Deep Collective Inference

John Moore, Jennifer Neville
Departments of Computer Science and Statistics
Purdue University, West Lafayette, IN
Email: moore269, neville@purdue.edu

Abstract

Collective inference is widely used to improve classification in network datasets. However, despite recent advances in deep learning and the successes of recurrent neural networks (RNNs), researchers have only just recently begun to study how to apply RNNs to heterogeneous graph and network datasets. There has been recent work on using RNNs for unsupervised learning in networks (e.g., graph clustering, node embedding) and for prediction (e.g., link prediction, graph classification), but there has been little work on using RNNs for node-based relational classification tasks. In this paper, we provide an end-to-end learning framework using RNNs for *collective inference*. Our main insight is to transform a node and its set of neighbors into an unordered sequence (of varying length) and use an LSTM-based RNN to predict the class label as the output of that sequence. We develop a collective inference method, which we refer to as *Deep Collective Inference* (DCI), that uses semi-supervised learning in partially-labeled networks and two label distribution correction mechanisms for imbalanced classes. We compare to several alternative methods on seven network datasets. DCI achieves up to a 12% reduction in error compared to the best alternative and a 25% reduction in error on average—over all methods, for all label proportions.

Introduction

Collective inference is widely used to improve classification in network datasets (see e.g., Macskassy and Provost 2007). This is because many network datasets have nodes with attributes values that are correlated across the links. For example, in protein-protein interaction networks, the functions of interacting proteins in the cell are typically correlated. Similarly in social networks, friends tends to share similar interests and preferences. Thus in a partially labeled network where the attribute values of some nodes are observed, but others are unobserved, it is often helpful to learn a *statistical relational model* (see e.g., Getoor and Taskar 2007) and apply the model using collective classification (see e.g., Sen et al. 2008) to jointly make predictions about the set of unlabeled nodes.

Recently, the use of recurrent neural networks (RNNs) and deep learning for both supervised and unsupervised tasks have produced significant performance gains across domains such as speech translation, image processing, and natural language processing (Lipton, Berkowitz, and Elkan 2015; Chung et al. 2014; Graves and Jaitly 2014; Vinyals et al. 2014; Ren, Kiros, and Zemel 2015). While research on neural network models has been active for decades, recent achievements with the models are due to the availability of larger datasets, combined with several insights on how to structure the form of the model, initialize the weights, guide the optimization process to avoid overfitting, and fix problems such as vanishing gradients.

However, in the majority of domains where RNNs have been applied successfully, the examples are structured either as vectors, sequences, or matrices. Due to heterogeneous structure of graph data, it is still a relatively open question as to how to best design and exploit RNNs for learning in graphs with heterogeneous structure. There has been work on using neural networks for unsupervised learning in networks (e.g., graph clustering (Tian et al. 2014), node embeddings (Perozzi, Al-Rfou, and Skiena 2014; Tang et al. 2015)). However, when these methods have been applied for classification (e.g., link prediction (Li et al. 2014), graph classification (Yanardag and Vishwanathan 2015), node classification (Grover and Leskovec 2016)), it is typically based on using the output of unsupervised learning as features in a basic predictive model (e.g., logistic regression). The majority of previous methods for collective classification have been based on graphical models (e.g., Taskar, Abbeel, and Koller 2002; Richardson and Domingos 2006; Neville and Jensen 2007). There has been relatively little work on neural network models for supervised, end-to-end classification in partially-labeled relational graphs. One exception is the work of Monner and Reggia (2013) on Recurrent Neural Collective Classification (RNCC).

In this work, we develop an RNN for semi-supervised *collective inference* in attributed networks. Specifically, we consider the node classification problem, where given a single partially-labeled attributed network, the goal is to learn a model to jointly predict the remaining unlabeled nodes in the network. We propose an RNN approach that uses semi-supervised learning to jointly model relational structure and attributes. Our main insights are: (1) to transform a node and its set of neighbors into a random order (i.e., sequence of varying length) and use an LSTM-based RNN (Hochreiter and Schmidhuber 1997) to predict the class label as the output of that sequence, (2) to use a data augmentation or objective function balancing to adjust for skewed class label distributions, and (3) to initialize predictions of unlabeled nodes with a basic relational RNN, instead of using only the known class label values for the first round of learning.

We compare our proposed method to several baselines and alternatives, including state-of-the-art approaches to semi-supervised relational learning (Pfeiffer III, Neville, and Bennett 2015), network embedding (Grover and Leskovec 2016), and RNCC (Monner and Reggia 2013). We show that our approach achieves a significant reduction in classification error. We consider seven network datasets and observed up to a 12% reduction in error compared to the best alternative and a 25% reduction in error on average—over all competing methods, for all label proportions. Our main contributions are the following:

- We develop an LSTM-based RNN for node-based relational learning and collective inference, which we refer to this method as *Deep Collective Inference* (DCI).
- We show DCI outperforms state-of-the-art methods for semi-supervised collective classification using: graphical models, node embeddings, and recurrent neural networks.
- We evaluate the efficacy of our modeling choices and show that: (1) sequences of neighbors based on random orderings work better than ordering by connectivity, (2) a combination of data augmentation and cross entropy balancing helps to offset class label skew significantly during learning, (3) initializing class label predictions for unknown labels using a basic relational model improves performance compared to stacking (Kou and Cohen 2007) where class probabilities are used as features instead.

