Break the Ceiling: Stronger Multi-scale Deep Graph Convolutional Networks

Sitao Luan[1,2,*], **Mingde Zhao**[1,2,*], **Xiao-Wen Chang**[1], **Doina Precup**[1,2,3]
{sitao.luan@mail, mingde.zhao@mail, chang@cs, dprecup@cs}.mcgill.ca
[1]McGill University; [2]Mila; [3]DeepMind
[*]Equal Contribution

Abstract

Recently, neural network based approaches have achieved significant progress for solving large, complex, graph-structured problems. Nevertheless, the advantages of multi-scale information and deep architectures have not been sufficiently exploited. In this paper, we first analyze key factors constraining the expressive power of existing Graph Convolutional Networks (GCNs), including the activation function and shallow learning mechanisms. Then, we generalize spectral graph convolution and deep GCN in block Krylov subspace forms, upon which we devise two architectures, both scalable in depth however making use of multi-scale information differently. On several node classification tasks, the proposed architectures achieve state-of-the-art performance.

1 Introduction & Motivation

Many real-world problems can be modeled as graphs [14, 18, 25, 12, 27, 7]. Inspired by the success of Convolutional Neural Networks (CNNs) [20] in computer vision [22], graph convolution defined on graph Fourier domain stands out as the key operator and one of the most powerful tools for using machine learning to solve graph problems. In this paper, we focus on spectrum-free Graph Convolutional Networks (GCNs) [2, 29], which have demonstrated state-of-the-art performance on many transductive and inductive learning tasks [7, 18, 25, 3, 4].

One major problem of the existing GCNs is the low expressive power limited by their shallow learning mechanisms [38, 36]. There are mainly two reasons why people have not yet achieved an architecture that is scalable in depth. First, this problem is difficult: considering graph convolution as a special form of Laplacian smoothing [21], networks with multiple convolutional layers will suffer from an over-smoothing problem that makes the representation of even distant nodes indistinguishable [38]. Second, some people think it is unnecessary: for example, [2] states that it is not necessary for the label information to totally traverse the entire graph and one can operate on the multi-scale coarsened input graph and obtain the same flow of information as GCNs with more layers. Acknowledging the difficulty, we hold on to the objective of deepening GCNs since the desired compositionality[1] will yield easy articulation and consistent performance for problems with different scales.

In this paper, we break the performance ceiling of the GCNs. First, we analyze the limits of the existing GCNs brought by the shallow learning mechanisms and the activation functions. Then, we show that any graph convolution with a well-defined analytic spectral filter can

[1]The expressive power of a sound deep NN architecture should be expected to grow with the increment of network depth [19, 16].

be written as a product of a block Krylov matrix and a learnable parameter matrix in a special form. Based on this, we propose two GCN architectures that leverage multi-scale information in different ways and are scalable in depth, with stronger expressive powers and abilities to extract richer representations of graph-structured data. We also show that the equivalence of the two architectures can be achieved under certain conditions. For empirical validation, we test different instances of the proposed architectures on multiple node classification tasks. The results show that even the simplest instance of the architectures achieves state-of-the-art performance, and the complex ones achieve surprisingly higher performance, with or without validation sets.

2 Why Deep GCN Does Not Work Well?

2.1 Foundations

As in [11], we use bold font for vectors (*e.g.* $\boldsymbol{v}$), block vectors (*e.g.* $\boldsymbol{V}$) and matrix blocks (*e.g.* $\boldsymbol{V_i}$). Suppose we have an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$, where $\mathcal{V}$ is the node set with $|\mathcal{V}| = N$, $\mathcal{E}$ is the edge set with $|\mathcal{E}| = E$, $A \in \mathbb{R}^{N\times N}$ is a symmetric adjacency matrix and D is a diagonal degree matrix, *i.e.* $D_{ii} = \sum_j A_{ij}$. A diffusion process [6, 5] on $\mathcal{G}$ can be defined by a diffusion operator L, which is a symmetric matrix, *e.g.* graph Laplacian $L = D - A$, normalized graph Laplacian $L = I - D^{-1/2}AD^{-1/2}$ and affinity matrix $L = A + I$, *etc.*. In this paper, we use L for a general diffusion operator, unless specified otherwise. The eigendecomposition of L gives us $L = U\Lambda U^T$, where Λ is a diagonal matrix whose diagonal elements are eigenvalues and the columns of U are the orthonormal eigenvectors, named graph Fourier basis. We also have a feature matrix (graph signals) $\boldsymbol{X} \in \mathbb{R}^{N\times F}$ (which can be regarded as a block vector) defined on $\mathcal{V}$ and each node i has a feature vector $\boldsymbol{X_{i,:}}$, which is the i-th row of $\boldsymbol{X}$.

Spectral graph convolution is defined in graph Fourier domain *s.t.* $\boldsymbol{x} *_{\mathcal{G}} \boldsymbol{y} = U((U^T\boldsymbol{x}) \odot (U^T\boldsymbol{y}))$, where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$ and $\odot$ is the Hadamard product [7]. Following this definition, a graph signal $\boldsymbol{x}$ filtered by g_θ can be written as

$$\boldsymbol{y} = g_\theta(L)\boldsymbol{x} = g_\theta(U\Lambda U^T)\boldsymbol{x} = Ug_\theta(\Lambda)U^T\boldsymbol{x} \tag{1}$$

where g_θ is any function which is analytic inside a closed contour which encircles $\lambda(L)$, *e.g.* Chebyshev polynomial [7]. GCN generalizes this definition to signals with F input channels and O output channels and its network structure can be described as

$$\boldsymbol{Y} = \text{softmax}(L \text{ ReLU}(L\boldsymbol{X}W_0)\ W_1) \tag{2}$$

where

$$L \equiv \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2},\ \ \tilde{A} \equiv A + I,\ \ \tilde{D} \equiv \text{diag}(\textstyle\sum_j \tilde{A}_{1j}, \dots, \sum_j \tilde{A}_{Nj}) \tag{3}$$

This is called spectrum-free method [2] since it requires no explicit computation of eigendecomposition and operations on the frequency domain [38].

2.2 Problems

Suppose we deepen GCN in the same way as [18, 21], we have

$$\boldsymbol{Y} = \text{softmax}(L \text{ ReLU}(\cdots L \text{ ReLU}(L \text{ ReLU}(L\boldsymbol{X}W_0)\ W_1)\ W_2 \cdots)\ W_n) \equiv \text{softmax}(\boldsymbol{Y'}) \tag{4}$$

For this architecture, [21] gives an analysis on the effect of L without considering the ReLU activation function. Our analyses on (4) can be summarized in the following theorems.

Theorem 1. Suppose that $\mathcal{G}$ has k connected components and the diffusion operator L is defined as that in (3). Let $\boldsymbol{X} \in \mathbb{R}^{N\times F}$ be any block vector and let W_j be any non-negative parameter matrix with $\|W_j\|_2 \le 1$ for $j = 0, 1, \dots$. If $\mathcal{G}$ has no bipartite components, then in (4), as $n \to \infty$, $\text{rank}(\boldsymbol{Y'}) \le k$.

Proof See Appendix A. □

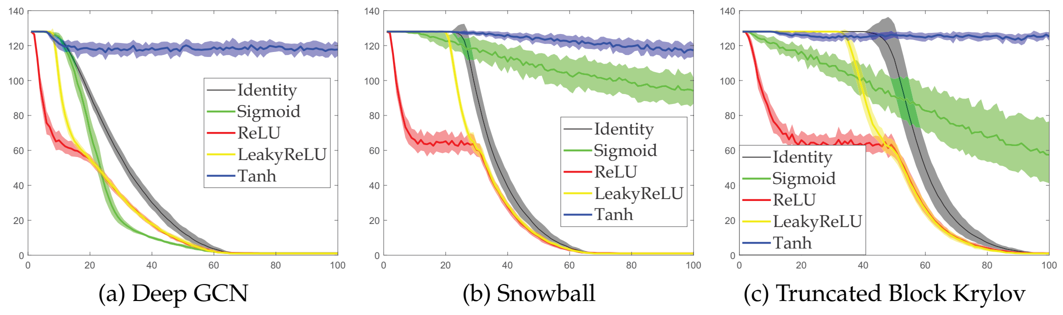
Conjecture 1. Theorem 1 still holds without the non-negative constraint on the parameter matrices.

