Dissecting the Diffusion Process in Linear Graph Convolutional Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph Convolutional Networks (GCNs) have attracted more and more attentions in recent years. A typical GCN layer consists of a linear feature propagation step and a nonlinear transformation step. Recent works show that a linear GCN can achieve comparable performance to the original non-linear GCN while being much more computationally efficient. In this paper, we dissect the feature propagation steps of linear GCNs from a perspective of continuous graph diffusion, and analyze why linear GCNs fail to benefit from more propagation steps. Following that, we propose Decoupled Graph Convolution (DGC) that decouples the terminal time and the feature propagation steps, making it more flexible and capable of exploiting a very large number of feature propagation steps. Experiments demonstrate that our proposed DGC improves linear GCNs by a large margin and makes them competitive with many modern variants of non-linear GCNs.

1 Introduction

Recently, Graph Convolutional Networks (GCNs) have successfully extended the powerful representation learning ability of modern Convolutional Neural Networks (CNNs) to the graph data [8]. A graph convolutional layer typically consists of two stages: linear feature propagation and nonlinear feature transformation. Simple Graph Convolution (SGC) [22] simplifies GCNs by removing the nonlinearities between GCN layers and collapsing the resulting function into a single linear transformation, which is followed by a single linear classification layer and then becomes a linear GCN. SGC can achieve comparable performance to canonical GCNs while being much more computationally efficient and using significantly fewer parameters. Thus, we mainly focus on linear GCNs in this paper.

Although being comparable to canonical GCNs, SGC still suffers from a similar issue as non-linear GCNs, that is, more (linear) feature propagation steps K will degrade the performance catastrophically. This issue is widely characterized as the "over-smoothing" phenomenon. Namely, node features become smoothed out and indistinguishable after too many feature propagation steps [11].

In this work, through a dissection of the diffusion process of linear GCNs, we characterize a fundamental limitation of SGC. Specifically, we point out that its feature propagation step amounts to a very coarse finite difference with a fixed step size $\Delta t = 1$, which results in a large numerical error. And because the step size is fixed, more feature propagation steps will inevitably lead to a large terminal time $T = K \cdot \Delta t \to \infty$ that over-smooths the node features.

To address these issues, we propose Decoupled Graph Convolution (DGC) by decoupling the terminal time T and propagation steps K. In particular, we can flexibly choose a continuous terminal time T for the optimal tradeoff between under-smoothing and over-smoothing, and then fix the terminal time while adopting more propagation steps K. In this way, different from SGC that over-smooths with more propagation steps, our proposed DGC can obtain a more fine-grained finite difference approximation with more propagation steps, which contributes to the final performance both theoretically and empirically. Extensive experiments show that DGC (as a linear GCN) improves over

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

SGC significantly and obtains state-of-the-art results that are comparable to many modern non-linear GCNs. Our main contributions are summarized as follows:

- We investigate SGC by dissecting its diffusion process from a continuous perspective, and characterize why it cannot benefit from more propagation steps.
- We propose Decoupled Graph Convolution (DGC) that decouples the terminal time T and the propagation steps K, which enables us to choose a continuous terminal time flexibly while benefiting from more propagation steps from both theoretical and empirical aspects.
- Experiments show that DGC outperforms canonical GCNs significantly and obtains state-of-the-art (SOTA) results among linear GCNs, which is even comparable to many competitive non-linear GCNs. We think DGC can serve as a strong baseline for the future research.

2 Related Work

Graph convolutional networks (GCNs). To deal with non-Euclidean graph data, GCNs are proposed for direct convolution operation over graph, and have drawn interests from various domains. GCN is firstly introduced for a spectral perspective [27, 8], but soon it becomes popular as a general message passing algorithm in the spatial domain. Many variants have been proposed to improve its performance, such as GraphSAGE [6] with LSTM and GAT with attention mechanism [20].

Over-smoothing issue. GCNs face a fundamental problem compared to standard CNNs, *i.e.*, the over-smoothing problem. Li *et al.* [11] offer a theoretical characterization of over-smoothing based on linear feature propagation. After that, many researchers have tried to incorporate effective mechanisms in CNNs to alleviate over-smoothing. DeepGCNs [10] shows that residual connection and dilated convolution can make GCNs go as deep as CNNs, although increased depth does not contribute much. Methods like APPNP [9] and JKNet [26] avoid over-smoothing by aggregating multi-scale information from the first hidden layer. DropEdge [17] applies dropout to graph edges and find it enables training GCNs with more layers. PairNorm [28] regularizes the feature distance to be close to the input distance, which will not fail catastrophically but still decrease with more layers.

Continuous GCNs. Deep CNNs have been widely interpreted from a continuous perspective, *e.g.,* ResNet [7] as the Euler discretization of Neural ODEs [12, 4]. This viewpoint has recently been borrowed to understand and improve GCNs. GCDE [16] directly extends GCNs to a Neural ODE, while CGNN [23] devises a GCN variant inspired by a new continuous diffusion. Our method is also inspired by the connection between discrete and continuous graph diffusion, but alternatively, we focus on their numerical gap and characterize how it affects the final performance.

3 Dissecting Linear GCNs from Continuous Dynamics

In this section, we make a brief review of SGC [22] in the context of semi-supervised node classification task, and further point out its fundamental limitations.

