GeniePath: Graph Neural Networks with Adaptive Receptive Paths

Ziqi Liu,[†] Chaochao Chen,[†] Longfei Li,[†] Jun Zhou,[†]
Xiaolong Li,[†] Le Song,[†‡] Yuan Qi[†]
[†]Ant Financial Services Group, China, [‡]Georgia Institute of Technology, Atlanta, USA
{ziqiliu, chaochao.ccc, longyao.llf, jun.zhoujun, xl.li, yuan.qi}@antfin.com, lsong@cc.gatech.edu

Abstract

We present, GeniePath, a scalable approach for learning adaptive receptive fields of neural networks defined on permutation invariant graph data. In GeniePath, we propose an adaptive path layer consists of two complementary functions designed for breadth and depth exploration respectively, where the former learns the importance of different sized neighborhoods, while the latter extracts and filters signals aggregated from neighbors of different hops away. Our method works in both transductive and inductive settings, and extensive experiments compared with competitive methods show that our approaches yield state-of-the-art results on large graphs.

Introduction

In this paper, we study the representation learning task involving data that lie in an irregular domain, i.e. graphs. Many data have the form of a graph, e.g., social networks (Perozzi, Al-Rfou, and Skiena 2014), citation networks (Sen et al. 2008), biological networks (Zitnik and Leskovec 2017), and transaction networks (Liu et al. 2017). We are interested in graphs with permutation invariant properties, i.e., the ordering of the neighbors for each node is irrelevant to the learning tasks. This is in opposition to temporal graphs (Kostakos 2009).

Convolutional Neural Networks (CNN) have been proven successful in a diverse range of applications involving images (He et al. 2016a) and sequences (Gehring et al. 2016). Recently, interests and efforts have emerged in the literature trying to generalize convolutions to graphs (Hammond, Vandergheynst, and Gribonval 2011; Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2016; Hamilton, Ying, and Leskovec 2017a), which also brings in new challenges.

Unlike image and sequence data that lie in regular domains, graph data are irregular in nature, making the receptive field of each neuron different for different nodes in the graph. Assuming a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $i \in \mathcal{V}$, $|\mathcal{E}|$ edges $(i, j) \in \mathcal{E}$, the sparse adjacency matrix $A \in \mathbb{R}^{N \times N}$, diagonal node degree matrix D ($D_{ii} = \sum_{ij} A_{ij}$), and a matrix of node features $X \in \mathbb{R}^{N,P}$. Consider the following calculations in typical graph convolutional networks: $H^{(t+1)} = \sigma\big(\phi(A) H^{(t)} W^{(t)}\big)$ at the t-th layer parameterized by $W^{(t)} \in \mathbb{R}^{K \times K}$, where $H^{(t)} \in \mathbb{R}^{N \times K}$ denotes the intermediate embeddings of N nodes at the t-th layer. In the case where $\phi(A) = D^{-1}A$ and we ignore the activation function σ, after T times iterations, it yields $H^{(T)} = \phi(A)^T H^{(0)} W^1$, with the T-th order transition matrix $\phi(A)^T$ (Gagniuc 2017) as the pre-defined receptive field. That is, the depth of layers T determines the extent of neighbors to exploit, and any node j which satisfies $d(i, j) \leq T$, where d is the shortest path distance, contributes to node i's embedding, with the importance weight pre-defined as $\phi(A)^T_{ij}$. Essentially, the receptive field of one target node in graph domain is equivalent to the subgraph that consists of nodes along paths to the target node.

Is there a specific path in the graph contributing mostly to the representation? Is there an adaptive and automated way of choosing the receptive fields or paths of a graph convolutional network? It seems that the current literature has not provided such a solution. For instance, graph convolution neural networks which lie in spectral domain (Bruna et al. 2013) heavily rely on the graph Laplacian matrix (Chung 1997) $L = I - D^{-1/2}AD^{-1/2}$ to define the importance of the neighbors (and hence receptive field) for each node. Approaches that lie in spatial domain define convolutions directly on the graph, with receptive field more or less hand-designed. For instance, GraphSage (Hamilton, Ying, and Leskovec 2017b) used the mean or max of a fixed-size neighborhood of each node, or an LSTM aggregator which needs a pre-selected order of neighboring nodes. These pre-defined receptive fields, either based on graph Laplacian in the spectral domain, or based on uniform operators like mean, max operators in the spatial domain, thus limit us from discovering meaningful receptive fields from graph data adaptively. For example, the performance of GCN (Kipf and Welling 2016) based on graph Laplacian could deteriorate severely if we simply stack more and more layers to explore deeper and wider receptive fields (or paths), even though the situation could be alleviated, to some extent, if adding residual nets as extra add-ons (Hamilton, Ying, and Leskovec 2017b; Veličković et al. 2017).

