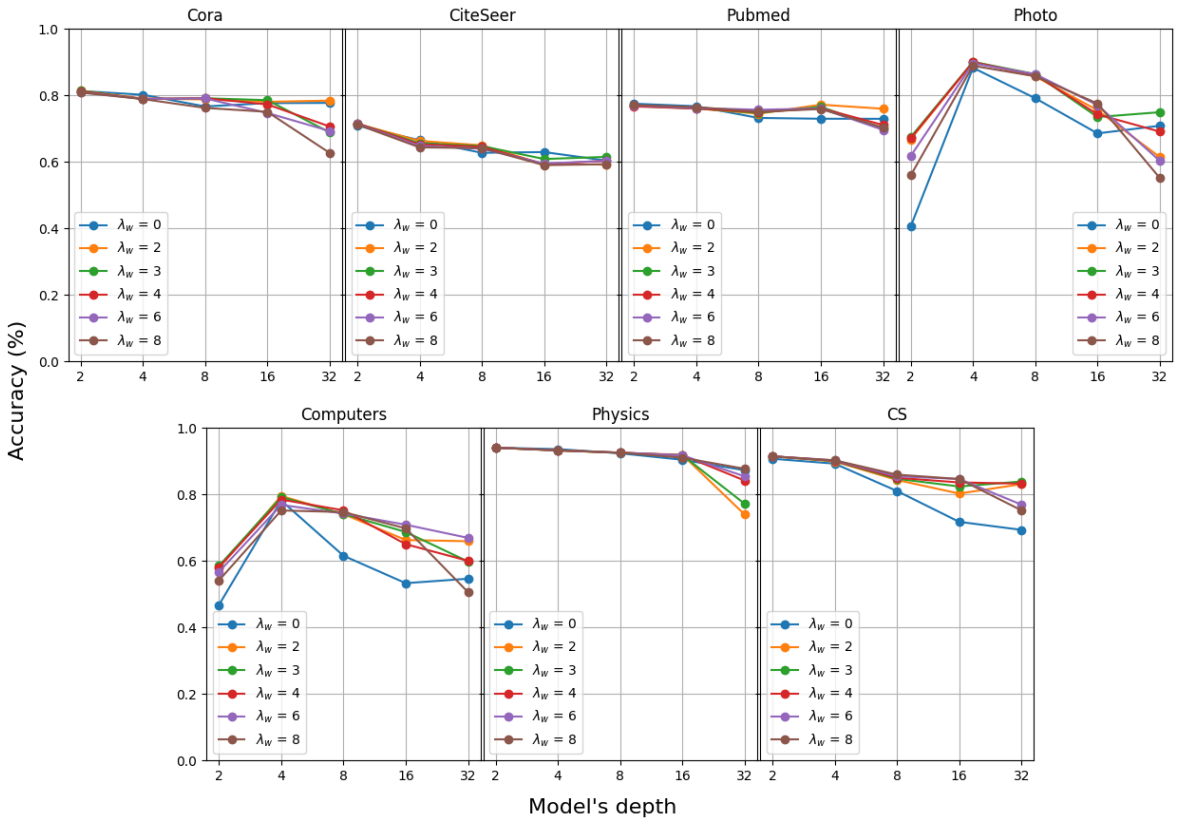

Figure 9.19: Comparison between a ResGCN with and without the proposed regularization across 7 datasets for varying depth. We include results for different values of λ_w.

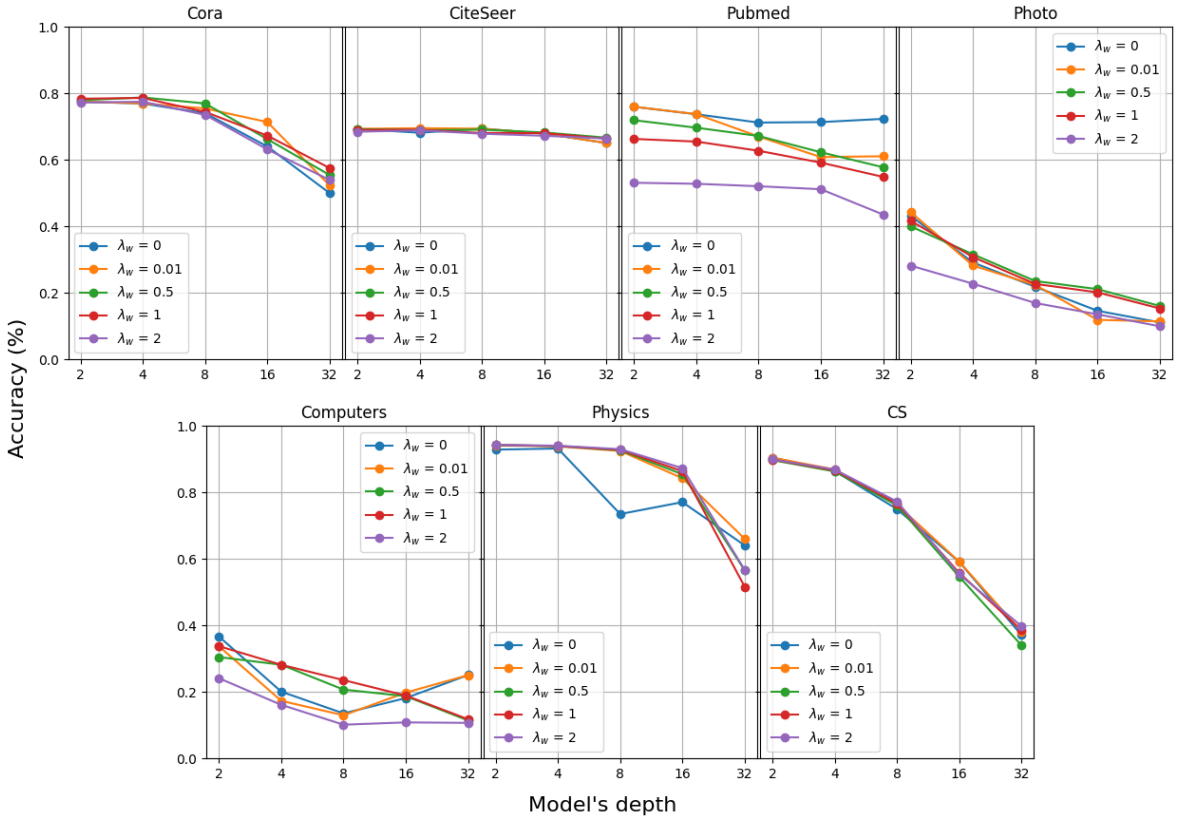

Figure 9.20: Comparison between a SGC with and without the proposed regularization across 7 datasets for varying depth. We include results for different values of λ_w.

5 Plots on variable number of SGC layers

Figure 9.21 presents the performance of SGC models with varying number of layers on the *CiteSeer, Pubmed, Computers,* and *Physics* datasets. For the *Physics* dataset, we could not train a model with 8 SGC layers due to hardware limitations.

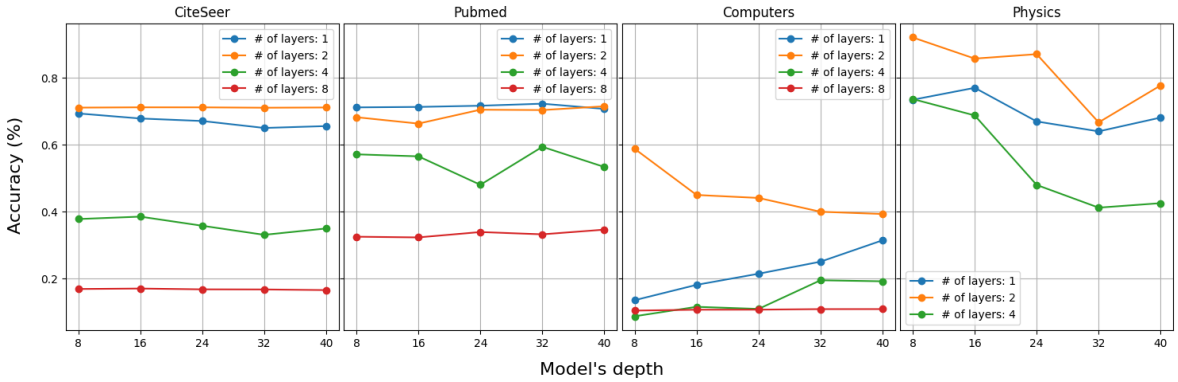

Figure 9.21: Comparison between SGC models that have different number of SGC layers stacked across 4 different datasets for varying depth. In every depth all models have access to the same information. We only vary the number of trainable weight matrices (i.e., the number of SGC layers).

6 Hyperparameters

Table 9.1: Hyperparameter search space used for finding the optimal configuration.

Hyperparameter	Search Space
Learning Rate (lr)	{1e-4, 6e-4, 1e-3, 6e-3, 1e-2, 6e-2}
Hidden Dimension	{64, 128, 256}
Number of Layers	{2, 4, 8, 16, 24, 32, 40}
Weight Decay	{5e-4, 1e-3, 0}
Epochs	{200, 1500, 3000}
λ_w	{0, 0.01, 0.5, 1, 2, 3, 4, 6, 8, 10, 12}

7 Lemma 8 Proof

Lemma 25. *In a power iteration method with convergence rate equal to $(1-\alpha)$, in order to achieve tolerance error at most tol, the number of iterations (L) needed is given by:*

$$L \sim \frac{log_{10}(tol)}{log_{10}(1-\alpha)} \implies tol \sim (1-\alpha)^L.$$

Proof: Let us assume that the power iteration is given by $x_{k+1} = Px_k$, where P is the transition matrix with $\lambda_1(P) = 1$ and $\lambda_2(P) = (1-\alpha)$, as explained in subsection 2.1. Considering that in a power iteration method the convergence rate (cr) is given by:

$$cr = \frac{\lambda_2(P)}{\lambda_1(P)} = \frac{1-\alpha}{1} = 1-\alpha.$$

In a power iteration method, with P as transition matrix and u_i denoting its i-th eigenvector, we can get the following results regarding the convergence of the method:

$$x_0 = c_1u_1 + ... + c_nu_n, \text{ where } c_i\text{'s are constants.}$$

$$x_k = P^kx_0 = c_1\lambda_1^ku_1 + ... + c_n\lambda_n^ku_n$$

$$x_k = c_1\lambda_1^k\left(u_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^k u_2 + ... + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^k u_n\right) \to c_1\lambda_1^ku_1. \tag{9.1}$$

From Equation 9.1 we observe that the slowest convergent factor is the one associated with the second largest eigenvalue. The tolerance error between consecutive iterations is given mainly (approached better) by the slowest convergent to zero i.e., the one associated with the second largest eigenvalue. Hence we get:

$$\left(\frac{\lambda_2}{\lambda_1}\right)^k \sim tol \implies (1-\alpha)^k \sim tol \implies k \sim \frac{log_{10}(tol)}{log_{10}(1-\alpha)}.$$

8 Deriving Equation 4.10

Starting from Equation 4.9, substituting $\beta_l = 1, \forall l$, and disregarding the activation functions leads to:

$$H^{(l+1)} = \left((1-\alpha_l)\hat{A}H^{(l)} + \alpha_l H^{(0)}\right)W^{(l)}.$$

Substituting $H^{(l)}$ once yields:

$$H^{(l+1)} = \Big((1-\alpha_l)\hat{A}\Big(\Big((1-\alpha_{l-1})\hat{A}H^{(l-1)} + \alpha_{l-1}H^{(0)}\Big)W^{(l-1)}\Big) + \alpha_l H^{(0)}\Big)W^{(l)} =$$

$$\hat{A}^2(1-\alpha_l)(1-\alpha_{l-1})H^{(l-1)}W^{(l)}W^{(l-1)} + \hat{A}(1-\alpha_l)\alpha_{l-1}H^{(0)}W^{(l)}W^{(l-1)} + \alpha_l H^{(0)}W^{(l)}.$$

Repeatedly substituting $H^{(k)}$ until k reaches to zero, and writing products of $(1-\alpha_k)$ in the $\prod(\cdot)$ format, leads to the formula presented in Equation 4.10.

9 Training Statistics

Figure 9.22 and Figure 9.23 illustrate the training loss evolution for two variants of the *Star* dataset. When the star path length is short (Figure 9.22), all models effectively learn the underlying task. However, as the length increases (Figure 9.23), residual methods exhibit inferior performance, struggling to integrate distant information from the star's leaves.

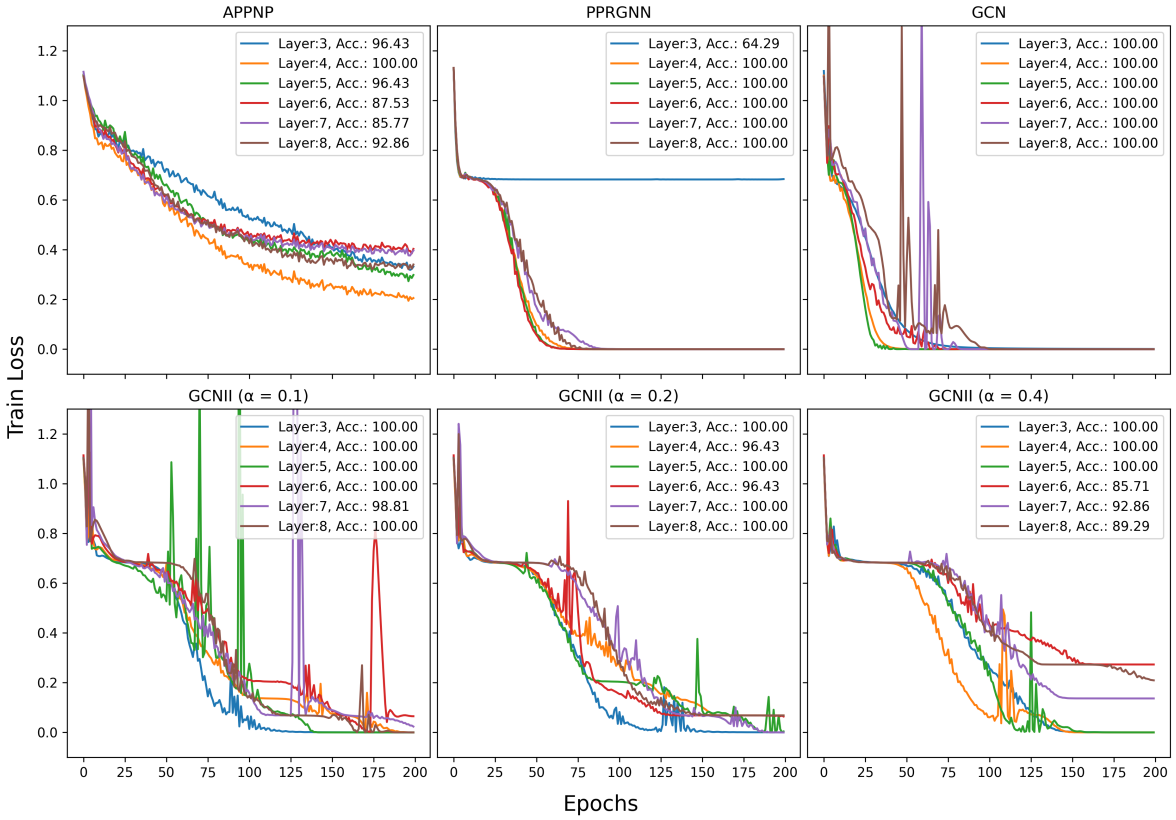

Figure 9.22: Training loss evolution of APPNP, PPRGNN, GCN, and three variants of GCNII on a variant of the *Star* dataset with 3 paths of length 3.