3.1 Review of Simple Graph Convolution (SGC)

Define a graph as $\mathcal{G} = (\mathcal{V}, \mathbf{A})$, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ denotes the vertex set of n nodes, and $\mathbf{A} \in \mathbb{R}^{n\times n}$ is an adjacency matrix where a_{ij} denotes the edge weight between node v_i and v_j. The degree matrix $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ of $\mathbf{A}$ is a diagonal matrix with its i-th diagonal entry as $d_i = \sum_j a_{ij}$. Each node v_i is represented by a d-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^d$, and we denote the feature matrix as $\mathbf{X} \in \mathbb{R}^{n\times d} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$. Each node belongs to one out of C classes, denoted by a one-hot vector $\mathbf{y}_i \in \{0, 1\}^C$. In node classification problems, only a subset of nodes $\mathcal{V}_l \subset \mathcal{V}$ are labeled and we want to predict the labels of the rest nodes $\mathcal{V}_u = \mathcal{V} \backslash \mathcal{V}_l$.

SGC shows that we can obtain similar performance with a simplified GCN,

$$\hat{\mathbf{Y}}_{\mathrm{SGC}} = \mathrm{softmax}\left(\mathbf{S}^K \mathbf{X} \mathbf{\Theta}\right), \tag{1}$$

which pre-processes the node features $\mathbf{X}$ with K linear propagation steps, and then applies a linear classifier with parameter $\mathbf{\Theta}$. Specifically, at the step k, each feature $\mathbf{x}_i$ is computed by aggregating features in its local neighborhood, which can be done in parallel over the whole graph for K steps,

$$\mathbf{X}^{(k)} \leftarrow \mathbf{S}\mathbf{X}^{(k-1)}, \text{ where } \mathbf{S} = \widetilde{\mathbf{D}}^{-\frac{1}{2}} \widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-\frac{1}{2}} \quad \Longrightarrow \quad \mathbf{X}^{(K)} = \mathbf{S}^K \mathbf{X}. \tag{2}$$

Here $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix augmented with the self-loop $\mathbf{I}$, $\widetilde{\mathbf{D}}$ is the degree matrix of $\widetilde{\mathbf{A}}$, and $\mathbf{S}$ denotes the symmetrically normalized adjacency matrix. This step exploits the local graph structure to smooth out the noise in each node.

At last, SGC applies a multinomial logistic regression (*a.k.a.* softmax regression) with parameter $\mathbf{\Theta}$ to predict the node labels $\hat{\mathbf{Y}}_{\mathrm{SGC}}$ from the node features of the last propagation step $\mathbf{X}^{(K)}$:

$$\hat{\mathbf{Y}}_{\mathrm{SGC}} = \mathrm{softmax}\left(\mathbf{X}^{(K)}\mathbf{\Theta}\right). \tag{3}$$

Because both the feature propagation ($\mathbf{S}^K\mathbf{X}$) and classification ($\mathbf{X}^{(K)}\mathbf{\Theta}$) steps are linear, SGC is essentially a linear version of GCN that only relies on linear features from the input.

3.2 Equivalence between SGC and Graph Heat Equation

Previous analysis of linear GCNs focuses on their asymptotic behavior as propagation steps $K \to \infty$ (discrete), known as the over-smoothing phenomenon [11]. In this work, we instead provide a novel *non-asymptotic* characterization of linear GCNs from the corresponding *continuous dynamics*, graph heat equation [5]. A key insight is that we notice that the propagation of SGC can be seen equivalently as a (coarse) numerical discretization of the graph diffusion equation, as we show below.

Graph Heat Equation (GHE) is a well-known generalization of the heat equation on graph data, which is widely used to model graph dynamics with applications in spectral graph theory [5], time series [13], combinational problems [14], *etc*. In general, GHE can be formulated as follows:

$$\begin{cases} \frac{d\mathbf{X}_t}{dt} & = -\mathbf{L}\mathbf{X}_t, \\ \mathbf{X}_0 & = \mathbf{X}, \end{cases} \tag{4}$$

where $\mathbf{X}_t$ $(t \geq 0)$ refers to the evolved input features at time t, and $\mathbf{L}$ refers to the graph Laplacian matrix. Here, for the brevity of analysis, we take the symmetrically normalized graph Laplacian for the augmented adjacency $\widetilde{\mathbf{A}}$ and overload the notation as $\mathbf{L} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\left(\widetilde{\mathbf{D}} - \widetilde{\mathbf{A}}\right)\widetilde{\mathbf{D}}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{S}$.

As GHE is a continuous dynamics, in practice we need to rely on numerical methods to solve it. We find that SGC can be seen as a coarse finite difference of GHE. Specifically, we apply the forward Euler method to Eq. (4) with an interval Δt:

$$\hat{\mathbf{X}}_{t+\Delta t} = \hat{\mathbf{X}}_t - \Delta t\mathbf{L}\hat{\mathbf{X}}_t = \hat{\mathbf{X}}_t - \Delta t(\mathbf{I} - \mathbf{S})\hat{\mathbf{X}}_t = \left[(1 - \Delta t)\mathbf{I} + \Delta t\mathbf{S}\right]\hat{\mathbf{X}}_t. \tag{5}$$

By involving the update rule for K forward steps, we will get the final features $\hat{\mathbf{X}}_T$ at the terminal time $T = K \cdot \Delta t$:

$$\hat{\mathbf{X}}_T = [\mathbf{S}^{(\Delta t)}]^K\mathbf{X}, \text{ where } \mathbf{S}^{(\Delta t)} = (1 - \Delta t)\mathbf{I} + \Delta t\mathbf{S}. \tag{6}$$

Comparing to Eq. (2), we can see that the Euler discretization of GHE becomes SGC when the step size $\Delta t = 1$. Specifically, the diffusion matrix $\mathbf{S}^{(\Delta t)}$ reduces to the SGC diffusion matrix $\mathbf{S}$ and the final node features, $\hat{\mathbf{X}}_T$ and $\mathbf{X}^{(K)}$, become equivalent. Therefore, SGC with K propagation steps is essentially a finite difference approximation to GHE with K forward steps (step size $\Delta t = 1$ and terminal time $T = K$).