Background

We define a graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$ where $v_i \in \mathbf{V}$ with $i \in [1, n]$ is a node and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ is the edge set. If $e_{ij} \in \mathbf{E}$, there is an edge between v_i and v_j, otherwise there is not. Let $\mathcal{N}_i$ correspond to the set of neighbors of v_i, that is $\mathcal{N}_i = \{v_j \mid e_{ij} \in \mathbf{E}\}$. Let $\mathbf{F}, \mathbf{Y}$ be the feature and label set over the nodes, respectively. Each $v_i \in \mathbf{V}$ has a corresponding feature vector $\mathbf{f}_i \in \mathbf{F}$. For relational classification, the input network is partially labeled and thus only some of the nodes have an associated class label (i.e., if $y_j \in \mathbf{Y}$ then v_j is labeled). The goal of relational classification is to learn a model from the partially labeled network and use the model to make predictions $\hat{\mathbf{y}}$ for the unlabeled nodes $\{v_k\}$ $s.t.$ $y_k \notin \mathbf{Y}$. In this work we assume that Y is binary and can only take values $\{0, 1\}$. Moreover each prediction $\hat{y}_i \in [0, 1]$ represents the probability of that the class label value for v_i is 1. Let $\mathbf{V_U}, \mathbf{V_L}$ refer to nodes that are unlabeled and labeled, respectively.

Recurrent Neural Networks

RNNs have been used extensively in sequence prediction problems. The vanilla RNN can be described by the following equations:

$$\mathbf{h^{(t)}} = \sigma(W^{hx}\mathbf{x^{(t)}} + W^{hh}\mathbf{h^{(t-1)}} + \mathbf{b_h})$$

$$\hat{\mathbf{y}}^{(\mathbf{t})} = \mathit{softmax}(W^{yh}\mathbf{h^{(t)}} + \mathbf{b_y})$$

$$\mathit{softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum e^{z_i}} \text{for the j}^{th} \text{ entry in } \mathbf{z}$$

where t is the sequence length, $\mathbf{x^{(t)}}$ is the current element in the sequence, and $\mathbf{h^{(t-1)}}$ is the network's previous state. W^{hx}, W^{hh}, and W^{yh} are the matrices of weights between input and hidden layer, hidden to hidden layers, and hidden to output layers. $\mathbf{b_h}$ and $\mathbf{b_y}$ are bias parameters. σ is the continuous, differentiable activation function. For prediction, the output $\hat{\mathbf{y}}^{(\mathbf{t})}$ at each timestep t is calculated given current hidden state $\mathbf{h^{(t)}}$. RNNs are known to have problems with exploding or vanishing gradients. However, the LSTM architecture overcomes the exploding/vanishing gradient problem by carefully designing the hidden units such that gradients behave well (Hochreiter and Schmidhuber 1997). Thus, in this work we use the LSTM architecture.

Deep Collective Inference

In this paper, we develop a *deep collective inference* (DCI) method, which uses an RNN for collective classification in relational network data. We refer to weights in any RNN as both the weights and bias terms.

Problem Definition and Input Specification

We assume as input a partially labeled graph $< G, \mathbf{F}, \mathbf{Y} >$. The goal is to learn a predictive model from the labeled nodes $\mathbf{V_L}$ and use the model to make predictions for the unlabeled nodes $\mathbf{V_U} = \mathbf{V} - \mathbf{V_L}$. We first specify a non-collective version of our method to generate seed predictions known as Deep Relational Inference (DRI).

Examples are constructed as follows: for a node v_i, the target output is the class label y_i and the input is the node's features $\mathbf{f}_i$ and the features of its neighbors $\{\mathbf{f}_j \mid v_j \in \mathcal{N}_i\}$. We form a model to learn a mapping of the inputs $[\mathbf{f}_i, \{\mathbf{f}_j\}_{v_j \in \mathcal{N}_i}]$ to the output y_i. Since each node has a different number of neighbors, we will transform the input into an unordered sequence (of varying length). First, we randomly order the list of neighbors (i.e., $[v_{j_1}, v_{j_2}, ..., v_{j_{|\mathcal{N}_i|}}]$) and then we use the associated features as a sequential input to the model:

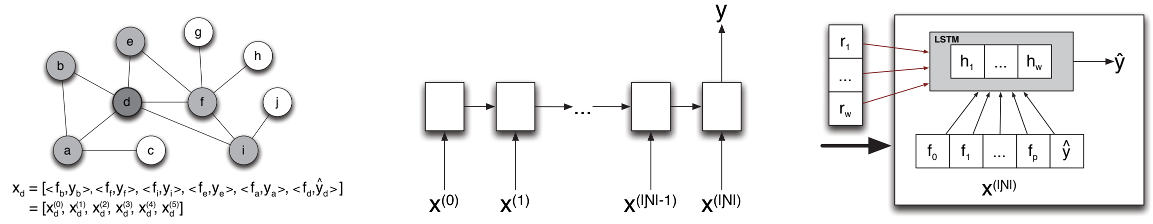
$$\begin{aligned}\mathbf{x}_i &= [\mathbf{f}_{j_1}, \mathbf{f}_{j_2}, ..., \mathbf{f}_{j_{|\mathcal{N}_i|}}, \mathbf{f}_i] \\ &= [\mathbf{x}_i^{(0)}, \mathbf{x}_i^{(1)}, ..., \mathbf{x}_i^{(|\mathcal{N}_i|)}] \end{aligned} \quad (1)$$

