Collective Vertex Classification Using Recursive Neural Network

Qiongkai Xu
The Australian National University
Data61 CSIRO
Xu.Qiongkai@data61.csiro.au

Qing Wang
The Australian National University
qing.wang@anu.edu.au

Chenchen Xu
The Australian National University
Data61 CSIRO
Xu.Chenchen@data61.csiro.au

Lizhen Qu
The Australian National University
Data61 CSIRO
Qu.Lizhen@data61.csiro.au

Abstract

Collective classification of vertices is a task of assigning categories to each vertex in a graph based on both vertex attributes and link structure. Nevertheless, some existing approaches do not use the features of neighbouring vertices properly, due to the noise introduced by these features. In this paper, we propose a graph-based recursive neural network framework for collective vertex classification. In this framework, we generate hidden representations from both attributes of vertices and representations of neighbouring vertices via recursive neural networks. Under this framework, we explore two types of recursive neural units, naive recursive neural unit and long short-term memory unit. We have conducted experiments on four real-world network datasets. The experimental results show that our framework with long short-term memory model achieves better results and outperforms several competitive baseline methods.

Introduction

In everyday life, graphs are ubiquitous, e.g. social networks, sensor networks, and citation networks. Mining useful knowledge from graphs and studying properties of various kinds of graphs have been gaining popularity in recent years. Many studies formulate their graph problems as predictive tasks such as vertex classification (London and Getoor 2014), link prediction (Tang et al. 2015), and graph classification (Niepert, Ahmed, and Kutzkov 2016).

In this paper, we focus on vertex classification task which studies the properties of vertices by categorising them. Algorithms for classifying vertices are widely adopted in web page analysis, citation analysis and social network analysis (London and Getoor 2014). Naive approaches for vertex classification use traditional machine learning techniques to classify a vertex only based on the attributes or features provided by this vertex, e.g. such attributes can be words of web pages or user profiles in a social network. Another series of approaches are *collective vertex classification*, where instances are classified simultaneously as opposed to independently. Based on the observation that information of neighbouring vertices may help classifying current vertex, some approaches incorporate attributes of neighbouring vertices into classification process, which however introduce noise at the same time and result in reduced performance (Chakrabarti, Dom, and Indyk 1998; Myaeng and Lee 2000). Other approaches incorporate the labels of its neighbours. For instance, the Iterative Classification Approach (ICA) integrates the label distribution of neighbouring vertices to assist classification (Lu and Getoor 2003) and the Label Propagation approach (LP) fine-tunes predictions of the vertex using the labels of its neighbouring vertices (Wang and Zhang 2008). However, labels of neighbouring vertices are not representative enough for learning sophisticated relationships of vertices, while using their attributes directly would involve noise. We thus need an approach that is capable of capturing information from neighbouring vertices, while in the mean time reducing noise of attributes. Utilizing representations learned from neural networks instead of neighbouring attributes or labels is one of the possible approaches (Schmidhuber 2015). As graphs normally provide rich structural information, we exploit the neural networks with sufficient complexity for capturing such structures.

Recurrent neural networks were developed to utilize sequence structures by processing input in order, in which the representation of the previous node is used to generate the representation of the current node. Recursive neural networks exploit representation learning on tree structures. Both of the approaches achieve success in learning representation of data with implicit structures which indicate the order of processing vertices (Tang, Qin, and Liu 2015; Tai, Socher, and Manning 2015). Following these work, we explore the possibility of integrating graph structures into recursive neural network. However, graph structures, especially cyclic graphs, do not provide such an processing order. In this paper, we propose a graph-based recursive neural network (GRNN), which allows us to transform graph structures to tree structures and use recursive neural networks to learn representations for the vertices to classify. This framework consists of two main components, as illustrated in Figure 1:

1. **Tree construction:** For each vertex to classify v_t, we generate a search tree rooted at v_t. Starting from v_t, we add its neighbouring vertices into the tree layer by layer.