3.3 Revealing the Fundamental Limitations of SGC

Based on the above analysis, we theoretically characterize several fundamental limitations of SGC: feature over-smoothing, large numerical errors and large learning risks. All proofs are in Appendix.

Theorem 1 (Oversmoothing from a spectral view)**.** *Assume that the eigendecomposition of the Laplacian matrix as* $\mathbf{L} = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$*, with eigenvalues* λ_i *and eigenvectors* $\mathbf{u}_i$*. Then, the heat equation (Eq. (4)) admits a closed-form solution at time t, known as the heat kernel* $\mathbf{H}_t = e^{-t\mathbf{L}} = \sum_{i=1}^{n} e^{-\lambda_i t}\mathbf{u}_i\mathbf{u}_i^\top$*. As* $t \to \infty$*,* $\mathbf{H}_t$ *asymptotically converges to a non-informative equilibrium as* $t \to \infty$*, due to the non-trivial (i.e., positive) eigenvalues vanishing:*

$$\lim_{t\to\infty} e^{-\lambda_i t} = \begin{cases} 0, & \textit{if } \lambda_i > 0 \\ 1, & \textit{if } \lambda_i = 0 \end{cases}, \; i = 1, \ldots, n. \tag{7}$$

Remark 1. In SGC, $T = K \cdot \Delta t = K$. Thus according to Theorem 1, a large number of propagation steps $K \to \infty$ will inevitably lead to over-smoothed non-informative features.

Theorem 2 (Numerical errors)**.** *For the initial value problem in Eq. (4) with finite terminal time T, the numerical error of the forward Euler method in Eq. (5) with K steps can be upper bounded by*

$$\left\|\mathbf{e}_T^{(K)}\right\| \leq \frac{T\|\mathbf{L}\|\|\mathbf{X}_0\|}{2K}\left(e^{T\|\mathbf{L}\|}-1\right). \tag{8}$$

Remark 2. Since $T = K$ in SGC, the upper bound reduces to $c \cdot \left(e^{T\|\mathbf{L}\|}-1\right)$ (c is a constant). We can see that the numerical error will increase exponentially with more propagation steps.

Theorem 3 (Learning risks)**.** *Consider a simple linear regression problem $(\mathbf{X}, \mathbf{Y})$ on graph, where the observed input features $\mathbf{X}$ are generated by corrupting the ground truth features $\mathbf{X}_c$ with the following inverse graph diffusion with time T^* :*

$$\frac{d\widetilde{\mathbf{X}}_t}{dt} = \mathbf{L}\widetilde{\mathbf{X}}_t, \quad \textit{where } \widetilde{\mathbf{X}}_0 = \mathbf{X}_c \textit{ and } \widetilde{\mathbf{X}}_{T^*} = \mathbf{X}. \tag{9}$$

Denote the population risk with ground truth features as $R(\mathbf{W}) = \mathbb{E}\left\|\mathbf{Y} - \mathbf{X}_c\mathbf{W}\right\|^2$ and that of Euler method applied input $\mathbf{X}$ (Eq. (5)) as $\hat{R}(\mathbf{W}) = \mathbb{E}\left\|\mathbf{Y} - \left[\mathbf{S}^{(\Delta t)}\right]^K \mathbf{X}\mathbf{W}\right\|^2$. Supposing that $\mathbb{E}\|\mathbf{X}_c\|^2 = M < \infty$, we have the following upper bound:

$$\hat{R}(\mathbf{W}) \leq R(\mathbf{W}) + \|\mathbf{W}\|^2\left(M\left\|e^{T^\star\mathbf{L}}\right\|^2\left\|e^{-T^\star\mathbf{L}} - e^{-\hat{T}\mathbf{L}}\right\|^2 + \mathbb{E}\left\|\mathbf{e}_{T^\star}^{(K)}\right\|^2\right). \tag{10}$$

Remark 3. Following Theorem 3, we can see that the upper bound can be minimized by finding an optimal terminal time such that $\hat{T} = T^\star$ and minimizing the numerical error $\left\|\mathbf{e}_{T^\star}^{(K)}\right\|$. While SGC fixes the step size $\Delta t = 1$, thus T and K are coupled together, which makes it less flexible to minimize the risk in Eq. (10).

4 The Proposed Decoupled Graph Convolution (DGC)

In this section, we introduce our proposed Decoupled Graph Convolution (DGC) and discuss how it overcomes the above limitations of SGC.