To address the above challenges, (1) we first formulate the space of functionals wherein any eligible aggregator func-

[1]We collapse $\prod_{i=0}^{T} W^{(i)}$ as W, and $H^{(0)} = X \in \mathbb{R}^{N \times P}$.

tions should satisfy, for permutation invariant graph data; (2) we propose adaptive path layer with two complementary components: adaptive breadth and depth functions, where the adaptive breadth function can adaptively select a set of significant important one-hop neighbors, and the adaptive depth function can extract and filter useful and noisy signals up to long order distance. (3) experiments on several datasets empirically show that our approaches are quite competitive, and yield state-of-the-art results on large graphs. Another remarkable result is that our approach is less sensitive to the depth of propagation layers.

Intuitively our proposed adaptive path layer guides the breadth and depth exploration of the receptive fields. As such, we name such adaptively learned receptive fields as receptive paths.

Graph Convolutional Networks

Generalizing convolutions to graphs aims to encode the nodes with signals lie in the receptive fields. The output encoded embeddings can be further used in end-to-end supervised learning (Kipf and Welling 2016) or unsupervised learning tasks (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016).

The approaches lie in spectral domain heavily rely on the graph Laplacian operator $L = I - D^{-1/2}AD^{-1/2}$ (Chung 1997). The real symmetric positive semidefinite matrix L can be decomposed into a set of orthonormal eigenvectors which form the graph Fourier basis $U \in \mathbb{R}^{N \times N}$ such that $L = U\Lambda U^\top$, where $\Lambda = \mathrm{diag}(\lambda_1, ..., \lambda_n) \in \mathbb{R}^{N \times N}$ is the diagonal matrix with main diagonal entries as ordered nonnegative eigenvalues. As a result, the convolution operator in spatial domain can be expressed through the element-wise Hadamard product in the Fourier domain (Bruna et al. 2013). The receptive fields in this case depend on the kernel U.

Kipf and Welling (2016) further propose GCN to design the following approximated localized 1-order spectral convolutional layer:

$$H^{(t+1)} = \sigma\big(\tilde{A}H^{(t)}W^{(t)}\big), \tag{1}$$

where $\tilde{A}$ is a symmetric normalization of A with self-loops, i.e. $\tilde{A} = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$, $\hat{A} = A + I$, $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$, $H^{(t)} \in \mathbb{R}^{N,K}$ denotes the t-th hidden layer with $H^{(0)} = X$, $W^{(t)}$ is the layer-specific parameters, and σ denotes the activation functions.

GCN requires the graph Laplacian to normalize the neighborhoods in advance. This limits the usages of such models in inductive settings. One trivial solution (GCN-mean) instead is to average the neighborhoods:

$$H^{(t+1)} = \sigma\big(\tilde{A}H^{(t)}W^{(t)}\big), \tag{2}$$

where $\tilde{A} = \hat{D}^{-1}\hat{A}$ is the row-normalized adjacency matrix.

More recently, Hamilton et al. (2017a) proposed GraphSAGE, a method for computing node representation in inductive settings. They define a series of aggregator functions in the following framework:

$$H^{(t+1)} = \sigma\Big(\mathrm{CONCAT}\big(\phi(A, H^{(t)}), H^{(t)}\big)W^{(t)}\Big), \tag{3}$$

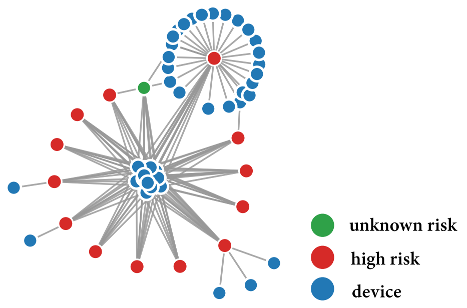


Figure 1: A fraud detection case: accounts with high risk (red nodes), accounts with unknown risk (green nodes), and devices (blue nodes). An edge between an account and a device means that the account has logged in via the device during a period.

where $\phi(\cdot)$ is an aggregator function over the graph. For example, their mean aggregator $\phi(A, H) = D^{-1}AH$ is nearly equivalent to GCN-mean (Eq. 2), but with additional residual architectures (He et al. 2016a). They also propose max pooling and LSTM (Long short-term memory) (Hochreiter and Schmidhuber 1997) based aggregators. However, the LSTM aggregator operates on a random permutation of nodes' neighbors that is not a permutation invariant function with respect to the ordering of neighborhoods, thus makes the operators hard to understand.