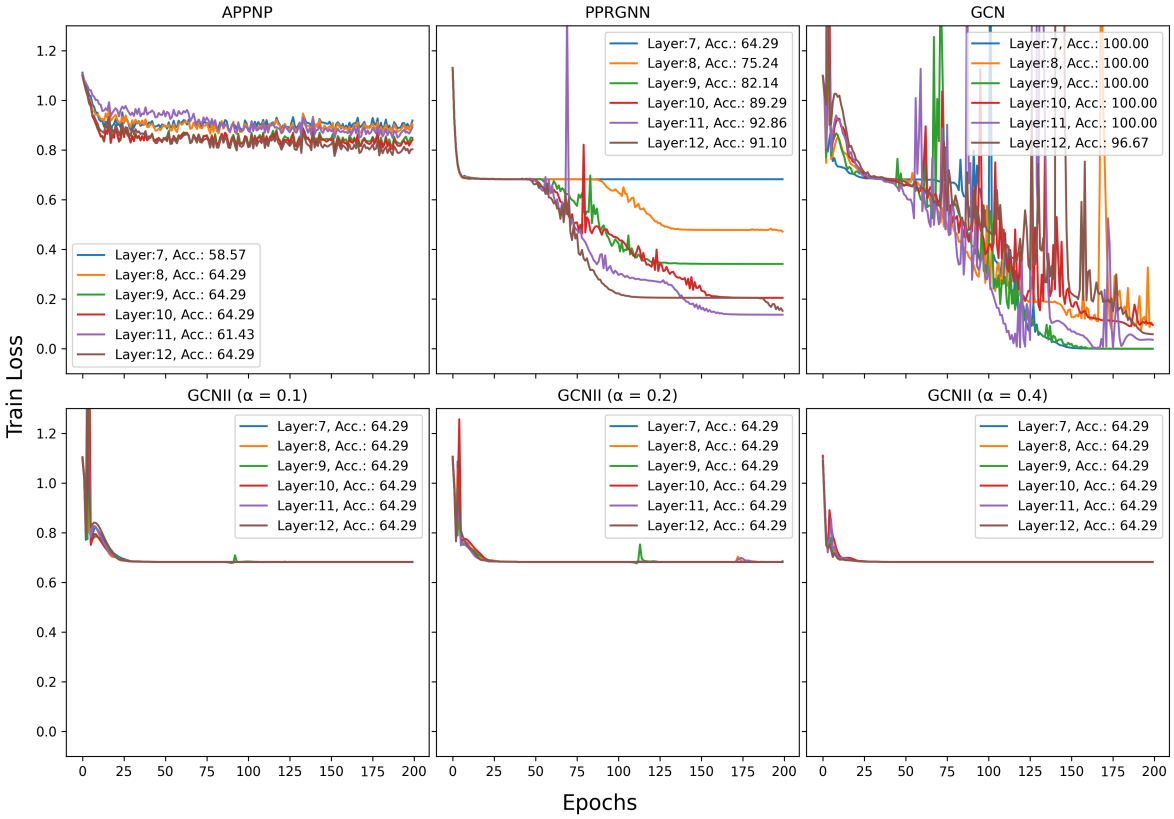

Figure 9.23: Training loss evolution of APPNP, PPRGNN, GCN, and three variants of GCNII on a variant of the *Star* dataset with 3 paths of length 7.

We also compare the models in terms of average training time consumption for two variants of the *Star* dataset. Figure 9.24 shows that for short-range interactions, GCNII is faster. However, as path length increases, other methods require less time and scale more efficiently, maintaining nearly constant runtime. In contrast, GCNII variants require progressively more time to compute the output as path length grows.

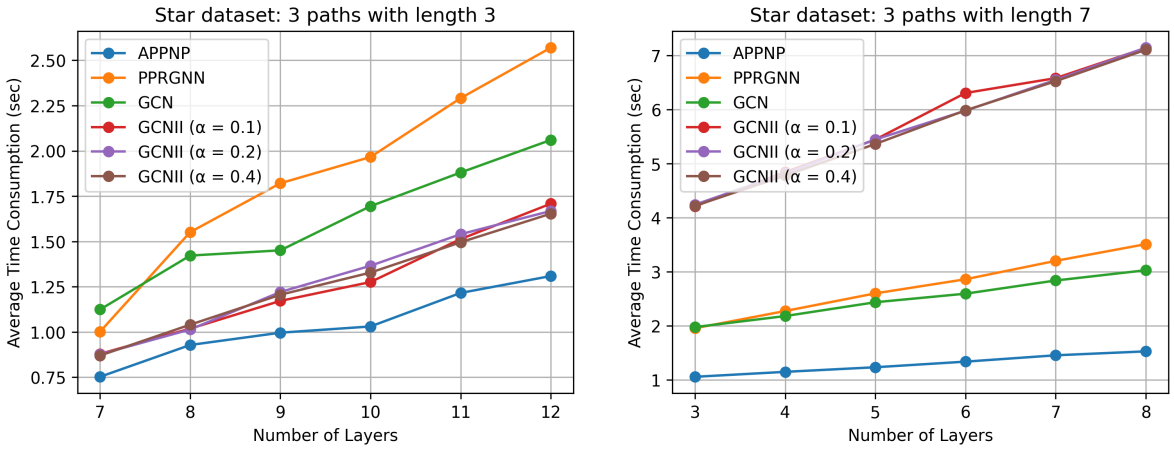

Figure 9.24: Average training time consumption of APPNP, PPRGNN, GCN, and three GCNII variants across two versions of the *Stars* dataset.

10 Statistical Analysis

Tables 9.2 and 9.3 report Friedman and Conover test results. Table 9.2 shows per-dataset Friedman tests (10 runs each), all of which reject the null hypothesis ($p < 0.05$). Holm-adjusted Conover post-hoc tests identify significant classifier pairs. An asterisk "*" denotes that all pairwise differences are significant.
Table 9.3 summarizes the global Friedman test on dataset-wise mean accuracies. The result is not significant, indicating that there is no clear winner across the evaluated datasets.

Table 9.2: Per-dataset Friedman test and Conover post-hoc results (Holm-adjusted). Each row shows a dataset (10 runs): the Friedman χ^2, its p-value, and any significant classifier pairs.

Dataset	χ^2	p_{Friedman}	Significant pairs (Conover)
Cora	19.56	2.1e-4	APPNP-GCNII, APPNP-PPRGNN, APPNP-GCN
CiteSeer	25.49	1.2e-5	*
Pubmed	25.32	1.3e-5	All except of APPNP-PPRGNN pair
Photo	30.00	1.4e-6	*
Computers	23.76	2.8e-5	All except of APPNP-PPRGNN pair
Arxiv	28.92	2.3e-6	*

"*" means all pairs are significant at $\alpha = 0.05$.

Table 9.3: Global Friedman test (on dataset-means) and Conover post-hoc (Holm-adjusted).

Analysis	χ^2	p_{Friedman}	Significant pairs (Conover)
All datasets	4.8	0.18	-

11 Hyperparameters

Table 9.4: Hyperparameter search space used for finding the optimal configuration.

Hyperparameter	Search Space
Learning Rate (lr)	{1e-4, 1e-3, 1e-2, 2e-2, 1e-1}
Hidden Dimension	{32, 64, 128, 256}
Number of Layers	{2, 4, 8, 16, 32, 64}
Weight Decay	{5e-4, 1e-3}

12 Partially trained GCNII

Figure 9.25 presents the comparison of fully-trained GCNII and partially trained GCNII across seven benchmark datasets. Each subplot reports mean test accuracy over 10 ran-

dom seeds, using the original hyperparameter settings from the GCNII authors. Across all datasets, partially trained GCNII performs at least as well as the fully-trained GCNII. This behavior aligns with expectations: GCNII already employs identity mappings and residual connections to resist oversmoothing and achieves near state-of-the-art accuracy, leaving limited room for further improvement. Nonetheless, the consistent positive results confirm that our partial training paradigm can complement even highly optimized architectures. We also conducted a hyperparameter sweep over larger width values on the fully-trained GCNII and observed no significant performance gains, confirming that improvements arise specifically from the use of the partial training mechanism.

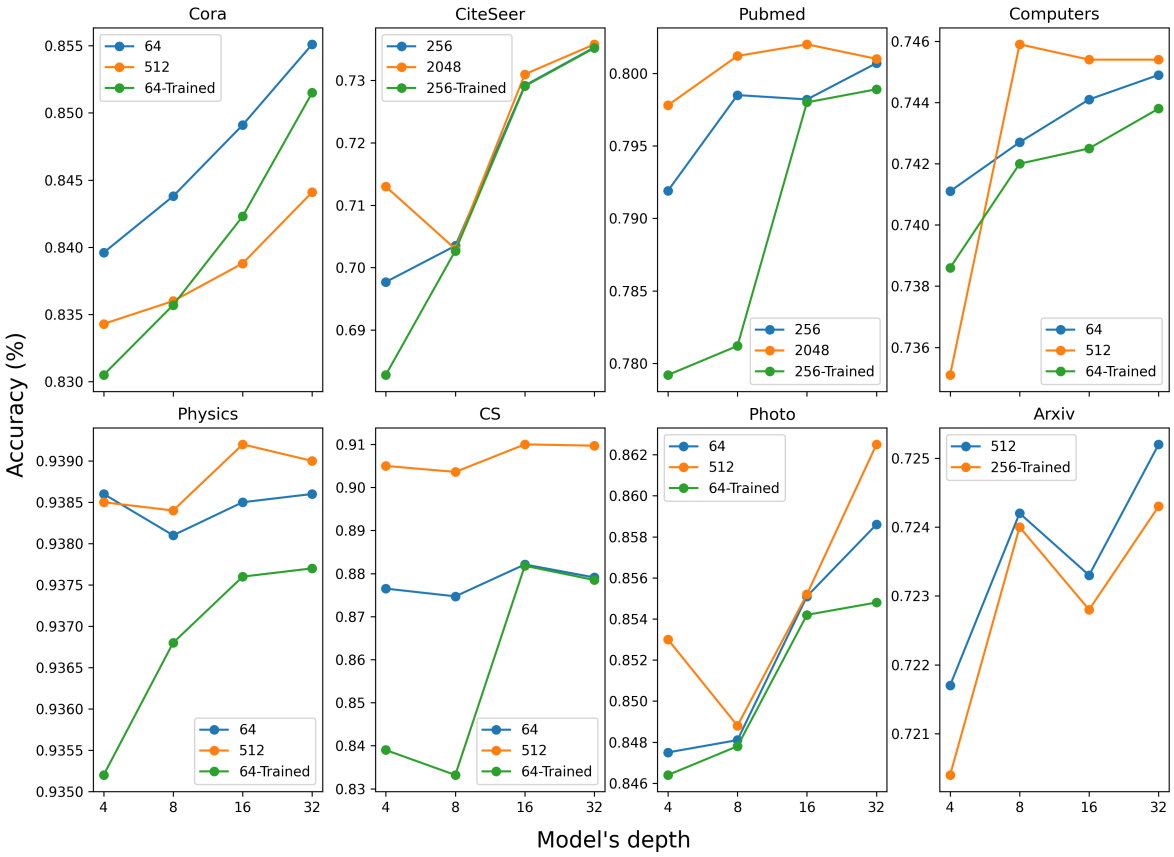

Figure 9.25: Comparison between a fully trained GCNII and different configurations (in terms of width) of partially trained GCNIIs across 8 datasets for varying depth. The trainable layer is always the second. Experiments were limited to the reported widths due to hardware constraints.

13 Extended Table 5.2

We present the extended version of Table 5.2 of the main text.

Table 9.5: Performance comparison of GCN models with different width and depth, as well as different placement of the trainable layer.

Accuracy (%) & std						
Dataset	(Width, Depth)	Position				
		2	4	8	16	32
Cora	(512,8)	**79.65 ± 0.62**	78.07 ± 1.31	73.49 ± 2.28	-	-
	(2048,8)	**81.85 ± 0.34**	81.09 ± 0.74	78.06 ± 1.12	-	-
	(8192,8)	81.25 ± 0.71	**81.47 ± 0.89**	81.04 ± 0.56	-	-
	(512,32)	35.29 ± 12.74	36.60 ± 11.21	38.83 ± 9.03	**39.65 ± 10.99**	33.55 ± 12.14
	(2048,32)	76.90 ± 2.23	77.09 ± 2.05	**77.43 ± 1.88**	77.22 ± 1.95	76.99 ± 2.22
	(8192,32)	78.87 ± 0.45	78.44 ± 0.58	**79.07 ± 0.73**	78.83 ± 0.43	78.97 ± 0.61
CiteSeer	(512, 8)	**67.58 ± 1.25**	62.13 ± 1.18	53.47 ± 1.85	-	-
	(2048, 8)	**71.20 ± 0.49**	67.20 ± 1.18	61.14 ± 1.11	-	-
	(8192, 8)	**70.78 ± 1.45**	69.52 ± 1.08	66.39 ± 0.74	-	-
	(512, 32)	34.35 ± 12.25	32.40 ± 9.81	29.01 ± 9.86	27.72 ± 11.78	**30.12 ± 10.94**
	(2048, 32)	57.36 ± 6.46	55.76 ± 5.72	57.22 ± 2.87	56.72 ± 3.84	**58.69 ± 3.80**
	(8192, 32)	**70.18 ± 0.64**	68.21 ± 0.87	65.45 ± 1.06	65.45 ± 0.99	65.99 ± 1.09
Pubmed	(512, 8)	**78.07 ± 0.85**	76.99 ± 0.88	72.68 ± 2.19	-	-
	(2048, 8)	77.18 ± 0.66	**77.53 ± 0.52**	76.72 ± 0.66	-	-
	(8192, 8)	74.05 ± 2.67	76.20 ± 1.43	**77.75 ± 0.67**	-	-
	(512, 32)	66.91 ± 14.76	68.56 ± 11.22	**68.99 ± 11.79**	68.32 ± 13.51	65.56 ± 18.07
	(2048, 32)	**78.55 ± 0.40**	78.22 ± 0.48	78.06 ± 0.61	77.96 ± 0.93	78.24 ± 0.77
	(8192, 32)	**78.63 ± 0.44**	78.57 ± 0.51	78.53 ± 0.45	78.56 ± 0.57	78.57 ± 0.45
Photo	(512, 8)	**84.53 ± 1.32**	83.54 ± 0.74	66.85 ± 5.44	-	-
	(2048, 8)	**89.68 ± 0.12**	89.41 ± 0.18	82.59 ± 0.36	-	-
	(8192, 8)	**90.26 ± 0.21**	89.82 ± 0.22	87.73 ± 0.18	-	-
	(512, 32)	22.69 ± 15.00	22.15 ± 11.70	24.42 ± 15.12	**25.68 ± 15.29**	21.61 ± 12.61
	(2048, 32)	41.12 ± 17.04	44.96 ± 15.92	50.78 ± 13.71	**51.76 ± 13.70**	50.19 ± 13.14
	(8192, 32)	**72.32 ± 2.93**	71.39 ± 2.04	70.58 ± 1.27	70.66 ± 1.95	71.62 ± 1.80
Computers	(512, 8)	**64.74 ± 2.14**	63.81 ± 1.57	59.70 ± 1.03	-	-
	(2048, 8)	**73.10 ± 0.26**	72.22 ± 0.22	64.77 ± 0.71	-	-
	(4096, 8)	**74.80 ± 0.00**	73.15 ± 0.00	66.52 ± 0.00	-	-
	(512, 32)	8.43 ± 5.18	8.83 ± 5.24	**12.71 ± 11.89**	10.82 ± 5.48	8.33 ± 5.21
	(2048, 32)	45.74 ± 19.44	48.15 ± 19.88	49.38 ± 19.14	50.85 ± 17.11	**50.97 ± 17.32**
	(4096, 32)	65.00 ± 0.00	**67.62 ± 0.00**	65.17 ± 0.00	66.28 ± 0.00	64.84 ± 0.00
Physics	(512, 8)	**92.88 ± 0.14**	92.68 ± 0.11	90.97 ± 0.71	-	-
	(2048, 8)	92.51 ± 0.06	92.47 ± 0.12	**92.66 ± 0.17**	-	-
	(8192, 8)	92.46 ± 0.06	92.41 ± 0.09	**92.65 ± 0.13**	-	-
	(512, 32)	39.50 ± 27.73	37.52 ± 26.90	**40.04 ± 27.52**	38.29 ± 26.26	33.93 ± 26.08
	(2048, 32)	81.54 ± 24.21	80.75 ± 24.78	**86.35 ± 14.05**	85.72 ± 13.94	86.09 ± 13.89
	(8192, 32)	92.47 ± 0.18	**92.53 ± 0.16**	92.44 ± 0.09	92.40 ± 0.10	92.43 ± 0.12
CS	(512, 8)	**87.16 ± 1.33**	86.83 ± 1.39	79.82 ± 1.76	-	-
	(2048, 8)	88.34 ± 0.09	**88.45 ± 0.16**	88.20 ± 0.30	-	-
	(8192, 8)	87.17 ± 0.09	87.45 ± 0.08	**88.93 ± 0.12**	-	-
	(512, 32)	7.17 ± 5.28	**9.04 ± 7.78**	5.92 ± 3.92	5.99 ± 3.75	7.30 ± 3.53
	(2048, 32)	36.07 ± 14.23	38.16 ± 10.80	41.19 ± 11.22	39.35 ± 9.42	**41.39 ± 11.59**
	(8192, 32)	**82.96 ± 0.92**	82.32 ± 1.53	82.62 ± 0.30	82.56 ± 0.36	82.37 ± 0.43

14 Main Text Formulas

The generic GNN equation as defined in main text is:

$$H^{(l+1)} = \sigma\big(\big(\alpha\hat{A}H^{(l)} + \beta H^{(0)} + \gamma H^{(l-1)}\big) \times \big(\delta W^{(l)} + \epsilon I\big)\big), \tag{9.2}$$

where $\alpha, \beta, \gamma, \delta$ and ϵ are preselected parameters that determine the convolutional GNN architecture.