Note the last feature vector in the sequence $\mathbf{f}_i$ corresponds to the features of the target node. Here $\mathbf{x}^{(t)}$ refers to the t^{th} neighbor (i.e., element) in the sequence and $\mathbf{x}_i^{(|\mathcal{N}_i|)}$ refers to the target node. Thus the resulting inputs to the RNN will be a set of feature vector sequences: $\mathbf{X_L} = \{\mathbf{x}_i \mid \forall v_i \in \mathbf{V}_L\}$. We train DRI using a canonical RNN learning algorithm outlined in Algorithm 1. Then we perform inference to generate seed predictions on the test set, $\hat{\mathbf{y}}_\mathbf{i}^{\mathbf{0}}$ $\forall$ $v_i \in \mathbf{V_U}$. Predictions are obtained from the class with highest probability for v_i.

RNN for Collective Inference

To extend DRI to the collective inference setting we augment each feature vector with a class label value, i.e.,

$$\mathbf{x}_i = \big[[\mathbf{f}_{j_1}, y_{j_1}], [\mathbf{f}_{j_2}, y_{j_2}], ..., [\mathbf{f}_i, \hat{y}_i]\big]$$


b
e
g
h
d
f
j
a
c
i
$x_d = [<f_b,y_b>, <f_f,y_f>, <f_i,y_i>, <f_e,y_e>, <f_a,y_a>, <f_d,\hat{y}_d>]$
$= [x_d^{(0)}, x_d^{(1)}, x_d^{(2)}, x_d^{(3)}, x_d^{(4)}, x_d^{(5)}]$
y
$X^{(0)}$
$X^{(1)}$
$X^{(|N|-1)}$
$X^{(|N|)}$
r_1
...
r_w
LSTM
h_1
...
h_w
$\hat{y}$
f_0
f_1
...
f_p
$\hat{y}$
$X^{(|N|)}$

(a) Example network and unordered sequence for node v_d.

(b) Sequential structure of RNN.

(c) Internal details of RNN structure at end of sequence.

Figure 1: Illustration of model structure.

$$= [\mathbf{x}_i^{(0)}, \mathbf{x}_i^{(1)}, ..., \mathbf{x}_i^{(|\mathcal{N}_i|)}] \quad (2)$$

In this case, if $v_j \in \mathbf{V}_L$ then we append the true class label y_j. Otherwise, we can append $\hat{y}_j$. For the target instance v_i, we use $\hat{y}_i$, since using the true class would lead to obvious overfitting. This is illustrated for a small network in Figure 1a, where the sequence for node v_d is specified based on a random ordering of its neighbors. The structure of the RNN is illustrated in Figure 1b and 1c. r is the previous recurrent output and h refers to the hidden nodes. Any generic RNN architecture can be used; however, we specify an LSTM so that gradients behave well with hidden node size, $w = 10$ used in evaluations. Similar to other templated models, the length of the model is determined by the length of each sequence. Each square in Figure 1b represents a local RNN, with parameters $\mathbf{W}$ that are shared across each the replication in the sequence. Figure 1c shows the details of the local RNNs at the end of the sequence. The input consists of the feature vector for the example at that point in the sequence—the length depends on the number of node features in G. On the final hidden output of the RNN, the w outputs are aggregated using softmax to produce a predicted class label $\hat{y}$.

Label Skew Corrections

Skewed class label distributions motivated us to explore two different ways of correcting for skewed labels. The first involves generating more data from the rare class via data augmentation, while the second involves changing the objective function to balance between the classes. Our algorithm learns which method to use by evaluating their performance on a held out dataset where labels are known. We call this held out set $\mathbf{V_{L_{V_2}}}$ in Algorithm 3. It is constructed from a portion of the original validation set.

Data Augmentation In image classification, researchers perform data augmentation by adding noise in some way to an image and using the noisy image as another training example (Krizhevsky, Sutskever, and Hinton 2012). We follow this approach and generate additional training examples by replicating examples (nodes) and then add noise by swapping some attribute values (neighbors) across examples from the same class.

Algorithm 2 details our approach to data augmentation. It randomly selects two examples from the minority class, duplicates them, and then swaps at most 50% of the $\mathbf{x}$ attribute values across the vectors. This corresponds to randomly swapping the neighbors of the two duplicated nodes. In the line 23 of Alg. 2, the augmented dataset $\mathbf{X_{ret}}$ is formed by downsampling to get to the original size of $\mathbf{X_L}$. We use downsampling to ensure a fair comparison (i.e., equal training sizes) to other methods that do not use data augmentation. The algorithm returns the set $\mathbf{X_{ret}}$, which has a balanced class distribution. We can then simply train on $\mathbf{X_{ret}}$ whenever an RNN is trained in Algorithm 3.

