Machine Learning (2025) 114:211
https://doi.org/10.1007/s10994-025-06865-3

# Partially trained graph convolutional networks resist oversmoothing

**Dimitrios Kelesis**[1,2] · **Dimitris Fotakis**[1,3] · **Georgios Paliouras**[2]

Received: 26 March 2025 / Revised: 31 July 2025 / Accepted: 4 August 2025 /
Published online: 22 August 2025
© The Author(s) 2025

**Abstract**
In this work we investigate an observation made by Kipf and Welling (5th International Conference on Learning Representations, 2017), who suggested that untrained Graph Convolutional Networks (GCNs) can generate meaningful node embeddings. In particular, we investigate the effect of training only a single layer of a GCN or a GAT (Graph Attention Network), while keeping the rest of the layers frozen. We propose a basis on which the effect of the untrained layers and their contribution to the generation of embeddings can be predicted. Moreover, we show that network width influences the dissimilarity of node embeddings produced after the initial node features pass through the untrained part of the model. Additionally, we establish a connection between partially trained GCNs and oversmoothing, showing that they are capable of reducing it. We verify our theoretical results experimentally and show the benefits of using deep networks that resist oversmoothing, in a "cold start" scenario, where there is a lack of feature information for unlabeled nodes.

**Keywords** Graph neural networks · Oversmoothing · Partially trained networks

## 1 Introduction

Graph Neural Networks (GNNs) have shown promising results in diverse graph analytics tasks, including node classification (Zhang et al., 2018; Hamilton et al., 2017), link prediction (Liben-Nowell et al., 2003; Zhang & Chen, 2018), and graph classification (Klicpera et al., 2020). Consequently, they play a key role in graph representation learning, employing

Editor: Quanming Yao.

✉ Dimitrios Kelesis
dkelesis@iit.demokritos.gr

[1] School of Electrical Engineering and Computer Science, National Technical University of Athens, Athens, Greece

[2] National Center for Scientific Research "Demokritos", Athens, Greece

[3] Archimedes Research Unit, Athena RC, Athens, Greece

message passing schemes to aggregate information from neighboring nodes and construct informative node representations. One of the most prominent GNN models is the Graph Convolutional Network (GCN) (Kipf & Welling, 2017), which generates node representations by computing the average of the representations (embeddings) of each node's immediate neighbors.

At the same time, the improved performance of deep Convolutional Neural Networks (CNNs) compared to their shallow counterparts, has led to several attempts towards building deep GNNs, in order to solve graph analytics tasks more effectively (Li et al., 2018; Xu et al., 2018). Greater depth offers the potential for more accurate representational learning, by expanding the model's receptive field and integrating more information. Hence, there is a demand for the creation of new models capable of scaling efficiently to a significant number of layers. However, designing deep GNNs has proven to be challenging in practice, as increased depth often leads to performance degradation.

The performance decline observed in deep GNNs is primarily attributed to oversmoothing (Li et al., 2018; Xu et al., 2018; Klicpera et al., 2018; Wu et al., 2019), induced by graph convolutions. Graph convolution is a type of Laplacian operator and causes node representations to converge to a stationary point when applied iteratively (Li et al., 2018). At that point, all initial information (i.e., information captured by node features) is dispersed due to the Laplacian smoothing. Therefore, oversmoothing hurts the performance of deep GNNs by making node embeddings indistinguishable across different classes.

Several attempts have been made to reduce oversmoothing, belonging mainly into two categories: altering the architecture of the model and modifying the topology of the graph. Approaches introducing residual and skip connections (Chen et al., 2020; Xu et al., 2018) fall into the former category, while methods focusing on removing edges or nodes from the graph (Rong et al., 2020; Do et al., 2021) belong to the latter. More recent strategies explore the impact of the activation functions (Luan et al., 2019; Kelesis et al., 2023) or leverage the Dirichlet energy (Zhou et al., 2021) to reduce oversmoothing.

Most of the existing GNN approaches train the entire model to generate node embeddings. Kipf and Welling (2017) 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 (Zachary, 1977). Based on this observation, we investigate scenarios where only a single layer of the GCN is trained, while the remaining model parameters retain their initial random values. We refer to such models as partially trained ones, and we examine how altering the width of the model impacts its effectiveness in generating node embeddings. 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.

In summary, the main contributions of this work 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.

The paper is structured as follows. Section 2 introduces the notation and preliminaries essential for our analysis. In Sect. 3, we delve into our theoretical analysis. The experimental evaluation of the proposed method is detailed in Sect. 4, while Sect. 5 discusses related work. Finally, Sect. 6 offers a summary of our conclusions and outlines potential future directions for research.