Theorem 2. Suppose the n-dimensional $\boldsymbol{x}$ and $\boldsymbol{y}$ are independently sampled from a continuous distribution and the activation function $\mathrm{Tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ is applied to $[\boldsymbol{x}, \boldsymbol{y}]$ pointwisely, then

$$\mathbb{P}(\mathrm{rank}\,(\mathrm{Tanh}([\boldsymbol{x}, \boldsymbol{y}])) = \mathrm{rank}([\boldsymbol{x}, \boldsymbol{y}])) = 1$$

Proof See Appendix A. □

Theorem 1 shows that if we simply deepen GCN, the extracted features will degrade, *i.e.* $\boldsymbol{Y}'$ only contains the stationary information of the graph structure and loses all the local information in node for being smoothed. In addition, from the proof we see that the pointwise ReLU transformation is a conspirator. Theorem 2 tells us that Tanh is better at keeping linear independence among column features. We design a numerical experiment on synthetic data (see Appendix) to test, under a 100-layer GCN architecture, how activation functions affect the rank of the output in each hidden layer during the feedforward process. As Figure 1(a) shows, the rank of hidden features decreases rapidly with ReLU, while having little fluctuation under Tanh, and even the identity function performs better than ReLU (see Appendix for more comparisons). So we propose to replace ReLU by Tanh.



(a) Deep GCN (b) Snowball (c) Truncated Block Krylov

Figure 1: Changes in the number of independent features with the increment of network depth

3 Spectral Graph Convolution and Block Krylov Subspaces

3.1 Block Krylov Subspaces

Let $\mathbb{S}$ be a vector subspace of $\mathbb{R}^{F \times F}$ containing the identity matrix I_F that is closed under matrix multiplication and transposition. We define an inner product $\langle \cdot, \cdot \rangle_{\mathbb{S}}$ in the block vector space $\mathbb{R}^{N \times F}$ as follows [11]:

Definition 1 *A mapping $\langle \cdot, \cdot \rangle_{\mathbb{S}}$ from $\mathbb{R}^{N \times F} \times \mathbb{R}^{N \times F}$ to $\mathbb{S}$ is called a block inner product onto $\mathbb{S}$ if $\forall \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \in \mathbb{R}^{N \times F}$ and $\forall C \in \mathbb{S}$:*

1. *$\mathbb{S}$-linearity: $\langle \boldsymbol{X}, \boldsymbol{Y}C \rangle_{\mathbb{S}} = \langle \boldsymbol{X}, \boldsymbol{Y} \rangle_{\mathbb{S}} C$ and $\langle \boldsymbol{X} + \boldsymbol{Y}, \boldsymbol{Z} \rangle_{\mathbb{S}} = \langle \boldsymbol{X}, \boldsymbol{Z} \rangle_{\mathbb{S}} + \langle \boldsymbol{Y}, \boldsymbol{Z} \rangle_{\mathbb{S}}$*
2. *symmetry: $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle_{\mathbb{S}} = \langle \boldsymbol{Y}, \boldsymbol{X} \rangle_{\mathbb{S}}^T$*
3. *definiteness: $\langle \boldsymbol{X}, \boldsymbol{X} \rangle_{\mathbb{S}}$ is positive definite if $\boldsymbol{X}$ has full rank, and $\langle \boldsymbol{X}, \boldsymbol{X} \rangle_{\mathbb{S}} = 0_F$ iff $\boldsymbol{X} = 0$.*

There are mainly three ways to define $\langle \cdot, \cdot \rangle_{\mathbb{S}}$ [11]: 1) (Classical.) $\mathbb{S}^{\mathrm{Cl}} = \mathbb{R}^{F \times F}$ and $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle_{\mathbb{S}}^{\mathrm{Cl}} = \boldsymbol{X}^T \boldsymbol{Y}$; 2) (Global.) $\mathbb{S}^{\mathrm{Gl}} = cI_F,\ c \in \mathbb{R}$ and $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle_{\mathbb{S}}^{\mathrm{Gl}} = \mathrm{trace}(\boldsymbol{X}^T \boldsymbol{Y}) I_F$; 3) (Loop-interchange.) $\mathbb{S}^{\mathrm{Li}}$ is the set of diagonal matrices and $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle_{\mathbb{S}}^{\mathrm{Li}} = \mathrm{diag}(\boldsymbol{X}^T \boldsymbol{Y})$. The three definitions are all useful yet we will use the classical one for our contribution.

For further explanations, we give the definition of block vector subspace of $\mathbb{R}^{N \times F}$.

Definition 2 *Given a set of block vectors* $\{\boldsymbol{X}_k\}_{k=1}^m \subset \mathbb{R}^{N\times F}$*, the* $\mathbb{S}$*-span of* $\{\boldsymbol{X}_k\}_{k=1}^m$ *is defined as* $\mathrm{span}^{\mathbb{S}}\{\boldsymbol{X}_1,\dots,\boldsymbol{X}_m\} := \{\sum_{k=1}^{m} \boldsymbol{X}_k C_k : C_k \in \mathbb{S}\}$

Given the above definition, the order-m block Krylov subspace with respect to the matrix $A \in \mathbb{R}^{N\times N}$, the block vector $\boldsymbol{B} \in \mathbb{R}^{N\times F}$ and the vector space $\mathbb{S}$ can be defined as $\mathcal{K}_m^{\mathbb{S}}(A,\boldsymbol{B}) := \mathrm{span}^{\mathbb{S}}\{\boldsymbol{B}, A\boldsymbol{B},\dots,A^{m-1}\boldsymbol{B}\}$. The corresponding block Krylov matrix is defined as $K_m(A,\boldsymbol{B}) := [\boldsymbol{B}, A\boldsymbol{B},\dots,A^{m-1}\boldsymbol{B}]$.

3.2 Spectral Graph Convolution in Block Krylov Subspace Form

In this section, we show that any graph convolution with well-defined analytic spectral filter defined on $L \in \mathbb{R}^{N\times N}$ can be written as the product of a block Krylov matrix with a learnable parameter matrix in a specific form. We take $\mathbb{S} = \mathbb{S}^{\mathrm{Cl}} = \mathbb{R}^{F\times F}$.

For any real analytic scalar function g, its power series expansion around center 0 is

$$g(x) = \sum_{n=0}^{\infty} a_n x^n = \sum_{n=0}^{\infty} \frac{g^{(n)}(0)}{n!} x^n, \ |x| < R$$

where R is the radius of convergence.

The function g can be used to define a filter. Let $\rho(L)$ denote the spectrum radius of L and suppose $\rho(L) < R$. The spectral filter $g(L) \in \mathbb{R}^{N\times N}$ can be defined as

$$g(L) := \sum_{n=0}^{\infty} a_n L^n = \sum_{n=0}^{\infty} \frac{g^{(n)}(0)}{n!} L^n, \ \rho(L) < R$$

According to the definition of spectral graph convolution in (1), graph signal $\boldsymbol{X}$ is filtered by $g(L)$ as follows,

$$g(L)\boldsymbol{X} = \sum_{n=0}^{\infty} \frac{g^{(n)}(0)}{n!} L^n \boldsymbol{X} = \left[\boldsymbol{X}, L\boldsymbol{X}, L^2\boldsymbol{X}, \cdots\right] \left[\frac{g^{(0)}(0)}{0!} I_F, \frac{g^{(1)}(0)}{1!} I_F, \frac{g^{(2)}(0)}{2!} I_F, \cdots\right]^T = A'B'$$

where $A' \in \mathbb{R}^{N\times\infty}$ and $B' \in \mathbb{R}^{\infty\times F}$. We can see that A' is a block Krylov matrix and $\mathrm{Range}(A'B') \subseteq \mathrm{Range}(A')$. It is shown in [13, 11] that for $\mathbb{S} = \mathbb{R}^{F\times F}$ there exists a smallest m such that

$$\mathrm{span}^{\mathbb{S}}\{\boldsymbol{X}, L\boldsymbol{X}, L^2\boldsymbol{X}, \cdots\} = \mathrm{span}^{\mathbb{S}}\{\boldsymbol{X}, L\boldsymbol{X}, L^2\boldsymbol{X}, \dots, L^{m-1}\boldsymbol{X}\} \tag{5}$$

where m depends on L and $\boldsymbol{X}$ and will be written as $m(L,\boldsymbol{X})$ later. This means for any $k \geq m$, $L^k\boldsymbol{X} \in \mathcal{K}_m^{\mathbb{S}}(L,\boldsymbol{X})$. From (5), the convolution can be written as

$$g(L)\boldsymbol{X} = \sum_{n=0}^{\infty} \frac{g^{(n)}(0)}{n!} L^n \boldsymbol{X} \equiv \left[\boldsymbol{X}, L\boldsymbol{X}, \dots, L^{m-1}\boldsymbol{X}\right] \left[(\Gamma_0{}^{\mathbb{S}})^T, (\Gamma_1{}^{\mathbb{S}})^T, \cdots, (\Gamma_{m-1}^{\mathbb{S}})^T\right]^T \equiv K_m(L,\boldsymbol{X})\Gamma^{\mathbb{S}} \tag{6}$$

where $\Gamma_i^{\mathbb{S}} \in \mathbb{R}^{F\times F}$ for $i = 1,\dots,m-1$ are parameter matrix blocks. Then, a graph convolutional layer can be be generally written as

$$g(L)\boldsymbol{X}W' = K_m(L,\boldsymbol{X})\Gamma^{\mathbb{S}}W' = K_m(L,\boldsymbol{X})W^{\mathbb{S}} \tag{7}$$

where $W^{\mathbb{S}} \equiv \Gamma^{\mathbb{S}}W' \in \mathbb{R}^{mF\times O}$. The essential number of learnable parameters is $mF \times O$.

