# Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification

Chenyi Zhuang, Qiang Ma
Department of Informatics, Kyoto University, Kyoto, Japan
zhuang@db.soc.i.kyoto-u.ac.jp,qiang@i.kyoto-u.ac.jp

## ABSTRACT

The problem of extracting meaningful data through graph analysis spans a range of different fields, such as the internet, social networks, biological networks, and many others. The importance of being able to effectively mine and learn from such data continues to grow as more and more structured data become available. In this paper, we present a simple and scalable semi-supervised learning method for graph-structured data in which only a very small portion of the training data are labeled. To sufficiently embed the graph knowledge, our method performs graph convolution from different views of the raw data. In particular, a dual graph convolutional neural network method is devised to jointly consider the two essential assumptions of semi-supervised learning: (1) *local consistency* and (2) *global consistency*. Accordingly, two convolutional neural networks are devised to embed the local-consistency-based and global-consistency-based knowledge, respectively. Given the different data transformations from the two networks, we then introduce an unsupervised temporal loss function for the ensemble. In experiments using both unsupervised and supervised loss functions, our method outperforms state-of-the-art techniques on different datasets.

## KEYWORDS

Graph convolutional networks, Semi-supervised learning, Graph diffusion, Adjacency matrix, Pointwise mutual information

**ACM Reference Format:**
Chenyi Zhuang, Qiang Ma. 2018. Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.3186116

## 1 INTRODUCTION

The explosion in the availability of structured and semi-structured data (e.g., from the internet, social science, physics, biology, and computer science) has led to a wealth of research focusing on the analysis of graphs. Moreover, modern companies often wish to store information in the form of a corporate knowledge graph and apply a variety of machine learning and data mining techniques (e.g., graph mining [8], relational learning [19], security [25], knowledge embedding [27]) for different applications.


*WWW 2018, April 23–27, 2018, Lyon, France*

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186116

One problem of significant interest is the classification of graph nodes from only a small portion of labeled training data and the graph structure. In the context of machine learning, graph-based semi-supervised learning is one such technique that aims to use the graph structure to train models with higher accuracy when only a limited set of labeled data is available. Accordingly, in this paper, we propose a new general semi-supervised learning algorithm that can be applied to different kinds of graphs, such as social networks, knowledge graphs, citation networks, World Wide Web, and so on.

Conventionally, graph-based semi-supervised learning could be defined through the following loss function:

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{reg}, \tag{1}$$

where $\mathcal{L}_0$ denotes the supervised loss with respect to the labeled data and $\mathcal{L}_{reg}$ denotes the regularizer with respect to the graph structure. (Note that, in Eqs. (1), (2), and (3), we omit a model's own parameter regularization terms for simplicity.) Using an explicit graph-based regularizer $\mathcal{L}_{reg}$, the formulation of Eq. (1) smooths the label information in $\mathcal{L}_0$ over the whole graph. For instance, a graph Laplacian regularization term is typically used in $\mathcal{L}_{reg}$ [3] [5] [33], which relies on the prior assumption that connected nodes in the graph are likely to share the same label. However, this assumption might restrict the modeling capacity, as graph edges need not encode the node similarity, but could instead contain additional information.

To avoid this limitation, recent studies have attempted to consider both label and graph structure information in a convolutional manner [2] [9] [15]. In Eq. (2), instead of using an explicit graph-based regularizer in the loss function, a convolutional function *Conv* is derived to encode the graph structure directly.

$$\mathcal{L} = \mathcal{L}_0(Conv). \tag{2}$$

In approximate terms, the structure encoding *Conv* could be conducted in two domains: (1) the graph vertex domain [2] [32]; and (2) the graph spectral domain [9] [15]. In [24], the authors discussed the relationship between these two domains.

However, by using Eqs. (1) and (2), most of the related work have only considered the *local consistency* of a graph for knowledge embedding. To sufficiently embed the graph knowledge, we find that the *global consistency* of a graph has not been well investigated yet. Hence, in this paper, we propose a dual graph convolutional neural network method to jointly take both of them into consideration. The form of the loss function in the proposed strategy is

$$\mathcal{L} = \mathcal{L}_0(Conv_A) + \lambda(t)\mathcal{L}_{reg}(Conv_A, Conv_P). \tag{3}$$

The idea of our method is simple. First, during the convolutional process, sample input data (e.g., a feature vector of a node) pass through two kinds of convolutional networks: $Conv_A$ and $Conv_P$.

Using the graph adjacency matrix and positive pointwise mutual information matrix, the two convolutional networks encode the local and global structure information. Corresponding to the two essential assumptions in semi-supervised learning [32], $Conv_A$ embeds the *local-consistency*-based knowledge (i.e., nearby data points are likely to have the same label), whereas $Conv_P$ embeds the *global-consistency*-based knowledge (i.e., data points that occur in similar contexts tend to have the same label).

During training, a sample makes multiple passes through the two convolutional networks and the random layer-wise dropout [26]. Thus, different transformations of the sample are obtained. The output of either $Conv_A$ or $Conv_P$ is then used for supervised learning, e.g., $\mathcal{L}_0(Conv_A)$. However, to give better predictions, an ensemble-oriented regularizer $\mathcal{L}_{reg}(Conv_A, Conv_P)$ for these transformations is derived. By minimizing the difference between predictions from different transformations of an input sample, the regularizer combines the opinions of $Conv_A$ and $Conv_P$. Accordingly, $\mathcal{L}_{reg}(Conv_A, Conv_P)$ is an unsupervised loss function.

Overall, the main contributions of this work can be summarized as follows:

(1) In addition to the graph adjacency matrix-based convolution $Conv_A$, we propose a new convolutional neural network $Conv_P$ that depends on the positive pointwise mutual information (PPMI) matrix. Unlike $Conv_A$, which embeds local-consistency-based knowledge, we employ a random walk to construct a PPMI matrix that further embeds semantic information, i.e., global-consistency-based knowledge.
(2) In addition to the supervised learning on a small portion of labeled training data (i.e., $\mathcal{L}_0$ in Eq. (3)), an unsupervised loss function (i.e., $\mathcal{L}_{reg}$ in Eq. (3)) is proposed as a kind of regularizer to combine the output of different convolved data transformations. In a series of experiments considering both $Conv_A$ and $Conv_P$, our method is shown to yield better predictions than a single convolutional network.

## 2 BACKGROUND TO GRAPH-BASED SEMI-SUPERVISED LEARNING

Semi-supervised learning considers the general problem of learning from labeled and unlabeled data. Given a set of data points $\mathcal{X} = \{x_1, ..., x_l, x_{l+1}, ..., x_n\}$ and a set of labels $C = \{1, ..., c\}$, the first $l$ points have labels $\{y_1, ..., y_l\} \in C$ and the remaining points are unlabeled. The goal is to predict the labels of the unlabeled points.