## 2 Notations and preliminaries

### 2.1 Notations

We consider the task of semi-supervised node classification on a graph $G(V, E, X)$, where $|V| = N$ denotes the number of nodes, $u_i \in V$ represents the nodes, $(u_i, u_j) \in E$ stands for the edges, and $X = [x_1, ..., x_N]^T \in R^{N \times C}$ indicates the initial node features, with each node having a feature vector of dimensionality $C$. The edges collectively form an adjacency matrix $A \in R^{N \times N}$, 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 $(u_i, u_j)$. Node degrees are captured by a diagonal matrix $D \in R^{N \times N}$, where each element $d_i$ represents the sum of edge weights connected to node $i$. During the training phase, only the labels of a subset $V_l \subset V$ are accessible, where $|V_l| = N_{train}$. 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.

*GCN*, originally proposed by Kipf and Welling (2017), utilizes a feed forward propagation as:

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

where $H^{(l)} = [h_1^{(l)}, ..., h_N^{(l)}]$ are node representations (or hidden vectors or embeddings) at the $l$-th layer, with $h_i^{(l)}$ denoting the hidden representation of node $i$; $\hat{A} = \hat{D}^{-1/2}(A + I)\hat{D}^{-1/2}$ is the augmented symmetrically normalized adjacency matrix after self-loop addition, where $\hat{D}$ corresponds to the degree matrix; $\sigma(\cdot)$ is a nonlinear element-wise function, i.e. the activation function, where ReLU is a common choice in the literature, and $W^{(l)}$ is the trainable weight matrix of the $l$-th layer.

### 2.2 Understanding oversmoothing

Oversmoothing occurs as node representations converge to a fixed point with increasing network depth (Li et al., 2018). At that point, node representations contain information about the graph topology, disregarding the input features. This phenomenon was initially observed in Kipf and Welling (2017) and latter analyzed by Li et al. (2018). The latter demonstrated that each node's new representation is a weighted average of its own representa-

tion and that of its neighbors. This mechanism is a form of Laplacian smoothing operation, which allows the node representations within each (graph) cluster, i.e. highly connected subgraph, to become more similar. Increasing similarity of neighboring node embeddings often enhances performance in semi-supervised tasks. At the same time, however, stacking multiple convolutional layers intensifies the smoothing operation, leading to oversmoothing of node representations. Consequently, the hidden representations of all nodes become similar, resulting in information loss.

Oono and Suzuki (2020) have generalized the idea in Li et al. (2018) by considering that the ReLU activation function maps to a positive cone. In their analysis, oversmoothing is explained as the convergence to a subspace rather than to a fixed point. Following a similar approach, Cai and Wang (2020) presented a different perspective to the oversmoothing problem using Dirichlet energy.

It is worth looking closer at the main result in Oono and Suzuki (2020), which estimates the speed of convergence towards a subspace $M$, where the distance between node representations tends to zero. We denote as $d_M(X)$ the distance between the feature vector $X$ and the subspace where oversmoothing is prevalent. For this distance, Oono and Suzuki (2020) prove an interesting property.

**Theorem 1** *(Oono and Suzuki 2020) Let the largest singular value of the weight matrix $W_{lh}$ be $s_{lh}$ and $s_l = \prod_{h=1}^{H_l} s_{lh}$, where $W_{lh}$ is the weight matrix of layer $h$ and $H_l$ is the network's depth, following the notation of the original paper. Then it holds that $d_M(f_l(X)) \leq s_l \lambda d_M(X)$ for any $X \in R^{N \times C}$, where f( ) is the forward pass of a GNN layer (i.e. $\sigma(AXW_{lh})$).*

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 1** *(Oono and Suzuki 2020) 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 $\lambda$ 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 Oono and Suzuki (2020)]. We build upon this result, aiming to show that partially trained GCNs resist oversmoothing and enable deeper architectures.

### 2.3 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 (Glorot & Bengio, 2010). Glorot initialization proposes drawing each element of the weight matrix independently from the same zero-mean distri-

bution, 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 2** *(Bai-Yin's law Bai and Yin 1993) 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:*

$$s_{max}(Z) = \sqrt{N} + \sqrt{n} + o(\sqrt{n}). \tag{2}$$

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 2** *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 2 suggests). Therefore, the largest singular value in Eq. 2 is scaled by a factor of $1 / \sqrt{N}$.*

Corollary 2 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.

### 2.4 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 (Zhou et al., 2023) 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 Kipf and Welling (2017), 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.

## 3 Theoretical analysis

### 3.1 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 Eq. 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 Fig. 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.