2. **Recursive neural network construction:** We build a recursive neural network for the constructed tree, by aug-

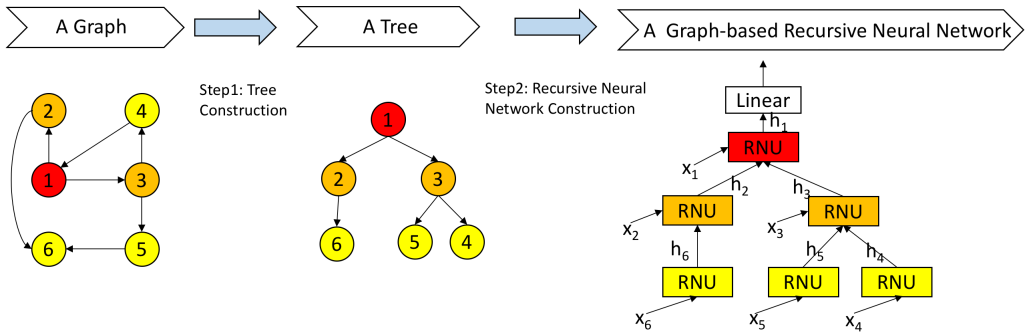

Figure 1: Main components of the Graph-based Recursive Neural Network framework (GRNN): 1) constructing a tree from the vertex to classify (vertex 1); 2) building a recursive neural network on the tree.

menting each vertex with one recursive neural unit. The inputs of each vertex are its features and hidden states of its child vertices. The output of a vertex is its hidden states.

Our main contributions are: (1) We introduce recursive neural networks to solve the collective vertex classification problem. (2) We propose a method that can transfer vertices for classification to a locally constructed tree. Based on the tree, recursive neural network can extract representations for target vertices. (3) Our experimental results show that the proposed approach outperforms several baseline methods. Particularly, we demonstrate that including information from neighbouring vertices can improve performance of classification.

Related Work

There has been a growing trend to represent data using graphs (Angles and Gutierrez 2008). Discovering knowledge from graphs becomes an exciting research area, such as vertex classification (London and Getoor 2014) and graph classification (Niepert, Ahmed, and Kutzkov 2016). Graph classification analyzes the properties of the graph as a whole, while vertex classification focuses on predicting labels of vertices in the graph. In this paper, we discuss the problem of vertex classification. The main-stream approaches for vertex classification are collective vertex classification (Lu and Getoor 2003) which classify vertices using information provided by neighbouring vertices. Iterative classification approach (Lu and Getoor 2003) models neighbours' label distribution as link features to facilitate classification. Label propagation approach (Wang and Zhang 2008) assigns a probabilistic label for each vertex and then fine-tunes the probability using graph structure. However, labels of neighbouring vertices are not representative enough to include all useful information. Some researchers tried to introduce attributes from neighbouring vertices to improve classification performance. Nevertheless, as reported in (Chakrabarti, Dom, and Indyk 1998; Myaeng and Lee 2000), naively incorporating these features may reduce the performance of classification, when original features of neighbouring vertices are too noisy.

Recently, some researchers analysed graphs using deep neural network technologies. Deepwalk (Perozzi, Al-Rfou, and Skiena 2014) is an unsupervised learning algorithm to learn vertex embeddings using link structure, while content of each vertex is not considered. Convolutional neural network for graphs (Niepert, Ahmed, and Kutzkov 2016) learns feature representations for the graphs as a whole. Recurrent neural collective classification (Monner and Reggia 2013) encodes neighbouring vertices via a recurrent neural network, which is hard to capture the information from vertices that are more than several steps away.

Recursive neural networks (RNN) are a series of models that deal with tree-structured information. RNN has been implemented in natural scenes parsing (Socher et al. 2011) and tree-structured sentence representation learning (Tai, Socher, and Manning 2015). Under this framework, representations can be learned from both input features and representations of child nodes. Graph structures are more widely used and more complicated than tree or sequence structures. Due to the lack of notable order for processing vertices in a graph, few studies have investigated the vertex classification problem using recursive neural network techniques. The graph-based recursive neural network framework proposed in this paper can generate the processing order for neural network according to the vertex to classify and the local graph structure.

Graph-based Recursive Neural Networks

In this section, we present the framework of *Graph-based Recursive Neural Networks* (GRNN). A graph $G = (V, E)$ consists of a set of vertices $V = \{v_1, v_2, \ldots, v_N\}$ and a set of edges $E \subseteq V \times V$. Graphs may contain cycles, where a cycle is a path from a vertex back to itself. Let $\mathcal{X} = \{x_1, x_2, \cdots, x_N\}$ be a set of feature vectors, where each $x_i \in \mathcal{X}$ is associated with a vertex $v_i \in V$, $\mathcal{L}$ be a set of labels, and $v_t \in V$ be a vertex to be classified, called *target vertex*. Then, the collective vertex classification problem is to predict the label y_t of v_t, such that

$$\hat{y}_t = \arg\max_{y_t \in \mathcal{L}} P_\theta(y_t | v_t, G, \mathcal{X}) \quad (1)$$

using a recursive neural network with parameters θ.