4.1 Formulation

Based on the analysis in Section 3.3, we need to resolve the coupling between propagation steps K and terminal time T caused by the fixed time interval $\Delta t = 1$. Therefore, we regard the terminal time T and the propagation steps K as two free hyper-parameters in the numerical integration via a flexible time interval. In this way, the two parameters can play different roles and cooperate together to attain better results: 1) we can flexibly choose T to tradeoff between under-smoothing and over-smoothing to find a sweet spot for each dataset; and 2) given an optimal terminal time T, we can also flexibly increase the propagation steps K for better numerical precision with $\Delta t = T/K \to 0$. In practice, a moderate number of steps is sufficient to attain the best classification accuracy, hence we can also choose a minimal K among the best for computation efficiency.

Formally, we propose our Decoupled Graph Convolution (DGC) as follows:

$$\hat{\mathbf{Y}}_{\text{DGC}} = \text{softmax}\left(\hat{\mathbf{X}}_T\mathbf{\Theta}\right), \text{ where } \hat{\mathbf{X}}_T = \text{ode\_int}(\mathbf{X}, \Delta t, K). \tag{11}$$

Here $\text{ode\_int}(\mathbf{X}, \Delta t, K)$ refers to the numerical integration of the graph heat equation that starts from $\mathbf{X}$ and runs for K steps with step size Δt . Here, we consider two numerical schemes: the forward Euler method and the Runge-Kutta (RK) method.

DGC-Euler. As discussed previously, the forward Euler gives an update rule as in Eq. (5). With terminal time T and step size $\Delta t = T/K$, we can obtain $\hat{\mathbf{X}}_T$ after K propagation steps:

$$\hat{\mathbf{X}}_T = \left[\mathbf{S}^{(T/K)}\right]^K \mathbf{X}, \text{ where } \mathbf{S}^{(T/K)} = (1 - T/K) \cdot \mathbf{I} + (T/K) \cdot \mathbf{S}. \tag{12}$$

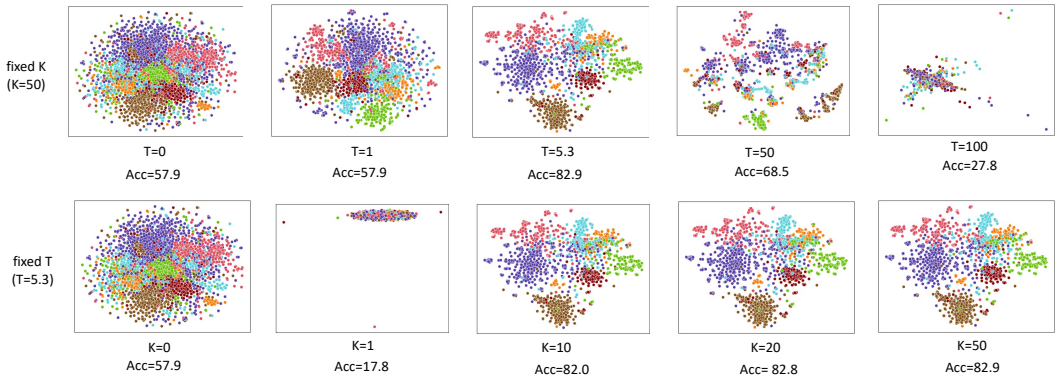

Figure 1: Input feature visualization of our DGC-Euler model with t-SNE [19] on the Cora dataset. Each point represents a node in the graph and its color denotes the class of the node.

DGC-RK. Alternatively, we can apply higher-order finite difference methods to achieve better numerical precision, at the cost of more function evaluations at intermediate points. One classical method is the 4th-order Runge-Kutta (RK) method, which proceeds with

$$\hat{\mathbf{X}}_{t+\Delta t} = \hat{\mathbf{X}}_t + \frac{1}{6}\Delta t\left(\mathbf{R}_1 + 2\mathbf{R}_2 + 2\mathbf{R}_3 + \mathbf{R}_4\right) \triangleq \mathbf{S}_{\mathrm{RK}}^{(\Delta t)}\hat{\mathbf{X}}_t, \tag{13}$$

where

$$\mathbf{R}_1 = \hat{\mathbf{X}}_k,\ \mathbf{R}_2 = \hat{\mathbf{X}}_k - \frac{1}{2}\Delta t\mathbf{L}\mathbf{R}_1,\ \mathbf{R}_3 = \hat{\mathbf{X}}_k - \frac{1}{2}\Delta t\mathbf{L}\mathbf{R}_2,\ \mathbf{R}_4 = \hat{\mathbf{X}}_k - \Delta t\mathbf{L}\mathbf{R}_3. \tag{14}$$

Replacing the propagation matrix $\mathbf{S}^{(T/K)}$ in DGC-Euler with the RK-matrix $\mathbf{S}_{\mathrm{RK}}^{(T/K)}$, we can get a 4th-order model, namely DGC-RK, whose numerical error can be reduced to $O(1/K^4)$ order.

Remark. In GCN [8], a self-loop $\mathbf{I}$ is heuristically introduced in the adjacency matrix $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ to prevent numerical instability with more steps K. Here, we notice that the DGC-Euler diffusion matrix $\mathbf{S}^{(\Delta t)} = (1-\Delta t)\mathbf{I} + \Delta t\mathbf{S}$ naturally incorporates the self-loop $\mathbf{I}$ into the diffusion process as a momentum term, where Δt flexibly tradeoffs information from the self-loop and the neighborhood. Therefore, in DGC, we can also remove the self-loop from $\widetilde{\mathbf{A}}$ and increasing K is still numerically stable with fixed T. We name the resulting model as DGC-sym with symmetrically normalized adjacency matrix $\mathbf{S}_{\mathrm{sym}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, which aligns with the canonical normalized graph Laplacian $\mathbf{L}_{\mathrm{sym}} = \mathbf{D}^{-\frac{1}{2}}\left(\mathbf{D} - \mathbf{A}\right)\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{S}_{\mathrm{sym}}$ in the spectral graph theory [5]. Comparing the two Laplacians from a spectral perspective, $\mathbf{L}$ has a smaller spectral range than $\mathbf{L}_{\mathrm{sym}}$ [22]. According to Theorem 2, $\mathbf{L}$ will have a faster convergence rate of numerical error.