The variance of the gradient flowing backwards is defined as:

$$Var\left[\Delta x_l\right] = n_l Var[w_l'] Var\left[\Delta y_l^{(i)}\right] = \frac{1}{2} n_l \left(\delta^2 Var[w_l] + \epsilon^2\right) Var\left[\Delta x_{l+1}^{(i)'}\right]. \tag{9.3}$$

Theorem 16 proof

Theorem 26. *The upper bound of the variance of the signals flowing forward in a generic GNN defined by Equation 9.2 is:*

$$Var[y_l^{(i)}] \leq n_l \cdot (d_i + \mathbb{1}(\beta \neq 0) + \mathbb{1}(\gamma \neq 0)) \cdot \left(\frac{\alpha^2}{2d_i^2} Var[y_{l-1}^{(i)}] + \frac{\gamma^2}{2} \cdot Var[y_{l-2}^{(i)}] + j(\alpha, \beta)\right) \cdot \left(\delta^2 Var[w_l] + \epsilon^2\right), \tag{9.4}$$

where n_l is the weight matrix dimension, d_i is the degree of node i, $\mathbb{1}(\cdot)$ is the indicator function $\alpha, \beta, \gamma, \delta$ and ϵ are constants depending on the underlying architecture of the model, and $j(\alpha, \beta)$ is defined in Lemma 3.

Proof: We let the initialized elements in $W^{(l)}$ be mutually independent and have the same distribution. We also assume that elements in $x_l^{(i)}$ are also mutually independent and have the same distribution and finally we assume that $x_l^{(i)}$ and $W^{(l)}$ are independent of each other. Following a similar analysis as the one presented in [15] we have:

$$Var[y_l^{(i)}] = n_l Var[\delta w_l x_l^{(i)'} + \epsilon x_l^{(i)'}], \tag{9.5}$$

where $y_l^{(i)}, x_l^{(i)'}$ and w_l represent the random variables of each element in the respective matrices. We let w_l have zero mean, hence the variance of the product is:

$$Var[y_l^{(i)}] = n_l \left(Var\left[\delta w_l x_l^{(i)'}\right] + Var\left[\epsilon x_l^{(i)'}\right] + 2Cov(\delta w_l x_l^{(i)'}, \epsilon x_l^{(i)'})\right) = \tag{9.6}$$

$$n_l \left(\delta^2 Var[w_l] E\left[\left(x_l^{(i)'}\right)^2\right] + \epsilon^2 Var\left[x_l^{(i)'}\right]\right) \leq n_l \cdot E\left[\left(x_l^{(i)'}\right)^2\right] \left(\delta^2 Var[w_l] + \epsilon^2\right). \tag{9.7}$$

The special form of $x_l^{(i)'}$ is not directly related to $y_{l-1}^{(i)}$. In order to bridge that gap we will proceed our analysis with inequalities resulted by the usage of the Cauchy–Schwarz–Bunyakovsky inequality (CSB).

Lemma 27.

$$E\left[\left(x_l^{(i)\prime}\right)^2\right] = E\left[\left(\frac{\alpha}{d_i}\sum_{j\in\hat{N}(i)} x_l^{(j)T} + \beta\cdot x_{(0)}^{(i)} + \gamma\cdot x_{(l-1)}^{(i)}\right)^2\right] \overset{CSB}{\leq}$$

$$(d_i + \mathbb{1}(\beta\neq 0) + \mathbb{1}(\gamma\neq 0))\times$$

$$\left(\frac{\alpha^2}{d_i^2}E\left[\left(x_l^{(i)}\right)^2\right] + \frac{\alpha^2}{d_i^2}\sum_{j\in N(i)}\left(x_l^{(j)T}\right)^2 + \beta^2\cdot E\left[\left(x_0^{(i)}\right)^2\right] + \gamma^2\cdot E\left[\left(x_{l-1}^{(i)}\right)^2\right]\right)$$

$$= (d_i + \mathbb{1}(\beta\neq 0) + \mathbb{1}(\gamma\neq 0))\left(\frac{\alpha^2}{d_i^2}E\left[\left(x_l^{(i)}\right)^2\right] + \gamma^2\cdot E\left[\left(x_{l-1}^{(i)}\right)^2\right] + j(\alpha,\beta)\right),$$

where $k_l^{(i)} = \sum_{j\in N(i)}\left(x_l^{(j)T}\right)^2$, *i.e. sum of neighbors representations except of self representation and* $j(\alpha,\beta) = \frac{\alpha^2\cdot k_l^{(i)}}{d_i^2} + \beta^2\cdot E\left[(\,x_0^{(i)}\,)^2\right]$.

Using Lemma 27, Equation 9.6 transforms to:

$$Var[y_l^{(i)}] \leq n_l\cdot(d_i + \mathbb{1}(\beta\neq 0) + \mathbb{1}(\gamma\neq 0))\times$$

$$\left(\frac{\alpha^2}{d_i^2}E\left[\left(x_l^{(i)}\right)^2\right] + \gamma^2\cdot E\left[\left(x_{l-1}^{(i)}\right)^2\right] + j(\alpha,\beta)\right)\cdot\left(\delta^2 Var[w_l] + \epsilon^2\right). \tag{9.8}$$

We have set $x_l^{(i)\prime} = \frac{\alpha}{d_i}\sum_{j\in\hat{N}(i)} x_l^{(j)T} + \beta\cdot x_{(0)}^{(i)} + \gamma\cdot x_{(l-1)}^{(i)}$ and $W_l' = \delta W_l + \epsilon I$. Letting w_{l-1}' (element of W_l' matrix) have a symmetric distribution around zero and $b_{l-1} = 0$, then y_{l-1} has zero mean and symmetric distribution around zero. In order to have symmetric distribution of w_{l-1}' around zero we should have a symmetric distribution of w_{l-1} around $-\epsilon$, based on the relationship between W_l and W_l'.

This leads to $E\left[\left(x_l^{(i)}\right)^2\right] = \frac{1}{2}Var[y_{l-1}]$, when activation function is ReLU. Applying that result to Inequality 9.8 yields:

$$Var[y_l^{(i)}] \leq n_l\cdot(d_i + \mathbb{1}(\beta\neq 0) + \mathbb{1}(\gamma\neq 0))\times$$

$$\left(\frac{\alpha^2}{2d_i^2}Var[y_{l-1}^{(i)}] + \frac{\gamma^2}{2}\cdot Var[y_{l-2}^{(i)}] + j(\alpha,\beta)\right)\cdot\left(\delta^2 Var[w_l] + \epsilon^2\right). \tag{9.9}$$

In a similar spirit, we also derive the lower bound for the variance. Considering that $x_l^{(i)} > 0$ for all l, i we have the following lemma:

Lemma 28.

$$E\left[\left(x_l^{(i)\prime}\right)^2\right] = E\left[\left(\frac{\alpha}{d_i}\sum_{j\in\hat{N}(i)} x_l^{(j)T} + \beta\cdot x_{(0)}^{(i)} + \gamma\cdot x_{(l-1)}^{(i)}\right)^2\right] \geq$$

$$\frac{\alpha^2}{d_i^2}E\left[\left(x_l^{(i)}\right)^2\right] + \gamma^2\cdot E\left[\left(x_{l-1}^{(i)}\right)^2\right] + j(\alpha,\beta).$$

This comes as a consequence of the fact that $\left(\sum_j m_j\right)^2 \geq \sum_j m_j^2$, *for all* $m_j \geq 0$.

Consequently, the lower bound of $Var\left[y_l^{(i)}\right]$ is given by:

$$Var[y_l^{(i)}] \geq n_l \left(\frac{\alpha^2}{2d_i^2} Var[y_{l-1}^{(i)}] + \frac{\gamma^2}{2} \cdot Var[y_{l-2}^{(i)}] + j(\alpha, \beta) \right) \cdot \left(\delta^2 Var[w_l] + \epsilon^2 \right). \quad (9.10)$$

15 Theorem 18 proof

Lemma 29.

$$Var\left[\Delta x_{l+1}^{(i)'}\right] = E\left[\left(\Delta x_{l+1}^{(i)'}\right)^2\right] = E\left[\left(\frac{\alpha}{d_i} \sum_{j \in \hat{N}(i)} \Delta x_{l+1}^{(j)T} + \gamma \cdot \Delta x_{(l)}^{(i)T}\right)^2\right]$$

$$\overset{CSB}{\leq} (d_i + \mathbb{1}(\gamma \neq 0)) \cdot \left(\frac{\alpha^2}{d_i^2} E\left[\left(\Delta x_{l+1}^{(i)}\right)^2\right] + \gamma^2 E\left[\left(\Delta x_l^{(i)}\right)^2\right] + q(\alpha) \right),$$

where $q(\alpha) = \frac{\alpha^2}{d_i^2} \cdot o_{l+1}^{(i)}$, *and* $o_{l+1}^{(i)} = \sum_{j \in N(i)} \left(\Delta x_{l+1}^{(j)T}\right)^2$, *i.e., the sum of gradients originating from the neighbors of the node, excluding self-originating gradient and* $\mathbb{1}(\cdot)$ *is the indicator function.*

Theorem 30. *The upper bound of the variance of the gradients flowing backward in a generic GNN defined by Equation 9.2 is:*

$$Var\left[\Delta x_l^{(i)}\right] \leq \frac{m_w}{1 - \gamma^2 m_w} \cdot \left(\frac{\alpha^2}{d_i^2} Var\left[\Delta x_{l+1}^{(i)}\right] + q(\alpha) \right), \quad (9.11)$$

with

$$m_w = \frac{1}{2} n_l \left(d_i + \mathbb{1}(\gamma \neq 0)\right) \cdot \left(\delta^2 Var[w_l] + \epsilon^2 \right),$$

where n_l *is the weight matrix dimension,* d_i *is the degree of node* i, $\mathbb{1}(\cdot)$ *is the indicator function* α, γ, δ *and* ϵ *are constants depending on the underlying architecture of the model, and* $q(\alpha)$ *is defined in Lemma 29.*

Proof: Using the fact that $E\left[\left(\Delta x_l^{(i)}\right)^2\right] = Var\left[\Delta x_l^{(i)}\right]$ and Lemma 29 to Equation 9.3 we get:

$$Var\left[\Delta x_l^{(i)}\right] = \frac{1}{2} n_l \left(\delta^2 Var[w_l] + \epsilon^2 \right) Var\left[\Delta x_{l+1}^{(i)'}\right] \leq \frac{1}{2} n_l \left(\delta^2 Var[w_l] + \epsilon^2 \right) (d_i + \mathbb{1}(\gamma \neq 0)) \cdot$$

$$\left(\frac{\alpha^2}{d_i^2} Var\left[\Delta x_{l+1}^{(i)}\right] + \gamma^2 Var\left[\Delta x_l^{(i)}\right] + q(\alpha) \right) =$$

$$m_w \cdot \left(\frac{\alpha^2}{d_i^2} Var\left[\Delta x_{l+1}^{(i)}\right] + \gamma^2 Var\left[\Delta x_l^{(i)}\right] + q(\alpha) \right) \implies$$

$$Var\left[\Delta x_l^{(i)}\right](1-\gamma^2 m_w) \leq m_w \cdot \left(\frac{\alpha^2}{d_i^2} Var\left[\Delta x_{l+1}^{(i)}\right] + q(\alpha)\right) \implies$$

$$Var\left[\Delta x_l^{(i)}\right] \leq \frac{m_w}{1-\gamma^2 m_w} \cdot \left(\frac{\alpha^2}{d_i^2} Var\left[\Delta x_{l+1}^{(i)}\right] + q(\alpha)\right).$$

In the final inequality we have assumed that $m_w < \gamma^{-2}$. If this condition is not satisfied, then the last inequality has the inverse direction and establishes the lower bound for the variance of the gradients.

Following a similar approach as in Appendix 14, we derive the lower bound of the variance of the gradients.

Lemma 31.

$$Var\left[\Delta x_{l+1}^{(i)'}\right] = E\left[\left(\Delta x_{l+1}^{(i)'}\right)^2\right] = E\left[\left(\frac{\alpha}{d_i}\sum_{j\in\hat{N}(i)} \Delta x_{l+1}^{(j)T} + \gamma \cdot \Delta x_{(l)}^{(i)T}\right)^2\right] \geq$$

$$\frac{\alpha^2}{d_i^2} E\left[\left(\Delta x_{l+1}^{(i)}\right)^2\right] + \gamma^2 E\left[\left(\Delta x_l^{(i)}\right)^2\right] + q(\alpha).$$

This comes as a consequence of the fact that $\left(\sum_j m_j\right)^2 \geq \sum_j m_j^2$, *for all* $m_j \geq 0$.

Consequently, the lower bound of $Var\left[\Delta x_l^{(i)}\right]$ is given by:

$$Var\left[\Delta x_l^{(i)}\right] \geq \frac{m_w}{(d_i + \mathbb{1}(\gamma \neq 0))}\left(\frac{\alpha^2}{d_i^2} Var\left[\Delta x_{l+1}^{(i)}\right] + \gamma^2 Var\left[\Delta x_l^{(i)}\right] + q(\alpha)\right) \implies$$

$$Var\left[\Delta x_l^{(i)}\right] \geq \frac{m'_w}{1-\gamma^2 m'_w} \cdot \left(\frac{\alpha^2}{d_i^2} Var\left[\Delta x_{l+1}^{(i)}\right] + q(\alpha)\right),$$

where $m'_w = m_w/(d_i + \mathbb{1}(\gamma \neq 0))$ and once again we assume that $m'_w < \gamma^{-2}$. If this condition is not satisfied, then the last inequality has the inverse direction and establishes the upper bound for the variance of the gradients.

16 Extended analysis about the "cold start" problem

Oversmoothing primarily affects deep GNNs, raising the question of the necessity of deep architectures. Many benchmark datasets in the literature do not require deep networks due to the homophilic nature of the data. Homophily implies that the valuable information for the majority of the graph nodes is typically within close neighbors (2 or 3 hops away). One particular scenario where deeper architectures might be beneficial is the "cold start" problem, where the majority of node features are missing, resembling to a recommender system encountering a new product or user. In this context, deeper GNNs may recover features from more distant nodes to generate informative representations. The "cold start" datasets that we use in these experiments are generated by removing feature vectors from unlabeled nodes and replacing them with all-zero vectors.

We present an extended version of Table 6.1:

Table 9.6: Comparison of different initialization methods on the "cold start" problem. We show accuracy percentage (%) for the test set. Only the features of the nodes in the training set are available to the model. We also show at what depth (i.e. # Layers) each model achieves its best accuracy, for both GCN and GAT.