In addition to the labeled and unlabeled points, graph-based semi-supervised learning also involves a given graph, denoted as an $n \times n$ matrix $A$. Each entry $a_{i,j} \in A$ indicates the *similarity* between data points $x_i$ and $x_j$. The *similarity* can be derived by calculating the distances among data points [33], or may be explicitly given by structured data, such as knowledge graphs [29], citation graphs [14], hyperlinks between documents [20], and so on. Therefore, the key problem of graph-based semi-supervised learning concerns how to embed the additional information of the graph for better label prediction. In approximate terms, we classify the different graph knowledge embeddings into two groups, i.e., explicit and implicit graph-based semi-supervised learning.

### 2.1 Explicit Graph-Based Semi-Supervised Learning

Explicit graph-based semi-supervised learning uses a graph-based regularizer (i.e., $\mathcal{L}_{reg}$ in Eq. (1)) to incorporate the information in the graph. Conventionally, a graph Laplacian regularizer is defined so as to incur a large penalty when similar data points $x_i$ and $x_j$ with a large $a_{i,j}$ are predicted to have different labels $f(x_i) \neq f(x_j)$.

$$\mathcal{L}_{reg} = \sum_{i,j} a_{i,j} ||f(x_i) - f(x_j)||^2 = \mathbf{f}^{\mathsf{T}} \Delta \mathbf{f}. \tag{4}$$

Eq. (4) presents an instance of the graph Laplacian regularizer, where $f(\cdot)$ is the label prediction function (e.g., a neural network). The unnormalized graph Laplacian is defined as $\Delta = A - D$, where $A$ is the adjacency matrix and $D$ is a diagonal matrix with each entry defined as $d_{i,i} = \sum_j a_{i,j}$.

Many related methods have been proposed as variants of Eq. (4). For instance, in [33], the authors proposed a label propagation algorithm based on Gaussian Random Fields. In [32], a PageRank-like algorithm is proposed to account for both local and global consistency in the graph. Recently, in [30], the authors proposed a sampling-based method. Instead of using the graph Laplacian $\Delta$, they derived a random walk-based sampling algorithm to obtain the positive and negative contexts for each data point. A feed-forward neural network method was then used for knowledge embedding.

### 2.2 Implicit Graph-Based Semi-Supervised Learning

As mentioned in Section 1, a convolutional process for implicit graph-based semi-supervised learning could be conducted in either the graph vertex domain or the graph spectral domain. We now introduce some work related to each case.

**Convolution in the vertex domain.** To apply convolution in the vertex domain, a data point $x_i$ will usually be transformed in a diffusive manner. A simple example of a $k$-hop localized linear transform is

$$Conv(x_i) = b_{i,i} x_i + \sum_{j \in \mathcal{N}(i,k)} b_{i,j} x_j, \tag{5}$$

where $\{b_{i,j}\}$ are some weights for filtering and $\mathcal{N}(i,k)$ denotes the set of neighbors connected to $x_i$ by a path of $k$ or fewer edges. Such a convolution is built on the idea of a diffusion kernel, which can be thought of as a measure of the level of connectivity between any two nodes in a graph. For example, by introducing a damping ratio, longer paths will be discounted more than shorter paths. A recent survey paper [10] compared several diffusion kernels on graphs.

Recently, several methods have used diffusion-based convolution. For instance, in [2], the authors proposed a diffusion-convolutional neural network. In their method, the $k$-hop diffusion-convolutional results (i.e., a large tensor) are directly used as the input to a neural network. As a result, significant amounts of memory are required to record the input. In [15], the authors proposed a scalable method that conducts 1-hop diffusion on each layer of a neural network. This approach obtained state-of-the-art performance in semi-supervised classification.

**Convolution in the spectral domain.** We first consider the most simple situation of scalar $x_i$. In this context, the input $X \in$

$\mathbb{R}^{n\times 1}$ is considered as a signal defined on the graph with $n$ nodes. As shown in Eq. (6), the spectral convolution on a graph can then be defined as the multiplication of the signal $X$ with a filter $g_\theta = diag(\theta)$ parametrized by $\theta \in \mathbb{R}^n$ in the graph Fourier domain.

$$Conv(X) = g_\theta * X = U g_\theta U^T X, \tag{6}$$

where $U$ is the matrix of eigenvectors of the normalized Laplacian matrix $\Delta = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, which plays the role of the Fourier transform. When $x_i$ has more than one feature, we can regard $X$ as a signal with multiple input channels. The transformation $U$ is then applied to each channel [13].

In [24], the authors discussed different ways to define graph spectral domains and explained the graph Fourier transform in detail. As Eq. (6) is computationally expensive for large graphs, recent studies [9] [12] [15] have attempted to reduce the computational complexity. For instance, in [12], the authors approximated $g_\theta$ using a truncated expansion in terms of Chebyshev polynomials.

**Relation between the two domains.** In [24], the authors identified the relation between convolutions in the vertex and spectral domains. That is, when the filter function $g_\theta$ is approximated as an order-$k$ polynomial, the spectral convolution can be interpreted as a $k$-hop diffusion convolution. This conclusion was later verified [15]. Thus, by approximating $g_\theta$ as an order-1 Chebyshev polynomial, after some derivation, it is equivalent to a 1-hop diffusion.

Our work is mainly inspired by [15]. In addition to adjacency matrix-based convolution, we further calculate a PPMI matrix for encoding the semantic information during the convolution. Furthermore, a new regularizer is proposed to combine the different convolutional results for better label prediction.

# 3 DUAL GRAPH CONVOLUTIONAL NETWORKS

## 3.1 Problem Definition and an Example

Following the notation in Sections 1 and 2, the input to our model includes a set of data points $\mathcal{X} = \{x_1, ..., x_l, x_{l+1}, ..., x_n\}$, the labels $\{y_1, ..., y_l\}$ for the first $l$ points, and the graph structure. Assuming that each point has at most $k$ features, the dataset is denoted as a matrix $X \in \mathbb{R}^{n\times k}$. As in previous studies [2] [9] [15] [32], the graph structure is represented by the adjacency matrix $A \in \mathbb{R}^{n\times n}$.[1] Given the input $X$, $\{y_1, ..., y_l\}$, and $A$, our model aims to predict the labels of the unlabeled points.

As shown in Figure 1a, we use the Karate club network [31] as an example to visualize some intermediate results. In this example, as the data points (i.e., graph nodes) do not have any features, we initialize $X$ as the identity matrix. Namely, each node is described by a different one-hot vector of length 34. Our objective is to classify all the nodes into two groups (labeled red and green) correctly.

In the remainder of this section, we introduce our method in three steps. First, for *local consistency*, we introduce the convolutional method using the graph adjacency matrix $A$. Another convolutional method based on a random walk is then proposed to encode the semantic information for *global consistency*. Finally, we introduce a regularizer for the ensemble.

[1] If the edges have attributes, we can add additional edge nodes for encoding. For detailed pre-processing, please refer to the NELL dataset in our experiments.