A recent work from (Veličković et al. 2017) proposed Graph Attention Networks (GAT) which uses attention mechanisms to parameterize the aggregator function $\phi(A, H; \Theta)$. This method is restricted to learn a direct neighborhood receptive field, which is not as general as our proposed approach which can explore receptive field in both breadth and depth directions.

Note that, this paper focuses mainly on works related to graph convolutional networks, but may omit some other types of graph neural networks.

Discussions

To summarize, the major effort put on this area is to design effective aggregator functions that can propagate signals around the T-th order neighborhood for each node. Few of them try to learn the meaningful paths that direct the propagation.

Why learning receptive paths is important? The reasons could be twofold: (1) graphs could be noisy, (2) different nodes play different roles, thus it yields different receptive paths. For instance, in Figure 1 we show the graph patterns from real fraud detection data. This is a bipartite graph where we have two types of nodes: accounts and devices (e.g. IP proxy). Malicious accounts (red nodes) tend to aggregate together as the graph tells. However, in reality, normal and malicious accounts could connect to the same IP proxy. As a result, we cannot simply tell from the graph patterns that the account labeled in "green" is definitely a malicious one. It is important to verify if this account behave in the similar patterns as other malicious accounts, i.e. according to the features of each node. Thus, node features could

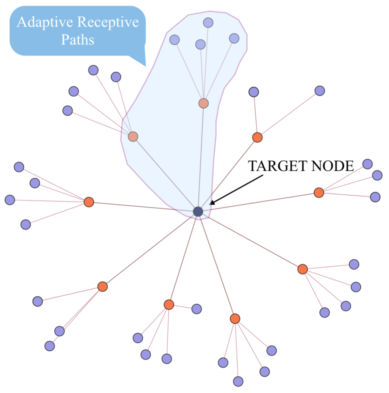


Figure 2: A motivated illustration of meaningful receptive paths (shaded) given all two-hops neighbors (red and blue nodes), with the black node as target node.

be additional signals to refine the importance of neighbors and paths. In addition, we should pay attention to the number of hops each node can propagate. Obviously, the nodes at frontier would not aggregate signals from a hub node, else it would make everything "over-spread".

We demonstrate the idea of learning receptive paths of graph neural networks in Figure 2. Instead of aggregating all the 2-hops neighbors to calculate the embedding of the target node (black), we aim to learn meaningful receptive paths (shaded region) that contribute mostly to the target node. The meaningful paths can be viewed as a subgraph associated to the target node. What we need is to do breadth/depth exploration and filter useful/noisy signals. The breadth exploration determines which neighbors are important, i.e. leads the direction to explore, while the depth exploration determines how many hops of neighbors away are still useful. Such signal filterings on the subgraphs essentially learn receptive paths.

Proposed Approaches

In this section, we first discuss the space of functionals satisfying the permutation invariant requirement for graph data. Then we design neural network layers which satisfy such requirement while at the same time have the ability to learn "receptive paths" on graph.

Permutation Invariant

We aim to learn a function f that maps $\mathcal{G}, \mathcal{H}$, i.e. the graph and associatd (latent) feature spaces, into the range $\mathcal{Y}$. We have node i, and the 1-order neighborhood associated with i, i.e. $\mathcal{N}(i)$, and a set of (latent) features in vector space $\mathbb{R}^K$ belongs to the neighborhood, i.e. $\{h_j | j \in \mathcal{N}(i)\}$. In the case of permutation invariant graphs, we assume the learning task is independent of the order of neighbors.

Now we require the (aggregator) function f acting on the neighbors must be invariant to the orders of neighbors under any random permutation σ, i.e.

$$f(\{h_1, ..., h_j, ...\} | j \in \mathcal{N}(i)) = f(\{h_{\sigma(1)}, ..., h_{\sigma(j)}, ...\} | j \in \mathcal{N}(i)) \tag{4}$$

Algorithm 1: A generic algorithm of GeniePath.

```
Input: Depth $T$, node features $H^{(0)} = X$, adjacency matrix $A$.
Output: $\Theta$ and $\Phi$
1 while not converged do
2     for $t = 1$ to $T$ do
3         $H^{(tmp)} = \phi(A, H^{(t-1)}; \Theta)$ (breadth function)
4         $H^{(t)} = \varphi\Big(H^{(tmp)} | \big\{H^{(\tau)} | \tau \in \{t-1, ..., 0\}\big\}; \Phi\Big)$ (depth function)
5     end
6     Backpropagation based on loss $\mathcal{L}(H^{(T)}, \cdot)$
7 end
8 return $\Theta$ and $\Phi$
```

Theorem 1 (Permutation Invariant) *A function f operating on the neighborhood of i can be a valid function, i.e. with the assumption of permutation invariant to the neighbors of i, if and only if the f can be decomposed into the form $\rho(\sum_{j \in \mathcal{N}(i)} \phi(h_j))$ with mappings ϕ and ρ.*

Proof sketch. It is trivial to show that the sufficiency condition holds. The necessity condition holds due to the Fundamental Theorem of Symmetric Functions (Zaheer et al. 2017).