Dataset	Method	GCN		GAT	
		Accuracy (%) & std	#L	Accuracy (%) & std	#L
Cora	Xavier Normal	64.86 ± 0.7	4	59.47 ± 1.0	3
	Xavier Uniform	62.52 ± 3.9	4	59.63 ± 1.3	3
	Kaiming Normal	68.35 ± 1.9	6	60.73 ± 4.5	4
	Kaiming Uniform	62.78 ± 5.1	6	57.36 ± 5.3	4
	VIRGO	73.01 ± 1.0	26	70.68 ± 1.5	18
	G-Init	**74.04** ± 1.7	25	**72.34** ± 2.0	27
CiteSeer	Xavier Normal	41.95 ± 0.2	4	39.75 ± 6.3	4
	Xavier Uniform	40.17 ± 5.0	5	36.63 ± 0.6	3
	Kaiming Normal	44.62 ± 1.8	6	37.09 ± 9.1	5
	Kaiming Uniform	44.20 ± 2.5	7	41.68 ± 1.2	4
	VIRGO	49.18 ± 1.4	18	41.01 ± 9.5	13
	G-Init	**49.75** ± 0.7	27	**49.31** ± 0.4	30
Pubmed	Xavier Normal	64.54 ± 0.9	4	59.76 ± 1.5	4
	Xavier Uniform	62.83 ± 2.8	4	46.91 ± 6.5	4
	Kaiming Normal	68.48 ± 1.5	6	60.25 ± 2.7	4
	Kaiming Uniform	65.93 ± 5.7	5	56.93 ± 5.2	4
	VIRGO	71.55 ± 1.5	14	63.06 ± 9.8	14
	G-Init	**71.65** ± 1.8	23	**72.24** ± 1.1	32
Physics	Xavier Normal	94.00 ± 0.1	2	93.26 ± 0.1	1
	Xavier Uniform	93.98 ± 0.0	1	92.9 ± 0.1	2
	Kaiming Normal	**94.00** ± 0.0	2	93.45 ± 0.0	1
	Kaiming Uniform	93.98 ± 0.0	1	92.68 ± 0.1	2
	VIRGO	82.34 ± 6.1	8	85.58 ± 4.3	5
	G-Init	93.99 ± 0.0	1	**93.62** ± 0.1	2
CS	Xavier Normal	89.95 ± 0.2	1	90.78 ± 0.1	1
	Xavier Uniform	90.17 ± 0.4	2	90.71 ± 0.1	1
	Kaiming Normal	90.13 ± 0.3	3	90.77 ± 0.1	1
	Kaiming Uniform	90.22 ± 0.4	2	90.71 ± 0.1	1
	VIRGO	71.28 ± 1.9	6	76.85 ± 3.7	6
	G-Init	**90.28** ± 0.2	3	**90.82** ± 0.1	3
Photo	Xavier Normal	83.19 ± 5.1	5	85.58 ± 0.8	4
	Xavier Uniform	84.50 ± 1.4	3	85.02 ± 0.7	3
	Kaiming Normal	86.53 ± 0.6	5	**86.35** ± 0.9	4
	Kaiming Uniform	87.11 ± 0.6	4	86.29 ± 1.2	4
	VIRGO	83.00 ± 3.5	6	77.60 ± 4.0	6
	G-Init	**87.56** ± 1.2	4	86.07 ± 0.9	4
Computers	Xavier Normal	68.83 ± 6.5	4	70.26 ± 1.7	2
	Xavier Uniform	42.73 ± 6.7	2	70.55 ± 1.4	2
	Kaiming Normal	75.18 ± 3.0	4	74.35 ± 3.5	4
	Kaiming Uniform	72.42 ± 2.5	5	68.87 ± 2.9	3
	VIRGO	75.17 ± 2.7	6	69.91 ± 3.6	7
	G-Init	**78.03** ± 1.0	5	**76.28** ± 1.7	5

17 t-SNE plots

We present t-SNE [142] plots for all datasets using a 32-layer GCN model initialized with the methods investigated in this study. T-SNE results are exhibited for layers 1, 9, 17 and 25. Specifically, we compare the t-SNE plots of a GCN initialized with the proposed G-Init method against the generally second-best performing initialization method, namely, Kaiming Normal. The t-SNE plots validate that our proposed method attains high accuracy by generating meaningful representations that reduce oversmoothing. In contrast, alternative initialization methods lead to the mixing of node embeddings, contributing to a degradation in performance. We observe this difference in the mixing of node representations as depth increases across the majority of datasets.

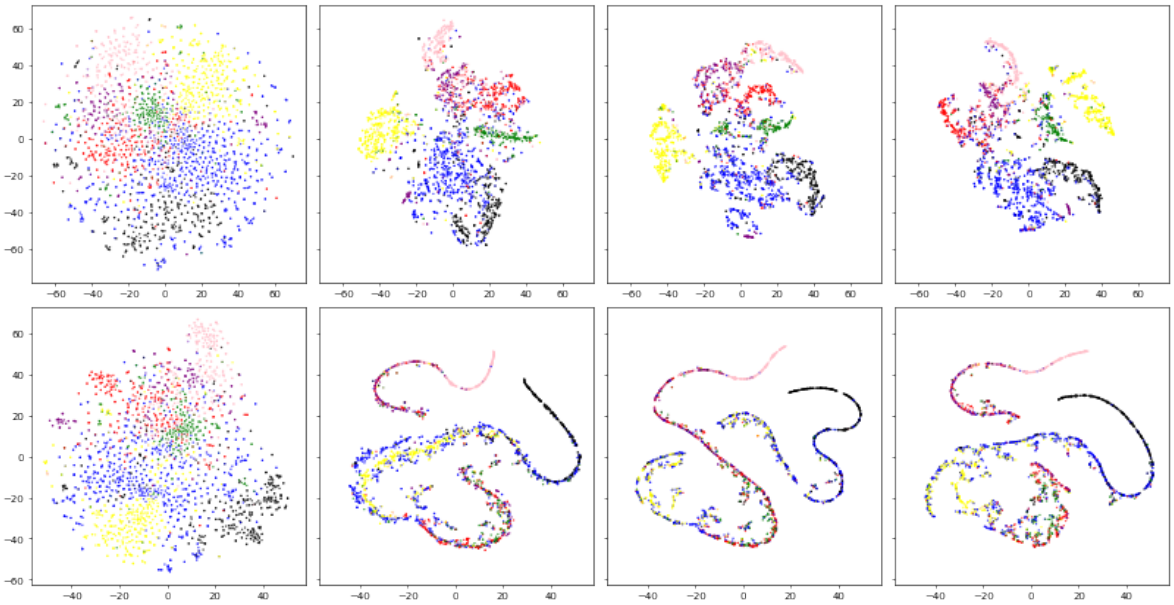

Figure 9.26: T-SNE plot of *Cora* dataset. The upper row presents results for a G-Init initialized 32-layer GCN, while the lower row showcases results for a Kaiming Normal initialized 32-layer GCN.

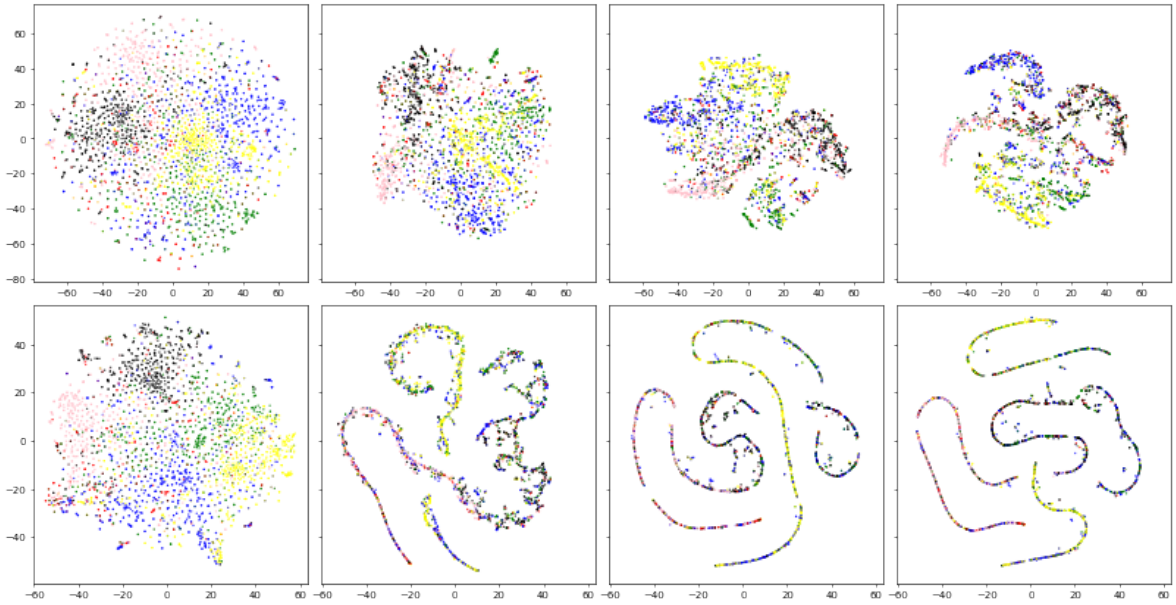

Figure 9.27: T-SNE plot of *Citeseer* dataset. The upper row presents results for a G-Init initialized 32-layer GCN, while the lower row showcases results for a Kaiming Normal initialized 32-layer GCN.

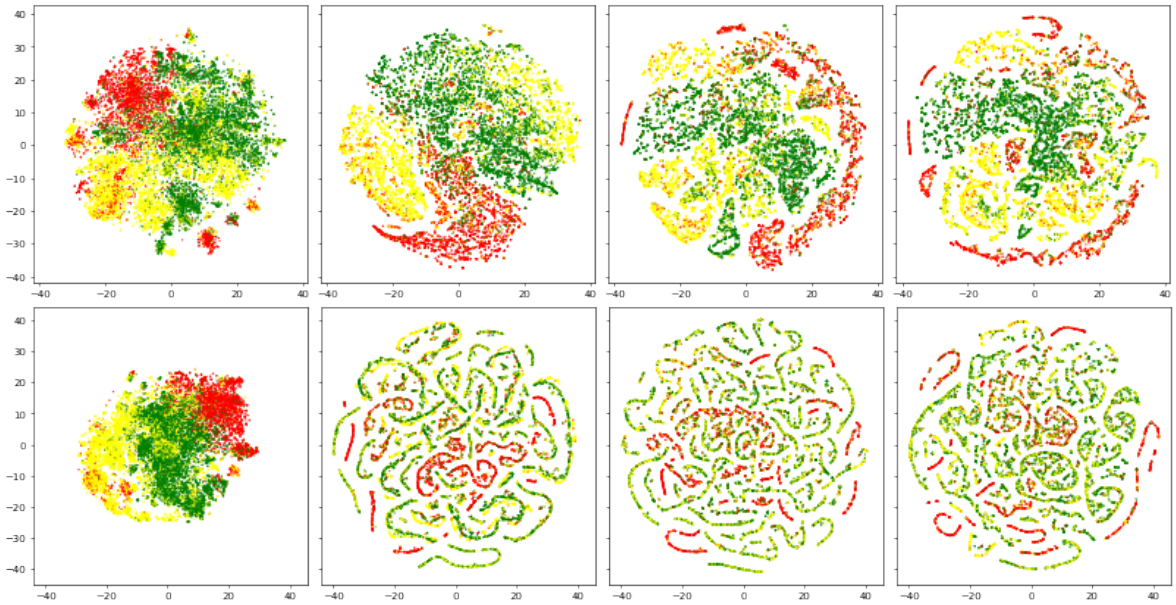

Figure 9.28: T-SNE plot of *Pubmed* dataset. The upper row presents results for a G-Init initialized 32-layer GCN, while the lower row showcases results for a Kaiming Normal initialized 32-layer GCN.

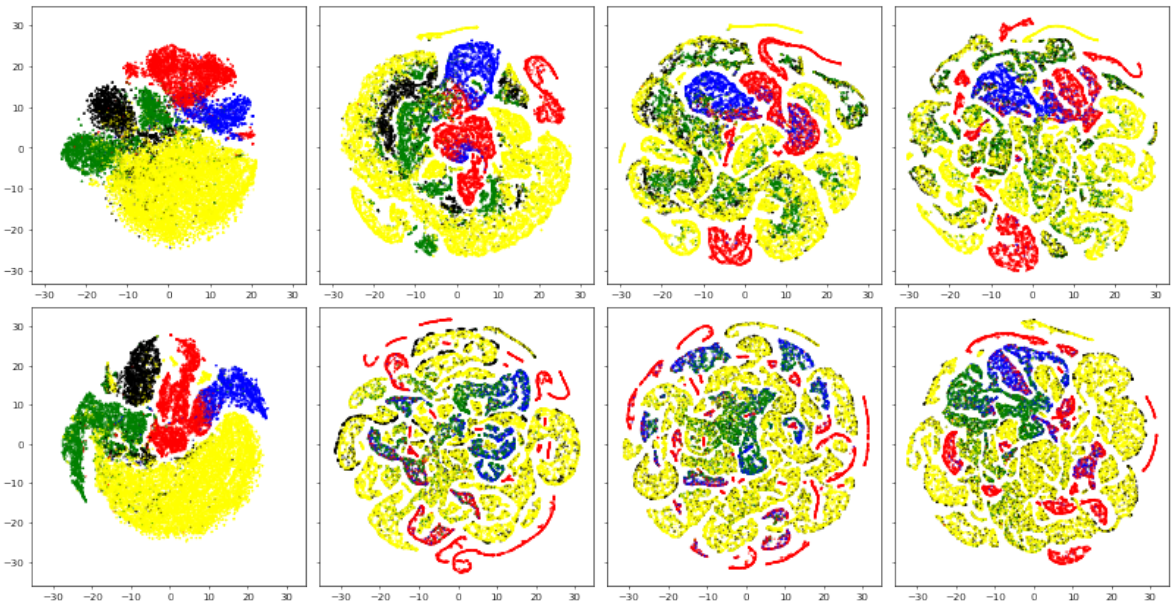

Figure 9.29: T-SNE plot of *Physics* dataset. The upper row presents results for a G-Init initialized 32-layer GCN, while the lower row showcases results for a Kaiming Normal initialized 32-layer GCN.

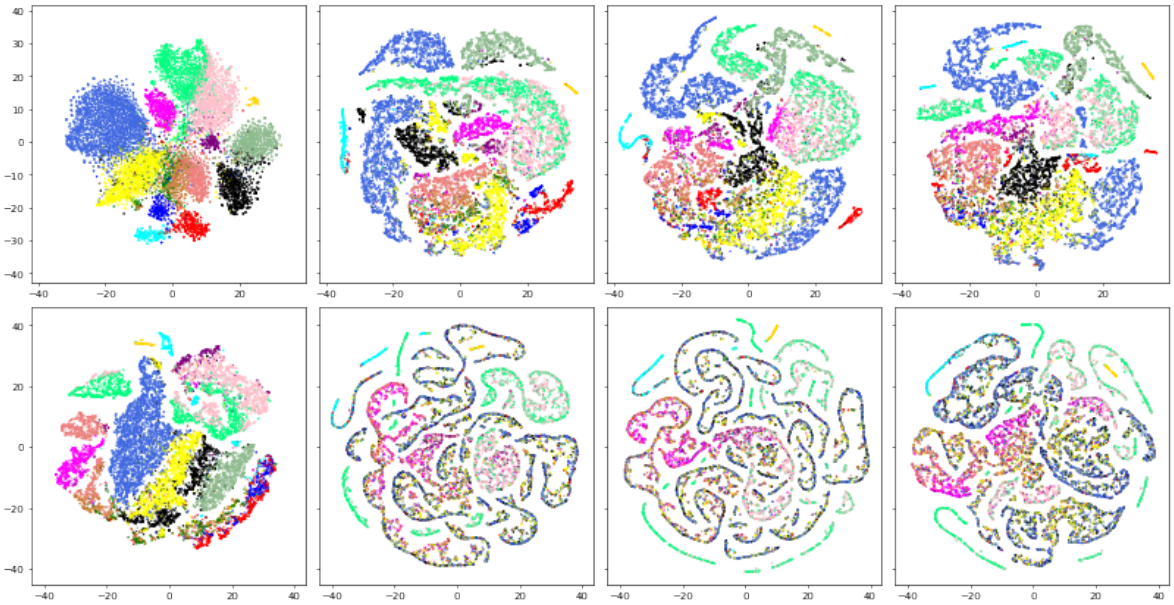

Figure 9.30: T-SNE plot of *CS* dataset. The upper row presents results for a G-Init initialized 32-layer GCN, while the lower row showcases results for a Kaiming Normal initialized 32-layer GCN.

18 Hyperparameters

Table 9.7: Hyperparameter search space used for finding the optimal configuration.

Hyperparameter	Search Space
Learning Rate (lr)	{1e-4, 1e-3, 1e-2}
Hidden Dimension	{64, 128, 256}
Number of Layers	{2, 4, 8, 16, 32, 64}
Weight Decay	{5e-4, 1e-3}
Batch Size	{16, 32, 64}

19 Lemma 21 Proof

Lemma 21 *For a network of depth dep the total gradient reaching to the l-th layer* $\left(\textit{i.e., } \frac{dJ}{dw^{(l)}_{old_{i,j}}} \right)$ *in order to update* $W^{(l)}$ *is bounded by:*

$$\frac{dJ}{dw^{(l)}_{old_{i,j}}} \leq \alpha^{(dep-l)} \cdot G_l.$$

Where $\mathcal{J}$ *is the model's loss function (i.e. Cross Entropy),* α *stands for the ReLU slope and* G_l *is the upper bound of gradients of the output of the subsequent layer (l+1) with respect to the weight elements of* $W^{(l)}$.