4.2 Verifying the Benefits of DGC

Here we demonstrate the advantages of DGC both theoretically and empirically.

Theoretical benefits. Revisiting Section 3.3, DGC can easily alleviate the limitations of existing linear GCNs shown in Remarks 1, 2, 3 by decoupling T and K.

- For **Theorem 1**, by choosing a fixed terminal time T with optimal tradeoff, increasing the propagation steps K in DGC will not lead to over-smoothing as in SGC;
- For **Theorem 2**, with T is fixed, using more propagation steps ($K \to \infty$) in DGC will help minimize the numerical error $\left\|\mathbf{e}_T^{(K)}\right\|$ with a smaller step size $\Delta t = T/K \to 0$;
- For **Theorem 3**, by combining a flexibly chosen optimal terminal time T^* and minimal numerical error with a large number of steps K, we can get minimal learning risks.

Empirical evidence. To further provide an intuitive understanding of DGC, we visualize the propagated input features of our proposed DGC-Euler on the Cora dataset in Figure 1. The first row shows that there exists an optimal terminal time T^* for each dataset with the best feature separability (*e.g.*,

Table 1: A comparison of propagation rules. Here $\mathbf{X}^{(k)} \in \mathcal{X}$ represents input features after k feature propagation steps and $\mathbf{X}^{(0)} = \mathbf{X}$; $\mathbf{H}^{(k)}$ denotes the hidden features of non-linear GCNs at layer k; $\mathbf{W}$ denotes the weight matrix; σ refers to a activation function; α, β are coefficients.

Method	Type	Propagation rule
GCN [8]	Non-linear	$\mathbf{H}^{(k)} = \sigma\left(\mathbf{S}\mathbf{H}^{(k-1)}\mathbf{W}^{(k-1)}\right)$
APPNP [9]	Non-linear	$\mathbf{H}^{(k)} = (1-\alpha)\mathbf{S}\mathbf{H}^{(k-1)} + \alpha\mathbf{H}^{(0)}$
CGNN [23]	Non-linear	$\mathbf{H}^{(k)} = (1-\alpha)\mathbf{S}\mathbf{H}^{(k-1)}\mathbf{W} + \mathbf{H}^{(0)}$
SGC [22]	Linear	$\mathbf{X}^{(k)} = \mathbf{S}\mathbf{X}^{(k-1)})$
DGC-Euler (ours)	Linear	$\mathbf{X}^{(k)} = (1 - T/K) \cdot \mathbf{X}^{(k-1)} + (T/K) \cdot \mathbf{S}\mathbf{X}^{(k-1)}$

5.3 for Cora). Either a smaller T (under-smooth) or a larger T (over-smooth) will mix the features up and make them more indistinguishable, which eventually leads to lower accuracy. From the second row, we can see that, with fixed optimal T, too large step size Δt (*i.e.,* too small propagation steps K) will lead to feature collapse, while gradually increasing the propagation steps K makes the nodes of different classes more separable and improve the overall accuracy.

4.3 Discussions

To highlight the difference of DGC to previous methods, we summarize their propagation rules in Table 1. For non-linear methods, GCN [8] uses the canonical propagation rule which has the oversmoothing issue, while APPNP [9] and CGNN [23] address it by further aggregating the initial hidden state $\mathbf{H}^{(0)}$ repeatedly at each step. In particular, we emphasize that our DGC-Euler is different from APPNP in terms of the following aspects: 1) DGC-Euler is a linear model and propagates on the input features $\mathbf{X}^{(k-1)}$, while APPNP is non-linear and propagates on non-linear embedding $\mathbf{H}^{(k-1)}$; 2) at each step, APPNP aggregates features from the *initial* step $\mathbf{H}^{(0)}$, while DGC-Euler aggregates features from the *last* step $\mathbf{X}^{(k-1)}$; 3) APPNP aggregates a large amount $(1-\alpha)$ of the propagated features $\mathbf{S}\mathbf{H}^{(k-1)}$ while DGC-Euler only takes a small step Δt (T/K) towards the new features $\mathbf{S}\mathbf{X}^{(k-1)}$. For linear methods, SGC has several fundamental limitations as analyzed in Section 3.3, while DGC addresses them by flexible and fine-grained numerical integration of the propagation process.

Our dissection of linear GCNs also suggests a different understanding of the over-smoothing problem. As shown in Theorem 1, over-smoothing is an inevitable phenomenon of (canonical) GCNs, while we can find a terminal time to achieve an optimal tradeoff between under-smoothing and over-smoothing. However, we cannot expect more layers can bring more profit if the terminal time goes to infinity, that is, the benefits of more layers can only be obtained under a proper terminal time.

5 Experiments

In this section, we conduct a comprehensive analysis on DGC and compare it against both linear and non-linear GCN variants on a collection of benchmark datasets.