## 3.2 Local Consistency Convolution: $Conv_A$

By directly utilizing the state-of-the-art method proposed in [15], we formulate the graph-structure-based convolution $Conv_A$ as a type of feed-forward neural network. Given the input feature matrix $X$ and adjacency matrix $A$, the output of the $i$-th hidden layer of the network $Z^{(i)}$ is defined as:

$$Conv_A^{(i)}(X) = Z^{(i)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} Z^{(i-1)} W^{(i)}). \tag{7}$$

$\tilde{A} = A + I_n$, where $I_n \in \mathbb{R}^{n\times n}$ is the identity matrix, is the adjacency matrix with self-loops and $\tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$. Accordingly, $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix. $Z^{(i-1)}$ is the output of the $(i-1)$-th layer, and $Z^{(0)} = X$. $W^{(i)}$ are the trainable parameters of the network, and $\sigma(\cdot)$ denotes an activation function (e.g., ReLU, Sigmoid).

In [15], the authors derived Eq. (7) from the spectral-domain-based convolution in [9] [12], i.e., Eq. (6). In this context, the parameters $W^{(i)}$ in Eq. (7) correspond to the parameters $\theta$ of the filtering function $g_\theta$. A detailed explanation is presented in [15].

The role of $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} Z^{(i-1)}$ in Eq. (7) is to exactly conduct a 1-hop diffusion process in each layer. Namely, a node's feature vector is enriched by linearly adding all the feature vectors of its neighbors. This discovery inspired the proposed concept. That is, this method can be further improved by reducing the exceptions to *local consistency* in semi-supervised learning: nearby points are likely to have the same label. For example, from Figure 1a, as the directly connected data points $x_8$ and $x_{30}$ have different labels, their convolved feature vectors should not be similar. However, Eq. (7) cannot deal with such an exception in an effective manner.

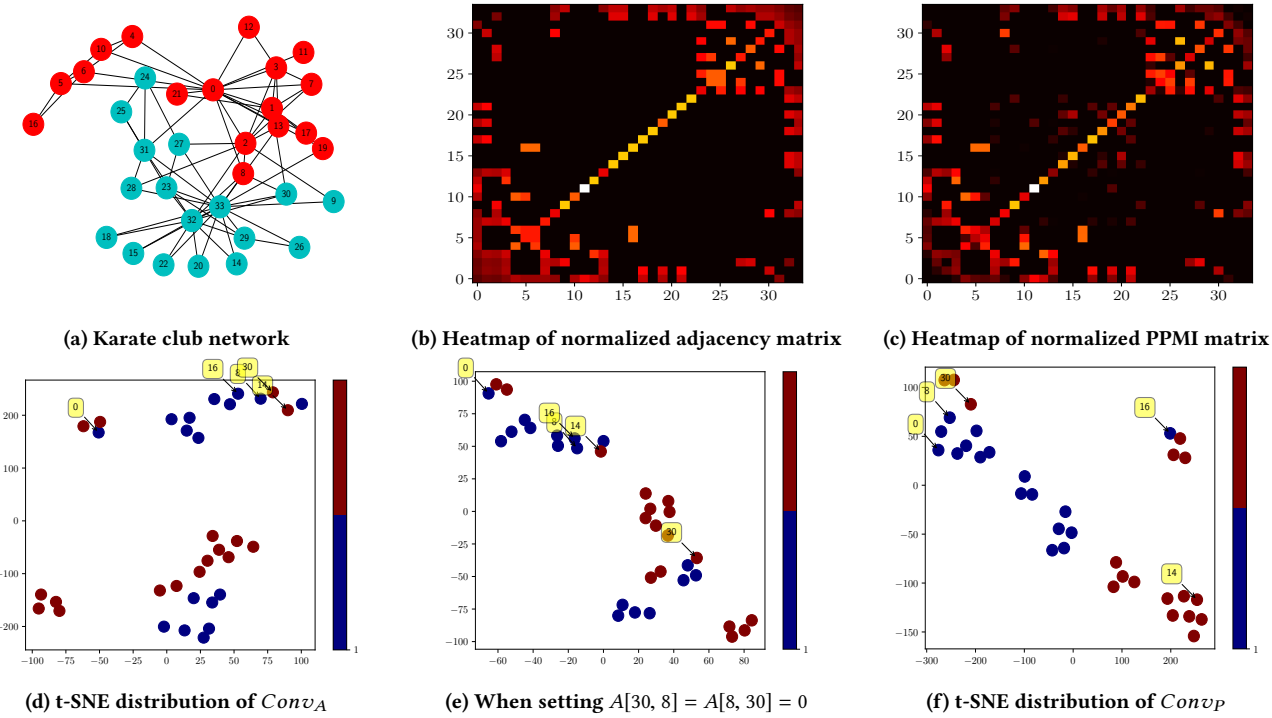
To verify our idea, we visualize the Karate club network's convolved result, i.e., the output of $Conv_A$, using t-stochastic neighbor embeddings (SNEs) [17]. Given $X$ (an identity matrix) and the normalized $\tilde{A}$ (see Figure 1b), a neural network with two hidden layers is constructed and all the parameters, i.e., $W^{(1)}$ and $W^{(2)}$, are randomly initialized. No training is conducted.

From the visualized results in Figure 1d, as expected, $x_8$ and $x_{30}$ are close together. However, they belong to different groups. To verify the proposed concept, we manually delete the edge between $x_8$ and $x_{30}$, i.e., setting $A[8, 30] = A[30, 8] = 0$. As a result, Figure 1e presents the new t-SNE distribution of all 34 data points, where $x_8$ and $x_{30}$ are far apart. Hence, the attendant problem is how to automatically reduce the number of such exceptions.

In the next subsection, we introduce a PPMI-based convolution method. By encoding semantic information, this method allows different latent representations to be learnt for each data point. By devising an ensemble, we then automatically reduce the number of exceptions while avoiding the introduction of additional noise.

## 3.3 Global Consistency Convolution: $Conv_P$

In addition to the graph structure information defined by the adjacency matrix $A$, we further apply PPMI to encode the semantic information, which is denoted as a matrix $P \in \mathbb{R}^{n\times n}$. We first calculate a frequency matrix $F$ using a random walk. Based on $F$, we then calculate $P$ and explain why it leverages knowledge from the frequency to semantics. Finally, we define the $P$-based convolution function $Conv_P$.