Algorithm 1 Tree Construction Algorithm

Require: Graph G, Target vertex v_t, Tree depth d
Ensure: Tree $T = (V_T, E_T)$

1. Let Q be a queue
2. $V_T = \{v_t\}$, $E_T = \emptyset$
3. Q.push(v_t)
4. **while** Q is not empty **do**
5. v = Q.pop()
6. **if** T.depth(v) $\leq d$ **then**
7. **for** w in G.outgoingVertices(v) **do**
8. // add w to T as child of v
9. $V_T = V_T \cup \{w\}$
10. $E_T = E_T \cup \{(v, w)\}$
11. Q.push(w)
12. **end for**
13. **end if**
14. **end while**
15. **return** T

Tree Construction

In a neural network, neurons are arranged in layers and different layers are processed following a predefined order. For example, recurrent neural networks process inputs in sequential order and recursive neural networks deal with tree-structures in a bottom-up manner. However, graphs, particularly cyclic graphs, do not have an explicit order. How to construct an ordered structure from a graph is challenging.

Given a graph $G = (V, E)$, a target vertex $v_t \in V$ and tree depth $d \in \mathbb{N}$, we can construct a tree $T = (V_T, E_T)$ rooted at v_t using breadth-first-search, where V_T is a vertex set, E_T is an edge set, and $(v, w) \in E_T$ means an edge from parent vertex v to child vertex w. The depth of a vertex v in T is the length of the path from v_t to v, denoted as T.depth(v). The depth of a tree is maximum depth of vertices in T. We use G.outgoingVertices(v) to denote a set of outgoing vertices from v, i.e. $\{w|(v, w) \in G\}$. The tree construction algorithm is described in Algorithm 1. Firstly, a first-in-first-out queue (Q) is initialized with v_t (lines 1-3). The algorithm iteratively check the vertices in Q. If there is a vertex whose depth is less than d, we pop it out from Q (line 6), add all its neighbouring vertices in G as its children in T and push them to the end of Q (lines 9-11).

In general, there are two approaches to deal with cycles in a graph. One is to remove vertices that have been visited, and the other is to keep duplicate vertices. Fig 2.a describes an example of a graph with a cycle between v_1 and v_2. Let us start with the target vertex v_1. In the first approach, there will be no child vertex for v_2, since v_1 is already visited. The corresponding tree is shown in Fig 2.b. In the second approach, we will add v_1 as a child vertex to v_2 iteratively and terminate after certain steps. The corresponding tree is illustrated in Fig 2.c. When generating the representation of a vertex, say v_2, any information from its neighbours may help. We thus include v_1 as a child vertex of v_2. In this paper, we use the second manner for tree construction.

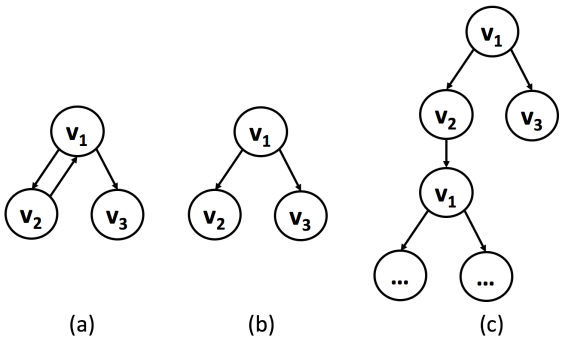

Figure 2: (a) A graph with a cycle; (b) A tree constructed without duplicate vertices; (c) A tree constructed with duplicate vertices.

Recursive Neural Network Construction

Now we construct a recursive neural unit (RNU) for each vertex $v_k \in T$. Each RNU takes a feature vector x_k and hidden states of its child vertices as input. We explore two kinds of recursive neural units which are discussed in (Socher et al. 2011; Tai, Socher, and Manning 2015).