5.1 Performance on Semi-supervised Node Classification

Setup. For semi-supervised node classification, we use three standard citation networks, Cora, Citeseer, and Pubmed [18] and adopt the standard data split as in [8, 20, 25, 24, 16]. Here we compare our DGC against several representative non-linear and linear methods that also adopts the standard data split. For non-linear GCNs, we include 1) classical baselines like GCN [8], GAT [21], GraphSAGE [6], APPNP [9] and JKNet [26]; 2) spectral methods using graph heat kernel [25, 24]; and 3) continuous GCNs [16, 23]. For linear methods, we present the results of Label Propagation [29], DeepWalk [15], SGC (linear GCN) [22] as well as its regularized version SGC-PairNorm [28]. For DGC, we adopt the Euler scheme, *i.e.,* DGC-Euler (Eq. (12)) by default for simplicity. We report results averaged over 10 random runs. Dataset statistics and training details are in Appendix.

We compare DGC against both linear and non-linear baselines for the semi-supervised node classification task, and the results are shown in Table 2.

Table 2: Test accuracy (%) of semi-supervised node classification on citation networks.

Type	Method	Cora	Citeseer	Pubmed
Non-linear	GCN [8]	81.5	70.3	79.0
	GAT [20]	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
	GraphSAGE [6]	82.2	71.4	75.8
	JKNet [26]	81.1	69.8	78.1
	APPNP [9]	83.3	71.8	80.1
	GWWN [25]	82.8	71.7	79.1
	GraphHeat [24]	83.7	72.5	80.5
	CGNN [23]	84.2 ± 0.6	71.8 ± 0.7	76.8 ± 0.6
	GCDE [16]	83.8 ± 0.5	72.5 ± 0.5	79.9 ± 0.3
Linear	Label Propagation [29]	45.3	68.0	63.0
	DeepWalk [15]	70.7 ± 0.6	51.4 ± 0.5	76.8 ± 0.6
	SGC [22]	81.0 ± 0.0	71.9 ± 0.1	78.9 ± 0.0
	SGC-PairNorm [28]	81.1	70.6	78.2
	DGC (ours)	$\mathbf{83.5 \pm 0.0}$	$\mathbf{74.5 \pm 0.2}$	$\mathbf{80.2 \pm 0.1}$

Table 3: Test accuracy (%) of fully-supervised node classification on citation networks.

Type	Method	Cora	Citeseer	Pubmed
Non-linear	GCN [8]	85.77	73.58	88.13
	GAT [20]	86.37	74.32	87.62
	JK-MaxPool [26]	89.6	77.7	-
	JK-Concat [26]	89.1	78.3	-
	JK-LSTM [26]	85.8	74.7	-
	APPNP [9]	90.21	79.8	86.29
Linear	SGC [22]	85.82	78.08	83.27
	DGC (ours)	$\mathbf{88.2 \pm 0.1}$	$\mathbf{79.0 \pm 0.2}$	$\mathbf{88.7 \pm 0.0}$

DGC v.s. linear methods. We can see that DGC shows significant improvement over previous linear methods across three datasets. In particular, compared to SGC (previous SOTA methods), DGC obtains 83.5 *v.s.* 81.0 on Cora, 74.5 *v.s.* 71.9 on Citeseer and 80.2 *v.s.* 78.9 on Pubmed. This shows that in real-world datasets, a flexible and fine-grained integration by decoupling T and K indeed helps improve the classification accuracy of SGC by a large margin.

DGC v.s. non-linear models. Table 2 further shows that DGC, as a linear model, even outperforms many non-linear GCNs on semi-supervised tasks. First, DGC improves over classical GCNs like GCN [8], GAT [20] and GraphSAGE [6] by a large margin. Also, DGC is comparable to, and sometimes outperforms, many modern non-linear GCNs. For example, DGC shows a clear advantage over multi-scale methods like JKNet [26] and APPNP [9]. DGC is also comparable to the spectral methods based on graph heat kernel, *e.g.,* GWWN [25], GraphHeat [24], while being much more efficient as a simple linear model. Besides, compared to non-linear continuous models like GCDE [16] and CGNN [23], DGC also achieves comparable accuracy only using a simple linear dynamic.

5.2 Performance on Fully-supervised Node Classification

Setup. For fully-supervised node classification, we also use the three citation networks, Cora, Citeseer and Pubmed, but instead randomly split the nodes in three citation networks into 60%, 20% and 20% for training, validation and testing, following the previous practice in [26]. Here, we include the baselines that also have reported results on the fully supervised setting, such as GCN [8], GAT [20] (reported baselines in [26]), and the three variants of JK-Net: JK-MaxPool, JK-Concat and JK-LSTM [26]. Besides, we also reproduce the result of APPNP [9] for a fair comparison. Dataset statistics and training details are described in Appendix.