(a) Karate club network (b) Heatmap of normalized adjacency matrix (c) Heatmap of normalized PPMI matrix

(d) t-SNE distribution of $Conv_A$ (e) When setting $A[30, 8] = A[8, 30] = 0$ (f) t-SNE distribution of $Conv_P$

**Figure 1: Illustration of our proposed method. (a) shows the Karate club network [31]; (b) and (c) present heatmaps of the normalized adjacency matrix $\tilde{A}$ and the PPMI matrix $P$, respectively; (d), (e), and (f) present the t-stochastic neighbor embedding (SNE) distributions of the different convolved results obtained by $Conv_A$ and $Conv_P$. Note that no training was conducted.**

**Algorithm 1** Calculate Frequency Matrix $F$

**Input:** adjacency matrix $A$, path length $q$, window size $w$, walks per node $\gamma$.
**Output:** frequency matrix $F \in \mathbb{R}^{n \times n}$.
Initialize $F$ with zeros
**for** each node $x_i \in \mathcal{X}$ **do**
  set $x_i$ as a path root
  **for** $i = 0$ to $\gamma$ **do**
    $S = RandomWalk(A, x_i, q)$ ▷ get a path $S$ using Eq. (8)
    Uniformly sample all pairs $(x_n, x_m) \in S$ within $w$
    **for** each pair $(x_n, x_m)$ **do**
      $F_{n,m} += 1.0$; $F_{m,n} += 1.0$
    **end for**
  **end for**
**end for**

**Calculating frequency matrix $F$.** The Markov chain describing the sequence of nodes visited by a random walker is called a random walk. If the random walker is on node $x_i$ at time $t$, we define the state as $s(t) = x_i$. The transition probability of jumping from the current node $x_i$ to one of its neighbors $x_j$ is denoted as $p(s(t+1) = x_j | s(t) = x_i)$. In our problem setting, given the adjacency matrix $A$, we assign:

$$p(s(t+1) = x_j | s(t) = x_i) = A_{i,j} / \sum_j A_{i,j}. \tag{8}$$

Algorithm 1 describes the process of calculating $F$ using a random walk. The time complexity is $O(n\gamma q^2)$; as the parameters $\gamma$ and $q$ are small integers, $F$ can be calculated quickly. Furthermore, the algorithm could be parallelized by conducting several random walks simultaneously on different parts of a graph.

Random walks have been used as a similarity measure for a variety of problems in recommendation [11], graph classification [1], and semi-supervised learning [30]. In our method, we use a random walk to calculate the semantic similarity between nodes.

**Calculating PPMI.** After calculating the frequency matrix $F$, the $i$-th row in $F$ is the row vector $F_{i,:}$ and the $j$-th column in $F$ is the column vector $F_{:,j}$. $F_{i,:}$ corresponds to a node $x_i$ and $F_{:,j}$ corresponds to a context $c_j$. Based on Algorithm 1, the contexts are defined as all nodes in $\mathcal{X}$. The value of an entry $F_{i,j}$ is the number of times that $x_i$ occurs in context $c_j$. Based on $F$, we calculate the PPMI matrix $P \in \mathbb{R}^{n \times n}$ as:

$$\begin{aligned} p_{i,j} &= \frac{F_{i,j}}{\sum_{i,j} F_{i,j}}; \\ p_{i,*} &= \frac{\sum_j F_{i,j}}{\sum_{i,j} F_{i,j}}; \\ p_{*,j} &= \frac{\sum_i F_{i,j}}{\sum_{i,j} F_{i,j}}; \\ P_{i,j} &= max\{pmi_{i,j} = log(\frac{p_{i,j}}{p_{i,*}\, p_{*,j}}), 0\}. \end{aligned} \tag{9}$$

Applying Eq. (9) encodes the semantic information in $P$. That is, $p_{i,j}$ is the estimated probability that node $x_i$ occurs in context $c_j$; $p_{i,*}$ is the estimated probability of node $x_i$; and $p_{*,j}$ is the estimated probability of context $c_j$. Based on the definition of statistical independence, if $x_i$ and $c_j$ are independent (i.e., $x_i$ occurs in $c_j$ by pure random chance), then $p_{i,j} = p_{i,*}p_{*,j}$, and thus $pmi_{i,j} = 0$. Accordingly, if there is a semantic relation between $x_i$ and $c_j$, then $p_{i,j}$ is expected to be greater than if $x_i$ and $c_j$ are independent. Hence, when $p_{i,j} > p_{i,*}p_{*,j}$, $pmi_{i,j}$ should be positive. If node $x_i$ is unrelated to context $c_j$, $pmi_{i,j}$ may be negative. As we are focusing on pairs $(x_i, c_j)$ that have a semantic relation, our method uses a nonnegative *pmi*.

PPMI has been extensively investigated in terms of natural language processing (NLP) [4] [16] [28]. Indeed, the PPMI metric is known to perform well on semantic similarity tasks [4]. However, to the best of our knowledge, we are the first to introduce PPMI to the field of graph-based semi-supervised learning. Furthermore, using a novel PPMI-based convolution, our method applies the concept of *global consistency*: graph nodes that occur in similar contexts tend to have the same label.

Figure 1c visualizes the normalized PPMI matrix $P$ of the Karate club network. Compared with the adjacency matrix of this network (shown in Figure 1b), there are at least two obvious differences: (1) $P$ has reduced the effect of the hub nodes, e.g., $x_0$ and $x_{33}$; and (2) $P$ has initiated more latent relations among different data points, which cannot be characterized by the adjacency matrix $A$.

**PPMI-based convolution.** In addition to the convolution $Conv_A$, which is based on the similarity defined by the adjacency matrix $A$, another feed-forward neural network $Conv_P$ is derived from the similarity defined by the PPMI matrix $P$. This convolutional neural network is given by:

$$Conv_P^{(i)}(X) = Z^{(i)} = \sigma(D^{-\frac{1}{2}}PD^{-\frac{1}{2}}Z^{(i-1)}W^{(i)}), \tag{10}$$

where $P$ is the PPMI matrix and $D_{i,i} = \sum_j P_{i,j}$ for normalization. Obviously, applying diffusion based on such a node-contextual matrix $P$ ensures *global consistency*. Additionally, by using the same neural network structure as $Conv_A$, the two can be combined very concisely.

Figure 1f presents the t-SNE distribution of the output of $Conv_P$ applied to the Karate club network. We find that $x_8$ and $x_{30}$ are now slightly farther away from each other. A more encouraging result is that, compared with the results shown in Figures 1d and 1e, $Conv_P$ has correctly classified data points $x_0$ and $x_{14}$. However, $x_{16}$ is not assigned an ideal latent representation by $Conv_P$. Accordingly, in the next subsection, we introduce a novel ensemble method to jointly consider both the local and global consistencies.

## 3.4 Ensemble of Local and Global Consistencies

To jointly consider the *local consistency* and *global consistency* for semi-supervised learning, we must overcome the challenge of having very few labeled training data. That is, as the training data are limited, a general ensemble method (e.g., by concatenating the output of $Conv_A$ and $Conv_P$) cannot be utilized. In Appendix A, we discuss this issue in detail. The lack of training data causes the ensemble method introduced in Appendix A to achieve worse performance than non-ensemble methods. Hence, in addition to supervised learning using training data, we further derive an unsupervised regularizer for the ensemble.

Figure 2 presents the architecture of our dual graph convolutional networks method. In addition to training $Conv_A$ using the labeled data (i.e., $\mathcal{L}_0(Conv_A)$ in Eq. (3)), an unsupervised regularizer (i.e., $\mathcal{L}_{reg}(Conv_A, Conv_P)$ in Eq. (3)) is introduced to train $Conv_P$ against the posterior probabilities of a previously trained model, i.e., the trained $\mathcal{L}_0(Conv_A)$. To explain this self-ensembling method, the remainder of this section describes the calculation of $\mathcal{L}_0(Conv_A)$ and $\mathcal{L}_{reg}(Conv_A, Conv_P)$ sequentially. The learning algorithm is then introduced.