Remark 2 (Associative Property) *If f is permutation invariant with respect to a neighborhood $\mathcal{N}(\cdot)$, $g \circ f$ is still permutation invariant with respect to the neighborhood $\mathcal{N}(\cdot)$ if g is independent of the order of $\mathcal{N}(\cdot)$. This allows us to stack the functions in a cascading manner.*

In the following, we will use the permutation invariant requirement and the associative property to design propagation layers. It is trivial to check that the LSTM aggregator in GraphSAGE is not a valid function under this condition, even though it could be possibly an appropriate aggregator function in temporal graphs.

Adaptive Path Layer

Our goal is to learn receptive paths while propagate signals along the learned paths rather than the pre-defined paths. The problem is equivalent to determining a proper subgraph through breadth (which one-hop neighbor is important) and depth (the importance of neighbors at the t-th hop away) expansions for each node.

To explore the breadth of the receptive paths, we learn an adaptive breadth function $\phi(A, H^{(t)}; \Theta)$ parameterized by Θ, to aggregate the signals by adaptively assigning different importances to different one-hop neighbors. To explore the depth of the receptive paths, we learn an adaptive depth function $\varphi\Big(h_i^{(t)} | \big\{h_i^{(\tau)} | \tau \in \{t-1, ..., 0\}\big\}; \Phi\Big)$ parameterized by Φ (shared among all nodes $i \in \mathcal{V}$) that could further extract and filter the aggregated signals at the t-th order distance away from the anchor node i by modeling the dependencies among aggregated signals at various depths.

We summarize the overall "GeniePath" algorithm in Algorithm 1.

The parameterized functions in Algorithm 1 are optimized by the customer-defined loss functions. This may includes supervised learning tasks (e.g. multi-class, multi-label), or unsupervised tasks like the objective functions defined in (Perozzi, Al-Rfou, and Skiena 2014).

Now it remains to specify the adaptive breadth and depth functions, i.e. $\phi(.)$ and $\varphi(.)$. We make the adaptive path layer more concrete as follows:

$$h_i^{(\mathbf{tmp})} = \tanh\left(W^{(t)^\top} \sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha(h_i^{(t)}, h_j^{(t)}) \cdot h_j^{(t)}\right) \tag{5}$$

then

$$\begin{aligned} i_i &= \sigma(W_i^{(t)^\top} h_i^{(\mathbf{tmp})}), & f_i &= \sigma(W_f^{(t)^\top} h_i^{(\mathbf{tmp})}) \\ o_i &= \sigma(W_o^{(t)^\top} h_i^{(\mathbf{tmp})}), & \tilde{C} &= \tanh(W_c^{(t)^\top} h_i^{(\mathbf{tmp})}) \\ C_i^{(t+1)} &= f_i \odot C_i^{(t)} + i_i \odot \tilde{C}, \end{aligned}$$

and finally,

$$h_i^{(t+1)} = o_i \odot \tanh(C_i^{(t+1)}). \tag{6}$$

The first equation (Eq. (5)) corresponds to $\phi(.)$ and the rest gated units correspond to $\varphi(.)$. We maintain for each node i a memory C_i (initialized as $C_i^{(0)} \leftarrow \vec{0} \in \mathbb{R}^K$) , and gets updated as the receptive paths being explored $t = 0, 1, ..., T$. At the $(t+1)$-th layer, i.e. while we are exploring the neighborhood $\{j | d(i,j) \leq t+1\}$, we have the following two complementary functions.

Adaptive breadth function. Eq. (5) assigns the importance of any one-hop neighbors' embedding $h_j^{(t)}$ by the parameterized generalized linear attention operator $\alpha(\cdot,\cdot)$ as follows

$$\alpha(x, y) = \text{softmax}_y\left(v^\top \tanh(W_s^\top x + W_d^\top y)\right), \tag{7}$$

where $\text{softmax}_y f(\cdot, y) = \frac{\exp f(\cdot,y)}{\sum_{y'} \exp f(\cdot,y')}$.