Naive Recursive Neural Unit (NRNU)

Each NRNU for a vertex $v_k \in T$ takes a feature vector x_k and the aggregation of the hidden states from all children of v_k. The transition equations of NRNU are given as follows:

$$\widetilde{h_k} = \max_{v_r \in C(v_k)} \{h_r\} \tag{2}$$

$$h_k = \tanh(W^{(h)} x_k + U^{(h)} \widetilde{h_k} + b^{(h)}) \tag{3}$$

where $C(v_k)$ is the set of child vertices of v_k, $W^{(h)}$ and $U^{(h)}$ are weight matrix, and $b^{(h)}$ is the bias. The generated hidden state h_k of v_k is related to the input vector x_k and aggregated hidden state $\widetilde{h_k}$. Different from summing up all hidden states as in (Tai, Socher, and Manning 2015), we use max pooling for $\widetilde{h_k}$ (see Eq 2). This is because, in real-life situations, the number of neighbours for a vertex can be very large and some of them are irrelevant for the vertex to classify (Zhang et al. 2013). We use G-NRNN to refer to the graph-based naive recursive neural network which incorporates NRNU as recursive neural units.

Long Short-Term Memory Unit (LSTMU)

LSTMU is one variation on RNU, which can handle the long term dependency problem by introducing memory cells and gated units (Hochreiter and Schmidhuber 1997). LSTMU is composed of an input gate i_k, a forget gate f_k, an output gate o_k, a memory cell c_k and a hidden state h_k. The transition equations of LSTMU are given as follows:

$$\widetilde{h_k} = \max_{v_r \in C(v_k)} \{h_r\} \tag{4}$$

$$i_k = \sigma(W^{(i)} x_k + U^{(i)} \widetilde{h_k} + b^{(i)}) \tag{5}$$

$$f_{kr} = \sigma(W^{(f)} x_k + U^{(f)} h_r + b^{(f)}) \tag{6}$$

$$o_k = \sigma(W^{(o)} x_k + U^{(o)} \widetilde{h_k} + b^{(o)}) \tag{7}$$

$$u_k = \tanh(W^{(u)} x_k + U^{(u)} \widetilde{h_k} + b^{(u)}) \tag{8}$$

$$c_k = i_k \odot u_k + \sum_{v_r \in C(v_k)} f_{kr} \odot c_r \tag{9}$$

$$h_k = o_k \odot \tanh(c_k) \tag{10}$$

where $C(v_k)$ is the set of child vertices of v_k, $v_r \in C(v_k)$, x_k is corresponding feature vector of the child vertex v_k, $\odot$ is element-wise multiplication and σ is sigmoid function, $W^{(*)}$ and $U^{(*)}$ are weight matrices, and $b^{(*)}$ are the biases. $\widetilde{h_k}$ is a vector aggregated from the hidden states of the child vertices. We use G-LSTM to refer to the graph-based long short-term memory network which incorporates LSTMU as recursive neural units.

After constructing GRNN, we calculate the hidden states of all vertices in T from leaves to root, then we use a softmax classifier to predict label y_t of the target vertex v_t using its hidden states h_t (see Eq 11).

$$P_\theta(y_t|v_t, G, \mathcal{X}) = softmax(W^{(s)}h_t + b^{(s)}) \tag{11}$$

$$\hat{y_t} = \arg\max_{y_t \in \mathcal{L}} P_\theta(y_t|v_t, G, \mathcal{X}) \tag{12}$$

Cross-entropy $J(\theta) = -\frac{1}{N}\sum_{t=1}^{N} \log P_\theta(y_t|v_t, G, \mathcal{X})$ is used as cost function, where N is the number of vertices in a training set.

Experimental Setup

To verify the effectiveness of our approach, we have conducted experiments on four datasets and compared our approach with three baseline methods. We will describe the datasets, baseline methods, and experimental settings.

Datasets

We have tested our approach on four real-world network datasets.

- **Cora** (McCallum et al. 2000) is a citation network dataset which is composed of 2708 scientific publications and 5429 citations between publications. All publications are classified into seven classes: *Rule Learning (RU)*, *Genetic Algorithms (GE)*, *Reinforcement Learning (RE)*, *Neural Networks (NE)*, *Probabilistic Methods (PR)*, *Case Based (CA)* and *Theory (TH)*.
- **Citeseer** (Giles, Bollacker, and Lawrence 1998) is another citation network dataset which is larger than Cora. Citeseer is composed of 3312 scientific publications and 4723 citations. All publications are classified into six classes: *Agents*, *AI*, *DB*, *IR*, *ML* and *HCI*.
- **WebKB** (Craven et al. 1998) is a website network collected from four computer science departments in different universities which consists of 877 web pages, 1608 hyper-links between web pages. All websites are classified into five classes: *faculty*, *students*, *project*, *course* and *other*.
- **WebKB-sim** is a network dataset which is generated from WebKB based on the cosine similarity between each vertex and its top 3 similar vertices according to their feature vectors (Baeza-Yates, Ribeiro-Neto, and others 1999). We use same feature vectors as the ones in WebKB. This dataset is used to demonstrate the effectiveness of our framework on datasets which may not have explicit relationship represented as edges between vertices, but can be treated as graphs whose edges are based on some metrics such as similarity of vertices.

