Graph Neural Networks

Mohammed Abdul Qaathir
Koppolu Anudeep
Palakshigari Sasidhar Manjunath

Abstract

Deep Learning algorithms directly work on the data with only the features represented in eucledian space, but there are lots of cases where data is generated from non eucledian space and they are represented as graphs with relationships and interdependency between objects(nodes). There are many algorithms proposed for handling such data.

Algorithms Used

We have used following algorithms:

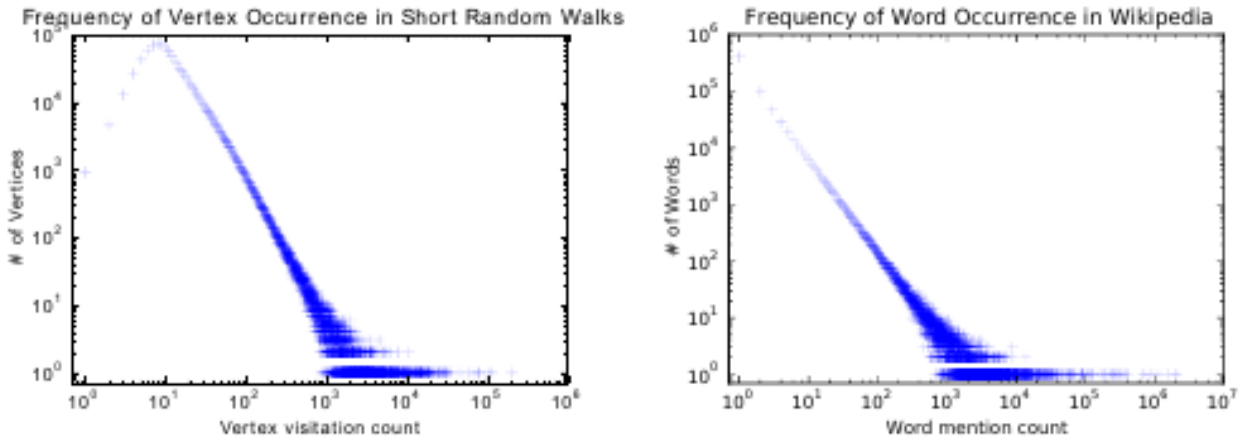
***DeepWalk**

***Graph Convolutional Networks**

Deepwalk

Here the model treats each node as a word, and makes random walks in the graphs to make sentences.

From the generated sentences the model then applies word2vec algorithms to find the vector representation of the nodes.



(a) YouTube Social Graph (b) Wikipedia Article Text

Figure 2: The distribution of vertices appearing in short random walks (2a) follows a power-law, much like the distribution of words in natural language (2b).

Deepwalk Algorithm

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

for each $v_j \in \mathcal{W}_{v_i}$ **do**
  for each $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do**
    $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
    $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
  end for
end for

Graph Convolutional NN

Spectral based graph convolutional algorithms take O(N^3) time complexity,so with first order approximation of the ChebNet (which defines a filter as Chebyshev polynomials of the diagonal matrix of eigenvalues), the convolution operation becomes

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$$

Multilayer graph convolution

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) .$$

Here $H^{(l)}$ is the convoluted output of the current layer, $H^{(0)}$ is the input feature matrix, W is the learnable parameter of the model.

Semi supervised learning using GCN

We have used the GCN algorithm for semi supervised learning. Where there will be some nodes labelled and other nodes unlabelled, we have to predict the label for the unlabelled nodes,

The loss function of the model uses only the nodes with labels into consideration and backpropagate the errors to learn W.

Combined Model

Word representations can be thought of as features for the word, where each element of the vector can be thought of as some feature.

So the output vector we got from DeepWalk represent the structural features of each node. We added these features also to the node's feature and made a new feature matrix, and used that feature matrix for training the GCN.

Reasoning

In this figure, A and B are connected To a subgraph which is highly Connected then there will be lots of Similar random walks starting from A and B, if a real time data depends On such structures of graph then these can be modelled reasonably using Deepwalk but deepwalk works only on the graph not on the features, so using Deepwalk along with GCN can help the model to work with such structures and the features of the nodes.

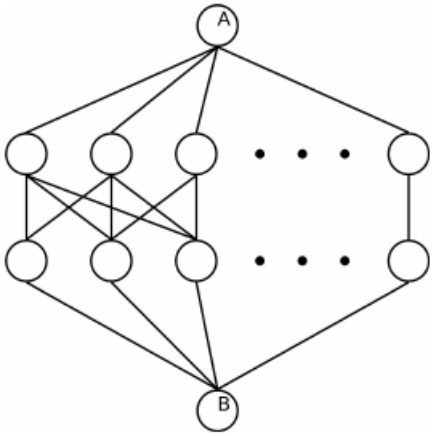


Figure 2.