Adaptive depth function. The gated unit i_i with sigmoid output (in the range $(0,1)$) is used to extract newly useful signals from $\tilde{C}$, and be added to the memory as $i_i \odot \tilde{C}$. The gated unit f_i is used to filter useless signals from the old memory given the newly observed neighborhood $\{j | d(i,j) \leq t+1\}$ by $f_i \odot C_i^{(t)}$. As such, we are able to filter the memory for each node i as $C_i^{(t+1)}$ while we extend the depth of the neighborhood. Finally, with the output gated unit o_i and the latest memory $C_i^{(t+1)}$, we can output node i's embedding at $(t+1)$-th layer as $h_i^{(t+1)}$.

The overall architecture of adaptive path layer is illustrated in Figure 3. We let $h_i^{(0)} = W_x^\top X_i$ with $X_i \in \mathbb{R}^P$ as node i's feature. The parameters to be optimized are weight matrix $W_x \in \mathbb{R}^{P,K}$, $\Theta = \{W, W_s, W_d \in \mathbb{R}^{K,K}, v \in \mathbb{R}^K\}$, and $\Phi = \{W_i, W_f, W_o, W_c \in \mathbb{R}^{K,K}\}$ that are extremely compact (at the same scale as other graph neural networks).

Note that the adaptive path layer with only adaptive breadth function $\phi(.)$ reduces to the proposal of GAT, but with stronger nonlinear representation capacities. Our generalized linear attention can assign symmetric importance with constraint $W_s = W_d$. We do not limit $\varphi(\cdot)$ as a LSTM-like function but will report this architecture in experiments.

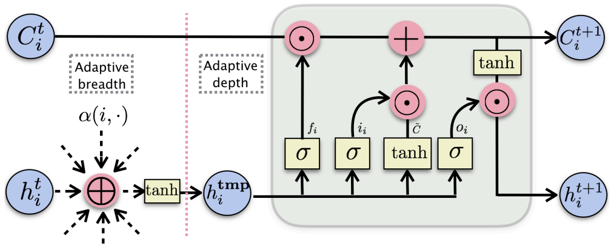


Figure 3: A demonstration for the architecture of GeniePath. Symbol $\bigoplus$ denotes the operator $\sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha(h_i^{(t)}, h_j^{(t)}) \cdot h_j^{(t)}$.

A variant. Next, we propose a variant called "GeniePath-lazy" that postpones the evaluations of the adaptive depth function $\varphi(.)$ at each layer. We rather propagate signals up to T-th order distance by merely stacking adaptive breadth functions. Given those hidden units $\{h_i^{(0)}, ..., h_i^{(t)}, ..., h_i^{(T)}\}$, we add adaptive depth function $\varphi(.)$ on top of them to further extract and filter the signals at various depths. We initialize $\mu_i^{(0)} = W_x^\top X_i$, and feed $\mu_i^{(T)}$ to the final loss functions. Formally given $\{h_i^{(t)}\}$ we have:

$$\begin{aligned} i_i &= \sigma(W_i^{(t)^\top} \text{CONCAT}(h_i^{(t)}, \mu_i^{(t)})), \\ f_i &= \sigma(W_f^{(t)^\top} \text{CONCAT}(h_i^{(t)}, \mu_i^{(t)})), \\ o_i &= \sigma(W_o^{(t)^\top} \text{CONCAT}(h_i^{(t)}, \mu_i^{(t)})), \\ \tilde{C} &= \tanh(W_c^{(t)^\top} \text{CONCAT}(h_i^{(t)}, \mu_i^{(t)})), \\ C_i^{(t+1)} &= f_i \odot C_i^{(t)} + i_i \odot \tilde{C}, \\ \mu_i^{(t+1)} &= o_i \odot \tanh(C_i^{(t+1)}). \end{aligned} \tag{8}$$

Permutation invariant. Note that the adaptive breadth function $\phi(\cdot)$ in our case operates on a linear transformation of neighbors, thus satisfies the permutation invariant requirement stated in Theorem 1. The function $\varphi(\cdot)$ operates on layers at each depth is independent of the order of any 1-step neighborhood, thus the composition of $\varphi \circ \phi$ is still permutation invariant.

Efficient Numerical Computation

Our algorithms could be easily formulated in terms of linear algebra, thus could leverage the power of Intel MKL (Intel 2007) on CPUs or cuBLAS (Nvidia 2008) on GPUs. The major challenge is to numerically efficient calculate Eq. (5). For example, GAT (Veličković et al. 2017) in its first version performs attention over all nodes with masking applied by way of an additive bias of $-\infty$ to the masked entries' coefficients before apply the softmax, which results into $\mathcal{O}(N^2)$ in terms of computation and storage.

Table 1: Dataset summary.