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 Eq. 1 to represent the final node embeddings, as follows:

$$\begin{aligned} H^{(L)} &= \hat{A}^{L-1} \cdot X \left( \cdot \prod_{i=1}^{j-1} W^{(i)} \right) \cdot W^{(j)} \cdot \left( \prod_{i=j+1}^{L-1} W^{(i)} \right) \\ &= \hat{A}^{L-1} \cdot X \cdot W^{left} \cdot W^{(j)} \cdot W^{right}, \end{aligned} \tag{3}$$

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.

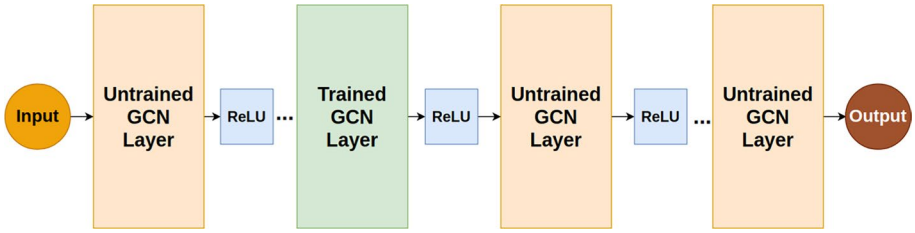



**Fig. 1** Architectural diagram of a partially trained GCN

### 3.2 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 $w^{1,2}_{\cdot,\cdot}$, 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:

$$w^{1,2}_{1,1} = \sum_{i=1}^{d} w^1_{1,i} w^2_{i,1}. \tag{4}$$

Each term of the sum in Eq. 4 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 1** *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 $(\chi^2)$ distribution. Therefore,

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

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

Lemma 1 shows that each term in the sum of Eq. 4 follows a $\chi_1^2$ 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 $\chi_1^2$ distribution.

**Lemma 2** *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.

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

$$M_F(t) = \left(\frac{1/4}{1/4 - t^2}\right)^{1/2}. \tag{5}$$

Eq. 5 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 2 to the equations of the Variance-Gamma distribution we get: i) $E[F] = 0$, and ii) $Var(F) = 4$.

Combining the results of Lemma 1 and Lemma 2, 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 2, 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 Eq. 4 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.

### 3.3 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 Eq. 3, 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 Eq. 3, which means that $W^{right}$ disappears. Equation 3 is transformed as follows:

$$H^{(L)} = \left(B \cdot W^{left}\right) \cdot W^{(j)}. \tag{6}$$

**Lemma 3** *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 3, 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 3 comes from the definition of a multivariate Gaussian random vector and the impact of an affine transformation on its distribution. Lemma 3 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 $\mu$ and a positive-definite covariance matrix $\Sigma$, e.g., $B_i \sim N_p(\mu, \Sigma)$. Lastly, we make the mild assumption that $\mu = 0$.

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 $\hat{\Sigma}$, $\sigma_i$ represents the standard deviation of each feature column in $B$, computed column-wise. That is, $\sigma_i$ 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:

$$C_{lower} \cdot S = \left( \min_j \frac{1}{\sigma_j} \right)^2 S \leq R \leq \left( \max_j \frac{1}{\sigma_j} \right)^2 S = C_{upper} \cdot S, \tag{7}$$

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:

$$\frac{N_{train}}{d} C_{lower} \cdot S \leq R' \leq \frac{N_{train}}{d} C_{upper} \cdot S, \tag{8}$$

where $d$ the width of the trainable layer.

Therefore, the presence of $W^{left}$ in Eq. 6 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 Eq. 7 with Eq. 8 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.

### 3.4 Untrained weight matrices and oversmoothing

Conceptually, as discussed in Sect. 3.3, 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 2 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 1, 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.

# 4 Experiments

## 4.1 Experimental setup

*Datasets:* We conduct experiments on well-known benchmark datasets for node classification: *Cora, CiteSeer, Pubmed*, utilizing the same data splits as in Kipf and Welling (2017), 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 Shchur et al. (2018). To assess the real world applicability of our partial training paradigm, we extended our evaluation to the *Arxiv* dataset (Hu et al., 2020) as well. Dataset statistics can be found in Appendix C.

*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 (Kipf & Welling, 2017). To further validate the proposed method, we also experiment with the GAT model (Velickovic et al., 2018), 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 (Chen et al., 2020). 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 attention 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.