3.3 Deep GCN in the Block Krylov Subspace Form

Since the spectral graph convolution can be simplified as (6)(7), we can build deep GCN in the following way.

Suppose that we have a sequence of analytic spectral filters $G = \{g_0, g_1, \dots, g_n\}$ and a sequence of pointwise nonlinear activation functions $H = \{h_0, h_1, \dots, h_n\}$. Then, a deep spectral graph convolution network can be written as

$$\mathbf{Y} = \mathrm{softmax}\left\{g_n(L)\, h_{n-1}\left\{\cdots g_2(L)\, h_1\left\{g_1(L)\, h_0\left\{g_0(L)\boldsymbol{X}W_0'\right\} W_1'\right\} W_2' \cdots\right\} W_n'\right\} \tag{8}$$

Define

$$\boldsymbol{H_0} = \boldsymbol{X}, \qquad \boldsymbol{H_{i+1}} = h_i\{g_i(L)\boldsymbol{H_i}W_i\},\ \ i = 0, \dots, n-1$$

Then, we have

$$\boldsymbol{Y} = \text{softmax}\{g_n(L)\boldsymbol{H_n}W_n'\}$$

From (7) and (8), we see we can write

$$\boldsymbol{H_{i+1}} = h_i\{K_{m_i}(L, \boldsymbol{H_i})W_i^{\mathbb{S}_i}\},\ \ m_i \equiv m(L, \boldsymbol{H_i})$$

It is easy to see that, when $g_i(L) = I$, (8) is a fully connected network [21]; when $n = 1$, $g_0(L) = g_1(L) = L$, where L is defined in (3), it is just GCN [18]; when $g_i(L)$ is defined by the Chebyshev polynomial [15], $W_i' = I$, (8) is ChebNet [7].

3.4 Difficulties & Inspirations

In the last subsection, we gave a general form of deep GCN in the block Krylov form. Following this idea, we can leverage the existing block Lanczos algorithm [11, 10] to find m_i and compute orthogonal basis of $\mathcal{K}_{m_i}^{\mathbb{S}}(L, \boldsymbol{H_i})$ which makes the filter coefficients compact [25] and improve numerical stability. But there are some difficulties in practice:

1. During the training phase, $\boldsymbol{H_i}$ changes every time when parameters are updated. This makes m_i a variable and thus requires adaptive size for parameter matrices $W_i^{\mathbb{S}_i}$.
2. For classical inner product, the QR factorization that is needed in block Lanczos algorithm [11] is difficult to be implemented in backpropagation framework.

Despite implementation intractability, block Krylov form is still meaningful for constructing GCNs that are scalable in depth as we illustrate below.

For each node $v \in \{1, \dots, N\}$ in the graph, denote $N(v)$ as the set of its neighbors and $N^k(v)$ as the set of its k-hop neighbors. Then, $L\boldsymbol{X}(v, :)$ can be interpreted as a weighted mean of the feature vectors of v and $N(v)$. If the network goes deep as (4), $\boldsymbol{Y}'(v, :)$ becomes the "weighted mean" of the feature vectors of v and $N^{(n+1)}(v)$ (not exactly weighted mean because we have ReLU in each layer). As the scope grows, the nodes in the same connected component tend to have the same (global) features, while losing their individual (local) features, which makes them indistinguishable. Such phenomenon is recognized as "oversmoothing" [21]. Though it is reasonable to assume that the nodes in the same cluster share many similar properties, it will be harmful to omit the individual differences between each node.

Therefore, the inspiration from the block Krylov form is that, to get a richer representation of each node, we need to concatenate the multi-scale information (local and global) together instead of merely doing smoothing in each hidden layer. If we have a smart way to stack multi-scale information, the network will be scalable in depth. To this end, we naturally come up with a densely connected architecture [17], which we call *snowball* network and a compact architecture, which we call the *truncated Krylov* network, in which the multi-scale information is used differently.

4 Deep GCN Architectures

4.1 Snowball

The block Krylov form inspires first an architecture that concatenates multi-scale features incrementally, resulting in a densely-connected graph network (Figure 2(a)) as follows:

$$\begin{aligned}
&\boldsymbol{H_0} = \boldsymbol{X},\ \ \boldsymbol{H_{l+1}} = f\left(L\left[\boldsymbol{H_0}, \boldsymbol{H_1}, \dots, \boldsymbol{H_l}\right]W_l\right),\ \ l = 0, 1, \dots, n-1\\
&\boldsymbol{C} = g\left(\left[\boldsymbol{H_0}, \boldsymbol{H_1}, \dots, \boldsymbol{H_n}\right]W_n\right)\\
&\text{output} = \text{softmax}\left(L^p\boldsymbol{C}W_C\right)
\end{aligned} \tag{9}$$

where $W_l \in \mathbb{R}^{\left(\sum_{i=0}^{l} F_i\right)\times F_{l+1}}$, $W_n \in \mathbb{R}^{\left(\sum_{i=0}^{n} F_i\right)\times F_C}$ and $W_C \in \mathbb{R}^{F_C\times F_O}$ are learnable parameter matrices, F_{l+1} is the number of output channels in layer l; f and g are pointwise activation functions; $\boldsymbol{H_l}$ are extracted features; $\boldsymbol{C}$ is the output of a classifier of any kind, *e.g.*, a fully connected neural network or even an identity layer, in which case $\boldsymbol{C} = [\boldsymbol{H_0}, \boldsymbol{H_1}, \ldots, \boldsymbol{H_n}]$; $p \in \{0, 1\}$. When $p = 0$, $L^p = I$ and when $p = 1$, $L^P = L$, which means that we project $\boldsymbol{C}$ back onto graph Fourier basis, which is necessary when the graph structure encodes much information. Following this construction, we can stack all learned features as the input of the subsequent hidden layer, which is an efficient way to concatenate multi-scale information. The size of input will grow like a snowball and this construction is similar to DenseNet [17], which is designed for regular grids (images). Thus, some advantages of DenseNet are naturally inherited, *e.g.*, alleviate the vanishing-gradient problem, encourage feature reuse, increase the variation of input for each hidden layer, reduce the number of parameters, strengthen feature propagation and improve model compactness.

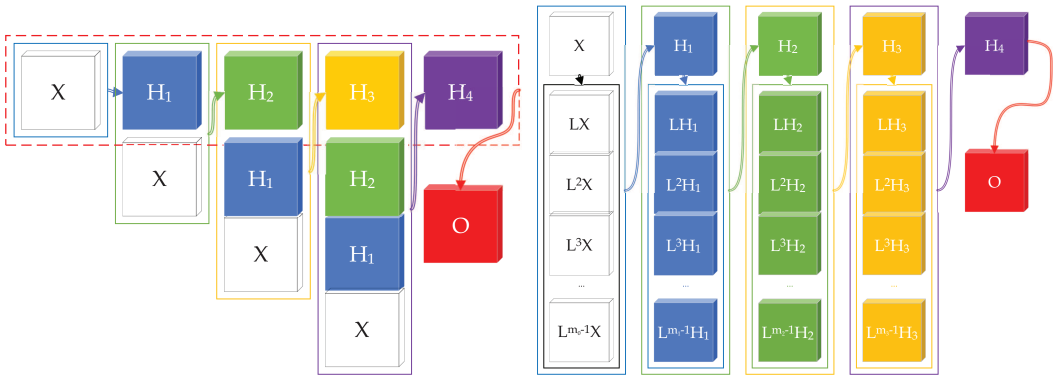
4.2 Truncated Krylov

The block Krylov form inspires then an architecture that concatenates multi-scale features directly together in each layer. However, as stated in Section 3.4, the fact that m_i is a variable makes GCN difficult to be merged into the block Krylov framework. Thus we compromise and set m_i as a hyperparameter and get a truncated block Krylov network (Figure 2(b)) as shown below:

$$\begin{aligned} &\boldsymbol{H_0} = \boldsymbol{X}, \;\; \boldsymbol{H_{l+1}} = f\left(\left[\boldsymbol{H_l}, L\boldsymbol{H_l} \ldots, L^{m_l-1}\boldsymbol{H_l}\right] W_l\right), \;\; l = 0, 1, \ldots, n-1 \\ &\boldsymbol{C} = g\left(\boldsymbol{H_n} W_n\right) \\ &\text{output} = \text{softmax}\left(L^p \boldsymbol{C} W_C\right) \end{aligned} \tag{10}$$

where $W_l \in \mathbb{R}^{(m_l F_l) \times F_{l+1}}$, $W_n \in \mathbb{R}^{F_n \times F_C}$ and $W_C \in \mathbb{R}^{F_C \times F_O}$ are learnable parameter matrices; f and g are activation functions; $\boldsymbol{C}$ is the output of a classifier of any kind; $p \in \{0, 1\}$. In the truncated Krylov network, the local information will not be diluted in each layer because in each layer l, we start the concatenation from $L^0\boldsymbol{H_l}$ so that the extracted local information can be kept.