Dateset	V	E	# Classes	# Features	Label rate (train / test)
Pubmed	$19,717$	$88,651$	3	500	0.3% / 5.07%
BlogCatalog[1]	$10,312$	$333,983$	39	0	50% / 40%
BlogCatalog[2]	$10,312$	$333,983$	39	128	50% / 40%
PPI	$56,944$	$818,716$	121	50	78.8% / 9.7%
Alipay	$981,748$	$2,308,614$	2	$4,000$	20.5% / 3.2%

Instead, all we really need is the attention entries in the same scale as adjacency matrix A, i.e. $|\mathcal{E}|$. Our trick is to build two auxiliary sparse matrices $L \in \{0,1\}^{|\mathcal{E}|,N}$ and $R \in \{0,1\}^{|\mathcal{E}|,N}$ both with $|\mathcal{E}|$ entries. For the i-th row of L and R, we have $L_{ii'} = 1$, $R_{ij'} = 1$, and (i',j') corresponds to an edge of the graph. After that, we can do $LHW_s + RHW_d$ for the transformations on all the edges in case of generalized linear form in Eq. (7). As a result, our algorithm complexity and storage is still in linear with $\mathcal{O}(|\mathcal{E}|)$ as other type of GNNs.

Experiments

In this section, we first discuss the experimental results of our approach evaluated on various types of graphs compared with strong baselines. We then study its abilities of learning adaptive depths. Finally we give qualitative analyses on the learned paths compared with graph Laplacian.

Datasets

Transductive setting.

The *Pubmed* (Sen et al. 2008) is a type of citation networks, where nodes correspond to documents and edges to undirected citations. The classes are exclusive, so we treat the problem as a multi-class classification problem. We use the exact preprocessed data from Kipf and Welling (2016).

The *BlogCatalog*[1] (Zafarani and Liu 2009) is a type of social networks, where nodes correspond to bloggers listed on BlogCatalog websites, and the edges to the social relationships of those bloggers. We treat the problem as a multi-label classification problem. Different from other datasets, the BlogCatalog[1] has no explicit features available, as a result, we encode node ids as one-hot features for each node, i.e. $10,312$ dimensional features. We further build dataset *BlogCatalog*[2] with 128 dimensional features decomposed by SVD on the adjacency matrix A.

The *Alipay* dataset (Liu et al. 2017) is a type of Account-Device Network, built for detecting malicious accounts in the online cashless payment system at Alipay. The nodes correspond to users' accounts and devices logged in by those accounts. The edges correspond to the login relationships between accounts and devices during a time period. Node features are counts of login behaviors discretized into hours and account profiles. There are 2 classes in this dataset, i.e. malicious accounts and normal accounts. The Alipay dataset is random sampled during one week. The dataset consists of $82,246$ disjoint subgraphs.

Inductive setting.

The *PPI* (Hamilton, Ying, and Leskovec 2017a) is a type of protein-protein interaction networks, which consists of 24 subgraphs with each corresponds to a human tissue (Zitnik and Leskovec 2017). The node features are extracted by positional gene sets, motif gene sets and immunological signatures. There are 121 classes for each node from gene ontology. Each node could have multiple labels, then results into a multi-label classification problem. We use the exact preprocessed data provided by Hamilton et al. (2017a). There are 20 graphs for training, 2 for validation and 2 for testing.

We summarize the statistics of all the datasets in Table 1.

Experimental Settings

Comparison Approaches We compare our methods with several strong baselines.

(**1**) MLP (multilayer perceptron), utilizes only the node features but not the structures of the graph. (**2**) node2vec (Grover and Leskovec 2016). Note that, this type of methods built on top of lookup embeddings cannot work on problems with multiple graphs, i.e. on datasets Alipay and PPI, because without any further constraints, the entire embedding space cannot guarantee to be consistent during training (Hamilton, Ying, and Leskovec 2017a). (**3**) Chebyshev (Defferrard, Bresson, and Vandergheynst 2016), which approximates the graph spectral convolutions by a truncated expansion in terms of Chebyshev polynomials up to T-th order. (**4**) GCN (Kipf and Welling 2016), which is defined in Eq. (1). Same as Chebyshev, it works only in the transductive setting. However, if we just use the normalization formulated in Eq. (2), it can work in an inductive setting. We denote this variant of GCN as GCN-mean. (**5**) GraphSAGE (Hamilton, Ying, and Leskovec 2017a), which consists of a group of pooling operators and skip connection architectures as discussed in section . We will report the best results of GraphSAGEs with different pooling strategies as GraphSAGE*. (**6**) Graph Attention Networks (GAT) (Veličković et al. 2017) is similar to a reduced version of our approach with only adaptive breadth function. This can help us understand the usefulness of adaptive breadth and depth function.