**Calculating $\mathcal{L}_0(Conv_A)$.** Assuming there are $c$ different labels for prediction, the $softmax$ activation function is applied row-wise to the output $Z^A \in \mathbb{R}^{n\times c}$ given by $Conv_A$. The output of the $softmax$ layer is denoted as $\hat{Z}^A \in \mathbb{R}^{n\times c}$. $\mathcal{L}_0(Conv_A)$, which evaluates the cross-entropy error over all labeled data points, is calculated as:

$$\mathcal{L}_0(Conv_A) = -\frac{1}{|\mathcal{Y}_L|}\sum_{l\in\mathcal{Y}_L}\sum_{i=1}^{c} Y_{l,i} ln \hat{Z}^A_{l,i}, \tag{11}$$

where $\mathcal{Y}_L$ is the set of data indices whose labels are observed for training and $Y \in \mathbb{R}^{n\times c}$ is the ground truth.

**Calculating $\mathcal{L}_{reg}(Conv_A, Conv_P)$.** The calculation of $\mathcal{L}_{reg}$ is given by:

$$\mathcal{L}_{reg}(Conv_A, Conv_P) = \frac{1}{n}\sum_{i=1}^{n} \| \hat{Z}^P_{i,:} - \hat{Z}^A_{i,:} \|^2 . \tag{12}$$

Similar to $Conv_A$, after applying the $softmax$ activation function, the output of $Conv_P$ is denoted as $\hat{Z}^P \in \mathbb{R}^{n\times c}$. Over all $n$ data points, we introduce an unsupervised loss function that minimizes the mean squared differences between $\hat{Z}^P$ and $\hat{Z}^A$.

By looking at the formulation of Eq. (12), we could regard the unsupervised loss function as training $Conv_P$ against $Conv_A$. That is, after the $\mathcal{L}_0$-based training (i.e., Eq. (11)), the softmaxed scores in $\hat{Z}^A \in \mathbb{R}^{n\times c}$ are then interpreted as a posterior distribution over the $c$ labels. By minimizing the loss function in Eq. (12), despite different transformations by $Conv_A$, $Conv_P$, and random layer-wise dropout, the final predictions given by each model should then be the same.

As shown in Figure 2, the key to our model is to share the model parameters (i.e., neural network weights $W$ in Eqs. (7) and (10)) in $Conv_A$ and $Conv_P$. By doing so, our model can jointly consider the opinions of both $Conv_A$ and $Conv_P$. Although sharing the same parameters $W$, the different diffusions (i.e., $A$ and $P$) and random dropout may cause the predictions of $Conv_A$ and $Conv_P$ (i.e., $\hat{Z}^A$ and $\hat{Z}^P$) to differ. However, we know that each data point is assigned to only one class. Therefore, the model (which is characterized by the parameters $W$) is expected to give the same prediction from $Conv_A$ and $Conv_P$, i.e., minimizing Eq. (12). As a result, the trained parameters $W$ have considered the opinions from both $Conv_A$ and $Conv_P$.

We are aware that a recently proposed transformation/stability loss [23] is based on a similar principle as our ensemble method. By explicitly incorporating the prior knowledge (i.e., the diffusion matrices $A$ and $P$ in our method) during the data transformation stage, our ensemble method can be regarded as a further extension

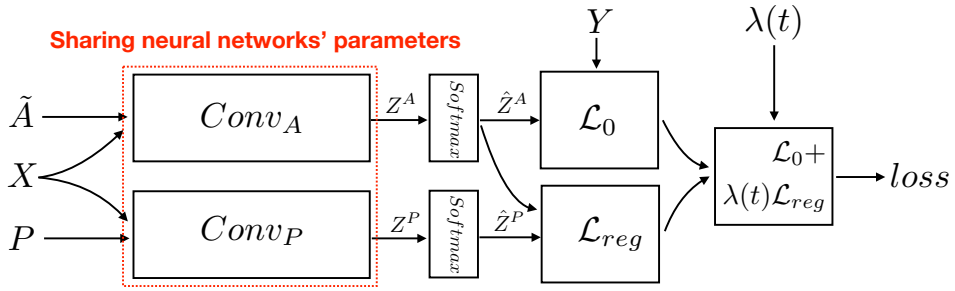



**Figure 2: Architecture of our method. Inputs are: $X \in \mathbb{R}^{n\times k}$, masked $Y \in \mathbb{R}^{n\times c}$, $\tilde{A} \in \mathbb{R}^{n\times n}$, $P \in \mathbb{R}^{n\times n}$, and time $t$.**

of theirs. Namely, by using multiple neural networks, different prior knowledge can be embedded during the data transformation stage.

**The final model.** Algorithm 2 describes the training process of our dual graph convolutional networks method. The loss function is defined as a weighted sum of $\mathcal{L}_0(Conv_A)$ and $\mathcal{L}_{reg}(Conv_A, Conv_P)$. A dynamic weight function is devised to implement the idea described above. That is, at the beginning of the training process (i.e., small $t$), the loss function is mainly dominated by the supervised entry $\mathcal{L}_0$. After obtaining a posterior distribution over the labels using $Conv_A$, increasing $\lambda(t)$ forces our model to simultaneously consider the knowledge encoded in $Conv_P$.

As our method consists of two simple feed-forward neural networks, conventional parameter update strategies can be applied. Instead of Stochastic Gradient Decent (SGD), which updates the parameters for each training example, our implementation uses Batch Gradient Decent (BGD), in which the full training dataset is used for every training iteration. Although BGD is relatively slow, it is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces. In the case of very large training datasets that cannot be fully loaded into memory, SGD or Minibatch Gradient Decent are good memory-efficient extensions. A recent survey [22] discusses the different gradient decent methods in detail.

**Algorithm 2** Dual Graph Convolutional Networks

**Require:**
  $X \in \mathbb{R}^{n\times k}$: the data feature matrix
  $Y \in \mathbb{R}^{n\times c}$: the masked ground truth
  $\mathcal{Y}_L$: the indices of training data for masking
  $\tilde{A}$ and $P \in \mathbb{R}^{n\times n}$: the two diffusion matrices
  $\lambda(t)$: a dynamic weight function
  $h$: the number of hidden convolution layers
**Return:**
  The trained model, i.e., the parameters $W^{(1)}$, ..., $W^{(h)}$.
Randomly initialize $W^{(1)}$, ..., $W^{(h)}$ ▷ to construct the networks
**for** $t$ in range $[0, num\_of\_epochs]$ **do**
  $\hat{Z}^A \leftarrow softmax(Conv_A(X))$ ▷ Eq. (7)
  $\hat{Z}^P \leftarrow softmax(Conv_P(X))$ ▷ Eq. (10)
  $loss \leftarrow \mathcal{L}_0(\hat{Z}^A) + \lambda(t)\mathcal{L}_{reg}(\hat{Z}^A, \hat{Z}^P)$ ▷ Eqs. (11), (12)
  Updating $W$ using $\frac{\partial loss}{\partial W^{(1)}}$, ..., $\frac{\partial loss}{\partial W^{(h)}}$
  **if** convergence **then**
    break loop
  **end if**
**end for**

## 4 EXPERIMENTS

In this section, we present the results from several experiments to verify the performance of our method in graph-based semi-supervised learning tasks. We first introduce the five datasets used in the experiments, and then list the comparative methods and their implementation details. Finally, we present the experimental results and discuss the advantages and limitations of our method. Furthermore, in Appendix 1, we present and discuss some variants of our method for reference.