There are works on the analysis of error bounds of doing truncation in block Krylov methods [11]. But the results need many assumptions either on $\boldsymbol{X}$, *e.g.*, $\boldsymbol{X}$ is a standard Gaussian matrix [34], or on L, *e.g.*, some conditions on the smallest and largest eigenvalues of L have to be satisfied [28]. Instead of doing truncation for a specific function or a fixed $\boldsymbol{X}$, we are dealing with variable $\boldsymbol{X}$ during training. So we cannot get a practical error bound since we cannot put any restriction on $\boldsymbol{X}$ and its relation to L.



(a) Snowball (b) Truncated Block Krylov

Figure 2: Proposed Architectures

The Krylov subspace methods are often associated with low-rank approximation methods for large sparse matrices. Here we would like to mention [25] does low-rank approximation of L by the Lanczos algorithm. It suffers from the tradeoff between accuracy and efficiency: the information in L will be lost if L is not low-rank, while keeping more information via increasing the Lanczos steps will hurt the efficiency. Since most of the graphs we are dealing with have sparse connectivity structures, they are actually not low-rank, *e.g.*, the Erdős-Rényi graph $G(n,p)$ with $p = \omega(\frac{1}{n})$ [32] and examples in Appendix IV. Thus, we do not propose to do low-rank approximation in our architecture.

4.3 Equivalence of Linear Snowball GCN and Truncated Block Krylov Network

In this part, we will show that the two proposed architectures are inherently connected. In fact their equivalence can be established when using identify functions as f, identity layer as $\boldsymbol{C}$ and constraining the parameter matrix of truncated Krylov to be in a special form.

In linear snowball GCN, we can split the parameter matrix W_i into $i+1$ blocks and write it as $W_i = \left[(\boldsymbol{W}_i^{(1)})^T, \cdots, (\boldsymbol{W}_i^{(i+1)})^T\right]^T$ and then following (9) we have

$$\boldsymbol{H_0} = \boldsymbol{X},\ \boldsymbol{H_1} = L\boldsymbol{X}W_0,\ \boldsymbol{H_2} = L[\boldsymbol{X}, \boldsymbol{H_1}]W_1 = L\boldsymbol{X}\boldsymbol{W}_1^{(1)} + L^2\boldsymbol{X}\boldsymbol{W}_0^{(1)}\boldsymbol{W}_1^{(2)} = L[\boldsymbol{X}, L\boldsymbol{X}]\begin{bmatrix} \boldsymbol{I} & 0 \\ 0 & \boldsymbol{W}_0^{(1)} \end{bmatrix}\begin{bmatrix} \boldsymbol{W}_1^{(1)} \\ \boldsymbol{W}_1^{(2)} \end{bmatrix}, \cdots$$

As in (9), we have $\boldsymbol{C}W_C = L[\boldsymbol{H_0}, \boldsymbol{H_1}, \ldots, \boldsymbol{H_n}]W_C$. Thus we can write

$$\begin{aligned} &[\boldsymbol{H_0}, \boldsymbol{H_1} \cdots, \boldsymbol{H_n}]W_C \\ = \ &[\boldsymbol{X}, L\boldsymbol{X}, \cdots, L^n\boldsymbol{X}]\begin{bmatrix} \boldsymbol{I} & 0 & \cdots & 0 \\ 0 & \boldsymbol{I} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{W}_0^{(1)} \end{bmatrix}\begin{bmatrix} \boldsymbol{I} & 0 & \cdots & 0 \\ 0 & \boldsymbol{I} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{W}_1^{(1)} \end{bmatrix}\cdots\begin{bmatrix} \boldsymbol{I} & 0 & \cdots & 0 \\ 0 & \boldsymbol{W}_{n-1}^{(n)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{W}_{n-1}^{(1)} \end{bmatrix}\begin{bmatrix} \boldsymbol{W}_C^{(1)} \\ \boldsymbol{W}_C^{(2)} \\ \vdots \\ \boldsymbol{W}_C^{(n)} \end{bmatrix} \end{aligned}$$

which is in the form of (7), where the parameter matrix is the multiplication of a sequence of block diagonal matrices whose entries consist of identity blocks and blocks from other parameter matrices. Though the two proposed architectures stack multi-scale information in different ways, *i.e.* incremental and direct respectively, the equivalence reveals that the truncated block Krylov network can be constrained to leverage multi-scale information in a way similar to the snowball architecture. While it is worth noting that when there are no constraints, truncated Krylov is capable of achieving more than what snowball does.

4.4 Relation to Message Passing Framework

We denote the concatenation operator as $\|$. If we consider L as a general aggregation operator which aggregates node features with its neighborhood features, we see that the two proposed architectures both have close relationships with message passing framework [12], which are illustrated in the following table, where $N^0(v) = \{v\}$, M_t is a message function, U_t is a vertex update function, $\boldsymbol{m}_v^{(t+1)}, \boldsymbol{h}_v^{(t+1)}$ are messages and hidden states at each node respectively, $\boldsymbol{m}^{(t+1)} = [\boldsymbol{m}_1^{(t+1)}, \cdots, \boldsymbol{m}_N^{(t+1)}]^T$, $\boldsymbol{h}^{(t+1)} = [\boldsymbol{h}_1^{(t+1)}, \cdots, \boldsymbol{h}_N^{(t+1)}]^T$ and σ is a nonlinear activation function.

Compared to our proposed architectures, we can see that the message passing paradigm cannot avoid oversmoothing problem because it does not leverage multi-scale information in each layer and will finally lose local information. An alternate solution to address the oversmoothing problem could be to modify the readout function to $\hat{y} = R(\{\boldsymbol{h}_v^{(0)}, \boldsymbol{h}_v^{(1)}, \ldots, \boldsymbol{h}_v^{(T)} | v \in \mathcal{V}\})$.

5 Experiments

On node classification tasks, we test 2 instances of the snowball GCN and 1 instance of the truncated Krylov GCN, which include linear snowball GCN ($f = g =$ identity, $p = 1$), snowball GCN ($f =$ Tanh, $g =$ identity, $p = 1$) and truncated Krylov ($f = g =$ Tanh, $p = 0$). The test cases include on public splits [37, 25] of Cora, Citeseer and PubMed[2], as well as the crafted smaller splits that are more difficult [25, 21, 31]. We compare the instances

[2]Source code to be found at `https://github.com/PwnerHarry/Stronger_GCN`

Table 1: Algorithms in Matrix and Nodewise Forms

	Forms	
Algorithms	Matrix	Nodewise
Message Passing	$\boldsymbol{m}^{(t+1)} = M_t(A, \boldsymbol{h}^{(t)})$ $\boldsymbol{h}^{(t+1)} = U_t(\boldsymbol{h}^{(t)}, \boldsymbol{m}^{(t+1)})$	$\boldsymbol{m}_v^{(t+1)} = \sum_{w \in N(v)} M_t(\boldsymbol{h}_v^{(t)}, \boldsymbol{h}_w^{(t)}, e_{vw})$ $\boldsymbol{h}_v^{(t+1)} = U_t(\boldsymbol{h}_v^{(t)}, \boldsymbol{m}_v^{(t+1)})$
GraphSAGE-GCN	$\boldsymbol{m}^{(t+1)} = L\boldsymbol{h}^{(t)}$ $\boldsymbol{h}^{(t+1)} = \sigma(\boldsymbol{m}^{(t+1)} W_t)$	$\boldsymbol{m}_v^{(t+1)} = \text{mean}(\{\boldsymbol{h}_v^{(t)}\} \cup \{\boldsymbol{h}_{N(v)}^{(t)}\})$ $\boldsymbol{h}_v^{(t+1)} = \sigma(W_t^T \boldsymbol{m}_v^{(t+1)})$
Snowball	$\boldsymbol{m}^{(t+1)} = L[\boldsymbol{h}^{(0)} \Vert \dots \Vert \boldsymbol{h}^{(t)}]$ $\boldsymbol{h}_v^{(t+1)} = \sigma(\boldsymbol{m}^{(t+1)} W_t)$	$\boldsymbol{m}_v^{(t+1)} = \Vert_{i=0}^{t} \text{mean}(\{\boldsymbol{h}_v^{(i)}\} \cup \{\boldsymbol{h}_{N(v)}^{(i)}\})$ $\boldsymbol{h}_v^{(t+1)} = \sigma(W_t^T \boldsymbol{m}_v^{(t+1)})$
Truncated Krylov	$\boldsymbol{m}^{(t+1)} = \boldsymbol{h}^{(t)} \Vert \dots \Vert L^{m_t - 1} \boldsymbol{h}^{(t)}$ $\boldsymbol{h}^{(t+1)} = \sigma(\boldsymbol{m}^{(t+1)} W_t)$	$\boldsymbol{m}_v^{(t+1)} = \Vert_{i=0}^{m_t - 1} \text{mean}(\cup_{k=0}^{i} \{\boldsymbol{h}_{N^k(v)}^{(t)}\})$ $\boldsymbol{h}_v^{(t+1)} = \sigma(W_t^T \boldsymbol{m}_v^{(t+1)})$