## 4.2 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 Figs. 2 and 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 Sect. 3. 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*

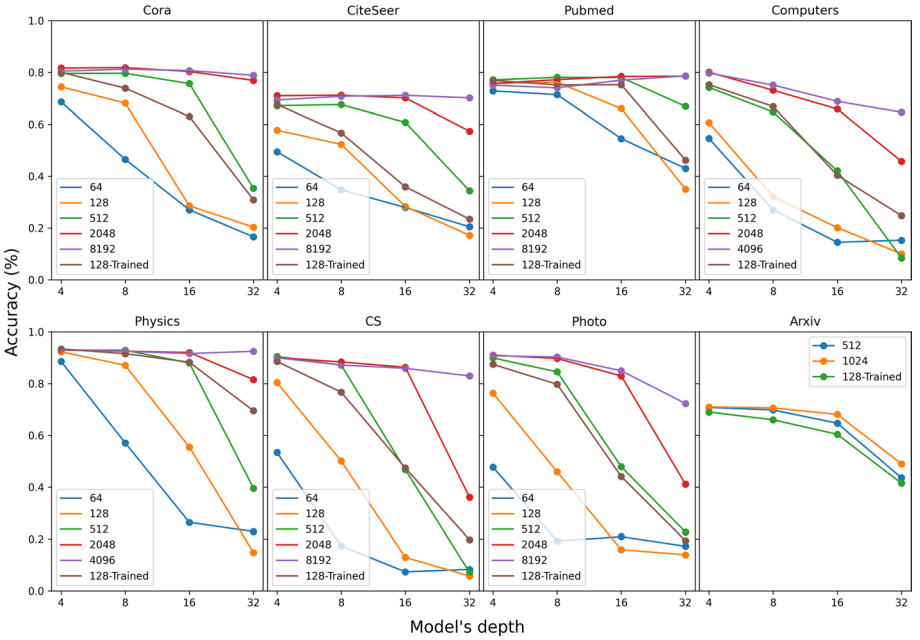



**Fig. 2** Comparison between a fully trained GCN and different configurations (in terms of width) of partially trained GCNs across 8 datasets for varying depth. The trainable layer is always the second

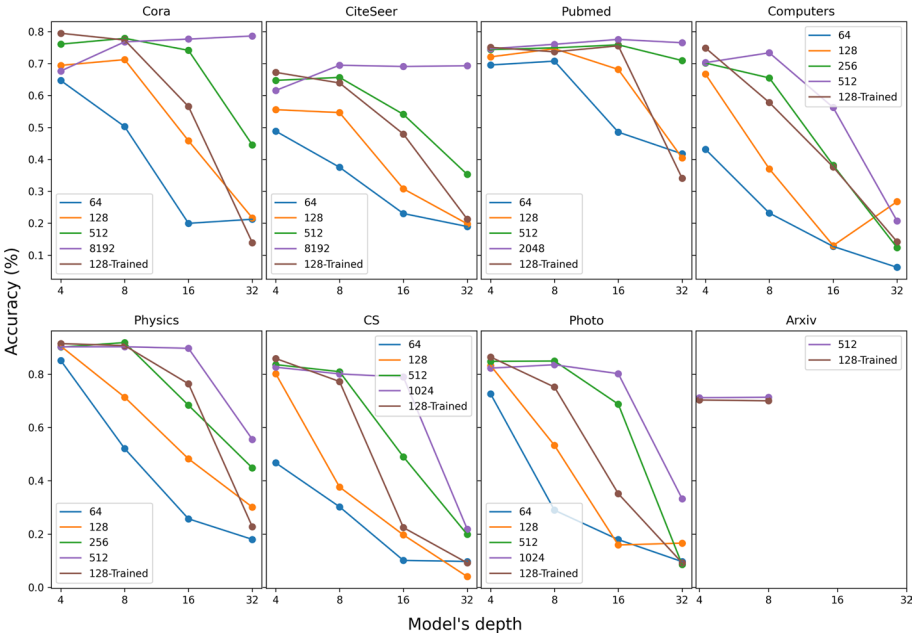



**Fig. 3** Comparison between a fully trained GAT and different configurations (in terms of width) of partially trained GATs across 8 datasets for varying depth. The trainable layer is always the second. *Arxiv* experiments were limited to 8 layers due to hardware constraints

dataset. Across all depths, both partially trained variants yield consistent accuracy improvements 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 results highlight 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, Figs. 4 and 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 in Sect. 4.1, we also experimented with the more advanced GCNII across varying network depths. The results of those experiments, which can be found in Appendix A, 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 without any features. This setup, initially proposed in Zhao and Akoglu (2020), requires leveraging information from distant nodes to create meaningful embeddings for the new item. In order to simulate it,