We use abstracts of publications in Cora and Citeseer, and contents of web pages in WebKB to generate features of vertices. For the above datasets, all words are stemmed first, then stop words and words with document frequency less than 10 are discarded. A dictionary is generated by including all these words. We have 1433, 3793, and 1703 for Cora, Citeseer and WebKB, respectively. Each vertex is represented by a bag-of-words vector where each dimension indicates absence or occurrence of a word in the dictionary of the corresponding dataset[1].

Baseline Methods

We have implemented the following three baseline methods:

- **Logistic Regression (LR)** (Hosmer Jr and Lemeshow 2004) predicts the label of a vertex using its own attributes through a logistic regression model.
- **Iterative classification approach (ICA)** (Lu and Getoor 2003; London and Getoor 2014) utilizes the combination of link structure and vertex features as input of a statistical machine learning model. We use two variants of ICA: **ICA-binary** uses the occurrence of labels of neighbouring vertices, **ICA-count** uses the frequency of labels of neighbouring vertices.
- **Label propagation (LP)** (Wang and Zhang 2008; London and Getoor 2014) uses a statistical machine learning to give a label probability for each vertex, then propagates the label probability to all its neighbours. The propagation steps are repeated until all label probabilities converge.

To make experiments consistent, logistic regression is used for all statistical machine learning components in ICA and LP. We run 5 iterations for each ICA experiment and 20 iterations for each LP experiment[2].

Experimental Settings

In our experiments, we split each dataset into two parts: training set and testing set, with different proportions (70% to 95% for training). For each proportion setting, we randomly generate 5 pairs of training and testing sets. For each experiment on a pair of training and testing sets, we run 10 epochs on the training set and record the highest Micro-F1 score (Baeza-Yates, Ribeiro-Neto, and others 1999) on the testing set. Then we report the averaged results from the experiments with the same proportion setting. According to preliminary experiments, the learning rate is set to 0.1 for LR, ICA and LP and 0.01 for all GRNN models. We empirically set number of hidden states to 200 for all GRNN models. Adagrad (Duchi, Hazan, and Singer 2011) is used as the optimization method in our experiments.

[1]Cora, Citeseer and WebKB can be downloaded from LINQS. We will publish WebKB-sim along with our code.

[2]According to our preliminary experiments, LP converges slower than ICA.