Results

Dataset	Accuracy
Cora	80.12%
Citeseer	67.89

The accuracy is comparable to the actual accuracy we get using just the GCN, for citeseer the accuracy actually dropped by 3%, one main reason for the accuracy drop is the actual feature matrix for the data is in binary, but the output given by DeepWalk is real valued feature so integrating these features was difficult.

Model Hyper Parameters

We used a hyperparameter C which is a real number. We used this parameter to change the real valued DeepWalk output to binary output, that is an elements value is less than C then we made that to 0 else its 1. this parameter was crucial for the the accuracy varies from 65% to 80% by changing the value of C. Another Hyperparameter is H the representation size of the DeepWalk output. When testing the model we ran DeepWalk for 20 iterations due to time constraints, which is very less had we ran for more iterations accuracy might have increased.

About Dataset

CORA: The Cora dataset consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

CITESEER: The CiteSeer dataset consists of 3312 scientific publications classified into one of six classes. The citation network consists of 4732 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 3703 unique words.

Other models surveyed

* **GraphSage**

* **Large-scale Graph Convolution Networks (LGCN)**

Graph Sage

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

LGCN

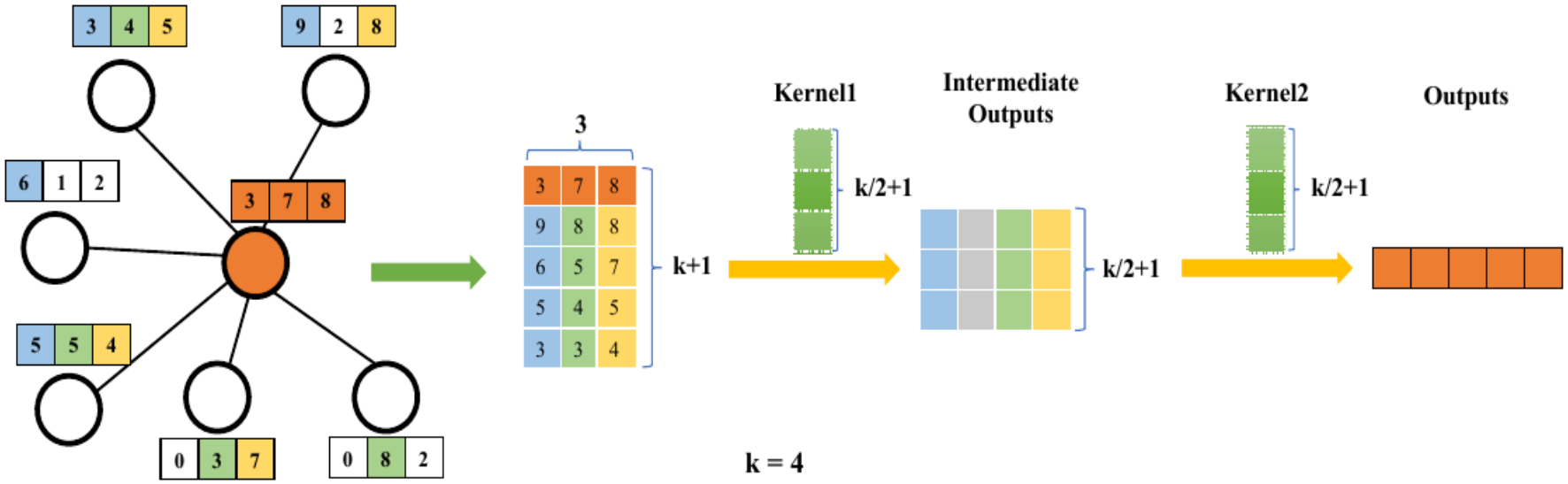


Figure 2: An illustration of a learnable graph convolutional layer (LGCL). We consider a node with 6 adjacent nodes. Each node has three features, represented by a 3-component feature vector. This layer selects $k = 4$ nodes in the neighborhood and employs a 1-D CNN to produce a new vector representation of five features for the central node, color-coded in orange. The left part describes the process of selecting the k-largest values for each feature from neighboring nodes. It can be seen from the graph that there are 6 neighbors. Since $k = 4$, for each feature, four largest values are selected from the neighborhood based on the ranking. For example, the results of this selection process for the first feature is {9, 6, 5, 3} out of {9, 6, 5, 3, 0, 0}. By repeating the same process for the other two features, we obtain $(k + 1)$ 3-component feature vectors, including that of the orange node itself. Concatenating them gives a 1-D data of grid-like structure, which has $(k + 1)$ positions and 3 channels. Afterwards, a 1-D CNN is applied to generate the final feature vector. Specifically, we use two convolutional layers with a kernel size of $(k/2 + 1)$ and without padding. The numbers of output channels are 4 and 5, respectively. In practice, the 1-D CNN can be any CNN model, as long as the final output is a vector, serving as the new feature representation of the central node.

Thank You