Proof: Firstly, let us define the gradient of the output of a ReLU function with respect to the input as follows:

$$\frac{d(ReLU(x))}{dx} = \begin{cases} \alpha, & x > 0. \\ 0, & x \leq 0. \end{cases}$$

Let us consider now a function of nested ReLUs, like the neural networks under investigation, i.e. $f(x) = g(\sigma(g(\sigma(..g(\sigma(g(x)))..))))$, where $\sigma(\cdot)$ is the ReLU function applied repeatedly n times, and $g(\cdot)$ is a function that multiplies its input with a value. Then the gradient of $f(\cdot)$ with respect to x (x is a vector) is given as:

$$\frac{d(f(x))}{dx} = \begin{cases} \alpha^n \cdot \frac{d(g(x))}{dx}, & g(x) > 0, g(\sigma(\cdot)) > 0. \\ 0, & \text{otherwise.} \end{cases} \tag{9.12}$$

This formula indicates the relation between slope, layer index, depth and largest singular value (we only care about the largest). Let as use the notion of J for the loss function of the network (i.e. Cross Entropy). The relation comes from the following formulas (for each weight element in the weight matrix):

$$w^{(l)}_{new_{i,j}} = w^{(l)}_{old_{i,j}} + \eta \cdot \frac{dJ}{dw^{(l)}_{old_{i,j}}}.$$

$$\frac{dJ}{dw^{(l)}_{old_{i,j}}} = \alpha^{(depth-l)} \cdot \frac{dO^{(l+1)}}{w^{(l)}_{old_{i,j}}} \text{ or } 0.$$

Where the 0 value comes from Equation 9.12. GNNs' forward pass is similar to $f(\cdot)$, hence performing the backpropagation leads to gradient calculation of the form of Equation 9.12. So the update rule adds (or subtracts) to each weight a big number (contains exponential factor). We prove, that $\frac{dJ}{dw_{old_{i,j}}}$ tends to be closer to zero as the layer index gets smaller, because when the gradient travels backwards the more distant it travels the more probable it is to die. There exist many components (gradients of ReLU functions) in the total gradient so it is probable one of them to be zero and the total gradient to be zero.
Given Assumption 19, we get an upper bound regarding the gradient of the output of the subsequent layer with respect to each weight element of $W^{(l)}$ $\left(\text{i.e., } \frac{dO^{(l+1)}}{w^{(l)}_{old_{i,j}}} \leq G_l\right)$.
So we will have that:

$$\eta \cdot \frac{dJ}{dw_{old_{i,j}}} \leq \eta \cdot \alpha^{(depth-l)} \cdot G_l \leq \alpha^{(depth-l)} \cdot B_l, \tag{9.13}$$

where B_l is an upper bound on the product of the learning rate η with G_l, i.e. $B_l = \eta \cdot G_l$.

20 Lemma 22 Proof

Lemma 22 *While model's loss, through gradients, flows backwards some weight elements do not receive updates, because we have dying ReLUs (i.e. ReLUs outputting zero) [126]. The total number of weight elements getting updated at layer l is bounded by:*

$$\#\{w^{(l)}_{i,j}\} \leq p^{(dep-l)} \cdot d^2,$$

where d is the largest of the two dimensions of $W^{(l)}$, i.e. there are at most d^2 elements in $W^{(l)}$, if it is a square matrix.

Proof: Let us assume that the number of weights that get updated is reduced as the gradient flows backward. This is due to the fact that, the further the gradient "travels" the more gradients of ReLUs will exist within it. Hence the greater is the probability at least one of them to be zero. In fact, we define the probability of a weight element to get updated and the number of weight elements to be updated in layer l as:

$$P\{w^{(l)}_{i,j} = updated\} = p^{(depth-l)}.$$

$$\#\{w^{(l)}_{i,j}\} = p^{(depth-l)} \cdot (i \cdot j) \leq p^{(depth-l)} \cdot d^2. \tag{9.14}$$

Where i, j are the dimensions of the weight matrix and d is the largest of them.
The probability for a weight element to get updated has the aforementioned form, because we have used the Assumption 20 regarding the probability of a ReLU unit not to output zero, which is at most p. Thus, the probability of getting updated is the probability of the gradient flowing backwards to reach the weight element, which in turn means not to have any ReLU component equal to zero.
The total number of elements that get updated is upper bounded (using union bound) on the sum of the probabilities of all elements of the weight matrix. In fact, every weight matrix has $i \cdot j$ elements where i, j are the matrix dimensions and $d = max(i, j)$.

21 Theorem 23 & Theorem 24 Proofs

Theorem 23 *The upper bound of the largest singular value of the weight matrix $W^{(l)}$ at layer l for a GNN model, utilizing a ReLU activation function, depends on the slope of the function. That bound is given per layer and shows the effect of each iteration of updates on the weight matrix. We denote with W_{old} and W_{new} the weight matrices before and after the update during an iteration of the training process respectively.*

$$max(s_l(W^{(l)}_{new})) \leq ||W^{(l)}_{old}||_F + \sqrt{3} \cdot p^{\left(\frac{dep-l}{2}\right)} \cdot d \cdot \alpha^{(dep-l)} \cdot B_l,$$

where $B_l = \eta \cdot G_l$, dep is network's depth, d is the largest dimension of $W^{(l)}$ matrix, p is the upper bound of the probability of ReLU not to output zero and α is ReLU's slope.

Proof: Regarding the singular values of a matrix (denoted by s_l), it is known that:

$$max(s_l(W^{(l)})) = ||W^{(l)}||_2 \leq ||W^{(l)}||_F = \sqrt{\sum |w^{(l)}_{i,j}|^2},$$

the matrix here being $W^{(l)}$, the weight matrix at layer *l*. Based on Lemma 21, a new weight, during the weight update process will be given by: $w_{new} \sim \alpha^{(depth-l)} \cdot B_l$. Since only $\#\{w^{(l)}_{i,j}\}$ weight elements are getting updated per layer (Lemma 22), in each iteration the Frobenius norm increases by a value, that depends on layer index, depth and slope. Specifically, if we define $\alpha^{(depth-l)} \cdot B_l = K_l$, in order to simplify the formulas, we have:

$$max(s_l(W^{(l)}_{new})) \leq ||W^{(l)}_{new}||_F = \sqrt{\sum |w^{(l)}_{new_{i,j}}|^2} \rightarrow$$

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} |w^{(l)}_{new_{i,j}}|^2 + \sum_{not\_updated} |w^{(l)}_{new_{i,j}}|^2} \xrightarrow{9.13}$$

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} |w^{(l)}_{old_{i,j}} + K_l|^2 + \sum_{not\_updated} |w^{(l)}_{old_{i,j}}|^2} \rightarrow$$

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} \left(|w^{(l)}_{old_{i,j}}|^2 + |K_l|^2 + 2w^{(l)}_{old_{i,j}} K_l\right) + \sum_{not\_updated} |w^{(l)}_{old_{i,j}}|^2}$$

Using the fact that $w^{(l)}_{old_{i,j}} <= K_l$, because K_l contains an exponential term and weights are initialized to values close to zero we get:

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} \left(|w^{(l)}_{old_{i,j}}|^2 + 3|K_l|^2\right) + \sum_{not\_updated} |w^{(l)}_{old_{i,j}}|^2} \xrightarrow{\sqrt{A+B} \leq \sqrt{A} + \sqrt{B}}$$

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} |w^{(l)}_{old_{i,j}}|^2 + \sum_{not\_updated} |w^{(l)}_{old_{i,j}}|^2} + \sqrt{\sum_{updated} 3|K_l|^2} \xrightarrow{Lemma2(9.14)}$$

In the second square root we sum over the updated weight elements and K_l is independent of them. Hence, we get:

$$max(s_l(W_{new}^{(l)})) \leq ||W_{old}^{(l)}||_F + \sqrt{3|K_l|^2 \cdot \#\{w_{i,j}^{(l)}\}}$$

$$max(s_l(W_{new}^{(l)})) \leq ||W_{old}^{(l)}||_F + \sqrt{3} \cdot |K_l| \cdot p^{\left(\frac{depth-l}{2}\right)} \cdot d$$

$$max(s_l(W_{new}^{(l)})) \leq ||W_{old}^{(l)}||_F + \sqrt{3} \cdot p^{\left(\frac{depth-l}{2}\right)} \cdot d \cdot \alpha^{(depth-l)} \cdot B_l.$$

So the slope of the function determines the upper bound of the Frobenius norm while training. In turn this norm is directly connected as an upper bound to the largest singular value of the matrix. In the aforementioned proof we have used previously defined Lemmas and Assumptions.

Theorem 24 *The lower bound of the largest singular value of the weight matrix $W^{(l)}$ at layer l for a GNN model, utilizing ReLU activation function depends on the slope of the function. That bound is given per layer and shows the effect of each iteration of updates on the weight matrix. We denote with W_{old} and W_{new} the weight matrices before and after the update during an iteration of the training process respectively.*

$$max(s_l(W_{new}^{(l)})) \geq \sqrt{2\sqrt{3} \cdot max(w_{old_{i,j}}^{(l)}) \cdot B_l'} \cdot \alpha^{\frac{dep-l}{2}}, \tag{9.15}$$

where $B_l' = \eta \cdot L_l$, L_l is the lower bound to the gradient with respect to the output of the subsequent layer $\left(\frac{dO^{(l+1)}}{w_{old_{i,j}}^{(l)}} \geq L_l\right)$, dep is the network's depth and α is ReLU's slope.

Proof: Let $e_1, ..., e_n$ denote the canonical basis of R^n. Let L_l is the lower bound to the gradient with respect to the output of the subsequent layer, i.e., L_l is the lower bound in a similar way that G_l is the upper bound $\left(\frac{dO^{(l+1)}}{w_{old_{i,j}}^{(l)}} \geq L_l\right)$. We define B_l' as $B_l' = \eta L_l$ and set $K_l' = \alpha^{(depth-l)} B_l'$.

$$max(s_l(W^{(l)})) \geq ||W^{(l)} e_j|| = \sqrt{\sum_{i=1}^{d} \left(w_{i,j}^{(l)}\right)^2} \geq max(w_{i,j}^{(l)}) \rightarrow$$

$$max(s_l(W_{new}^{(l)})) \geq \sqrt{\sum_{i=1}^{d} \left(w_{new_{i,j}}^{(l)}\right)^2} \rightarrow$$

$$max(s_l(W_{new}^{(l)})) \geq \sqrt{\sum_{i=1}^{d} \left(w_{old_{i,j}}^{(l)}\right)^2 + \sum_{updated} 3(K_l')^2} \xrightarrow{\text{at least one update}}$$

$$max(s_l(W_{new}^{(l)})) \geq \sqrt{\sum_{i=1}^{d} \left(w_{old_{i,j}}^{(l)}\right)^2 + 3(K_l')^2} \xrightarrow{\text{at most d elements}}$$

$$max(s_l(W_{new}^{(l)})) \geq \sqrt{(max(w_{old_{i,j}}^{(l)}))^2 + 3(K_l')^2} \xrightarrow{\sqrt{A+B} \geq \sqrt{2}(A \cdot B)^{1/4}}$$

$$max(s_l(W_{new}^{(l)})) \geq \sqrt{2\sqrt{3} \cdot max(w_{old_{i,j}}^{(l)}) \cdot K_l'} \rightarrow$$
$$max(s_l(W_{new}^{(l)})) \geq \sqrt{2\sqrt{3} \cdot max(w_{old_{i,j}}^{(l)}) \cdot B_l'} \cdot \alpha^{\frac{depth-l}{2}}.$$

22 Modifying learning rate: LR2GNN

As discussed in the main text, an alternative to changing the slope of ReLU is to modify the learning rate. In order to test this hypothesis and compare the results to the modified activation function, we performed some preliminary experiments with GCN. In these experiments, we determined a different learning rate per layer, using the validation part of each dataset and the chosen values are shown in Table 9.8. We did this only for the first 8 layers, as the process of tuning the rate proved very cumbersome and time-consuming. The main observation seems to be the improved performance over the vanilla GCN for the three simpler datasets, coupled with a low accuracy for the *Texas* dataset. The comparison against the modified activation function seems inconclusive, but the latter approach wins due to its simplicity. If one opted for the modified learning rate, a method to automatically adapt the rate in different layers would be needed. Deeper architectures extend the complexity and the pool, from which we would have to find the proper values of learning rates. Worth to mention, combining different values of LR per layer with Slope2GNN results in negligible improvements.

Table 9.8: Preliminary results of modifying the learning rate of GCN on *Cora, CiteSeer, Pubmed* and *Texas* datasets. The graph displays test accuracy of an 8-layer GCN, using the aforementioned learning rates.

Layer index	0	1	2	3	4	5	6	7
Learning Rate	$3 \cdot 10^{-3}$	10^{-4}	10^{-5}	$5 \cdot 10^{-5}$	10^{-5}	10^{-4}	10^{-4}	10^{-4}

Cora Accuracy: **76.43**	CiteSeer Accuracy: **64.96**	Pubmed Accuracy: **78.82**
Texas Accuracy: **16.32**		

23 Extended Experiments

Extended experimentation, regular datasets:
Table 9.9 is an expanded version of Table 7.1 (of the main text), showing average test node classification accuracy along with the respective standard deviations. We have also included GCNII [11], which is one of the best performing GNNs utilizing residual connections. GCNII reduces oversmoothing through "short circuiting" initial information to subsequent layers of the model. The aim of the comparison against GCNII is to show that by changing the slope of ReLU simple models are capable of producing comparable (up to a limit) results to more sophisticated architectures, that use residual connections.
Extended "Cold start":
Table 9.10 is an expanded version of Table 7.2 (of the main text) including (a) the performance of the GCNII method, (b) standard deviations for all models, (c) preliminary results on two additional datasets called Computers and Amazon from [147]. The "cold start"

Table 9.9: Performance comparison of vanilla GCN and GAT against SeLU and Slope2GNN enhanced versions of the same models, in *Cora, CiteSeer, Pubmed, Texas*. We include GCNII [11] in the comparison. Average test node classification accuracy (%) and standard deviation for networks of different depth. With **bold** the best performing method for each model (i.e., GCN, GAT and GCNII), depth and dataset.