Results. The results of the fully-supervised semi-classification task are basically consistent with the semi-supervised setting. As a linear method, DGC not only improves the state-of-the-art linear GCNs by a large margin, but also outperforms GCN [8], GAT [20] significantly. Beside, DGC is

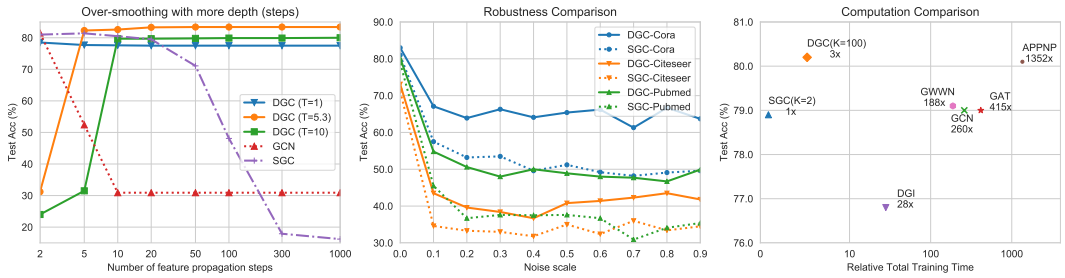

Figure 2: Left: test accuracy (%) with increasing feature propagation steps on Cora. Middle: comparison of robustness under different noise scales σ on three citation networks. Right: a comparison of relative total training time for 100 epochs on the Pubmed dataset, where the absolute total time of SGC (baseline) is 80$\pm$2 ms on a NVIDIA GeForce RTX 3090 GPU.

also comparable to multi-scale methods like JKNet [26] and APPNP [9], showing that a good linear model like DGC is also competitive for fully-supervised tasks.

5.3 Performance on Large Scale Datasets

Setup. More rigorously, we also conduct the comparison on a large scale node classification dataset, the Reddit networks [6]. Following SGC [22], we adopt the inductive setting, where we use the subgraph of training nodes as training data and use the whole graph for the validation/testing data. For a fair comparison, we use the same training configurations as SGC [22] and include its reported baselines, such as GCN [8], FastGCN [3], three variants of GraphSAGE [6], and RandDGI (DGI with randomly initialized encoder) [21]. We also include APPNP [9] for a comprehensive comparison.

Results. We can see DGC still achieves the best accuracy among linear methods and improve 0.9% accuracy over SGC. Meanwhile, it is superior to the three variants of GraphSAGE as well as APPNP. Thus, DGC is still the state-of-the-art linear GCNs and competitive against nonlinear GCNs on large scale datasets.

Table 4: Test accuracy (%) comparison with inductive methods on on a large scale dataset, Reddit. Reported results are averaged over 10 runs. OOM: out of memory.

Type	Method	Acc.
Non-linear	GCN [8]	OOM
	FastGCN [3]	93.7
	GraphSAGE-GCN [6]	93.0
	GraphSAGE-mean [6]	95.0
	GraphSAGE-LSTM [6]	95.4
	APPNP [9]	95.0
Linear	RandDGI [21]	93.3
	SGC [22]	94.9
	DGC (ours)	**95.8**

5.4 Empirical Understandings of DGC

Setup. Here we further provide a comprehensive analysis of DGC. First, we compare its over-smoothing behavior and computation time against previous methods. Then we analyze several factors that affect the performance of DGC, including the Laplacian matrix $\mathbf{L}$, the numerical schemes and the terminal time T. Experiments are conducted on the semi-supervised learning tasks and we adopt DGC-Euler with the default hyper-parameters unless specified.

Over-smoothing with increasing steps. In the left plot of Figure 2, we compare different GCNs with increasing model depth (non-linear GCNs) or propagation steps (linear GCNs) from 2 to 1000. Baselines include SGC [22], GCN [8], and our DGC with three different terminal time T (1, 5.3, 10). First we notice that SGC and GCN fail catastrophically when increasing the depth, which is consistent with the previously observed over-smoothing phenomenon. Instead, all three DGC variants can benefit from increased steps. Nevertheless, the final performance will degrade if the terminal time is either too small ($T = 1$, under-smoothing) or too large ($T = 10$, over-smoothing). DGC enables us to flexibly find the optimal terminal time ($T = 5.3$). Thus, we can obtain the optimal accuracy with an optimal tradeoff between under-smoothing and over-smoothing.

Robustness to feature noise. In real-world applications, there are plenty of noise in the collected node attributes, thus it is crucial for GCNs to be robust to input noise [2]. Therefore, we compare the

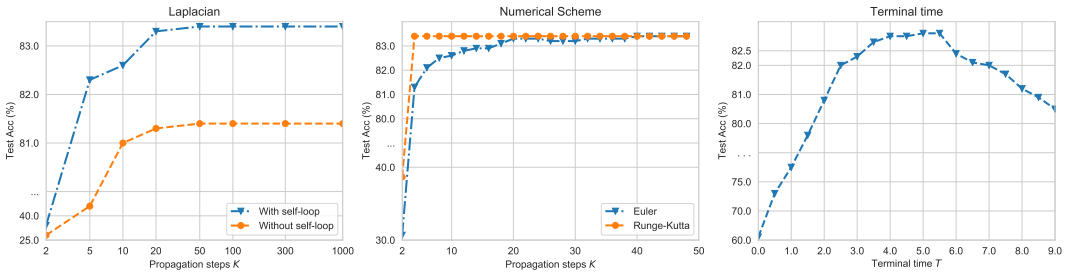

Figure 3: Algorithmic analysis of our proposed DGC. Left: test accuracy (%) of two kinds of Laplacian, $\mathbf{L} = \mathbf{I} - \mathbf{S}$ (with self-loop) and $\mathbf{L}_{\text{sym}} = \mathbf{I} - \mathbf{S}_{\text{sym}}$ (without self-loop), with increasing steps K and fixed time T on Cora. Middle: test accuracy (%) of two numerical schemes, Euler and Runge-Kutta, with increasing steps K and fixed T under fixed terminal time on Cora. Right: test accuracy (%) with varying terminal time T and fixed steps K on Cora.

robustness of SGC and DGC against Gaussian noise added to the input features, where σ stands for the standard deviation of the noise. Figure 2 (middle) shows that DGC is significantly more robust than SGC across three citation networks, and the advantage is clearer on larger noise scales.