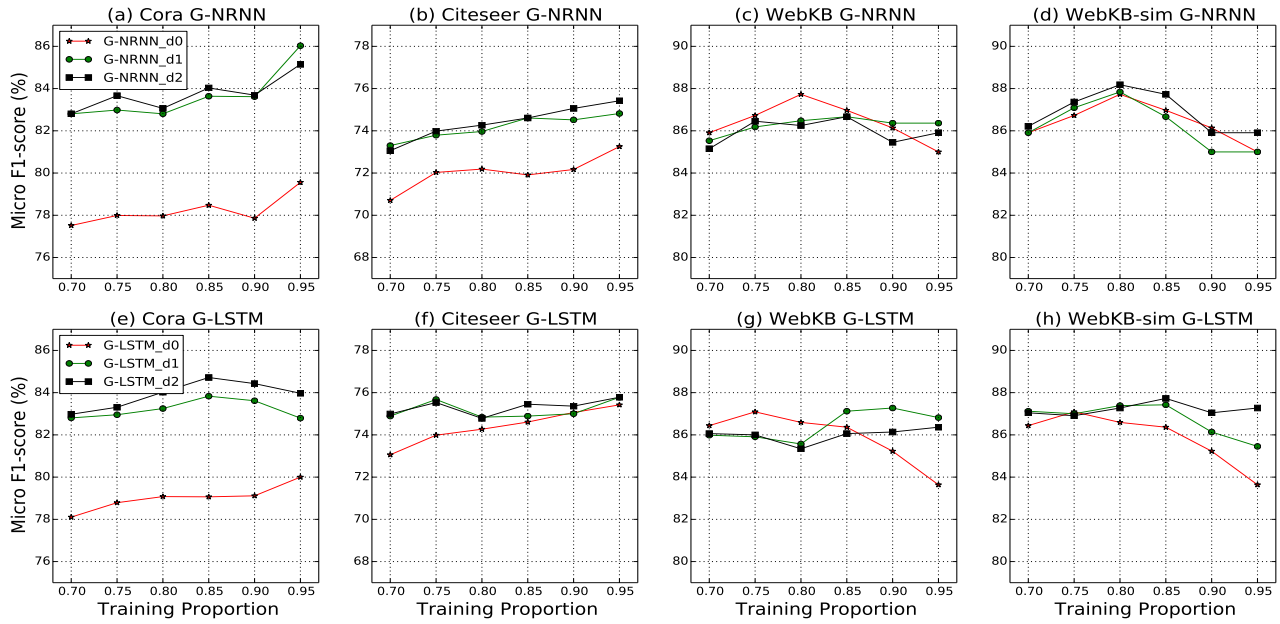

Figure 3: Comparison of G-NRNN and G-LSTM on four datasets: (a) Cora G-NRNN, (b) Citeseer G-NRNN, (c) WebKB G-NRNN, (d) WebKB-sim G-NRNN (e) Cora G-LSTM, (f) Citeseer G-LSTM, (g) WebKB G-LSTM and (h) WebKB-sim G-LSTM.

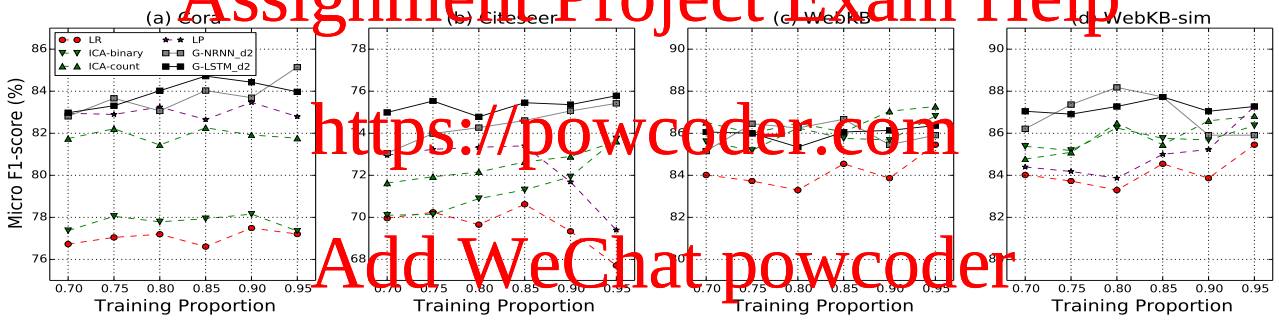

Figure 4: Comparison of G-LSTM with LR, LP and ICA on four datasets: (a) Cora, (b) Citeseer, (c) WebKB and (d) WebKB-sim.

Results and Discussion

Model and Parameter Selection

Figure 3 illustrates the performance of G-NRNN and G-LSTM on four datasets. We use G-NRNN_di and G-LSTM_di to refer to G-NRNN and G-LSTM over trees of depth i, respectively, where $i = 0, 1, 2$.

For the experiments on G-NRNN and G-LSTM over trees of different steps, $d1$ and $d2$ outperform $d0$ in most cases[3]. Particularly, the experiments with $d1$ and $d2$ perform better with more than 2% improvement than $d0$ on Cora, and $d1$ and d_2 enjoy a consistent improvement over $d0$ on Citeseer and WebKB-sim. This performance difference is also obvious in WebKB, when the training proportion is larger than 85%. These mean that introducing neighbouring vertices can improve the performance of classification and more neighbouring information can be obtained by increasing the depth of trees. Using same RNU setting, $d2$ outperforms $d1$ in most experiments on Cora, Citeseer and WebKB-sim. However, for WebKB, $d2$ does not always outperforms $d1$. That is to say, introducing more layers of vertices may help improving the performance, while the choice of the best tree depth depends on applications.

[3]When $d = 0$, each constructed tree contains only one vertex.

Table 1: Micro-F1 score (%) of G-LSTM model with different pooling strategies on Cora, Citeseer, WebKB and WebKB-sim.