against several methods under 2 experimental settings, with or without validations sets. The compared methods with validation sets include graph convolutional networks for fingerprint (GCN-FP) [8], gated graph neural networks (GGNN) [23], diffusion convolutional neural networks (DCNN) [1], Chebyshev networks (Cheby) [7], graph convolutional networks (GCN) [18], message passing neural networks (MPNN) [12], graph sample and aggregate (GraphSAGE) [14], graph partition neural networks (GPNN) [24], graph attention networks (GAT) [33], LanczosNet (LNet) [25] and AdaLanczosNet (AdaLNet) [25]. The copmared methods without validation sets include label propagation using ParWalks (LP) [35], Co-training [21], Self-training [21], Union [21], Intersection [21], GCN without validation [21], Multi-stage training [31], Multi-stage self-supervised (M3S) training [31], GCN with sparse virtual adversarial training (GCN-SVAT) [30] and GCN with dense virtual adversarial training (GCN-DVAT) [30].

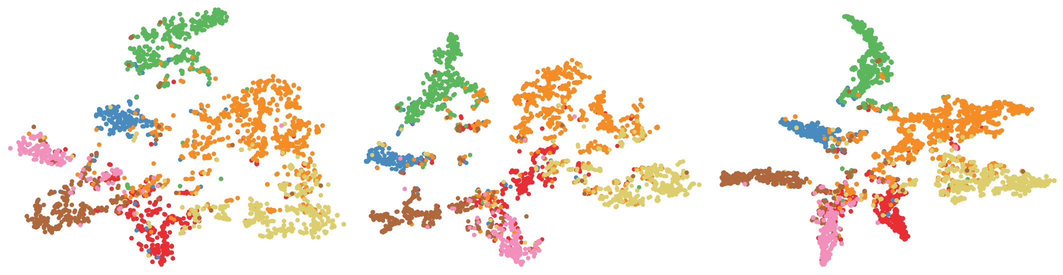

(a) Linear Snowball (b) Snowball (c) Truncated Krylov

Figure 3: t-SNE for the extracted features trained on Cora (7 classes) public (5.2%).

In Table 2 and 3, for each test case, we report the accuracy averaged from 10 independent runs using the best searched hyperparameters. These hyperparameters are reported in the appendix, which include learning rate and weight decay for the optimizers RMSprop or Adam for cases with validation or without validation, respectively, taking values in the intervals $[10^{-6}, 5 \times 10^{-3}]$ and $[10^{-5}, 10^{-2}]$, respectively, width of hidden layers taking value in the set $\{100, 200, \cdots, 5000\}$, number of hidden layers in the set $\{1, 2, \dots, 50\}$, dropout in $(0, 0.99]$, and the number of Krylov blocks taking value in $\{1, 2, \dots, 100\}$. An early stopping trick is also used to achieve better training. Specifically we terminate the training after 100 update steps of not improving the training loss.

We see that the instances of the proposed architectures achieve overwhelming performance in *all* test cases. We visualize a representative case using t-SNE [26] in Figure 3. From these visualization, we can see the instances can extract good features with small training data, especially for the truncated block Krylov network. Particularly, when the training splits are small, they perform astonishingly better than the existing methods. This may be explained by the fact that when there is less labeled data, larger scope of vision field is needed to make recognition of each node or to let the label signals propagate. We would also highlight that the linear snowball GCN can achieve state-of-the-art performance with much less computational cost. If $\mathcal{G}$ has no bipartite components, then in (4), as $n \to \infty$, $\text{rank}(\boldsymbol{Y}') \leq k$ almost surely.

Table 2: Accuracy without Validation

Algorithms	Cora						CiteSeer						PubMed			
	0.5%	1%	2%	3%	4%	5%	0.5%	1%	2%	3%	4%	5%	0.03%	0.05%	0.1%	0.3%
LP	56.4	62.3	65.4	67.5	69.0	70.2	34.8	40.2	43.6	45.3	46.4	47.3	61.4	66.4	65.4	66.8
Cheby	38.0	52.0	62.4	70.8	74.1	77.6	31.7	42.8	59.9	66.2	68.3	69.3	40.4	47.3	51.2	72.8
Co-training	56.6	66.4	73.5	75.9	78.9	80.8	47.3	55.7	62.1	62.5	64.5	65.5	62.2	68.3	72.7	78.2
Self-training	53.7	66.1	73.8	77.2	79.4	80.0	43.3	58.1	68.2	69.8	70.4	71.0	51.9	58.7	66.8	77.0
Union	58.5	69.9	75.9	78.5	80.4	81.7	46.3	59.1	66.7	66.7	67.6	68.2	58.4	64.0	70.7	79.2
Intersection	49.7	65.0	72.9	77.1	79.4	80.2	42.9	59.1	68.6	70.1	70.8	71.2	52.0	59.3	69.7	77.6
MultiStage	61.1	63.7	74.4	76.1	77.2		53.0	57.8	63.8	68.0	69.0		57.4	64.3	70.2	
M3S	61.5	67.2	75.6	77.8	78.0		56.1	62.1	66.4	70.3	70.5		59.2	64.4	70.6	
GCN	42.6	56.9	67.8	74.9	77.6	79.3	33.4	46.5	62.6	66.9	68.7	69.6	46.4	49.7	56.3	76.6
GCN-SVAT	43.6	53.9	71.4	75.6	78.3	78.5	47.0	52.4	65.8	68.6	69.5	70.7	52.1	56.9	63.5	77.2
GCN-DVAT	49	61.8	71.9	75.9	78.4	78.6	51.5	58.5	67.4	69.2	70.8	71.3	53.3	58.6	66.3	77.3
Linear Snowball	***69.5***	***74.1***	***79.4***	***80.4***	***81.3***	***82.4***	***56.8***	***64.3***	***68.8***	***71.0***	***72.2***	***72.3***	***64.1***	***69.5***	***72.9***	***79.3***
Snowball	***67.2***	***73.5***	***78.5***	***80.0***	***81.5***	***82.3***	***56.7***	***65.0***	***69.5***	***71.1***	***72.3***	***72.6***	***62.9***	***68.3***	***73.3***	***79.5***
Truncated Krylov	***72.2***	***75.7***	***79.2***	***80.9***	***82.5***	***82.8***	***60.9***	***66.2***	***69.0***	***71.8***	***72.4***	***73.2***	***69.1***	***71.8***	***76.1***	***80.1***

For each (column), the greener the cell, the better the performance. The redder, the worse. If our methods achieve better performance than all others, the corresponding cell will be in bold.

Table 3: Accuracy with Validation

Algorithms	Cora				CiteSeer			PubMed			
	0.5%	1%	3%	5.2% *public*	0.5%	1%	3.6% *public*	0.03%	0.05%	0.1%	0.3% *public*
Cheby	33.9	44.2	62.1	78.0	45.3	59.4	70.1	45.3	48.2	55.2	69.8
GCN-FP	50.5	59.6	71.7	74.6	43.9	54.3	61.5	56.2	63.2	70.3	76.0
GGNN	48.2	60.5	73.1	77.6	44.3	56.0	64.6	55.8	63.3	70.4	75.8
DCNN	59.0	66.4	76.7	79.7	53.1	62.2	69.4	60.9	66.7	73.1	76.8
MPNN	46.5	56.7	72.0	78.0	41.8	54.3	64.0	53.9	59.6	67.3	75.6
GraphSAGE	37.5	49.0	64.2	74.5	33.8	51.0	67.2	45.4	53.0	65.4	76.8
GAT	41.4	48.6	56.8	83.0	38.2	46.5	72.5	50.9	50.4	59.6	79.0
GCN	50.9	62.3	76.5	80.5	43.6	55.3	68.7	57.9	64.6	73.0	77.8
LNet	58.1	66.1	77.3	79.5	53.2	61.3	66.2	60.4	68.8	73.4	78.3
AdaLNet	60.8	67.5	77.7	80.4	53.8	63.3	68.7	61.0	66.0	72.8	78.1
Linear Snowball	***71.7***	***75.2***	***81.5***	***83.7***	***61.2***	***67.3***	***73.4***	***71.7***	***73.2***	***75.7***	***79.2***
Snowball	***72.9***	***76.2***	***81.1***	***83.3***	***62.4***	***67.0***	***73.2***	***70.9***	***73.0***	***76.1***	***79.5***
Truncated Krylov	***74.1***	***77.6***	***81.8***	***83.5***	***65.3***	***68.2***	***74.2***	***71.5***	***73.2***	***77.0***	***80.1***

6 Future Works

Future research of this like includes: 1) Investigating how the pointwise nonlinear activation functions influence block vectors, *e.g.*, the feature block vector $\boldsymbol{X}$ and hidden feature block vectors $\boldsymbol{H}_i$, so that we can find possible activation functions better than Tanh; 2) Finding a better way to leverage the block Krylov algorithms instead of conducting simple truncation.