### 4.1 Datasets

For comparison, we use the same datasets employed in previous studies [15] [30]. Specifically, there are three citation network datasets (i.e., Citeseer, Cora, and Pubmed) and one knowledge graph dataset (i.e., NELL). In addition, we constructed a simplified NELL dataset for further verification. Table 1 presents an overview of the five datasets; detailed descriptions are given below.

**Table 1: Overview of the Five Datasets**

| Dataset | #Nodes | #Features | #Edges | #Classes |
|---|---|---|---|---|
| Citeseer | 3,327 | 3,703 | 4,732 | 6 |
| Cora | 2,708 | 1,433 | 5,429 | 7 |
| Pubmed | 19,717 | 500 | 44,338 | 3 |
| NELL | 65,755 | 61,278 | 266,144 | 210 |
| Simplified NELL | 9,891 | 5,414 | 13,142 | 210 |

**Citeseer.** The Citeseer dataset contains 3,327 scientific publications classified into one of six classes. The citation network consists of 4,732 links. Each publication in this dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from a dictionary consisting of 3,703 unique words. Only 3.6% of the nodes are labeled for training.

**Cora.** Similar to the Citeseer dataset, Cora contains 2,708 scientific publications classified into one of seven classes. The citation

**Table 2: Values of the Hyper-Parameters in our DGCN**

| Dataset | $W^{(1)}$ | $W^{(2)}$ | Dropout rate | $w$ | $\eta$ |
|---|---|---|---|---|---|
| Citeseer, Cora, Pubmed | 32 | 6, 7, 3 | 10% | 2 | 0.05 |
| NELL | 64 | 210 | 10% | 2 | 0.002 |
| Simplified NELL | 96 | 210 | 30% | 2 | 0.001 |

network consists of 5,429 links. Each node is described by a 1,433-dimensional 0/1-valued vector. Only 5.2% of the nodes are labeled for training.

**Pubmed.** The Pubmed dataset contains 19,717 scientific publications classified into one of three classes. The citation network consists of 44,338 links. Each publication is described by a Term Frequency–Inverse Document Frequency (TF-IDF) vector drawn from a dictionary with 500 terms. Only 0.3% of the nodes are labeled for training.

**NELL.** The NELL dataset is extracted from the Never Ending Language Learning (NELL) knowledge graph [6]. By linking the selected NELL entities (9,891 in total) with text descriptions in ClueWeb09 [7], each relation in NELL is described as a triplet $(e_h, r, e_t)$. $e_h$ and $e_t$ are the head and tail entity vectors, respectively, and $r$ indicates the relation between them. By splitting each $(e_h, r, e_t)$ into two edges $(e_h, r_1)$ and $(r_2, e_t)$, we obtain a graph of 65,755 nodes (i.e., the total number of both entity and relation nodes) and 266,144 edges. By assigning a different one-hot vector to each relation node, the length of each feature vector is $5,414+55,864 = 61,278$. Only a single data point per class is labeled for training.

**Simplified NELL.** In simplified NELL, the relation information (i.e., $r$) has been removed and edges among entities have been directly added. By counting the co-occurrences of each $(e_h, e_t)$ pair in all triplets, a weighted adjacency matrix $A$ is constructed. After removing edges with small weights, we obtain a graph of 9,891 nodes (i.e., the number of all entities) and 13,142 edges. The simplified NELL dataset is intended to further verify that our dual convolutional networks-based method is more robust than the baselines (see Section 4.3).

## 4.2 Methods for Comparison

We require several state-of-the-art baselines for comparison. As most of the baselines have several hyper-parameters requiring fine-grained tuning, we selected baselines with public source code. As different baselines use different strategies to embed the graph knowledge, we ensured our baseline set had sufficient diversity. As a result, the following methods were selected.

- **DGCN.** This is the proposed method, as described in Algorithm 2. In our Dual Graph Convolutional Networks (DGCN) implementation, both $Conv_A$ and $Conv_P$ have two hidden layers. Namely, there are two separate $W$ vectors, $W^{(1)}$ and $W^{(2)}$, that need training in Algorithm 2. Table 2 presents detailed information about the implementation of our method for each dataset, including (1) size of the hidden layer; (2) layer-wise dropout rate; (3) window size $w$ in Algorithm 1; and (4) learning rate $\eta$. A detailed discussion of the temporal regularization weight $\lambda(t)$ in Algorithm 2 is given in Section 4.4. Our source code is available[2] for reference.
- **GCN.** The Graph Convolutional Networks (GCN) method [15] is a state-of-the-art technique that has obtained the highest performance among the baselines. It is derived from the related work of conducting graph convolutions in the spectral domain (i.e., Eq. (6)) [9] [12]. Our DGCN method is inspired by GCN. Obviously, DGCN would collapse into GCN if we set $\lambda(t) = 0$. The source code for GCN is publicly available[3].
- **PLANETOID.** Inspired by the Skipgram model [18] from NLP, PLANETOID [30] embeds the graph information using positive and negative samplings. During sampling, both the label information and graph structure are considered. As PLANETOID can be conducted in inductive and transductive manners, we report the better results. The source code for PLANETOID is publicly available[4].
- **DeepWalk.** By taking random walks on a graph, different paths are generated. By regarding the paths as "sentences," DeepWalk [21] generalizes language modeling techniques from sequences of words to paths in a graph. As our method also uses a random walk to calculate the PPMI matrix, this method represents an important comparison. The source code for DeepWalk source code is publicly available[5].

In addition to the state-of-the-art baselines described above, we compared some variants of our proposed method. For brevity, these self-comparisons are given in Appendix A.

**Table 3: Validation accuracies of all Methods: DGCN, GCN, PLANETOID, and DeepWalk**

| Method | Citeseer | Cora | Pubmed | NELL |
|---|---|---|---|---|
| DeepWalk [21] | 43.2% | 67.2% | 65.3% | 58.1% |
| PLANETOID [30] | 64.7% | 75.7% | 77.2% | 61.9% |
| GCN [15] | 70.3% | 81.5% | 79.0% | 66.0% |
| DGCN (our method) | **72.6%** | **83.5%** | **80.0%** | **74.2%** |

## 4.3 Results

Similar to the studies describing the baselines [15] [21] [30], our comparison uses the classification accuracy metric for quantitative evaluation. Table 3 summarizes the experimental results over the three citation network datasets (Citeseer, Cora, and Pubmed) and the knowledge graph dataset (NELL). Using the public source code of the comparative methods, we directly executed their programs and obtained the classification results reported in the table.