Algorithm 1 RNNTrain($\mathbf{X_t}$, $\mathbf{X_v}$, $\mathbf{Y}$, maxItr, performSwap)

```
1: if performSwap then
2:     Specify Canonical Cross Entropy as objective
3: else
4:     Specify Balanced Cross Entropy as objective
5: end if
6: repeat
7:     Initialize the structure of the RNN with random gaussian weights $\mathbf{W}$.
8:     Intialize Scores = int array of length maxItr ; Initialize $\mathbf{t_e} = \mathbf{0}$ ;
9:     Train RNN with 1 epoch over $\mathbf{X_t}$ to optimize over labels using specified objective function
10:    Scores[$\mathbf{t_e}$] = Calculate loss on $\mathbf{X_v}$ ; $\mathbf{t_e}$++ ;
11: until [earlyStopMet(Scores, tol)] OR [$\mathbf{t_e} \geq maxItr$]
12: return RNN with learned weights $\mathbf{W}$
```

Balanced Cross Entropy We use balancing to adapt the objective function to imbalanced classes and therefore gradient updates when performing backpropgation. Here, we specifically modify the Cross Entropy objective function, but the adaptation is general and can be applied to any objective function, which has separate terms for each class. Recall that the cross entropy objective is:

$$CE = \frac{1}{n} \sum_{i=1}^{|\mathbf{V_L}|} y_i log(\hat{y_i}) + (1 - y_i)(log(1 - \hat{y_i}))$$

where y_i is the true label of node v_i and $\hat{y_i}$ is the corresponding prediction. To balance the gradient updates, we can simply take the frequency of positive labels (C_+) and negative labels (C_-) in the training and validation sets and use these to scale the terms in the objective function. The new *balanced* objective function is then:

$$CE_B = \frac{1}{n} \sum_{i=1}^{|\mathbf{V_L}|} C_- y_i log(\hat{y_i}) + C_+(1 - y_i)(log(1 - \hat{y_i}))$$

Algorithm 2 SwapAug Method($\mathbf{X_L}, \mathbf{Y_L}$)

```
1: counts0 = countClasses($\mathbf{Y_L}$, 0)
2: counts1 = countClasses($\mathbf{Y_L}$, 1)
3: smallLabel = 0
4: if counts0>counts1 then
5:     smallLabel = 1
6: end if
7: classDiff = | counts0 − counts1 |
8: Let $\mathbf{X_S} \subseteq \mathbf{X_L}$ be the set of sequences that have smallLabel
9: initialize $\mathbf{newX}$ to list of size=classDiff+1
10: $j = 0$
11: while j < length($\mathbf{newX}$) do
12:     $j += 2$
13:     $n_1, n_2$ = Select two integers at random $\in$ [0, length($\mathbf{X_S}$) − 1]
14:     $\mathbf{x}_{t_1}$ = copy($\mathbf{X_S}[n_1]$), $\mathbf{x}_{t_2}$ = copy($\mathbf{X_S}[n_2]$)
15:     numSwaps = min(length($\mathbf{x}_{t_1}$)/2, length($\mathbf{x}_{t_2}$)/2)
16:     $t1$Idxs = sample numSwap integers $\in$ [0, length($\mathbf{x}_{t_1}$) − 1]
17:     $t2$Idxs = sample numSwap integers $\in$ [0, length($\mathbf{x}_{t_2}$) − 1]
18:     for each ($t1$idx, $t2$idx) in ($t1$Idxs, $t2$Idxs) do
19:         swap$\left(\mathbf{x}_{t_1}^{(t1\text{idx})}, \mathbf{x}_{t_2}^{(t2\text{idx})}\right)$
20:     end for
21:     Append $\mathbf{x}_{t_1}, \mathbf{x}_{t_2}$ to $\mathbf{newX}$
22: end while
23: $\mathbf{X_{ret}}$ = Sample (counts0+counts1) sequences randomly from $\mathbf{newX}$ + $\mathbf{X_L}$
24: return $\mathbf{X_{ret}}$
```

In our experiments $C_+ \leq C_-$, but the adapted objective is general and isn't specific to a given class. Intuitively in our case, since positive labels occur less frequently, we want to upweight their effect by using C_-, and since negative labels occur more frequently, we want to downweight their effect by C_+. The net effect is that both sets of positive and negative examples have equivalent impact on the gradient updates. The modified objective is used in Algorithm 1 if specified.

DCI Semi-supervised Learning

We now outline our algorithm to learn a *deep collective inference* (DCI) model. Algorithm 4 is a wrapper method that chooses the mechanism DCI should use to adjust for imbalanced classes. Nodes with $y_i \in \mathbf{Y_L}$ are split into training $\mathbf{V_{L_T}}$, validation 1 $\mathbf{V_{L_{V_1}}}$, and validation 2 $\mathbf{V_{L_{V_1}}}$ in line 1. In lines 2-3, we learn a model using either swapping or the balanced objective by calling Algorithm 3. In order to select the Swapping or Balanced Cross Entropy approach, we simply return the model that performs best on the held out validation set $\mathbf{V_{L_{V_2}}}$ (lines 4-8).