Acknowledgements

The authors wish to express sincere gratitude for the computational resources of Compute Canada provided by Mila, as well as for the proofreading done by Sitao and Mingde's good friend & coworker Ian P. Porada.

References

[1] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. *arXiv*, abs/1511.02136, 2015.

[2] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *arXiv*, abs/1611.08097, 2016.

[3] J. Chen, T. Ma, and C. Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[4] J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

[5] R. R. Coifman and S. Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[6] R. R. Coifman and M. Maggioni. Diffusion wavelets. *Applied and Computational Harmonic Analysis*, 21(1):53–94, 2006.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv*, abs/1606.09375, 2016.

[8] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[9] X. Feng and Z. Zhang. The rank of a random matrix. *Applied mathematics and computation*, 185(1):689–694, 2007.

[10] A. Frommer, K. Lund, M. Schweitzer, and D. B. Szyld. The radau–lanczos method for matrix functions. *SIAM Journal on Matrix Analysis and Applications*, 38(3):710–732, 2017.

[11] A. Frommer, K. Lund, and D. B. Szyld. Block Krylov subspace methods for functions of matrices. *Electronic Transactions on Numerical Analysis*, 47:100–126, 2017.

[12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[13] M. H. Gutknecht and T. Schmelzer. The block grade of a block krylov space. *Linear Algebra and its Applications*, 430(1):174–185, 2009.

[14] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *arXiv*, abs/1706.02216, 2017.

[15] D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[16] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[18] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv*, abs/1609.02907, 2016.

[19] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[20] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Q. Li, Z. Han, and X. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv*, abs/1801.07606, 2018.

[22] R. Li, S. Wang, F. Zhu, and J. Huang. Adaptive graph convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[23] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[24] R. Liao, M. Brockschmidt, D. Tarlow, A. L. Gaunt, R. Urtasun, and R. Zemel. Graph partition neural networks for semi-supervised classification. *arXiv preprint arXiv:1803.06272*, 2018.

[25] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv*, abs/1901.01484, 2019.

[26] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[27] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[28] C. Musco, C. Musco, and A. Sidford. Stability of the lanczos method for matrix function approximation. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1605–1624. Society for Industrial and Applied Mathematics, 2018.

[29] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *arXiv preprint arXiv:1211.0053*, 2012.

[30] K. Sun, H. Guo, Z. Zhu, and Z. Lin. Virtual adversarial training on graph convolutional networks in node classification. *arXiv preprint arXiv:1902.11045*, 2019.

[31] K. Sun, Z. Zhu, and Z. Lin. Multi-stage self-supervised learning for graph convolutional networks. *arXiv*, abs/1902.11038, 2019.

[32] L. V. Tran, V. H. Vu, and K. Wang. Sparse random graphs: Eigenvalues and eigenvectors. *Random Structures & Algorithms*, 42(1):110–134, 2013.

[33] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv*, abs/1710.10903, 2017.

[34] S. Wang, Z. Zhang, and T. Zhang. Improved analyses of the randomized power method and block lanczos method. *arXiv preprint arXiv:1508.06429*, 2015.

[35] X.-M. Wu, Z. Li, A. M. So, J. Wright, and S.-F. Chang. Learning with partially absorbing random walks. In *Advances in Neural Information Processing Systems*, pages 3077–3085, 2012.

[36] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *arXiv*, abs/1901.00596, 2019.

[37] Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[38] S. Zhang, H. Tong, J. Xu, and R. Maciejewski. Graph convolutional networks: Algorithms, applications and open challenges. In *International Conference on Computational Social Networks*, pages 79–91. Springer, 2018.

Appendices

A Proofs of Theorems 1 and 2

Lemma 1. Suppose that a graph $\mathcal{G}$ has k connected components $\{C_i\}_{i=1}^k$ and L is diffusion operator defined in (3). If $\mathcal{G}$ has no bipartite components, then $\lambda_i(L) \in (-1, 1]$ with

$$\lambda_1 = \cdots = \lambda_k = 1 > |\lambda_{k+1}| \geq \cdots \geq |\lambda_N|$$

Proof See Theorem 1 in [21]. □

Theorem 1. Suppose that $\mathcal{G}$ has k connected components and the diffusion operator L is defined as that in (3). Let $\boldsymbol{X} \in \mathbb{R}^{N\times F}$ be any block vector and let W_j be any non-negative parameter matrix with $\|W_j\|_2 \leq 1$ for $j = 0, 1, \ldots$. If $\mathcal{G}$ has no bipartite components, then in (4), as $n \to \infty$, $\text{rank}(\boldsymbol{Y}') \leq k$.

Proof Note that $\boldsymbol{Y}'$ is N by F. Certainly $\text{rank}(\boldsymbol{Y}') \leq k$ if $k \geq F$. In the following we assume $k < F$.

Let $\boldsymbol{Y_0} = \text{ReLU}(L\boldsymbol{X}W_0)$, then $\boldsymbol{Y_0}$ is a non-negative block vector. Since L and W_1 are non-negative as well, we have

$$L\text{ReLU}(L\boldsymbol{Y_0}W_1)W_2 = LL\boldsymbol{Y_0}W_1W_2 = L^2\boldsymbol{Y_0}W_1W_2$$

which is non-negative. In general, it is easy to see from (4), we have

$$\boldsymbol{Y}' = L^n\boldsymbol{Y_0}W_1W_2\cdots W_n$$

Thus, with the condition $\|W_j\|_2 \leq 1$ for any j, the i-th largest singular value of $\boldsymbol{Y}'$ satisfies

$$\sigma_i(\boldsymbol{Y}') \leq \sigma_i(L^n)\|\boldsymbol{Y_0}\|_2\|W_1\|_2\cdots\|W_n\|_2 \leq |\lambda_i(L)|^n\|\boldsymbol{Y_0}\|_2, \ \ i = 1, 2, \ldots, \min\{N, F\}$$

From Lemma 1 we can conclude that

$$\lim_{n\to\infty} \sigma_i(\boldsymbol{Y}') = 0, \ \ i = k+1, k+2, \ldots, \min\{N, F\}$$

Thus, $\lim_{n\to\infty} \text{rank}(Y') \leq k$. □

Theorem 2. Suppose the n-dimensional $\boldsymbol{x}$ and $\boldsymbol{y}$ are independently sampled from a continuous distribution and the activation function $\text{Tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ is applied to $[\boldsymbol{x}, \boldsymbol{y}]$ pointwisely, then

$$\mathbb{P}(\text{rank}\,(\text{Tanh}([\boldsymbol{x}, \boldsymbol{y}])) = \text{rank}([\boldsymbol{x}, \boldsymbol{y}])) = 1$$

Proof Since $\boldsymbol{x}$ and $\boldsymbol{y}$ are sampled from a continuous distribution, $\mathbb{P}(\text{rank}([\boldsymbol{x}, \boldsymbol{y}]) = 2) = 1$ (see [9]). Then

$$\begin{aligned}
&\mathbb{P}(\text{rank}(\text{Tanh}([\boldsymbol{x}, \boldsymbol{y}])) = \text{rank}([\boldsymbol{x}, \boldsymbol{y}])) \\
&= \mathbb{P}(\text{rank}(\text{Tanh}([\boldsymbol{x}, \boldsymbol{y}])) = \text{rank}([\boldsymbol{x}, \boldsymbol{y}]) \,|\, \text{rank}([\boldsymbol{x}, \boldsymbol{y}]) = 2)\mathbb{P}(\text{rank}([\boldsymbol{x}, \boldsymbol{y}]) = 2) \\
&\quad + \mathbb{P}(\text{rank}(\text{Tanh}([\boldsymbol{x}, \boldsymbol{y}])) = \text{rank}([\boldsymbol{x}, \boldsymbol{y}]) \,|\, \text{rank}([\boldsymbol{x}, \boldsymbol{y}]) < 2)\mathbb{P}(\text{rank}([\boldsymbol{x}, \boldsymbol{y}]) < 2) \\
&= \mathbb{P}(\text{rank}(\text{Tanh}([\boldsymbol{x}, \boldsymbol{y}])) = \text{rank}([\boldsymbol{x}, \boldsymbol{y}]) \,|\, \text{rank}([\boldsymbol{x}, \boldsymbol{y}]) = 2) \qquad (11)
\end{aligned}$$