Computation time. In practice, linear GCNs can accelerate training by pre-processing features with all propagation steps and storing them for the later model training. Since pre-processing costs much fewer time than training (<5% in SGC), linear GCNs could be much faster than non-linear ones. As shown in Figure 2 (right), DGC is slightly slower (3×) than SGC, but DGC achieves much higher accuracy. Even so, DGC is still much faster than non-linear GCNs (>100×).

Graph Laplacian. As shown in Figure 3 (left), in DGC, both the two Laplacians, $\mathbf{L}$ (with self-loop) and $\mathbf{L}_{\text{sym}}$ (without self-loop), can consistently benefit from more propagation steps without leading to numerical issues. Further comparing the two Laplacians, we can see that the augmented Laplacian $\mathbf{L}$ obtains higher test accuracy than the canonical Laplacian $\mathbf{L}_{\text{sym}}$ and requires fewer propagation steps K to obtain good results, which could also be understood from our analysis in Section 3.3.

Numerical scheme. By comparing different numerical schemes in Figure 3 (middle), we find that the Runge-Kutta method demonstrates better accuracy than the Euler method with a small K. Nevertheless, as K increases, the difference gradually vanishes. Thus, the Euler method is sufficient for DGC to achieve good performance and it is more desirable in term of its simplicity and efficiency.

Terminal time T. In Figure 3 (right), we compare the test accuracy with different terminal time T. We show that indeed, in real-world datasets, there exists a sweet spot that achieves the optimal tradeoff between under-smoothing and over-smoothing. In Table 5, we list the best terminal time and number of steps found by hyperopt [1] on two large graph datasets. We can see that T is almost consistent across different Laplacians on each dataset. This observation suggests that the optimal terminal time T^* is an intrinsic property of the dataset.

Table 5: Optimal configurations on the transductive task, Pubmed, and the inductive task, Reddit.

Dataset	Lap	T	K	Acc
Pubmed	$\mathbf{I} - \mathbf{S}$	5.1	4	**80.2**
	$\mathbf{I} - \mathbf{S}_{\text{sym}}$	5.2	580	79.2
Reddit	$\mathbf{I} - \mathbf{S}$	2.7	24	95.5
	$\mathbf{I} - \mathbf{S}_{\text{sym}}$	2.6	26	**95.8**

6 Conclusions

In this paper, we have proposed Decoupled Graph Convolution (DGC), which improves significantly over previous linear GCNs through decoupling the terminal time and feature propagation steps from a continuous perspective. Experiments show that our DGC is competitive with many modern variants of non-linear GCNs while being much more computationally efficient with much fewer parameters to learn.

Our findings suggest that, unfortunately, current GCN variants still have not shown significant advantages over a properly designed linear GCN. We believe that this would attract the attention of the community to reconsider the actual representation ability of current nonlinear GCNs and propose new alternatives that can truly benefit from nonlinear architectures.

References

[1] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015. 9

[2] Aleksandar Bojchevski and Stephan Günnemann. Certifiable robustness to graph perturbations. *NeurIPS*, 2019. 8

[3] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: fast learning with graph convolutional networks via importance sampling. *ICLR*, 2018. 8

[4] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *NeurIPS*, 2018. 2

[5] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997. 3, 5

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *NeurIPS*, 2017. 2, 6, 7, 8

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016. 2

[8] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017. 1, 2, 5, 6, 7, 8

[9] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *ICLR*, 2019. 2, 6, 7, 8

[10] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can gcns go as deep as cnns? *CVPR*, 2019. 2

[11] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *AAAI*, 2018. 1, 2, 3

[12] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *ICML*, 2018. 2

[13] Georgi S Medvedev. Stochastic stability of continuous time consensus protocols. *SIAM Journal on Control and Optimization*, 50(4):1859–1885, 2012. 3

[14] Georgi S Medvedev. The nonlinear heat equation on dense graphs and graph limits. *SIAM Journal on Mathematical Analysis*, 46(4):2743–2766, 2014. 3

[15] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. *SIGKDD*, 2014. 6, 7

[16] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019. 2, 6, 7

[17] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards deep graph convolutional networks on node classification. *ILCR*, 2019. 2

[18] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 6

[19] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 5

[20] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2018. 2, 6, 7

[21] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *ICLR*, 2019. 6, 8

[22] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *ICML*, 2019. 1, 2, 5, 6, 7, 8

[23] Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Continuous graph neural networks. *ICML*, 2020. 2, 6, 7

[24] Bingbing Xu, Huawei Shen, Qi Cao, Keting Cen, and Xueqi Cheng. Graph convolutional networks using heat kernel for semi-supervised learning. *arXiv preprint arXiv:2007.16002*, 2020. 6, 7

[25] Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. Graph wavelet neural network. *ICLR*, 2019. 6, 7

[26] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *ICML*, 2018. 2, 6, 7, 8

[27] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. *ICML*, 2016. 2

[28] Lingxiao Zhao and Leman Akoglu. PairNorm: Tackling oversmoothing in gnns. *ICLR*, 2020. 2, 6, 7

[29] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. *ICML*, 2003. 6, 7

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [No]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Proofs are all included in Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]