Algorithm 3 describes how to learn a DCI model from a partially-labeled network with a specified approach to adjusting for imbalanced class distributions. First, a DRI model is learned and applied to initialize the unlabeled prediction set $\mathbf{Y}_U^0$ (lines 1-8). Then, the algorithm starts the semi-supervised collective inference process (lines 12-26). One iteration of collective inference consists of the following steps. New attribute input examples, $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{X}}_{\mathbf{val}}$, are formed by concatenating the current predictions $\mathbf{Y}_U^{t_c}$ and the labels $\mathbf{Y_L}$ to $\mathbf{F}$ (lines 16-17). Note that neighbor orders are random when constructing $\tilde{\mathbf{X}}$ on each collective iteration. If

Algorithm 3 DCI_Apply(G, $\mathbf{F}$, $\mathbf{Y}$, $\mathbf{V}_U$, $\mathbf{V_{L_T}}$, $\mathbf{V_{L_{V_1}}}$, $\mathbf{V_{L_{V_2}}}$, maxItr, tol, performSwap)

```
1: //Train a DRI RNN to initialize seed predictions
2: Form $\mathbf{X}$ by constructing an unordered input sequence $\mathbf{x}_i$ from trainNodes, $\mathbf{F}$, $\mathbf{Y}$.
3: Form $\mathbf{X_{val}}$ similarly except with $\mathbf{V_{L_{V_1}}}$
4: if performSwap then
5:     $\mathbf{X}$ = SwapAug($\mathbf{X}$, $\mathbf{Y}_L$)
6: end if
7: $\mathcal{M}$ = RNNTrain($\mathbf{X}$, $\mathbf{X_{val}}$, $\mathbf{Y}_L$, maxItr, performSwap)
8: Use $\mathcal{M}$ to predict $\hat{y}_i^0$ for each $v_i \in \mathbf{V}_U$
9: Intialize both ScoresV1, ScoresV2 = int arrays of length maxItr; $\mathbf{t_c} = \mathbf{0}$ ;
10: Initialize the structure of the RNN with random weights $\mathbf{W}_{t_c}$.
11: //Start collective learning
12: repeat
13:     if $\mathbf{t_c} != \mathbf{0}$ then
14:         Set $\mathbf{W}_{t_c} = \mathbf{W}_{t_c-1}^{trained}$
15:     end if
16:     Form $\tilde{\mathbf{X}}$ by constructing $\mathbf{x}_i$ from $\mathbf{V_{L_T}}$, $\mathbf{F}$, $\mathbf{Y}$, $\hat{\mathbf{Y}}^{tc}$
17:     Form $\tilde{\mathbf{X}}_{\mathbf{val}}$ similarly except with $\mathbf{V_{L_{V_1}}}$
18:     if performSwap then
19:         $\tilde{\mathbf{X}}$ = SwapAug($\tilde{\mathbf{X}}$, $\mathbf{Y}_L$)
20:     end if
21:     Let RNN, $\mathbf{W}_{t_c}^{trained}$ = RNNTrain($\tilde{\mathbf{X}}$, $\tilde{\mathbf{X}}_{\mathbf{val}}$, $\mathbf{Y}_L$, maxItr, performSwap)
22:     Use the RNN to predict $\hat{y}_i^{tc+1}$ for each $v_i \in \mathbf{V}_U$ to form $\hat{\mathbf{Y}}^{tc+1}$
23:     ScoresV1[$\mathbf{t_c}$] = Calculate loss on $\mathbf{V_{L_{V_1}}}$
24:     ScoresV2[$\mathbf{t_c}$] = Calculate loss on $\mathbf{V_{L_{V_2}}}$
25:     $\mathbf{t_c}$++
26: until [earlyStopMet(ScoresV1, tol)] OR [$\mathbf{t_c} \geq maxItr$]
27: Let $\mathbf{t_{best}}$ be best collective model on $\mathbf{V_{L_{V_1}}}$ based on BAE
28: return RNN with learned weights $\mathbf{W}_{t_{best}}$, predictions $\hat{\mathbf{Y}}^{t_{best}}$, ScoresV2[$t_{best}$]
```

SwapAug is used in line 19, then $\tilde{\mathbf{X}}$ is transformed to reflect swapping. Otherwise, it is not transformed and CE_B is instead used as the objective function.

Next parameters are re-estimated (line 21) by training the RNN. Let the parameters of the RNN at iteration t_c be $\mathbf{W_{t_c}}$. We perform backpropagation and utilize early stopping methods based on the validation set. Any type of early stopping method (denoted `earlyStopMet`) on the validation set can be used. For each iteration, t_c, we obtain $\hat{y}_i^{t_c}$ for each $v_i \in \mathbf{V}_U$ from the collective RNN by performing inference on the unlabeled set (line 22). The predictions on the unlabeled set are used in the next collective iteration.

Except for the first iteration, note that we initialize weights according to the previous iteration's trained weights, i.e., let $W_{t_{c+1}} = W_{t_c}^{trained}$ instead of initializing randomly in lines 13-15.