For any fixed $\boldsymbol{x} \in \mathbb{R}^n$, suppose $\boldsymbol{x}$ and random $\boldsymbol{y}$ are linearly independent, but $\text{Tanh}(\boldsymbol{x})$ and $\text{Tanh}(\boldsymbol{y})$ are linearly dependent. Without loss of generality, we assume $x_n \neq 0$. Thus $\text{Tanh}(x_n) \neq 0$ and $\text{Tanh}(x_n) \neq 0$. Then we have

$$\frac{\text{Tanh}(y_i)}{\text{Tanh}(y_n)} = \frac{\text{Tanh}(x_i)}{\text{Tanh}(x_n)}, \quad i = 2, \ldots, n$$

Thus,

$$y_i = \text{Tanh}^{-1}\left(\frac{\text{Tanh}(x_i)\text{Tanh}(y_n)}{\text{Tanh}(x_n)}\right), \quad i = 2, \ldots, n$$

For any fixed $\boldsymbol{x}$, the set formed by all $\boldsymbol{y}$ satisfying the above equalities has dimension 1, and therefore its Lebesgue measure is 0, implying that

$$\mathbb{P}(\text{rank}(\text{Tanh}([\boldsymbol{x}, \boldsymbol{y}])) = 1 \,|\, \text{rank}([\boldsymbol{x}, \boldsymbol{y}]) = 2) = 0$$

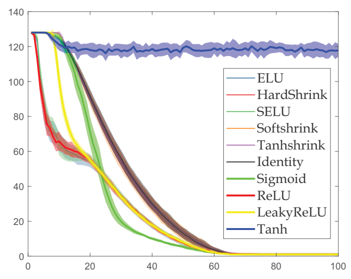
Then from (11) we can conclude the result holds. □

B Numerical Experiments on Synthetic Data

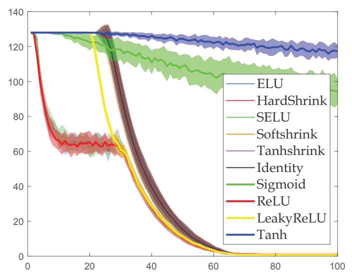
The goal of the experiments is to test which network structure with which kind of activation function has the potential to be extended to deep architecture. We measure this potential by the numerical rank of the output features in each hidden layer of the networks using synthetic data. The reason of choosing this measure can be explained by Theorem 2.2. We build the certain networks with depth 100 and the data is generated as follows.

We first randomly generate edges of an Erdős-Rényi graph $G(1000, 0.01)$, *i.e.* the existence of the edge between any pair of nodes is a Bernoulli random variable with $p = 0.01$. Then, we construct the corresponding adjacency matrix A of the graph which is a $\mathbb{R}^{1000\times1000}$ matrix. We generate a $\mathbb{R}^{1000\times500}$ feature matrix $\boldsymbol{X}$ and each of its element is drawn from $N(0, 1)$. We normalize A and $\boldsymbol{X}$ as [18] and abuse the notation $A, \boldsymbol{X}$ to denote the normalized matrices. We keep 3 blocks in each layer of truncated block Krylov network. The number of input channel in each layer depends on the network structures and the number of output channel is set to be 128 for all networks. Each element in every parameter matrix $W_i,\ i = 1, \ldots, 100$ is randomly sampled from $N(0, 1)$ and the size is $\mathbb{R}^{\#input\times\#output}$. With the synthetic $A, \boldsymbol{X}, W_i$, we simulate the feedforward process according to the network architecture and collect the numerical rank (at most 128) of the output in each of the 100 hidden layers. For each activation function under each network architecture, we repeat the experiments for 20 times and plot the mean results with standard deviation bars.

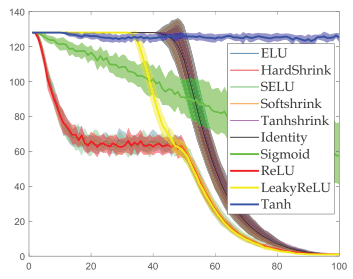
C Rank Comparison of Activation Functions and Networks



(a) GCN

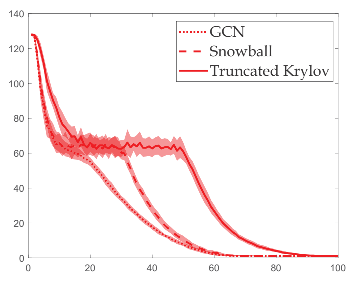


(b) Snowball

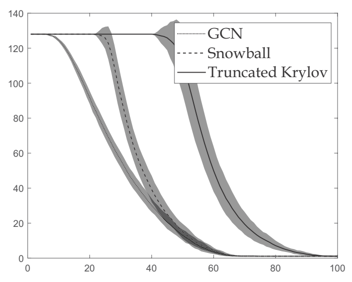


(c) Truncated Block Krylov

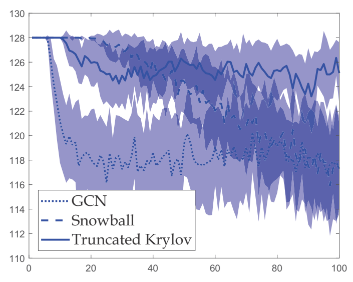
Figure 4: Column ranks of different activation functions with the same architecture



(a) ReLU

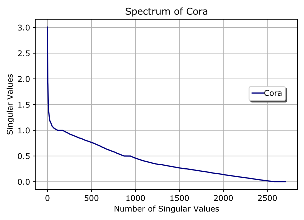


(b) Identity

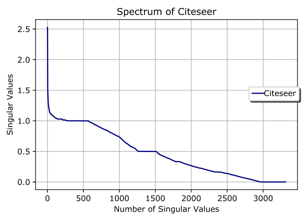


(c) Tanh

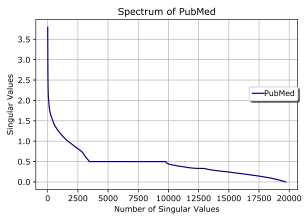
Figure 5: Column ranks of different architectures with the same activation function



(a) Cora



(b) Citeseer



(c) PubMed

Figure 6: Spectrum of the renormalized adjacency matrices for several datasets

D Spectrum of the Datasets

E Experiment Settings and Hyperparameters

The so-called public splits in [25] and the setting that randomly sample 20 instances for each class as labeled data in [37] is actually the same. Most of the results for the algorithms with validation are cited from [25], where they are reproduced with validation. However, some of them actually do not use validation in original papers and can achieve better results. In the paper, We compare with their best results.

We use NVIDIA apex amp mixed-precision plugin for PyTorch to accelerate our experiments. Most of the results were obtained from NVIDIA V100 clusters on Beluga of Compute-Canada, with minor part of them obtained from NVIDIA K20, K80 clusters on Helios Compute-Canada. The hyperparameters are searched using Bayesian optimization.

A useful tip is the smaller your training set is, the larger dropout probability should be set and the larger early stopping you should have.

Table 5 and Table 4 show the hyperparameters to achieve the performance in the experiments, for cases without and with validation, respectively. When conducting the hyperparameter search, we encounter memory problems: current GPUs cannot afford deeper and wider structures. But we do observe better performance with the increment of the network size. It is expected to achieve better performance with more advanced deep learning devices.