Method	Pooling Strategy	Datasets			
		Cora	Citeseer	WebKB	WebKB-sim
G-LSTM_d1	sum	83.05	74.81	86.21	87.42
	mean	**84.18**	74.77	86.21	**87.58**
	max	83.83	**74.89**	**87.12**	87.42
G-LSTM_d2	sum	84.03	75.05	85.61	87.42
	mean	84.47	75.33	**86.21**	87.42
	max	**84.72**	**75.45**	86.06	**87.73**

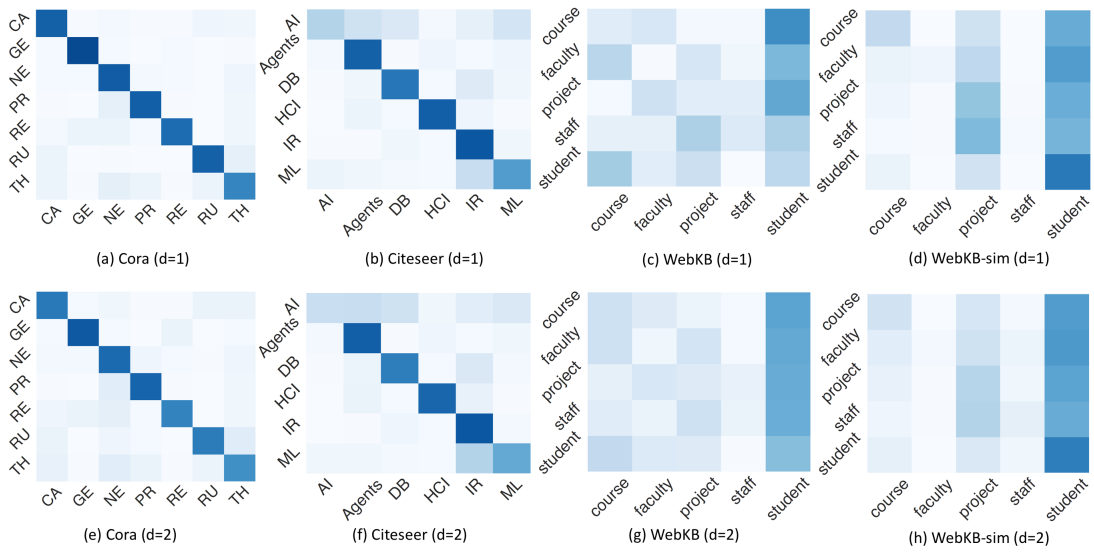

Figure 5: Distribution of label co-occurrence for Cora, Citeseer, WebKB and WebKB-sim, where $d = 1, 2$.

In Table 1, we compare three different pooling strategies used in G-LSTM[4], *sum*, *mean* and *max* pooling. We use 85% for training for all datasets here. In general, *mean* and *max* outperform *sum* which is used in (Tai, Socher, and Manning 2015). This is probably because the number of neighbours for a vertex can be very large and summing them up can make $\widetilde{h}$ large for some extreme cases. *max* slightly outperforms *mean* in our experiments, which is probably due to max pooling can select the most influential information of child vertices which filters out noise to some extend.

Baseline Comparison

We compare our approach with the baseline methods on four network datasets. As our models with d_2 provide better performance, we choose G-NRNN_d2 and G-LSTM_d2 as representative models.

In Figure 4.a and Figure 4.b, we compare our approach with the baseline methods on two citation networks, Cora and Citeseer. The collective vertex classification approaches, i.e. LP, ICA, G-NRNN and G-LSTM, largely outperform LR which only uses attributes of the target vertex. Both G-NRNN_d2 and G-LSTM_d2 consistently outperform all baseline methods on the citation networks, which indicates the effectiveness of our approach. In Figure 4.c and Figure 4.d, we compare our approach with the baseline methods on WebKB and WebKB-sim. For WebKB, our method obtains competitive results in comparison with ICA. LP works worse than LR on WebKB, where Micro-F1 score is less than 0.7. For this reason, it is not presented in Figure 4.c, and we will discuss this in detail in the next subsection. For WebKB-sim, our approaches consistently outperform all baseline methods, and G-LSTM_d2 outperforms G-NRNN_d1 when the training proportion is larger than 80%.

In general, G-LSTM_d2 outperforms G-NRNN_d2. This is likely due to the LSTMU's capability of memorizing information using memory cells. G-LSTM can thus better capture correlations between representations with long dependencies (Hochreiter and Schmidhuber 1997).