We repeat collective inference until the early stopping criteria is met on the validation set, $\mathbf{V_{L_{V_1}}}$, or until `maxItr` is reached. We return the RNN with predictions that performed best in terms of BAE on the validation set $\mathbf{V_{L_{V_1}}}$ (line 28).

The DCI algorithm is simultaneously learning the parameters of the RNN and making predictions for the unlabeled nodes—thus it is a semi-supervised approach to learning based on the full, albeit partially-labeled, graph. In addition, DCI uses an initialization approach where the predictions are initialized with the DRI model, which DCI uses on its first iteration of collective inference.

Algorithm 4 DCI(G, $\mathbf{F}$, $\mathbf{Y}$, $\mathbf{V}_U$, val1%, val2%, maxItr, tol)

```
1: Form $\mathbf{V_{L_T}}$, $\mathbf{V_{L_{V_1}}}$, and $\mathbf{V_{L_{V_2}}}$ from $\mathbf{Y_L}$ based on val1% and val2%
2: DCI_S_RNN, $\hat{\mathbf{Y}}_{\mathbf{S}}^{t_{best}}$, score_S = DCI_Apply($G$, $\mathbf{F}$, $\mathbf{Y}$, $\mathbf{V}_U$, $\mathbf{V_{L_T}}$, $\mathbf{V_{L_{V_1}}}$, $\mathbf{V_{L_{V_2}}}$, maxItr, tol, True)
3: DCI_B_RNN, $\hat{\mathbf{Y}}_{\mathbf{B}}^{t_{best}}$, score_B = DCI_Apply($G$, $\mathbf{F}$, $\mathbf{Y}$, $\mathbf{V}_U$, $\mathbf{V_{L_T}}$, $\mathbf{V_{L_{V_1}}}$, $\mathbf{V_{L_{V_2}}}$, maxItr, tol, False)
4: if $score\_S < score\_B$ then
5:     return DCI_S_RNN, $\hat{\mathbf{Y}}_{\mathbf{S}}^{t_{best}}$
6: else
7:     return DCI_B_RNN, $\hat{\mathbf{Y}}_{\mathbf{B}}^{t_{best}}$
8: end if
```

There are a number of DCI variations that are possible depending on different algorithmic decisions. We evaluate the algorithmic choices and found that they each improve performance significantly. See Section Empirical Evaluation for more details.

Time Complexity

DCI is scalable especially if run on a GPU. DCI takes $O(|\mathbf{E}| + |\mathbf{V}|)$ time since it's bottleneck is performing $|\mathbf{E}| + |\mathbf{V}|$ backpropagations. This could potentially be reduced to $b * |V|$ if performing truncated backprop to only b steps, which essentially corresponds to throwing away sequence input for large degree nodes.

Related Work

Relational Machine Learning

Relational machine learning (RML) methods seek to jointly model user labels given their attributes and relational structure (Getoor and Taskar 2007). In particular, semi-supervised RML approaches have been developed for partially-labeled networks (Xiang and Neville 2008; McDowell and Aha 2012; Lin and Cohen 2010). Many semi-supervised approaches to RML perform Expectation Maximization (EM), which can be divided into two basic steps: an **E-Step** that uses collective classification and an **M-Step** that optimizes the parameters given the current predicted labels. Current state-of-the-art methods use pseudolikelihood EM with a maximum entropy constraint in the inference step to produce better calibrated probability estimates (Pfeiffer III, Neville, and Bennett 2015). This method, which we refer to as $PLEM$, is the current best performing RML method for large-scale partially-labeled networks.

Neural Network Models for Graphs

Some recent work has used neural network models for graph clustering (Tian et al. 2014), labeling graphs (Yanardag and Vishwanathan 2015), and link prediction (Li et al. 2014). However, this work typically focuses on learning models of the graph structure alone and has not considered the development of node-based predictive models.

There has been some work on unsupervised learning of node embeddings, which are then used afterwards for learning predictive models for nodes. Line (Tang et al. 2015) uses a two-stage approach to learning an embedding such that nodes that are either well-connected or share many neighbors are close. Structural Deep Network Embedding (SDNE) (Wang, Cui, and Zhu 2016) generates a network embedding via an auto-encoder architecture where 1st and 2nd order neighbors are used in their objective function. DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) uses random walks and a skip-gram based approach. Node2Vec (Grover and Leskovec 2016) extends the skip-gram architecture from DeepWalk and performs various sampling strategies to sample neighborhoods differently. Currently, SDNE and Node2Vec are state of the art node embedding methods. While these methods have been shown to produce an embedding that is useful for subsequent classification, they do not directly learn to optimize class label predictions (i.e., the embedding is unsupervised) and moreover, they do not consider node attributes in their models.

The Graph Neural Network (GNN) (Scarselli et al. 2009) is a specially designed recurrent neural network for graph data, which assumes node attributes and optionally specifies attributes for edges. The model takes a whole graph as input to propagate information about each node. Unfortunately, GNNs cannot be directly applied to the node classification problem since GNN training examples consist of entire labeled graphs.