Table 4: Hyperparameters for Tests with Validation

Architecture	Dataset	Split	Accuracy		Corresponding Hyperparameters					
			Our Best	SOTA	learning rate	weight decay	width	depth/blocks	dropout	optimizer
linear Snowball	**Cora**	0.5%	71.69	60.8	3.0911E-05	4.7732E-02	500	17	0.37106	RMSprop
		1%	75.20	67.5	4.0685E-04	8.9625E-03	100	12	0.67302	RMSprop
		3%	81.50	77.7	1.0256E-05	2.2773E-02	1900	9	0.8031	RMSprop
		5.2% (public)	83.65	83.0	7.2385E-05	2.4428E-02	1400	12	0.91054	RMSprop
	CiteSeer	0.5%	61.21	53.8	2.0858E-03	1.6430E-02	2800	3	0.98144	RMSprop
		1%	67.26	63.3	1.7237E-03	4.0416E-02	5000	2	0.98661	RMSprop
		3.6% (public)	73.39	72.5	1.1486E-03	4.7276E-02	2800	2	0.98167	RMSprop
	Pubmed	0.03%	71.74	61.0	2.5377E-03	1.0830E-02	1700	2	0.98733	RMSprop
		0.05%	73.17	68.8	4.7712E-03	1.8898E-03	400	2	0.89331	RMSprop
		0.1%	75.68	73.4	3.7998E-04	1.6507E-02	100	11	0.17388	RMSprop
		0.3% (public)	79.22	79.0	8.1370E-04	4.0998E-02	2400	3	0.98527	RMSprop
Snowball	**Cora**	0.5%	72.90	60.8	1.7430E-04	2.4378E-02	100	23	0.6086	RMSprop
		1%	76.16	67.5	3.4202E-03	1.5751E-03	3900	2	0.98907	RMSprop
		3%	81.12	77.7	3.4523E-05	1.1752E-02	3800	5	0.16643	RMSprop
		5.2% (public)	83.32	83.0	2.4800E-05	3.5866E-02	3800	6	0.91983	RMSprop
	CiteSeer	0.5%	62.41	53.8	2.3734E-03	2.2992E-02	2500	2	0.97315	RMSprop
		1%	67.04	63.3	1.9889E-03	2.2401E-02	700	4	0.82512	RMSprop
		3.6% (public)	73.23	72.5	4.1985E-03	8.9302E-03	3400	1	0.96857	RMSprop
	Pubmed	0.03%	70.87	61.0	1.4998E-03	2.4265E-02	500	11	0.93233	RMSprop
		0.05%	73.03	68.8	1.4754E-03	3.0558E-02	400	5	0.72253	RMSprop
		0.1%	76.09	73.4	4.2362E-04	3.3066E-02	400	4	0.090822	RMSprop
		0.3% (public)	79.51	79.0	4.8091E-03	1.3221E-03	2800	1	0.98994	RMSprop
truncated Krylov	**Cora**	0.5%	74.11	60.8	1.4387E-04	9.4404E-03	3700	86	0.94346	RMSprop
		1%	77.55	67.5	3.0239E-03	1.8363E-03	4500	32	0.98817	RMSprop
		3%	81.81	77.7	1.5997E-03	1.8666E-04	2200	14	0.9814	RMSprop
		5.2% (public)	83.51	83.0	8.9107E-04	3.4034E-03	500	30	0.049966	RMSprop
	CiteSeer	0.5%	65.28	53.8	3.3864E-03	4.9753E-02	4600	34	0.9842	RMSprop
		1%	68.21	63.3	1.5359E-03	1.1404E-02	3700	22	0.91228	RMSprop
		3.6% (public)	74.18	72.5	2.8980E-03	4.2862E-02	2200	19	0.98689	RMSprop
	Pubmed	0.03%	71.45	61.0	4.5592E-04	3.7077E-03	5000	7	0.98677	RMSprop
		0.05%	73.24	68.8	3.8475E-03	9.1710E-03	1600	7	0.98375	RMSprop
		0.1%	77.01	73.4	3.8877E-03	1.5528E-02	3200	6	0.97849	RMSprop
		0.3% (public)	80.12	79.0	1.4288E-03	1.6897E-02	4200	7	0.017084	RMSprop

Table 5: Hyperparameters for Tests without Validation

Architecture	Dataset	Percentage	Accuracy		Correspondong Hyperparameters					
			Ours	SOTA	lr	weight_decay	hidden	layers/n_blocks	dropout	Optimizer
linear Snowball	**Cora**	0.5%	69.53	61.5	4.4438E-05	1.7409E-02	550	12	0.007753	Adam
		1%	74.12	69.9	1.0826E-03	3.3462E-03	1250	3	0.50426	Adam
		2%	79.43	75.9	2.4594E-06	9.6734E-03	1650	12	0.34073	Adam
		3%	80.41	78.5	2.8597E-05	3.4732E-02	900	15	0.039034	Adam
		4%	81.3	80.4	3.6830E-05	1.5664E-02	3750	4	0.93797	Adam
		5%	82.19	81.7	5.8323E-06	8.5940E-03	2850	5	0.14701	Adam
	CiteSeer	0.5%	56.76	56.1	4.5629E-03	2.0106E-03	300	3	0.038225	Adam
		1%	65.44	62.1	3.5530E-05	4.9935E-02	600	6	0.03556	Adam
		2%	68.78	68.6	6.1176E-06	3.0101E-02	1950	3	0.040484	Adam
		3%	71	70.3	2.1956E-05	4.3569E-02	3350	3	0.30207	Adam
		4%	72.23	70.8	9.1952E-05	4.6407E-02	3350	2	0.018231	Adam
		5%	72.21	71.3	3.7173E-03	1.9605E-03	2950	1	0.96958	Adam
	Pubmed	0.03%	64.133	62.2	1.0724E-03	8.1097E-03	64	4	0.8022	RMSProp
		0.05%	69.48	68.3	1.5936E-03	3.0236E-03	6	10	0.73067	RMSProp
		0.1%	72.93	72.7	4.9733E-03	1.3744E-03	128	3	0.91214	RMSProp
		0.3%	79.33	79.2	1.7998E-03	9.6753E-04	512	1	0.97483	RMSProp
Snowball	**Cora**	0.5%	67.15	61.5	9.8649E-04	1.0305E-02	1600	3	0.92785	Adam
		1%	73.47	69.9	1.4228E-04	1.3472E-02	100	13	0.68601	Adam
		2%	78.54	75.9	5.7111E-06	1.5544E-02	600	13	0.022622	Adam
		3%	79.97	78.5	4.0278E-05	2.7287E-02	4350	5	0.57173	Adam
		4%	81.49	80.4	1.4152E-05	2.3359E-02	2500	13	0.018578	Adam
		5%	81.82	81.7	1.2621E-03	1.5323E-02	3550	2	0.87352	Adam
	CiteSeer	0.5%	56.39	56.1	2.6983E-03	2.5370E-02	300	6	0.82964	Adam
		1%	65.04	62.1	1.6982E-03	1.5473E-02	2150	2	0.98611	Adam
		2%	69.48	68.6	9.7299E-05	4.9675E-02	2150	3	0.71216	Adam
		3%	71.09	70.3	1.7839E-04	3.0874E-02	2150	2	0.16549	Adam
		4%	72.32	70.8	5.6575E-05	3.5949E-02	4800	2	0.012576	Adam
		5%	72.8	71.3	2.8643E-04	1.6399E-02	2000	2	0.37308	Adam
	Pubmed	0.03%	62.94	62.2	1.2700E-03	1.4159E-03	128	4	0.76848	RMSProp
		0.05%	68.31	68.3	1.1224E-03	9.9166E-05	256	3	0.85496	RMSProp
		0.1%	73.29	72.7	6.0506E-04	1.0303E-03	256	2	0.97988	RMSProp
		0.3%	79.63	79.2	1.1416E-03	6.1543E-04	128	1	0.989	RMSProp
truncated Krylov	**Cora**	0.5%	72.96	61.5	3.3276E-03	1.0496E-04	128	18	0.76012	RMSProp
		1%	75.52	69.9	7.4797E-04	9.1736E-03	2048	20	0.98941	RMSProp
		2%	80.31	75.9	1.7894E-04	1.1079E-02	4096	16	0.97091	RMSProp
		3%	81.54	78.5	4.3837E-04	2.6958E-03	512	17	0.96643	RMSProp
		4%	82.47	80.4	3.6117E-03	4.1040E-04	64	25	0.021987	RMSProp
		5%	83.36	81.7	1.0294E-03	5.3882E-04	256	23	0.028392	RMSProp
	CiteSeer	0.5%	59.6	56.1	1.9790E-03	4.0283E-04	16	20	0.007761	RMSProp
		1%	65.95	62.1	7.8506E-04	8.2432E-03	64	24	0.28159	RMSProp
		2%	70.23	68.6	5.4517E-04	1.0818E-02	256	12	0.27027	RMSProp
		3%	71.81	70.3	1.4107E-04	5.0062E-03	1024	9	0.57823	RMSProp
		4%	72.36	70.8	4.8864E-06	1.8038E-02	4096	12	0.11164	RMSProp
		5%	72.24	71.3	2.1761E-03	1.1753E-02	5000	8	0.71473	Adam
	Pubmed	0.03%	69.07	62.2	6.8475E-04	2.8822E-02	4096	7	0.97245	RMSProp
		0.05%	71.77	68.3	2.3342E+04	2.2189E-03	1024	8	0.93694	RMSProp
		0.1%	76.07	72.7	4.2629E-04	4.1339E-03	2048	8	0.98914	RMSProp
		0.3%	80.04	79.2	2.2602E-04	3.3626E-02	2000	7	0.070573	Adam