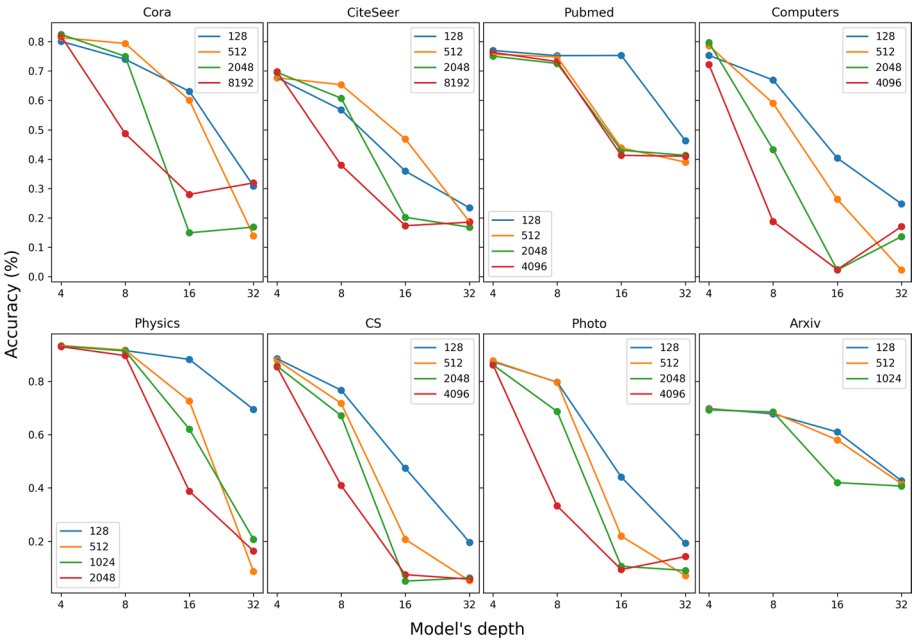



**Fig. 4** Comparison between different configurations (in terms of width) of fully trained GCNs across 8 datasets for varying depth

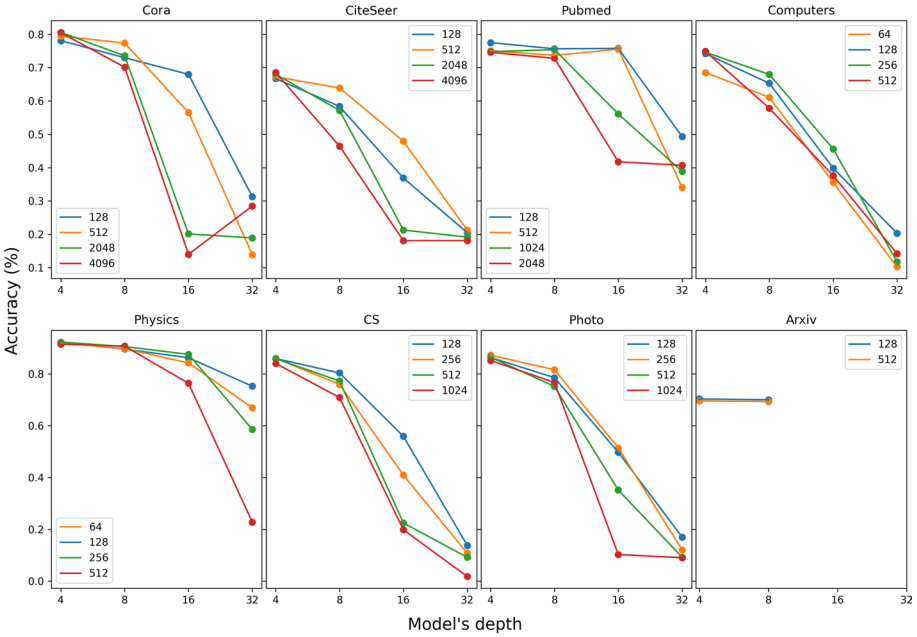



**Fig. 5** Comparison between different configurations (in terms of width) of fully trained GATs across 8 datasets for varying depth. *Arxiv* experiments were limited to 8 layers due to hardware constraints

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 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 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 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 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 experiments. According to Eq. 3, 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 1** Comparison of different model widths of partially trained GCNs and GATs with fully trained ones, in the "cold start" scenario

| 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 |

Bold values indicate the best performing configuration

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

Table 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 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 B.

*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 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 |

Bold values indicate the best performing configuration

**Table 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

Bold values indicate the best performing configuration

| Accuracy (%) & std | | |
|---|---|---|
| Dataset | Model | |
| | 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** |

Table 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-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 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 configuration. In our experiments, wide layers are placed lower in the model, but changing their position does not yield improvements.

**Table 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

| 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 |

Bold values indicate the best performing configuration

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

**Table 5** Peak memory and convergence metrics

| 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 |

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

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).