The work most closely related to ours is the Recurrent Neural Collective Classification (RNCC) method (Monner and Reggia 2013). During our initial work on DCI, we were unaware of the RNCC method, but the two methods are similar in that they both use neighbor information in LSTMs for collective classification. However, there are a number of key differences between the two methods:

- *Data*: RNCC does not adjust for imbalanced class label distributions, while DCI does.
- *Architecture*: RNCC's LSTM architecture models a node v_i's features separately from its neighbors' features (i.e., with separate weights). DCI's architecture models them jointly with shared weights. In addition, RNCC uses the hidden representation of neighbors as input features, while DCI does not.
- *Learning*: RNCC performs collective inference on every epoch of learning (i.e., epoch=collective iteration). In contrast, DCI waits until parameter estimation has locally converged in terms of epochs in order to perform the next round of collective inference. Also, RNCC uses a generalized LSTM learning algorithm (Monner and Reggia 2012), while DCI uses the standard LSTM backpropagation through time algorithm (Lipton, Berkowitz, and Elkan 2015).

In our experimental evaluation next, we compare to RNCC and also investigate the impact of a number of these algorithmic decisions in ablation studies.

Empirical Evaluation

Data

We employ seven datasets in our evaluations. Table 1 reports the number nodes, edges, density, and positive label proportion of each dataset.

Dataset	$\lvert V \rvert$	$\lvert E \rvert$	density	$P(+)$
Facebook	5906	73,374	4.2e-3	0.32
IMDB	7934	122,230	3.9e-3	0.164
Amz_DVD_20000	16,118	75,596	5.8e-3	0.5
Amz_DVD_7500	16,118	75,596	5.8e-3	0.21
Amz_Music_64500	56,891	272,544	1.7e-4	0.5
Amz_Music_7500	56,891	272,544	1.7e-4	0.08
Patents	881,187	5,302,712	1.4e-5	0.169

Table 1: Datasets

The Facebook dataset is a snapshot of the Purdue University Facebook network (Pfeiffer III, Neville, and Bennett 2015). Users have two attributes: religious views, and gender, and a class label: political views.

The Internet Movie Database (IMDB) is a movie dataset where we predict if a movie will have a gross revenue $\geq \$50$ million (Pfeiffer III, Neville, and Bennett 2015). Each movie (node) has 29 genre attributes and 9 boolean variables that record whether the average rating is greater than a particular value. Edges are created between movies that share two or more producers.

Amazon DVD 20000 is a subset of the Amazon copurchase data gathered by (Leskovec, Adamic, and Huberman 2007). Nodes correspond to DVD items and edges are created via DVD copurchases. Each node has 24 attributes describing the movie's genres. The prediction task is to determine whether the item has an Amazon salesrank < 20000. Amazon DVD 7500 is the same as Amazon DVD 20000 except that the threshold for class labels is salesrank $<$ 7500. This changes the class label distribution.

Amazon Music 64500 dataset is another subset of the Amazon data where nodes are song items, and each node has 22 attributes describing its styles. The class label threshold is salesrank $<$ 64500. Amazon Music 20000 uses the threshold for class labels is salesrank < 7500

The Patents citation network (Pfeiffer III, Neville, and Bennett 2015) consists of patent nodes and citations among them. Each patent has an attribute vector which records the TF/IDF values of it's top 50 words. We consider the "Computers" classification task where patents are predicted to be filed in "Primary Category 2" (computer related) or not.

Methodology

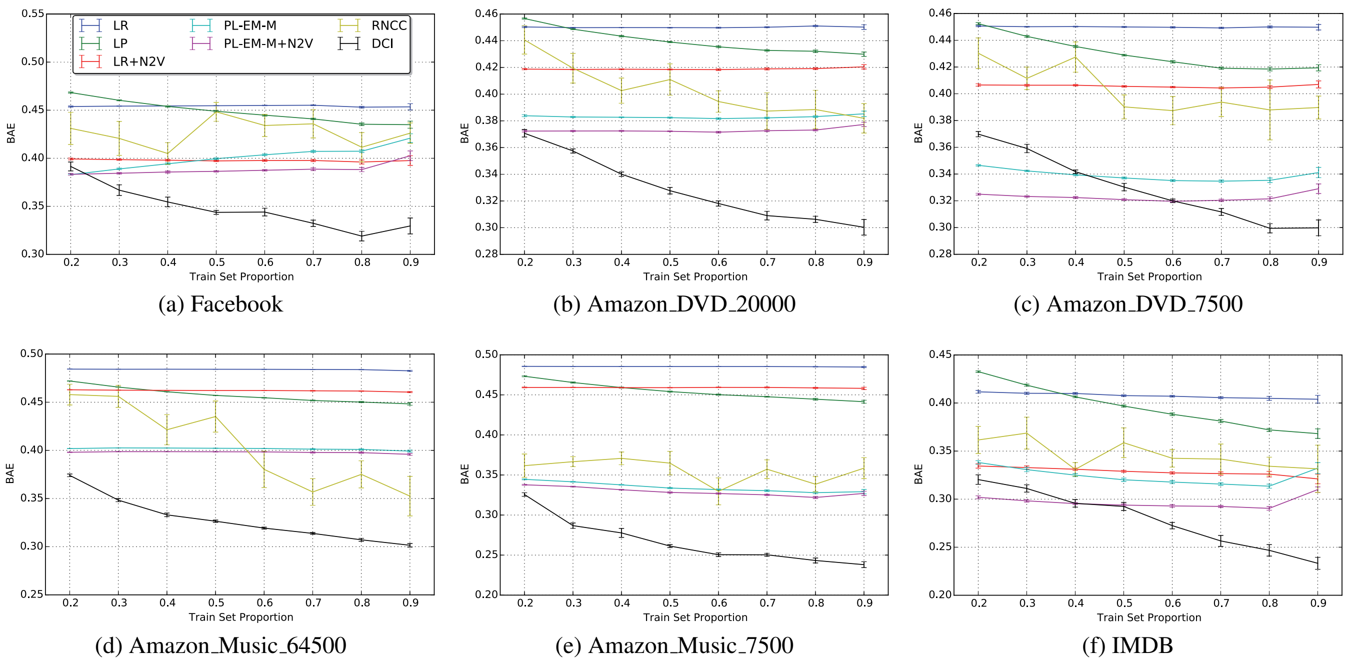
Our metric for algorithmic comparison is the Balanced Absolute Error (BAE), which normalizes error across classes. See Pfeiffer III, Neville, and Bennett (2015) for more detail. We run Algorithm 3 and plot performance for DRI and DCI using LSTMs.