Accuracy (%) and standard deviation							
Dataset	Method	# Layers					
		2	4	8	16	32	64
Cora	GCN	81.38± 0.5	78.09± 1.8	23.56± 9.2	14.92± 6.4	14.02± 3.5	12.48± 2.9
	GCN(Slope2GNN)	**81.84**± 0.3	**80.39**± 0.9	76.54± 1.5	**76.81**± 2.2	**73.21**± 1.7	**61.59**± 4.7
	GCN(SELU)	81.74± 0.4	79.85± 0.7	**79.41**± 0.5	76.38± 1.1	68.77± 5.7	26.85± 4.0
	GAT	77.79± 1.5	78.06± 1.6	33.11± 9.2	15.53± 9.0	14.67± 4.5	15.18± 5.5
	GAT(Slope2GNN)	**77.81**± 2.2	**79.54**± 1.2	**78.33**± 2.5	**75.00**± 1.9	**69.62**± 2.4	24.17± 9.5
	GAT(SELU)	76.59± 2.2	79.33± 1.1	78.19± 1.1	74.92± 1.6	69.36± 4.3	**30.55**± 4.9
	GCNII	**82.32**± 0.6	**82.89**± 0.6	**83.99**± 0.5	**84.77**± 0.7	**84.96**± 0.7	**85.39**± 0.6
CiteSeer	GCN	70.52± 0.6	65.21± 1.0	32.54± 9.3	17.73± 4.9	16.69± 4.8	17.65± 4.2
	GCN(Slope2GNN)	**70.55**± 0.4	**67.57**± 0.7	65.80± 1.2	57.10± 4.3	47.20± 3.8	**46.46**± 4.2
	GCN(SELU)	69.91± 0.4	67.53± 0.6	**67.27**± 0.9	**61.99**± 1.9	**47.90**± 9.2	27.10± 7.6
	GAT	**69.04**± 1.4	63.98± 2.8	31.33± 7.0	17.80± 1.9	18.67± 2.8	13.35± 4.3
	GAT(Slope2GNN)	68.24± 1.2	**67.27**± 1.0	65.55± 2.0	56.13± 2.5	49.16± 4.5	**26.24**± 3.5
	GAT(SELU)	66.81± 1.5	67.24± 1.1	**67.58**± 1.1	**62.23**± 3.8	**52.97**± 6.4	20.42± 2.8
	GCNII	**68.05**± 0.7	**67.75**± 1.3	**70.90**± 0.5	**72.69**± 0.4	**72.68**± 0.4	**72.77**± 0.9
Pubmed	GCN	77.65± 0.3	**76.75**± 0.7	49.08± 9.7	28.95± 9.9	38.49± 9.8	31.73± 9.9
	GCN(Slope2GNN)	**78.34**± 0.1	76.46± 1.0	**75.00**± 1.4	**76.35**± 1.7	**75.78**± 2.1	**71.68**± 1.7
	GCN(SELU)	77.80± 0.2	74.09± 1.3	73.43± 1.1	76.11± 2.1	71.80± 4.3	40.90± 0.9
	GAT	**77.33**± 0.7	**76.34**± 1.3	55.86± 7.5	29.88± 9.9	29.34± 9.9	28.62± 9.9
	GAT(Slope2GNN)	77.02± 0.7	75.59± 1.1	**74.20**± 1.7	73.81± 2.4	**74.63**± 1.1	38.01± 5.7
	GAT(SELU)	76.83± 1.0	72.62± 1.2	71.93± 1.8	**74.27**± 2.7	73.60± 2.2	**41.00**± 0.3
	GCNII	**78.57**± 0.6	**79.09**± 0.4	**79.85**± 0.5	**79.70**± 0.3	**79.77**± 0.2	**79.65**± 0.3
Texas	GCN	**59.01**± 2.7	**59.55**± 7.7	**57.48**± 6.5	39.91± 0.7	25.50± 0.6	27.39± 0.9
	GCN(Slope2GNN)	57.93± 4.2	57.66± 5.0	57.18± 6.2	**57.21**± 6.4	**55.79**± 6.5	46.31± 4.1
	GCN(SELU)	59.01± 2.7	58.02± 5.0	56.85± 6.2	52.88± 6.6	55.77± 6.2	**58.38**± 6.8
	GAT	**59.54**± 4.2	58.28± 6.5	**57.65**± 6.7	41.71± 4.0	18.64± 2.9	06.76± 1.9
	GAT(Slope2GNN)	56.67± 4.3	57.39± 5.2	56.85± 6.0	**57.53**± 6.1	54.14± 6.5	41.44± 2.3
	GAT(SELU)	57.21± 5.0	**58.65**± 5.8	57.47± 4.5	57.44± 6.9	**57.75**± 6.4	**56.30**± 5.1
	GCNII	**69.37**± 2.9	**70.72**± 4.3	**72.07**± 2.5	**70.00**± 2.2	**71.26**± 1.6	**71.35**± 1.1

problem requires deeper models, because nodes containing informative feature vectors are located further from each test node. Thus, architectures that utilize deep GNNs could effectively address the problem, if they avoid oversmoothing. The proposed method effectively enables deep architectures using simple models (i.e., GCN, GAT). Additionally, the new datasets indicate that the simple methods may perform better than more advanced ones, when assisted by our method. Removing the features of non-training nodes might be a real case scenario (although an extreme one) when it comes to recommender systems.

Table 9.10: Comparison of different models and activation functions on the "cold start" problem. We show accuracy percentage (%) and standard deviation on the test set using GCN and GAT as backbone GNN models and GCNII as a representative of residual GNNs. Only the features of the nodes in the training set, are available to the model. We also show at what depth (i.e. # Layers) each model achieves its best accuracy.

Dataset	Method	Accuracy (%) & std	#L
Cora	GCN	64.85± 1.2	4
	GCN(Slope2GNN)	**73.47**± 0.9	19
	GCN(SeLU)	73.00± 1.5	22
	GAT	59.74± 1.2	3
	GAT(Slope2GNN)	**72.78**± 1.4	20
	GAT(SeLU)	72.65± 1.6	23
	GCNII	**73.40**± 1.0	5
CiteSeer	GCN	41.94± 0.3	4
	GCN(Slope2GNN)	**49.88**± 1.1	21
	GCN(SeLU)	49.30± 1.2	19
	GAT	38.50± 4.7	4
	GAT(Slope2GNN)	**49.63**± 0.9	26
	GAT(SeLU)	49.38± 1.3	20
	GCNII	**59.84**± 3.0	31
Pubmed	GCN	60.16± 5.4	4
	GCN(Slope2GNN)	**72.61**± 0.8	32
	GCN(SeLU)	72.50± 0.8	23
	GAT	50.45± 8.9	4
	GAT(Slope2GNN)	71.14± 1.5	32
	GAT(SeLU)	**71.41**± 1.7	26
	GCNII	**78.25**± 0.6	7
Texas	GCN	32.40± 6.8	4
	GCN(Slope2GNN)	**33.33**± 6.9	6
	GCN(SeLU)	31.62± 5.9	3
	GAT	30.10± 8.5	2
	GAT(Slope2GNN)	**31.00**± 5.9	4
	GAT(SeLU)	30.81± 5.8	2
	GCNII	**57.66**± 0.0	1

Dataset	Method	Accuracy (%) & std	#L
Computers	GCN	18.43± 9.5	15
	GCN(Slope2GNN)	**77.07**± 1.9	5
	GCN(SeLU)	76.90± 2.0	5
	GAT	13.10± 8.5	5
	GAT(Slope2GNN)	**71.94**± 2.0	13
	GAT(SeLU)	71.81± 5.1	8
	GCNII	**61.00**± 7.0	31
Photo	GCN	16.00± 6.0	4
	GCN(Slope2GNN)	**82.07**± 1.7	5
	GCN(SeLU)	81.15± 3.5	4
	GAT	13.87± 7.4	8
	GAT(Slope2GNN)	**77.53**± 4.3	14
	GAT(SeLU)	77.40± 4.1	15
	GCNII	**53.80**± 3.2	32

Slope sensitivity results:
The choice of slope (i.e., $\alpha = 2$) that we proposed is based on Equation 7.5 (of the main text). We have verified experimentally that this slope value helps to reduce the effect of oversmoothing. However, this is not the only slope value that could be used. Therefore, we experimented with other slope values and present the results in Figure 9.31. Based on the Figure, it is evident that any value between 1.8 and 2.2 reduces oversmoothing, making the exact choice of value less important.

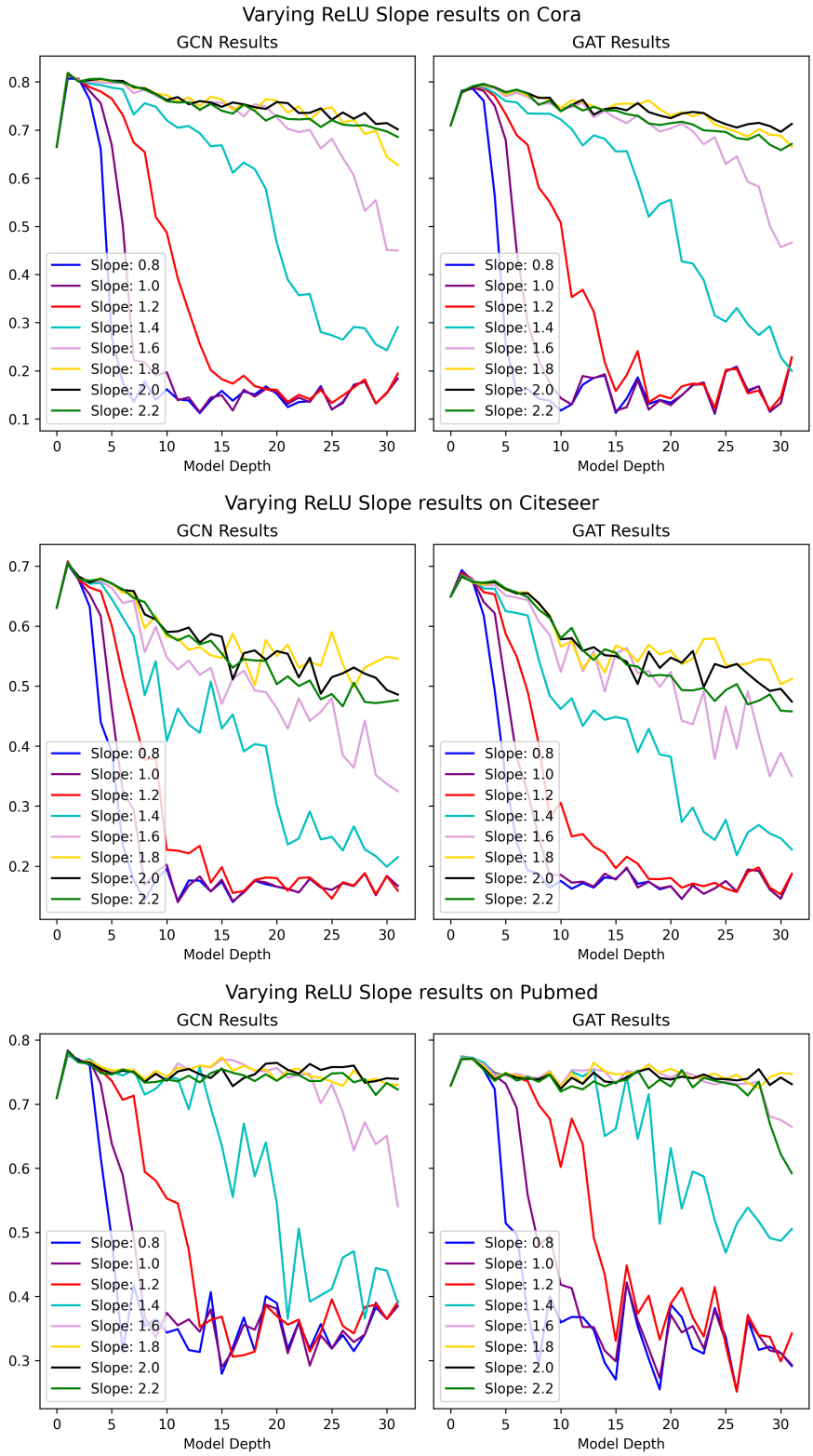

Figure 9.31: Comparison of test performance accuracy (%) of GCN and GAT with varying slope values in *Cora, CiteSeer* and *Pubmed*. We show the accuracy of the models in y-axis, while increasing their depth.

Visualising the effect of oversmoothing:
In order to understand the effect of oversmoothing in GNNs and show how our method alleviates it, we visualise node representations using t-SNE [142] for varying depth mod-

els. Figures 9.32, 9.33 and 9.34 visualise the representations for different depth levels and different ReLU slope values. Ideally, we would like nodes of the same color (same class) to be close together and well-separated from other classes. In all three datasets, we observe how oversmoothing (at slope = 1) leads to a mixing of the classes. We also see how the situation changes for slope = 2. In those graphs, node representations do not rapidly mix as the depth of the network increases.

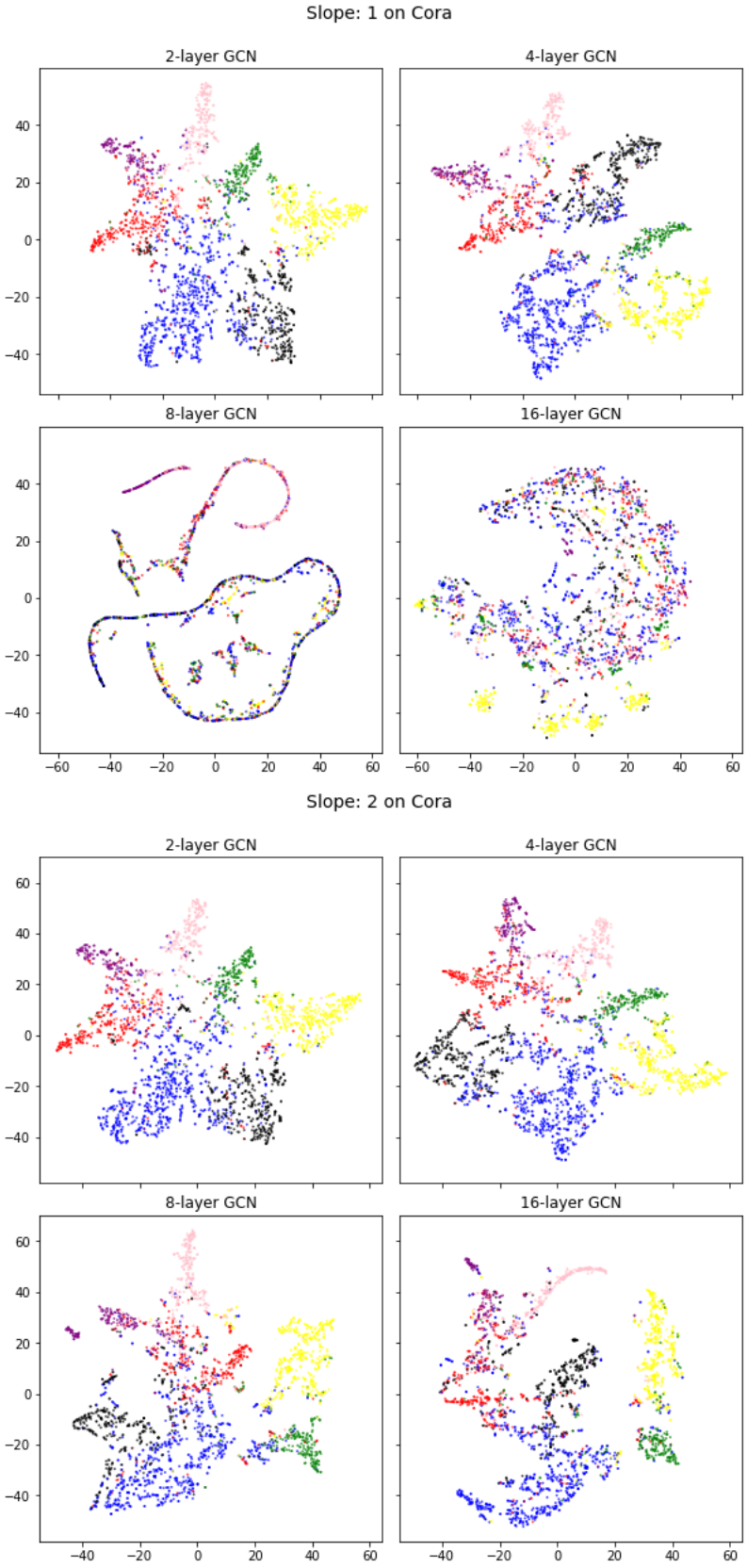

Figure 9.32: t-SNE of node representations of GCN and GAT on *Cora*, while increasing model's depth. Upper (lower) 4 figures show the results without (with) our method.