*Memory and convergence:*

Table 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 Sect. 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.

## 5 Related work

There is limited published work on partially trained graph neural networks, despite the observation made in the original paper by Kipf and Welling (2017). They observed that an untrained GCN can indeed generate meaningful node representations, but did not explore

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

the underlying reasons or mechanisms. They also did not study the effect of model width in such models.

An area of research that is marginally related to ours is the study of Neural Tangent Kernels (NTK), which formulates the relationship between the width of deep neural networks and kernel methods. NTK provides insights into the dynamics of neural network training using a system of differential equations, particularly in the limit of infinite width, where the network behavior becomes increasingly deterministic. While NTK has primarily been applied to deep neural networks, recent work has extended its applicability to GNNs (Du et al., 2019). However, the focus of NTK literature is on the theoretical understanding of graph classification tasks. In contrast, we examine mainly node classification tasks and further establish a connection between the behavior of these models and oversmoothing.

Finally, another relevant area of research is the ongoing efforts to reduce oversmoothing in GNNs. Several approaches have been proposed to address the problem, typically involving modifications to either the model architecture or the structure of the underlying graph (Chen et al., 2020; Rong et al., 2020). Chen et al. (2020) proposed GCNII which use residual connections and identity mapping in order to enable deep architectures and alleviate oversmoothing. That particular approach injects part of the initial information into higher layers of the network, implicitly downgrading the importance of intermediate layers, and forces the model to remain in the local neighborhood around each node. DropEdge (Rong et al., 2020) aimed to address oversmoothing by altering the graph topology, and removing edges at random, in order to slow down message passing. On the other hand, Jumping Knowledge Networks (JKNet) (Xu et al., 2018) introduce layer-wise skip connections and an adaptive aggregation mechanism to combine features from all intermediate layers, in order to avoid feature collapse in deep GNNs. Methods such as FAGCN (Bo et al., 2021) perform an explicit spectral decomposition of node signals and learn separate low and high-frequency filters to preserve discriminative information, while AKGNN (Ju et al., 2022) employs adaptive per-node kernel modules to dynamically balance smoothing and input signal retention. In contrast to those approaches, our method maintains the original graph topology without additional filter modules and uses the prominent GCN architecture, keeping the larger part of the model untrained and showing its capabilities in node classification tasks, even in the demanding "cold start" scenario.

## 6 Conclusion

In this work we proposed 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. As future work, we would like to shed more light to the dataset properties and the number of trainable layers that are required in the partially trained setup.

## Appendix A: Partially trained GCNII

Figure 6 presents the comparison of fully-trained GCNII and partially trained GCNII across eight benchmark datasets. Each subplot reports mean test accuracy over 10 random 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.

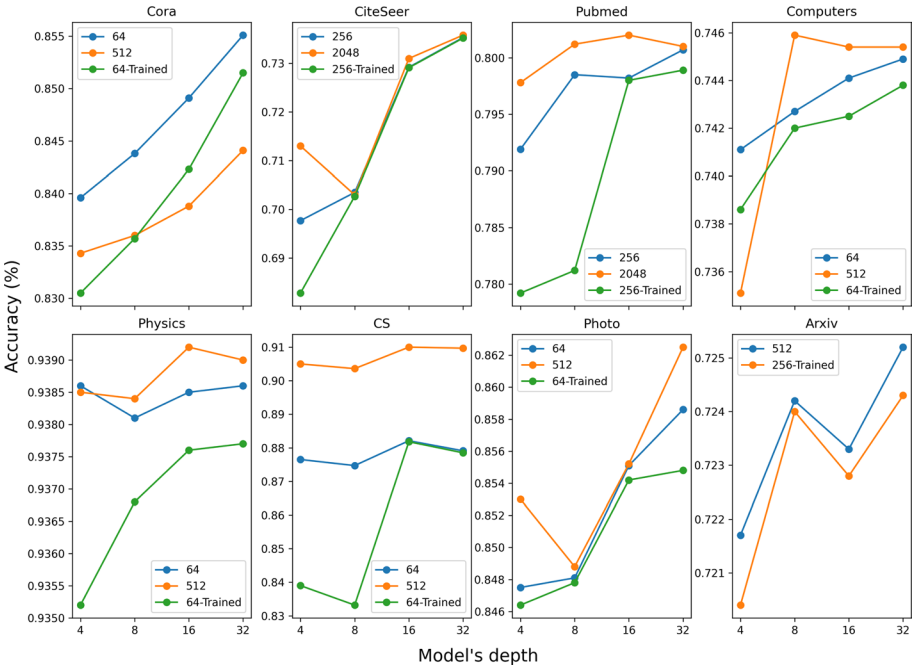



**Fig. 6** Comparison between a fully trained GCNII and different configurations (in terms of width) of partially trained GCNIIs across 8 datasets for varying depth. The trainable layer is always the second. Experiments were limited to the reported widths due to hardware constraints

## Appendix B: Extended Table 2

We present the extended version of Table 2 of the main text.