We pick the better results from GeniePath or GeniePath-lazy as GeniePath*. We found out that the residual architecture (skip connections) (He et al. 2016b) are useful for stacking deeper layers for various approaches and will report the approaches with suffix "-residual" to distinguish the contributions between graph convolution operators and skip connection architecture.

Table 2: Summary of testing results on Pubmed, BlogCatalog and Alipay in the transductive setting. In accordance with former benchmarks, we report accuracy for Pubmed, Macro-F1 for BlogCatalog, and F1 for Alipay.

Transductive

Methods	Pubmed	BlogCatalog[1]	BlogCatalog[2]	Alipay
MLP	71.4%	-	0.134	0.741
node2vec	65.3%	0.136	0.136	-
Chebyshev	74.4%	0.160	0.166	0.784
GCN	**79.0%**	0.171	0.174	0.796
GraphSAGE*	78.8%	0.175	0.175	0.798
GAT	78.5%	**0.201**	0.197	0.811
GeniePath*	78.5%	0.195	**0.202**	**0.826**

Table 3: Summary of testing Micro-F1 results on PPI in the inductive setting.

Inductive

Methods	PPI
MLP	0.422
GCN-mean	0.71
GraphSAGE*	0.768
GAT	0.81
GeniePath*	**0.979**

Table 4: A comparison of GAT, GeniePath, and additional residual "skip connection" on PPI.

Methods	PPI
GAT	0.81
GAT-residual	0.914
GeniePath	0.952
GeniePath-lazy	0.979
GeniePath-lazy-residual	**0.985**

Experimental Setups In our experiments, we implement our algorithms in TensorFlow (Abadi et al. 2016) with the Adam optimizer (Kingma and Ba 2014). For all the graph convolutional network-style approaches, we set the hyper-parameters include the dimension of embeddings or hidden units, the depth of hidden layers and learning rate to be same. For node2vec, we tune the return parameter p and in-out parameter q by grid search. Note that setting $p = q = 1$ is equivalent to DeepWalk (Perozzi, Al-Rfou, and Skiena 2014). We sample 10 walk-paths with walk-length as 80 for each node in the graph. Additionally, we tune the penalty of ℓ_2 regularizers for different approaches.

Transductive setting. In transductive settings, we allow all the algorithms to access to the whole graph, i.e. all the edges among the nodes, and all of the node features.

For pubmed, we set the number of hidden units as 16 with 2 hidden layers. For BlogCatalog[1] and BlogCatalog[2], we set the number of hidden units as 128 with 3 hidden layers. For Alipay, we set the number of hidden units as 16 with 7 hidden layers.

Inductive setting. In inductive settings, all the nodes in test are *completely unobserved* during the training procedure. For PPI, we set the number of hidden units as 256 with 3 hidden layers.

Classification

We report the comparison results of transductive settings in Table 2. For Pubmed, there are only 60 labels available for training. We found out that GCN works the best by stacking exactly 2 hidden convolutional layers. Stacking 2 more convolutional layers will deteriorate the performance severely. Fortunately, we are still performing quite competitive on this data compared with other methods. Basically, this dataset is quite small that it limits the capacity of our methods.

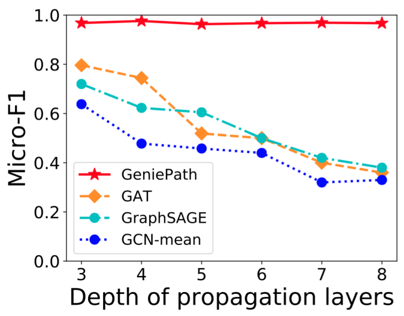
In BlogCatalog[1] and BlogCatalog[2], we found both GAT and GeniePath* work the best compared with other methods. Another suprisingly interesting result is that graph convolutional networks-style approaches can perform well even without explicit features. This works only in transductive setting, and the look-up embeddings (in this particular case) of testing nodes can be propagated to nodes in training.

The graph in Alipay (Liu et al. 2017) is relative sparse. We found that GeniePath* works quite promising on this large graph. Since the dataset consists of ten thousands of subgraphs, the node2vec is not applicable in this case.

We report the comparison results of inductive settings on PPI in Table 3. We use "GCN-mean" by averaging the neighborhoods instead of GCN for comparison. GeniePath* performs extremely promising results on this big graph, and shows that the adaptive depth function plays way important compared to GAT with only adaptive breadth function.

To further study the contributions of skip connection architectures, we compare GAT, GeniePath, and GeniePath-lazy with additional residual architecture ("skip connections") in Table 4. The results on PPI show that GeniePath is less sensitive to the additional "skip connections". The significant improvement of GAT on this dataset relies heavily on the residual structure.

In most cases, we found GeniePath-lazy converges faster than GeniePath and other GCN-style models.