On the basis of Table 3, the results are encouraging. That is, over all four datasets, our DGCN method outperformed all of the baselines. Specifically, the random walk based method (i.e., DeepWalk) did not perform well on graphs with a relatively low average degree. For example, on the Citeseer dataset (average degree of 2.84), DeepWalk achieved low classification accuracy. Comparing

[2] https://github.com/ZhuangCY/Coding-NN
[3] https://github.com/tkipf/gcn
[4] https://github.com/kimiyoung/planetoid
[5] https://github.com/phanein/deepwalk

**Table 4: Comparison of DGCN and GCN on Simplified NELL**

| % Labeled | 1% | 5% | 10% | 15% | 20% | 25% |
|---|---|---|---|---|---|---|
| DGCN | 26.0% | 62.6% | 70.4% | 70.6% | 72.0% | 72.8% |
| GCN | 20.4% | 53.0% | 63.4% | 64.6% | 68.6% | 69.8% |
| Margin | +5.6% | +9.6% | +7.0% | +6.0% | +3.4% | +3.0% |

PLANETOID and GCN, although they use a similar strategy of jointly considering both the graph structure and label information for knowledge embeddings, GCN produced better performance over all the datasets. One likely explanation is that, because of its sampling strategy, PLANETOID cannot fully embed all of the knowledge. For example, as the NELL dataset has 210 classes, the label-related knowledge derived by sampling would be sparse. As a result, PLANETOID achieved low accuracy on the NELL dataset.

In addition to the graph structure (i.e., local consistency) and label information, DGCN further considers the semantic context (i.e., global consistency) for each graph node. Compared with GCN, a PPMI matrix-based graph convolution improves the classification performance. By introducing an unsupervised loss function (Eq. (12)), two convolutional classifiers are combined in an ensemble manner. As a result, the model is more robust. The experimental results verify our claim, especially on the highly sparse NELL dataset, where our method outperformed the other baselines by a margin of 8.2%.

To further verify the robustness of our method, Table 4 compares DGCN and GCN on the new simplified NELL dataset. As discussed in Section 4.1, the relation information has been removed in the simplified NELL dataset. Furthermore, by directly adding edges among entities, there are many more noisy edges. Thus, the performance with this dataset will verify which method is more resistant to noise and sparsity.

In Table 4, the classification accuracies are presented with respect to different percentages of data points that were labeled for training. For a labeling rate of 1%, more than half of the 210 classes have no training data, and both DGCN and GCN obtained low accuracies. As the labeling percentage increased, the accuracy margin between GCN and DGCN became smaller. Namely, when there are not enough training data, DGCN performed much better than GCN. In other words, by introducing the PPMI matrix-based convolutional network and the unsupervised ensemble regularizer, our method is more robust in difficult situations, such as few training data, noise, and high sparsity.

## 4.4 Effect of Regularization Weight $\lambda(t)$

In line 14 of Algorithm 2, DGCN uses a temporal weight function $\lambda(t)$ to balance the trade-off between the supervised and unsupervised loss functions. In our implementations, we devised several different weight functions. Through a series of comparisons, we attempted to (1) identify the best classification performance and (2) verify the idea introduced in Section 3.4.

Figure 3 shows the shapes of the different weight functions. The X-axis represents the function variable $t$ and the Y-axis represents the function value. During training, $t$ is defined as the number of epochs. The maximum function value is fixed to be proportional

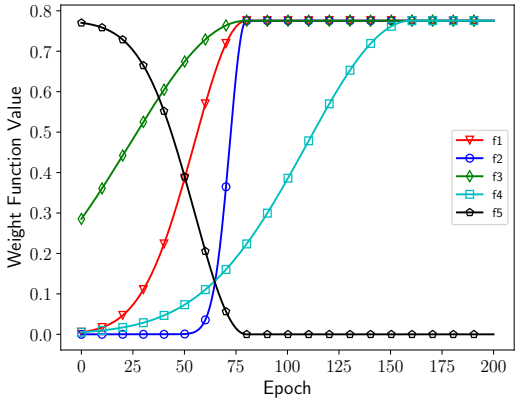



**Figure 3: Five different implementations of the weight function $\lambda(t)$ in Algorithm 2.**

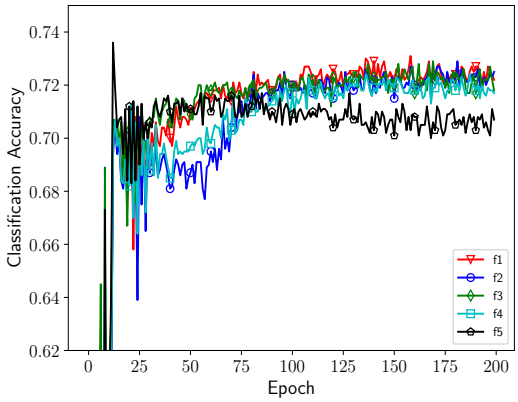



**Figure 4: Classification accuracy using different weight functions for training. Dataset: Citeseer.**

to the training rate. Functions $f1$, $f2$, and $f3$ have different incremental gradients, but reach the maximum value at the same epoch. Function $f4$ reaches the maximum value later than $f1$, $f2$, and $f3$. Unlike the other functions, the value of $f5$ decreases as the number of epochs increases.

Figure 4 shows the classification accuracy using the different weight functions as a function of the training epoch. From the similar performance obtained by $f1$, $f2$, and $f3$, generally speaking, our temporal ensemble method is stable enough to cope with different definitions of the weight. From the performance of all five functions, the classification accuracy clearly has a positive correlation with the function value. Namely, our regularizer embeds additional knowledge learned from the PPMI-based convolution $Conv_P$.

## 4.5 Effect of a Shifted PPMI Matrix $P$

A further evaluation was conducted to examine whether the performance of our method could be further improved by a shifted PPMI matrix $P$ and to verify that our ensemble method can effectively embed the knowledge from $P$.

**Table 5: Classification Accuracy Using Different Shifted PPMI Matrixes $P$ in DGCN Method**

| | Citeseer | Cora | Pubmed | NELL |
|---|---|---|---|---|
| $k = 1.0$ | 72.6% | 83.5% | 80.0% | 74.2% |
| $k = 2.0$ | 72.8% | 83.6% | 80.1% | 74.4% |
| $k = 5.0$ | 72.9% | 83.4% | 79.9% | **75.2%** |
| $k = 10.0$ | **73.2%** | 83.3% | 79.8% | 74.5% |
| $k = 100.0$ | 72.3% | 83.2% | 79.7% | 73.7% |

$$\forall P_{i,j} \in P, \quad P_{i,j} = max\{P_{i,j} - log(k), 0\}. \tag{13}$$

Eq. (13) presents the calculation of a shifted PPMI matrix, first introduced in [16] for word embedding. On the basis of the derivation in [16], the value of $k$ indicates the number of negative samplings required to calculate each entry of $P$. In research on semi-supervised learning, we are the first to verify whether such a shift can also be applied to understand a graph. Interesting results are observed in our experiments.