Dataset Comparison

To analyse co-occurrence of neighbouring labels, we compute the transition probability from target vertices to their neighbouring vertices. We first calculate the label co-occurrence matrix M^d, where $M^d_{i,j}$ indicates co-occurred times of labels l_i of target vertices v_i and labels l_j of d-step away vertices v_j. Then we obtain a transition probability matrix T^d, where $T^d_{i,j} = M^d_{i,j}/(\sum_k M^d_{i,k})$. The heat maps of T^d on four datasets are demonstrated in Figure 5.

For Cora and Citeseer, neighbouring vertices tend to share same labels. When d increases to 2, labels are still tightly correlated. That is probably why all ICA, LP G-NRNN and G-LSTM work well on Cora and Citeseer. In this situation, GRNN integrates features of d-step away vertices which may directly help classify a target vertex. For WebKB, correlation of labels is not clear, some label can be strongly related to more than two labels, e.g. *students* connects to *course*, *project* and *student*. Introducing vertices which are more steps away makes the correlation even worse for WebKB, e.g.all labels are most related to *student*. In this situation, LP totally fails, while ICA can learn the correlation of labels that are not same, i.e. *student* may relate to *course* instead of *student* itself. For this dataset, GRNN still achieves competitive results with the best baseline approach. For WebKB-sim, although *student* is still the label with highest frequency, the correlation between labels is clearer than WebKB, i.e. *project* relates to *project* and *student*. That is probably the reason why, the performance of our approach is good on WebKB-sim for both settings and G-LSTM_d2 achieves better results than G-NRNN_d2 when the training proportion is larger.

[4]As G-NRNN gives similar results, we only illustrate results for G-LSTM here

Conclusions and Future work

In this paper, we have presented a graph-based recursive neural network framework(GRNN) for vertex classification on graphs. We have compared two recursive units, NRNU and LSTMU within this framework. It turns out that LSTMU works better than NRNU on most experiments. Finally, the performance of our proposed methods outperformed several state-of-the-art statistical machine learning based methods.

In the future, we intend to extend this work in several directions. We aim to apply GRNN to large scale graphs. We also aim to improve the efficiency of GRNN and conduct time complexity analysis.

References

Angles, R., and Gutierrez, C. 2008. Survey of graph database models. *ACM Computing Surveys (CSUR)* 40(1):1.

Baeza-Yates, R.; Ribeiro-Neto, B.; et al. 1999. *Modern information retrieval*, volume 463. ACM press New York.

Chakrabarti, S.; Dom, B.; and Indyk, P. 1998. Enhanced hypertext categorization using hyperlinks. In *ACM SIGMOD Record*, volume 27, 307–318. ACM.

Craven, M.; DiPasquo, D.; Freitag, D.; and McCallum, A. 1998. Learning to extract symbolic knowledge from the world wide web. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 509–516. American Association for Artificial Intelligence.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12(Jul):2121–2159.

Giles, C. L.; Bollacker, K. D.; and Lawrence, S. 1998. Citeseer: An automatic citation indexing system. In *Proceedings of the 3rd ACM conference on Digital libraries*, 89–98. ACM.

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

Hosmer Jr, D. W., and Lemeshow, S. 2004. *Applied logistic regression*. John Wiley & Sons.

London, B., and Getoor, L. 2014. Collective classification of network data. *Data Classification: Algorithms and Applications* 399.

Lu, Q., and Getoor, L. 2003. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning*, volume 3, 496–503.

McCallum, A. K.; Nigam, K.; Rennie, J.; and Seymore, K. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3(2):127–163.

Monner, D. D., and Reggia, J. A. 2013. Recurrent neural collective classification. *IEEE transactions on neural networks and learning systems* 24(12):1932–1943.

Myaeng, S. H., and Lee, M.-h. 2000. A practical hypertext categorization method using links and incrementally available class information. In *Proceedings of the 23rd ACM International Conference on Research and Development in Information Retrieval*. ACM.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd International Conference on Machine Learning*.

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.

Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61:85–117.

Socher, R.; Lin, C. C.; Manning, C.; and Ng, A. Y. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, 129–136.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistic*. ACL.

Tang, J.; Chang, S.; Aggarwal, C.; and Liu, H. 2015. Negative link prediction in social media. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, 87–96. ACM.

Tang, D.; Qin, B.; and Liu, T. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1422–1432.

Wang, F., and Zhang, C. 2008. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering* 20(1):55–67.

Zhang, J.; Liu, B.; Tang, J.; Chen, T.; and Li, J. 2013. Social influence locality for modeling retweeting behaviors. In *Proceeding of the 23rd International Joint Conference on Artificial Intelligence*, volume 13, 2761–2767. Citeseer.