See Table 6.

**Table 6** 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 |

Bold values indicate the best performing configuration

## Appendix C: Dataset statistics

See Table 7.

**Table 7** The statistics of all datasets used in this work

| Datasets | # Nodes | # Edges | # Classes | # Features |
|---|---|---|---|---|
| Cora | 2708 | 5429 | 7 | 1433 |
| CiteSeer | 3327 | 4732 | 6 | 3703 |
| Pubmed | 19717 | 44338 | 3 | 500 |
| Physics | 34493 | 495924 | 5 | 8415 |
| CS | 18333 | 163788 | 15 | 6805 |
| Photo | 7487 | 119043 | 8 | 745 |
| Computers | 13381 | 245778 | 10 | 767 |
| Arxiv | 169343 | 1166243 | 40 | 128 |

**Acknowledgements** 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).

**Author contributions** D.K. wrote the manuscript, conducted experiments and prepared the figures, and the tables. All authors reviewed the manuscript.

**Funding** Open access funding provided by HEAL-Link Greece.

**Data availability** No datasets were generated or analysed during the current study.

## Declarations

**Conflict of interest** The authors declare no conflict of interest.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

## References

Bai, Z. D., & Yin, Y. Q. (1993). Limit of the smallest eigenvalue of a large dimensional sample covariance matrix. *The Annals of Probability, 21*(3), 1275–1294. https://doi.org/10.1214/aop/1176989118

Bo, D., Wang, X., Shi, C., et al. (2021). 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, pp 3950–3957, https://doi.org/10.1609/AAAI.V35I5.16514.

Cai, C. & Wang, Y. (2020). A note on over-smoothing for graph neural networks. CoRR abs/2006.13318. arxiv:https://arxiv.org/abs/2006.13318

Chen, M., Wei, Z., Huang, Z., et al. (2020). Simple and deep graph convolutional networks. In *Proceedings of the 37th international conference on machine learning, ICML 2020, 13–18 July 2020, virtual event, proceedings of machine learning research*, vol 119. PMLR, pp. 1725–1735, http://proceedings.mlr.press/v119/chen20v.html.

Do, T. H., Nguyen, D. M., Bekoulis, G., et al. (2021). Graph convolutional neural networks with node transition probability-based message passing and dropnode regularization. *Expert Systems with Applications, 174*, Article 114711. https://doi.org/10.1016/j.eswa.2021.114711

Du, S. S., Hou, K., Salakhutdinov, R., et al. (2019). Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In H. M. Wallach, H. Larochelle, A. Beygelzimer, et al (Eds.) *Advances in neural information processing systems 32: Annual conference on neural information processing systems 2019, NeurIPS 2019, December 8–14*, 2019, Vancouver, BC, Canada, pp. 5724–5734, https://proceedings.neurips.cc/paper/2019/hash/663fd3c5144fd10bd5ca6611a9a5b92d-Abstract.html.

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh, D. M. Titterington (Eds.) *Proceedings of the thirteenth international conference on artificial intelligence and statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13–15, 2010, JMLR Proceedings, vol 9*. JMLR.org, pp. 249–256, http://proceedings.mlr.press/v9/glorot10a.html.

Hamilton, W. L., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In I. Guyon, U. von Luxburg, S. Bengio, et al (Eds.) *Advances in neural information processing systems 30: Annual conference on neural information processing systems 2017*, December 4–9, 2017, Long Beach, CA, USA, pp. 1024–1034, https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.

Hu, W., Fey, M., Zitnik, M., et al. (2020). Open graph benchmark: Datasets for machine learning on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, et al (Eds.) *Advances in neural information processing systems 33: Annual conference on neural information processing systems 2020, NeurIPS 2020, December 6–12, 2020, virtual*, https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html.

Ju, M., Hou, S., Fan, Y., et al. (2022). Adaptive kernel graph neural network. In *Thirty-sixth AAAI conference on artificial intelligence, AAAI 2022, thirty-fourth conference on innovative applications of artificial intelligence, IAAI 2022, the twelveth symposium on educational advances in artificial intelligence, EAAI 2022 Virtual Event, February 22–March 1*, 2022. AAAI Press, pp 7051–7058, https://doi.org/10.1609/AAAI.V36I6.20664.