Table 5 presents the classification accuracy on all datasets for different values of $k$. When $k = 1.0$, no shifts are conducted on $P$. An interesting observation is that, compared with the Cora and Pubmed datasets, the shifted PPMI matrix allows our method to obtain significant improvements on the Citeseer and NELL datasets. As the classification performance on these datasets is relatively low (around 70% accuracy), there is a high probability that their given graphs have more noise than those associated with the Cora and Pubmed datasets (around 80% accuracy). By shifting the PPMI matrices of the Citeseer and NELL graphs (namely, conducting more negative samplings), the noise is reduced to a certain degree.

This observation proves that our ensemble method embeds useful knowledge from $Conv_P$. In related work, the adjacency matrix [9] [15], positive & negative sampling [30], and random walks [21] [30] are utilized to understand a graph. In our work, we have verified that the PPMI matrix can further be used to encode the global consistency of a graph.

## 5 CONCLUSIONS

In this paper, we have proposed a Dual Graph Convolutional Network method for graph-based semi-supervised learning. In addition to the often-considered local consistency, our method uses the PPMI to encode the global consistency. To jointly consider both the local and global consistencies, a dual neural network structure has been devised for the ensemble. Experiments on a variety of public datasets illustrate the effectiveness of our method for solving classification tasks. In the appendix, some variants of DGCN are derived to provide a deeper insight into our idea.

Our work provides a solution for combining the prior knowledge learned from different views of raw data. As a reasonable extension, in future work, we will seek more ways of understanding a graph. In other research, we will also investigate whether our method could be applied in research fields such as domain adaptation learning. The source and target domains' knowledge could be jointly embedded in our dual convolutional networks for adaptation.

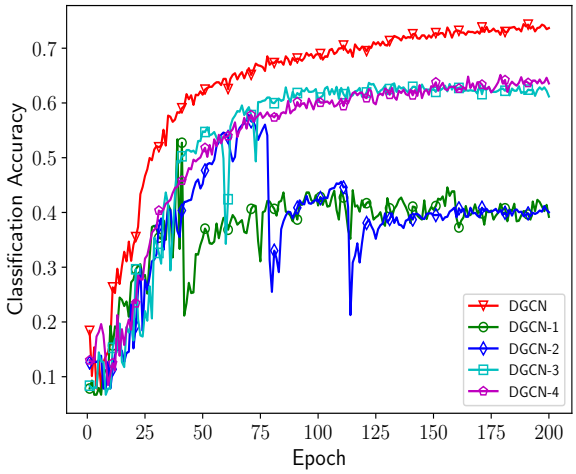



**Figure 5: Training processes of the different variants of our method. Dataset: NELL.**

## A VARIANTS OF OUR METHOD

In this appendix, some variants of our method are derived for further comparison. The methods are as follows.

- **DGCN:** The original method used in our experiments.
- **DGCN-1:** When the neural network parameters (i.e., $W^{(h)}$s) are not shared between $Conv_A$ and $Conv_P$.
- **DGCN-2:** Instead of Eq. (12), we use concatenation to form the ensemble. Namely, the final predictions are derived from the latent representations: $Conv_A \oplus Conv_P$. Before $softmax$, a dense layer is added to ensure the same shape as the label matrix $Y \in \mathbb{R}^{n \times c}$.
- **DGCN-3:** When the parameters are not shared between $Conv_A$ and $Conv_P$, and concatenation is used.
- **DGCN-4:** Without ensemble, when only $Conv_P$ is used.

Figure 5 shows the training processes of all the methods on the NELL dataset. For DGCN-1 and DGCN-2, there is a significant decrease in accuracy during training. Such a decline is very likely to have been caused by the conflicts between $Conv_A$ and $Conv_P$. In other words, both the sharing parameters and unsupervised regularizer are necessary for the ensemble when the number of training data is limited. As the parameters are not shared, DGCN-3 can be regarded as a linear combination of GCN [15] and $Conv_P$. DGCN-3 obtained similar performance to GCN. Accordingly, this observation has verified that jointly considering both $Conv_A$ and $Conv_P$ during training (i.e., DGCN) can significantly improve the linear DGCN-3. By only using $Conv_P$, DGCN-4 obtained a bit of better performance than DGCN-3.

Regarding the number of hidden layers in $Conv_*$, as the effect of this parameter has been investigated elsewhere in detail [15], we omit this discussion here.

## ACKNOWLEDGMENTS

This work is partly supported by JSPS KAKENHI (16K12532, 15J01402) and MIC SCOPE (172307001).

## REFERENCES


[1] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *the 47th Annual IEEE Symposium on Foundations of Computer Science*. 475–486.

[2] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1993–2001.

[3] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research* 7, Nov (2006), 2399–2434.

[4] John A Bullinaria and Joseph P Levy. 2007. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods* 39, 3 (2007), 510–526.

[5] Deng Cai, Xiaofei He, Jiawei Han, and Thomas S Huang. 2011. Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 8 (2011), 1548–1560.

[6] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. 2010. Toward an Architecture for Never-Ending Language Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, Vol. 5.

[7] Bhavana Dalvi, Aditya Mishra, and William W Cohen. 2016. Hierarchical semi-supervised classification with incomplete class hierarchies. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. 193–202.

[8] Maximilien Danisch, T-H Hubert Chan, and Mauro Sozio. 2017. Large Scale Density-friendly Graph Decomposition via Convex Programming. In *Proceedings of the 26th International Conference on World Wide Web*. 233–242.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.

[10] François Fouss, Kevin Francoisse, Luh Yen, Alain Pirotte, and Marco Saerens. 2012. An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification. *Neural networks* 31 (2012), 53–72.

[11] Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on knowledge and data engineering* 19, 3 (2007), 355–369.

[12] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis* 30, 2 (2011), 129–150.

[13] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[14] Ming Ji, Yizhou Sun, Marina Danilevsky, Jiawei Han, and Jing Gao. 2010. Graph regularized transductive classification on heterogeneous information networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 570–586.

[15] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*. 1–14.

[16] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*. 2177–2185.

[17] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[19] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A review of relational machine learning for knowledge graphs. *Proc. IEEE* 104, 1 (2016), 11–33.

[20] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[21] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[22] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[23] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. 2016. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems*. 1163–1171.

[24] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* 30, 3 (2013), 83–98.

[25] Milivoj Simeonovski, Giancarlo Pellegrino, Christian Rossow, and Michael Backes. 2017. Who Controls the Internet?: Analyzing Global Threats using Property Graph Traversals. In *Proceedings of the 26th International Conference on World Wide Web*. 647–656.

[26] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research* 15, 1 (2014), 1929–1958.

[27] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. 1067–1077.

[28] Peter D Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research* 37 (2010), 141–188.

[29] Derry Wijaya, Partha Pratim Talukdar, and Tom Mitchell. 2013. Pidgin: ontology alignment using web text as interlingua. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 589–598.

[30] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on Machine Learning*. 1–9.

[31] Wayne W Zachary. 1977. An information flow model for conflict and fission in small groups. *Journal of anthropological research* 33, 4 (1977), 452–473.

[32] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in neural information processing systems*. 321–328.

[33] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning*. 912–919.