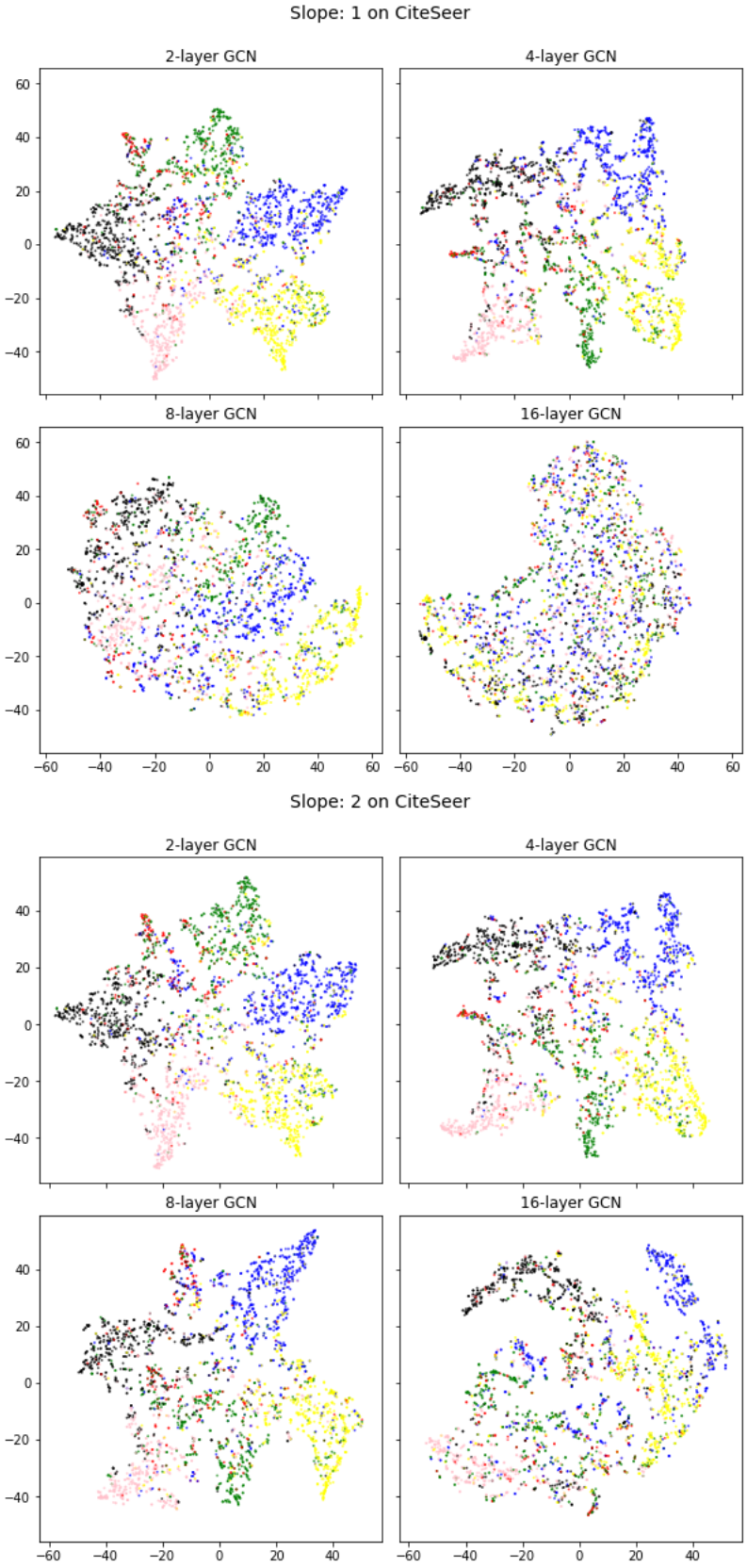

Figure 9.33: t-SNE of node representations of GCN and GAT on *CiteSeer*, while increasing model's depth. Upper (lower) 4 figures show the results without (with) our method.

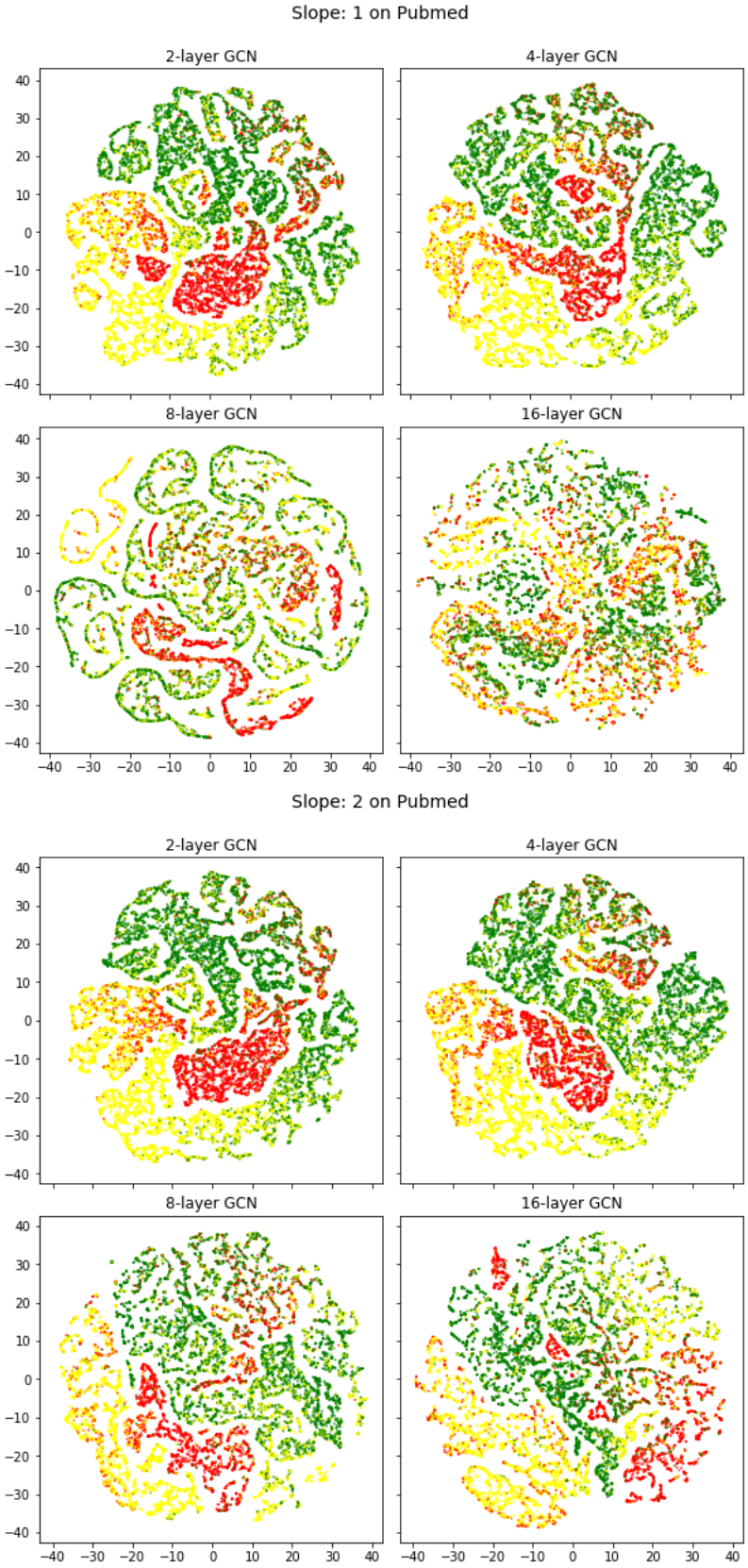

Figure 9.34: t-SNE of node representations of GCN and GAT on *Pubmed*, while increasing model's depth. Upper (lower) 4 figures show the results without (with) our method.

24 Dataset Statistics

Table 9.11: The statistics of all datasets used in this work.

Datasets	# Nodes	# Edges	# Classes	# Features
Cora	2708	10556	7	1433
CiteSeer	3327	9104	6	3703
Pubmed	19717	88648	3	500
Physics	34493	495924	5	8415
CS	18333	163788	15	6805
Photo	7650	238162	8	745
Computers	13752	491722	10	767
Arxiv	169343	1166243	40	128
Texas	183	309	5	1703

Bibliography

[1] D. Kelesis, D. Fotakis, and G. Paliouras, "Analyzing the effect of embedding norms and singular values to oversmoothing in graph neural networks," *arXiv*, vol. 2510.06066, 2025. [Online]. Available: https://arxiv.org/abs/2510.06066 7, 9, 35, 36, 37, 38

[2] ——, "Analyzing the effect of residual connections to oversmoothing in graph neural networks," *Mach. Learn.*, vol. 114, no. 8, p. 184, 2025. [Online]. Available: https://doi.org/10.1007/s10994-025-06822-0 7, 9, 35, 36, 37, 38

[3] ——, "Partially trained graph convolutional networks resist oversmoothing," *Mach. Learn.*, vol. 114, no. 10, p. 211, 2025. [Online]. Available: https://doi.org/10.1007/s10994-025-06865-3 7, 9, 35, 36, 37, 38

[4] ——, "Reducing oversmoothing through informed weight initialization in graph neural networks," *Applied Intelligence*, vol. 55, no. 7, p. 632, Apr 2025. [Online]. Available: https://doi.org/10.1007/s10489-025-06426-0 7, 9, 34, 35, 36, 37, 38

[5] D. Kelesis, D. Vogiatzis, G. Katsimpras, D. Fotakis, and G. Paliouras, "Reducing oversmoothing in graph neural networks by changing the activation function," in *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*, ser. Frontiers in Artificial Intelligence and Applications, K. Gal, A. Nowé, G. J. Nalepa, R. Fairstein, and R. Radulescu, Eds., vol. 372. IOS Press, 2023, pp. 1231–1238. [Online]. Available: https://doi.org/10.3233/FAIA230400 7, 9, 34, 35, 36, 37, 38

[6] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km 18, 31, 32, 51

[7] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=S1ldO2EFPr 19, 32, 33, 34, 54, 55, 56, 93, 94, 119, 133, 134, 137, 139

[8] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," *CoRR*, vol. abs/2006.05205, 2020. [Online]. Available: https://arxiv.org/abs/2006.05205 19, 32, 96

[9] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *6th International Conference on*

Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=SJiHXGWAZ 20, 43, 52

[10] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *CoRR*, vol. abs/2104.13478, 2021. [Online]. Available: https://arxiv.org/abs/2104.13478 21, 42, 43

[11] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 1725–1735. [Online]. Available: http://proceedings.mlr.press/v119/chen20v.html 22, 32, 50, 54, 61, 88, 93, 95, 97, 103, 110, 120, 183, 184

[12] A. Roth and T. Liebig, "Transforming pagerank into an infinite-depth graph neural network," *CoRR*, vol. abs/2207.00684, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2207.00684 22, 50, 88, 91, 97

[13] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in gnns," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=rkecl1rtwB 22, 32, 34, 57, 62, 88, 95, 113, 119, 134

[14] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=Hkx1qkrKPr 22, 32, 63

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852 23, 27, 64, 119, 120, 121, 123, 126, 128, 132, 171

[16] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=H1gL-2A9Ym 23, 49, 57, 79, 88, 89, 90, 97

[17] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 6861–6871. [Online]. Available: http://proceedings.mlr.press/v97/wu19e.html 23, 50, 80

[18] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *CoRR*, vol. abs/1704.01212, 2017. [Online]. Available: http://arxiv.org/abs/1704.01212 30, 42, 43, 45

[19] J. Klicpera, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=B1eWbxStPH 32, 42

[20] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 3538–3545. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16098 32, 33, 34, 45, 47, 54, 55

[21] C. Cai and Y. Wang, "A note on over-smoothing for graph neural networks," *CoRR*, vol. abs/2006.13318, 2020. [Online]. Available: https://arxiv.org/abs/2006.13318 32, 33, 34, 54, 64, 93, 133

[22] F. D. Giovanni, L. Giusti, F. Barbero, G. Luise, P. Lio, and M. M. Bronstein, "On over-squashing in message passing neural networks: The impact of width, depth, and topology," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 7865–7885. [Online]. Available: https://proceedings.mlr.press/v202/di-giovanni23a.html 32

[23] G. Li, M. Müller, A. K. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 9266–9275. [Online]. Available: https://doi.org/10.1109/ICCV.2019.00936 32, 47, 61, 80

[24] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 1024–1034. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html 33, 42, 43, 48

[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl 33, 41, 42, 43, 47, 51, 80, 97, 103, 104, 110, 125, 138

[26] H. Cai, V. W. Zheng, and K. C. Chang, "A comprehensive survey of graph embedding: Problems, techniques and applications," *CoRR*, vol. abs/1709.07604, 2017. [Online]. Available: http://arxiv.org/abs/1709.07604 41

[27] D. Liben-Nowell and J. M. Kleinberg, "The link prediction problem for social networks," in *Proceedings of the 2003 ACM CIKM International Conference*

on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003. ACM, 2003, pp. 556–559. [Online]. Available: https://doi.org/10.1145/956863.956972 41

[28] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 5171–5181. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/53f0d7c537d99b3824f0f99d62ea2428-Abstract.html 41

[29] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks," *J. Mach. Learn. Res.*, vol. 24, pp. 127:1–127:21, 2023. [Online]. Available: https://jmlr.org/papers/v24/20-998.html 41

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1106–1114. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html 41

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556 41

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 1–9. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7298594 41

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90 41, 60

[34] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *CoRR*, vol. abs/1808.03314, 2018. [Online]. Available: http://arxiv.org/abs/1808.03314 41

[35] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078 41

[36] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *CoRR*, vol. abs/1303.5778, 2013. [Online]. Available: http://arxiv.org/abs/1303.5778 41

[37] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019 41

[38] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *CoRR*, vol. abs/1611.08097, 2016. [Online]. Available: http://arxiv.org/abs/1611.08097 42

[39] J. E. Gerken, J. Aronsson, O. Carlsson, H. Linander, F. Ohlsson, C. Petersson, and D. Persson, "Geometric deep learning and equivariant neural networks," *Artif. Intell. Rev.*, vol. 56, no. 12, pp. 14 605–14 662, 2023. [Online]. Available: https://doi.org/10.1007/s10462-023-10502-7 42

[40] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, ser. JMLR Workshop and Conference Proceedings, M. Balcan and K. Q. Weinberger, Eds., vol. 48. JMLR.org, 2016, pp. 2990–2999. [Online]. Available: http://proceedings.mlr.press/v48/cohenc16.html 42

[41] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ 42, 48, 103, 110, 125, 138

[42] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=F72ximsx7C1 42

[43] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *2015 IEEE International Conference on Computer Vision Workshop, ICCV Workshops 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 832–840. [Online]. Available: https://doi.org/10.1109/ICCVW.2015.112 42, 43

[44] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 5425–5434. [Online]. Available: https://doi.org/10.1109/CVPR.2017.576 42, 43

[45] A. Poulenard and M. Ovsjanikov, "Multi-directional geodesic neural networks via equivariant convolution," *CoRR*, vol. abs/1810.02303, 2018. [Online]. Available: http://arxiv.org/abs/1810.02303 42

[46] T. S. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling, "Gauge equivariant convolutional networks and the icosahedral CNN," *CoRR*, vol. abs/1902.04615, 2019. [Online]. Available: http://arxiv.org/abs/1902.04615 42

[47] P. de Haan, M. Weiler, T. Cohen, and M. Welling, "Gauge equivariant mesh cnns: Anisotropic convolutions on geometric graphs," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=Jnspzp-oIZE 42

[48] N. Keriven and G. Peyré, "Universal invariant and equivariant graph neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 7090–7099. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/ea9268cb43f55d1d12380fb6ea5bf572-Abstract.html 42, 43

[49] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, pp. 452–473, 1977. [Online]. Available: http://www.jstor.org/stable/3629752 42, 103

[50] Y. Wang, Z. Li, and A. B. Farimani, "Graph neural networks for molecules," *CoRR*, vol. abs/2209.05582, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2209.05582 42, 43

[51] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 2224–2232. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/f9be311e65d81a9ad8150a60844bb94c-Abstract.html 42

[52] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, "Learning mesh-based simulation with graph networks," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=roNqYL0_XP 42

[53] Z. Li, N. B. Kovachki, C. B. Choy, B. Li, J. Kossaifi, S. P. Otta, M. A. Nabian, M. Stadler, C. Hundt, K. Azizzadenesheli, and A. Anandkumar, "Geometry-informed neural operator for large-scale 3d pdes," *CoRR*, vol. abs/2309.00583, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2309.00583 42

[54] G. Dong, M. Tang, Z. Wang, J. Gao, S. Guo, L. Cai, R. J. Gutierrez, B. Campbell, L. E. Barnes, and M. Boukhechba, "Graph neural networks in iot: A survey," *CoRR*, vol. abs/2203.15935, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.15935 42

[55] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. [Online]. Available: https://doi.org/10.1109/TNN.2008.2005605 42, 43

[56] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *2nd International Conference on Learning*

Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, Y. Bengio and Y. LeCun, Eds., 2014. [Online]. Available: http://arxiv.org/abs/1312.6203 42, 43

[57] R. Kondor and S. Trivedi, "On the generalization of equivariance and convolution in neural networks to the action of compact groups," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 2752–2760. [Online]. Available: http://proceedings.mlr.press/v80/kondor18a.html 42

[58] J. Klicpera, A. Bojchevski, and S. Günnemann, "Personalized embedding propagation: Combining neural networks on graphs with personalized pagerank," *CoRR*, vol. abs/1810.05997, 2018. [Online]. Available: http://arxiv.org/abs/1810.05997 43

[59] S. Batzner, A. Musaelian, L. Sun, M. Geiger, J. P. Mailoa, M. Kornbluth, N. Molinari, T. E. Smidt, and B. Kozinsky, "E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials," *Nature Communications*, vol. 13, no. 1, May 2022. [Online]. Available: http://dx.doi.org/10.1038/s41467-022-29939-5 43