Kelesis, D., Vogiatzis, D., Katsimpras, G., et al. (2023). Reducing oversmoothing in graph neural networks by changing the activation function. In K. Gal, A. Nowé, G. J. Nalepa , et al (Eds.) *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), Frontiers in Artificial Intelligence and Applications, vol 372*. IOS Press, pp. 1231–1238. https://doi.org/10.3233/FAIA230400.

Kipf, T. N. & Welling, M. (2017). 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, https://openreview.net/forum?id=SJU4ayYgl.

Klicpera, J., Bojchevski, A. & Günnemann, S. (2018). Personalized embedding propagation: Combining neural networks on graphs with personalized pagerank. CoRR abs/1810.05997. Arxiv:http://arxiv.org/abs/1810.05997,

Klicpera, J., Groß, J. & Günnemann, S. (2020). Directional message passing for molecular graphs. In *8th International conference on learning representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020*. OpenReview.net, https://openreview.net/forum?id=B1eWbxStPH.

Li, Q., Han, Z., & Wu, X. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In S. A. McIlraith, K. Q. Weinberger (Eds.) *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*. AAAI Press, pp. 3538–3545. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16098.

Liben-Nowell, D. & Kleinberg, J. M. (2003). 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, pp. 556–559, https://doi.org/10.1145/956863.956972.

Luan, S., Zhao, M., Chang, X., et al. (2019). Break the ceiling: Stronger multi-scale deep graph convolutional networks. In H. M. Wallach, H. Larochelle, A. Beygelzimer, et al (Eds.) *Advances in neural information processing systems 32: Annual conference on neural information processing systems 2019, NeurIPS 2019, December 8–14, 2019*, Vancouver, BC, Canada, pp. 10943–10953, https://proceedings.neurips.cc/paper/2019/hash/ccdf3864e2fa9089f9eca4fc7a48ea0a-Abstract.html.

Oono, K. & Suzuki, T. (2020). 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, https://openreview.net/forum?id=S1ldO2EFPr.

Rong, Y., Huang, W., Xu, T., et al. (2020). 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, https://openreview.net/forum?id=Hkx1qkrKPr.

Shchur, O., Mumme, M., Bojchevski, A., et al. (2018). Pitfalls of graph neural network evaluation. CoRR abs/1811.05868. arxiv:https://arxiv.org/abs/1811.05868.

Velickovic, P., Cucurull, G., Casanova, A., et al. (2018). 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, https://openreview.net/forum?id=rJXMpikCZ.

Wu, F., Jr. A. H. S., Zhang, T., et al. (2019). Simplifying graph convolutional networks. In: Chaudhuri K, Salakhutdinov R (eds) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA, Proceedings of machine learning research, vol. 97. PMLR, pp. 6861–6871, http://proceedings.mlr.press/v97/wu19e.html.

Xu, K., Li, C., Tian, Y., et al. (2018). Representation learning on graphs with jumping knowledge networks. In J. G. Dy, A. Krause (Eds.) *Proceedings of the 35th international conference on machine learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10–15, 2018, Proceedings of machine learning research, vol. 80*. PMLR, pp. 5449–5458, http://proceedings.mlr.press/v80/xu18c.html.

Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of an Thropological Research*, pp. 452–473. http://www.jstor.org/stable/3629752.

Zhang, M. & Chen, Y. (2018). Link prediction based on graph neural networks. In S. Bengio, H. M. Wallach , H. Larochelle, et al (Eds.) *Advances in neural information processing systems 31: Annual conference on neural information processing systems 2018, NeurIPS 2018, December 3–8, 2018*, Montréal, Canada, pp. 5171–5181. https://proceedings.neurips.cc/paper/2018/hash/53f0d7c537d99b3824f0f99d62ea2428-Abstract.html.

Zhang, M., Cui, Z., Neumann, M., et al. (2018). An end-to-end deep learning architecture for graph classification. In S. A. McIlraith, K. Q. Weinberger (Eds.) *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*. AAAI Press, pp. 4438–4445, https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17146.

Zhao, L. & Akoglu, L. (2020). Pairnorm: Tackling oversmoothing in gnns. In *8th International conference on learning representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020*. OpenReview.net, https://openreview.net/forum?id=rkecl1rtwB.

Zhou, C., Li, Q., Li, C., et al. (2023). A comprehensive survey on pretrained foundation models: A history from BERT to chatgpt. CoRR abs/2302.09419. https://doi.org/10.48550/ARXIV.2302.09419.

Zhou, K., Huang, X., Zha, D., et al. (2021). Dirichlet energy constrained learning for deep graph neural networks. CoRR abs/2107.02392. arxiv:https://arxiv.org/abs/2107.02392.

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.