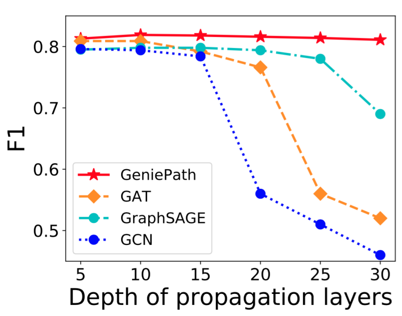


Figure 4: The classification measures with respect to the depths of propagation layers: PPI (**left**), Alipay (**right**).

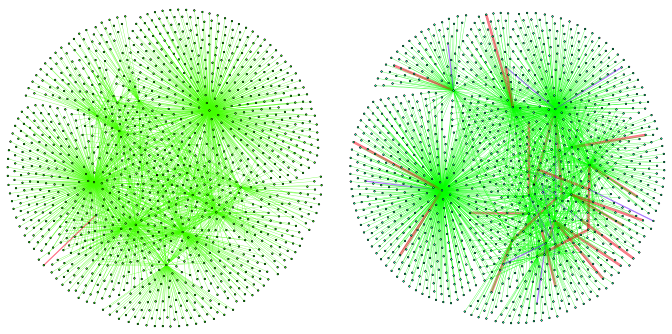


Figure 5: Graph Laplacian (**left**) v.s. Estimated receptive paths (**right**) with respect to the *black node* on PPI dataset: we retain all the paths to the black node in 2-hops that involved in the propagation, with edge thickness denotes the importance $w_{i,j} = \alpha(h_i, h_j)$ of edge (i, j) estimated in the first adaptive path layer. We also discretize the importance of each edge into 3 levels: Green ($w_{i,j} < 0.1$), Blue ($0.1 \leq w_{i,j} < 0.2$), and Red ($w_{i,j} \geq 0.2$).

Depths of Propagation

We show our results on classification measures with respect to the depths of propagation layers in Figure 4. As we stack more graph convolutional layers, i.e. with deeper and broader receptive fields, GCN, GAT and even GraphSAGE with residual architectures can hardly maintain consistently resonable results. Interestingly, GeniePath with adaptive path layers can adaptively learn the receptive paths and achieve consistent results, which is remarkable.

Qualitative Analysis

We show a qualitative analysis about the receptive paths with respect to a sampled node learned by GeniePath in Figure 5. It can be seen, the graph Laplacian assigns importance of neighbors at nearly the same scale, i.e. results into very dense paths (every neighbor is important). However, the GeniePath can help select significantly important paths (red ones) to propagate while ignoring the rest, i.e. results into much sparser paths. Such "neighbor selection" processes essentially lead the direction of the receptive paths and improve the effectiveness of propagation.

Conclusion

In this paper, we studied the problems of graph convolutional networks on identifying meaningful receptive paths. We proposed adaptive path layers with adaptive breadth and depth functions to guide the receptive paths. Our experiments on large benchmark data show that GeniePath significantly outperfoms state-of-the-art approaches, and are less sensitive to the depths of the stacked layers, or extent of neighborhood set by hand. The success of GeniePath shows that selecting appropriate receptive paths for different nodes is important. In future, we expect to further study the sparsification of receptive paths, and help explain the potential propagations behind graph neural networks in specific applications. In addition, studying temporal graphs that the ordering of neighbors matters could be another challenging problem.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; and Isard, M. 2016. Tensorflow: a system for large-scale machine learning.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

Chung, F. R. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 3844–3852.

Gagniuc, P. A. 2017. *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons.

Gehring, J.; Auli, M.; Grangier, D.; and Dauphin, Y. N. 2016. A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864. ACM.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017a. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017b. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

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

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 630–645. Springer.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Intel, M. 2007. Intel math kernel library.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kostakos, V. 2009. Temporal graphs. *Physica A: Statistical Mechanics and its Applications* 388(6):1007–1023.

Liu, Z.; Chen, C.; Zhou, J.; Li, X.; Xu, F.; Chen, T.; and Song, L. 2017. Poster: Neural network-based graph embedding for malicious accounts detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 2543–2545. New York, NY, USA: ACM.

Nvidia, C. 2008. Cublas library. *NVIDIA Corporation, Santa Clara, California* 15(27):31.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710. ACM.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine* 29(3):93.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Zafarani, R., and Liu, H. 2009. Social computing data repository at asu.

Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Poczos, B.; Salakhutdinov, R.; and Smola, A. 2017. Deep sets. *arXiv preprint arXiv:1703.06114*.

Zitnik, M., and Leskovec, J. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33(14):i190–i198.