[60] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5099–5108. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/d8bf84be3800d12f74d8b05e9b89836f-Abstract.html 43

[61] V. G. Satorras, E. Hoogeboom, and M. Welling, "E(n) equivariant graph neural networks," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021, pp. 9323–9332. [Online]. Available: http://proceedings.mlr.press/v139/satorras21a.html 43

[62] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, J. Lang, Ed. ijcai.org, 2018, pp. 3634–3640. [Online]. Available: https://doi.org/10.24963/ijcai.2018/505 43, 52

[63] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 11, pp. 4883–4894, 2020. [Online]. Available: https://doi.org/10.1109/TITS.2019.2950416 43, 52

[64] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 3837–3845. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/04df4d434d481c5bb723be1b6df1ee65-Abstract.html 43, 49

[65] C. K. Joshi, C. Bodnar, S. V. Mathis, T. Cohen, and P. Lio, "On the expressive power of geometric graph neural networks," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 15 330–15 355. [Online]. Available: https://proceedings.mlr.press/v202/joshi23a.html 43

[66] U. von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007. [Online]. Available: https://doi.org/10.1007/s11222-007-9033-z 43, 44

[67] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1088/1742-5468/2008/10/P10008 43

[68] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, 2013. [Online]. Available: https://doi.org/10.1109/MSP.2012.2235192 43, 45

[69] F. R. K. Chung, *Spectral Graph Theory*. American Mathematical Society, 1997. 44

[70] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. MIT Press, 2001, pp. 849–856. [Online]. Available: https://proceedings.neurips.cc/paper/2001/hash/801272ee79cfde7fa5960571fee36b9b-Abstract.html 44, 45

[71] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000. 44

[72] L. Lovász, "Random walks on graphs: A survey," *Combinatorics, Paul Erdos is Eighty*, vol. 2, no. 1, pp. 1–46, 1993. [Online]. Available: http://scholar.google.de/scholar.bib?q=info:llcRghStI1oJ:scholar.google.com/&output=citation&hl=de&as_sdt=0,5&ct=citation&cd=3 44

[73] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford Digital Library Technologies Project, Tech. Rep., 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.1768 44

[74] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal*, vol. 23, no. 2, pp. 298–305, 1973. [Online]. Available: http://eudml.org/doc/12723 44

[75] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds. MIT Press, 2003, pp. 321–328. [Online]. Available: https://proceedings.neurips.cc/paper/2003/hash/87682805257e619d49b8e0dfdc14affa-Abstract.html 45

[76] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, ser. Lecture Notes in Computer Science, B. Schölkopf and M. K. Warmuth, Eds., vol. 2777. Springer, 2003, pp. 144–158. [Online]. Available: https://doi.org/10.1007/978-3-540-45167-9_12 45

[77] Z. Zhong, C. Li, and J. Pang, "Hierarchical message-passing graph neural networks," *Data Min. Knowl. Discov.*, vol. 37, no. 1, pp. 381–408, 2023. [Online]. Available: https://doi.org/10.1007/s10618-022-00890-9 46

[78] Z. Zhang, J. Bu, M. Ester, J. Zhang, C. Yao, Z. Yu, and C. Wang, "Hierarchical graph pooling with structure learning," *CoRR*, vol. abs/1911.05954, 2019. [Online]. Available: http://arxiv.org/abs/1911.05954 46

[79] B. Gutteridge, X. Dong, M. M. Bronstein, and F. D. Giovanni, "Drew: Dynamically rewired message passing with delay," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 12 252–12 267. [Online]. Available: https://proceedings.mlr.press/v202/gutteridge23a.html 46

[80] B. Finkelshtein, X. Huang, M. M. Bronstein, and İ. İ. Ceylan, "Cooperative graph neural networks," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=ZQcqXCuoxD 46

[81] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *CoRR*, vol. abs/2012.09699, 2020. [Online]. Available: https://arxiv.org/abs/2012.09699 51

[82] L. Müller, M. Galkin, C. Morris, and L. Rampásek, "Attending to graph transformers," *Trans. Mach. Learn. Res.*, vol. 2024, 2024. [Online]. Available: https://openreview.net/forum?id=HhbqHBBrfZ 51

[83] G. Mialon, D. Chen, M. Selosse, and J. Mairal, "Graphit: Encoding graph structure in transformers," *CoRR*, vol. abs/2106.05667, 2021. [Online]. Available: https://arxiv.org/abs/2106.05667 51

[84] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T. Liu, "Do transformers really perform bad for graph representation?" *CoRR*, vol. abs/2106.05234, 2021. [Online]. Available: https://arxiv.org/abs/2106.05234 51

[85] C. Chen, Y. Wu, Q. Dai, H. Zhou, M. Xu, S. Yang, X. Han, and Y. Yu, "A survey on graph neural networks and graph transformers in computer vision: A task-oriented perspective," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 10 297–10 318, 2024. [Online]. Available: https://doi.org/10.1109/TPAMI.2024.3445463 51

[86] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=rklz9iAcKQ 51

[87] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 7444–7452. [Online]. Available: https://doi.org/10.1609/aaai.v32i1.12328 52

[88] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-GCN: A temporal graph convolutional network for traffic prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 9, pp. 3848–3858, 2020. [Online]. Available: https://doi.org/10.1109/TITS.2019.2935152 52

[89] F. Li, J. Feng, H. Yan, G. Jin, F. Yang, F. Sun, D. Jin, and Y. Li, "Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution," *ACM Trans. Knowl. Discov. Data*, vol. 17, no. 1, pp. 9:1–9:21, 2023. [Online]. Available: https://doi.org/10.1145/3532611 52

[90] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 4805–4815. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/e77dbaf6759253c7c6d0efc5690369c7-Abstract.html 52

[91] H. Gao, Y. Liu, and S. Ji, "Topology-aware graph pooling networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 12, pp. 4512–4518, 2021. [Online]. Available: https://doi.org/10.1109/TPAMI.2021.3062794 52

[92] M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, ser. Lecture Notes in Computer Science, A. Gangemi, R. Navigli, M. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai, and M. Alam, Eds., vol. 10843. Springer, 2018, pp. 593–607. [Online]. Available: https://doi.org/10.1007/978-3-319-93417-4_38 52

[93] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Y. Huang, I. King, T. Liu, and M. van Steen, Eds. ACM / IW3C2, 2020, pp. 2704–2710. [Online]. Available: https://doi.org/10.1145/3366423.3380027 52

[94] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, Eds., 2013, pp. 2787–2795. [Online]. Available: https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html 52

[95] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," 2020. [Online]. Available: https://arxiv.org/abs/2002.07962 53

[96] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 5363–5370. [Online]. Available: https://doi.org/10.1609/aaai.v34i04.5984 53

[97] J. Wu, M. Cao, J. C. K. Cheung, and W. L. Hamilton, "Temp: Temporal message passing for temporal knowledge graph completion," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds. Association for Computational Linguistics, 2020, pp. 5730–5746. [Online]. Available: https://doi.org/10.18653/v1/2020.emnlp-main.462 53

[98] Y. Yan, M. Hashemi, K. Swersky, Y. Yang, and D. Koutra, "Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks," in *IEEE International Conference on Data Mining, ICDM 2022, Orlando, FL, USA, November 28 - Dec. 1, 2022*, X. Zhu, S. Ranka, M. T. Thai, T. Washio, and X. Wu, Eds. IEEE, 2022, pp. 1287–1292. [Online]. Available: https://doi.org/10.1109/ICDM54844.2022.00169 54

[99] X. Wu, Z. Chen, W. W. Wang, and A. Jadbabaie, "A non-asymptotic analysis of oversmoothing in graph neural networks," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/forum?id=CJd-BtnwtXq 54

[100] Y. Song, C. Zhou, X. Wang, and Z. Lin, "Ordered GNN: ordering message passing to deal with heterophily and over-smoothing," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/forum?id=wKPmPBHSnT6 54

[101] G. Wang, R. Ying, J. Huang, and J. Leskovec, "Multi-hop attention graph neural networks," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, Z. Zhou, Ed. ijcai.org, 2021, pp. 3089–3096. [Online]. Available: https://doi.org/10.24963/ijcai.2021/425 54

[102] T. K. Rusch, M. M. Bronstein, and S. Mishra, "A survey on oversmoothing in graph neural networks," *CoRR*, vol. abs/2303.10993, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.10993 54, 58

[103] K. Zhou, X. Huang, D. Zha, R. Chen, L. Li, S. Choi, and X. Hu, "Dirichlet energy constrained learning for deep graph neural networks," *CoRR*, vol. abs/2107.02392, 2021. [Online]. Available: https://arxiv.org/abs/2107.02392 54, 57, 58, 88, 119

[104] D. Bo, X. Wang, C. Shi, and H. Shen, "Beyond low-frequency information in graph convolutional networks," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 3950–3957. [Online]. Available: https://doi.org/10.1609/aaai.v35i5.16514 55

[105] K. Huang, Y. G. Wang, M. Li, and P. Lio, "How universal polynomial bases enhance spectral graph neural networks: Heterophily, over-smoothing, and over-squashing," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=Z2LH6Va7L2 55

[106] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 3438–3445. [Online]. Available: https://doi.org/10.1609/aaai.v34i04.5747 57, 58, 64, 71

[107] J. Palowitch, A. Tsitsulin, B. A. Mayer, and B. Perozzi, "Graphworld: Fake graphs bring real insights for gnns," in *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, A. Zhang and H. Rangwala, Eds. ACM, 2022, pp. 3691–3701. [Online]. Available: https://doi.org/10.1145/3534678.3539203 57

[108] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 5449–5458. [Online]. Available: http://proceedings.mlr.press/v80/xu18c.html 59, 61, 92, 93

[109] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. V. Steeg, and A. Galstyan, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 21–29. [Online]. Available: http://proceedings.mlr.press/v97/abu-el-haija19a.html 59

[110] H. Dong, J. Chen, F. Feng, X. He, S. Bi, Z. Ding, and P. Cui, "On the equivalence of decoupled graph convolution network and label propagation," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, J. Leskovec, M. Grobelnik, M. Najork, J. Tang, and L. Zia, Eds. ACM / IW3C2, 2021, pp. 3651–3662. [Online]. Available: https://doi.org/10.1145/3442381.3449927 59

[111] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, R. Gupta, Y. Liu,

J. Tang, and B. A. Prakash, Eds. ACM, 2020, pp. 338–348. [Online]. Available: https://doi.org/10.1145/3394486.3403076 61

[112] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html 62

[113] M. Scholkemper, X. Wu, A. Jadbabaie, and M. T. Schaub, "Residual connections and normalization can provably prevent oversmoothing in gnns," 2025. [Online]. Available: https://arxiv.org/abs/2406.02997 62

[114] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016. [Online]. Available: http://arxiv.org/abs/1607.06450 62

[115] T. Cai, S. Luo, K. Xu, D. He, T. Liu, and L. Wang, "Graphnorm: A principled approach to accelerating graph neural network training," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021, pp. 1204–1215. [Online]. Available: http://proceedings.mlr.press/v139/cai21e.html 62

[116] K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu, "Towards deeper graph neural networks with differentiable group normalization," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/33dd6dba1d56e826aac1cbf23cdcca87-Abstract.html 62

[117] N. Huang, S. Villar, C. E. Priebe, D. Zheng, C. Huang, L. Yang, and V. Braverman, "From local to global: Spectral-inspired graph neural networks," *CoRR*, vol. abs/2209.12054, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2209.12054 63

[118] W. Huang, Y. Li, W. Du, R. Y. D. Xu, J. Yin, L. Chen, and M. Zhang, "Towards deepening graph neural networks: A gntk-based optimization perspective," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=tT9t_ZctZRL 63

[119] S. Maskey, R. Paolino, A. Bacho, and G. Kutyniok, "A fractional graph laplacian approach to oversmoothing," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/2a514213ba899f2911723a38be8d4096-Abstract-Conference.html 64

[120] T. Fang, Z. Xiao, C. Wang, J. Xu, X. Yang, and Y. Yang, "Dropmessage: Unifying random dropping for graph neural networks," *Proceedings of the AAAI Conference*

on Artificial Intelligence, vol. 37, no. 4, p. 4267–4275, Jun. 2023. [Online]. Available: http://dx.doi.org/10.1609/aaai.v37i4.25545 64

[121] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 22 092–22 103. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/fb4c835feb0a65cc39739320d7a51c02-Paper.pdf 64

[122] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, ser. JMLR Proceedings, Y. W. Teh and D. M. Titterington, Eds., vol. 9. JMLR.org, 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html 64, 104, 119, 120, 126

[123] J. Li, Y. Song, X. Song, and D. Wipf, "On the initialization of graph neural networks," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 19 911–19 931. [Online]. Available: https://proceedings.mlr.press/v202/li23y.html 65, 126

[124] A. Jaiswal, P. Wang, T. Chen, J. F. Rousseau, Y. Ding, and Z. Wang, "Old can be gold: Better gradient flow can make vanilla-gcns great again," in *NeurIPS*, 2022. 65

[125] X. Han, T. Zhao, Y. Liu, X. Hu, and N. Shah, "Mlpinit: Embarrassingly simple GNN training acceleration with MLP initialization," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=P8YIphWNEGO 65

[126] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying relu and initialization: Theory and numerical examples," *CoRR*, vol. abs/1903.06733, 2019. [Online]. Available: http://arxiv.org/abs/1903.06733 66, 135, 137, 180

[127] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the 30th International Conference on Machine Learning (ICML '13)*, Atlanta, Georgia, USA, 2013, pp. 3–6. 66

[128] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1511.07289 66

[129] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *CoRR*, vol. abs/1706.02515, 2017. [Online]. Available: http://arxiv.org/abs/1706.02515 66, 134

[130] S. Luan, M. Zhao, X. Chang, and D. Precup, "Break the ceiling: Stronger multi-scale deep graph convolutional networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 10 943–10 953. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/ccdf3864e2fa9089f9eca4fc7a48ea0a-Abstract.html 67

[131] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *CoRR*, vol. abs/1811.05868, 2018. [Online]. Available: http://arxiv.org/abs/1811.05868 80, 97

[132] H. Weyl, "Das asymptotische verteilungsgesetz der eigenwerte linearer partieller differentialgleichungen (mit einer anwendung auf die theorie der hohlraumstrahlung)," *Mathematische Annalen*, vol. 71, pp. 441–479, 1912. [Online]. Available: http://eudml.org/doc/158545 89

[133] R. Courant and D. Hilbert, *Methods of Mathematical Physics*. New York: Wiley, 1989, vol. 1. 89

[134] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html 97, 110, 125

[135] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675–701, 1937. 98

[136] Z. D. Bai and Y. Q. Yin, "Limit of the Smallest Eigenvalue of a Large Dimensional Sample Covariance Matrix," *The Annals of Probability*, vol. 21, no. 3, pp. 1275 – 1294, 1993. [Online]. Available: https://doi.org/10.1214/aop/1176989118 104

[137] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, H. Peng, J. Li, J. Wu, Z. Liu, P. Xie, C. Xiong, J. Pei, P. S. Yu, and L. Sun, "A comprehensive survey on pretrained foundation models: A history from BERT to chatgpt," *CoRR*, vol. abs/2302.09419, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.09419 104

[138] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *CoRR*, vol. abs/1811.05868, 2018. [Online]. Available: http://arxiv.org/abs/1811.05868 110, 125

[139] T. Tao and V. Vu, "Random matrices: The circular law," 2008. 120

[140] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," *CoRR*, vol. abs/2007.08663, 2020. [Online]. Available: https://arxiv.org/abs/2007.08663 125

[141] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. S. Pande, "Moleculenet: A benchmark for molecular machine learning," *CoRR*, vol. abs/1703.00564, 2017. [Online]. Available: http://arxiv.org/abs/1703.00564 125

[142] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html 127, 176, 187

[143] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html 134

[144] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016. [Online]. Available: http://arxiv.org/abs/1607.06450 134

[145] Y. Bengio, P. Y. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. [Online]. Available: https://doi.org/10.1109/72.279181 137

[146] H. Pei, B. Wei, K. C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=S1e2agrFvS 138

[147] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *CoRR*, vol. abs/1811.05868, 2018. [Online]. Available: http://arxiv.org/abs/1811.05868 183