We use 10 trials for all datasets. For each trial, nodes are randomly assigned to the labeled set, $\mathbf{V_L}$, Then the unlabeled set is formed from the remainder: $\mathbf{V_U} = \mathbf{V} - \mathbf{V_L}$. The following plots show performance as the proportion of the data used in the training set size is varied (i.e., $\frac{|\mathbf{V_L}|}{\mathbf{V}}$). For example, 0.2 corresponds to 20% labeled nodes and 80% unlabeled in the network. For DCI, 12% and 3% of nodes of the whole dataset is used for $\mathbf{V_{L_{V_1}}}$ and $\mathbf{V_{L_{V_1}}}$, respectively, which accounts for a total of 15% for validation. Thus when the training proportion is 0.2, DCI uses 5% for training, 15% for validation, and 80% for testing. Degree is concatenated to each $\mathbf{f}_i \in F$ to compare to previous work (Pfeiffer III, Neville, and Bennett 2015). For our implementation, we use Theano under the library known as Blocks (van Merriënboer et al. 2015). All evaluations are performed using three computer clusters with 20 Xeon cores each and memory ranging from 64gb-256gb ram.

Models

We compare our proposed method DCI to baselines and state-of-the-art alternative methods. Unless otherwise specified, each method uses all available labeled nodes, $\mathbf{V_L}$, for training.

- LP: A label propagation baseline that uses a weighted vote of the predicted/true labels of neighbors to make predictions (Macskassy and Provost 2007).
- LR: An independent logistic regression baseline that uses only node features (no relational data) to predict the label optimized using L2 regularization.
- LR+N2V: LR but utilizing new attributes by concatenating node2vec (Grover and Leskovec 2016) features of size 128 and the original attribute vectors of each node.
- PLEM: The MaxEntInf adjusted semi-supervised relational learning method from (Pfeiffer III, Neville, and Bennett 2015), specially designed for data with imbalanced class labels which is currently state-of-the art.
- PLEM+N2V: PLEM with added node2vec features. Note that this is not an existing method, we add node embedding features to PLEM to ascertain whether they improve relational learning methods.
- DCI: Weights of LSTMs are initialized (except after first iteration of collective inference) with sampled values from the Gaussian distribution. We apply Batched Gradient Descent where *batch size* $=$ 100. The maximum number of epochs for any network is 200. Early Stopping is used to check if performance on the validation set does not improve in the last 10 epochs. If no improvement, training stops early, and the model performing best on the validation set is chosen. DCI was run for 100 collective iterations with the same early stopping criterion. We used $w = 10$ (number of hidden nodes). We did not perform hyperparameter optimization, though this could further improve results. DCI further splits $\mathbf{V_L}$ into training $\mathbf{V_{L_T}}$, validation 1 $\mathbf{V_{L_{V_1}}}$, and validation 2 $\mathbf{V_{L_{V_2}}}$ sets.
- RNCC: RNCC is a similar model to DCI (Monner and Reggia 2013). We implement a version as close to it as possible. We use their RNN architecture where a given node v_i's attributes a_i are modeled via separate weights than neighborhood node's attributes. We utilize hidden states learned from a non-collective version of RNCC as the first hidden states to be used in collective inference. However, we use a canonical LSTM and standard backpropagation for learning, rather than the specific learning rules from (Monner and Reggia 2013). RNCC is learned with exact same hyperparameters and learning settings as



(a) Facebook (b) Amazon_DVD_20000 (c) Amazon_DVD_7500

(d) Amazon_Music_64500 (e) Amazon_Music_7500 (f) IMDB

Figure 2: DCI compared to alternatives LP, LR, RNCC, LR+N2V, PLEM, PLEM+N2V.

described for DCI, with the exception of collective iterations. Since epoch=collective iteration for RNCC, we utilize Max Collective iteration = 200. Lastly, training and validation sets are exactly similar to DCI's, except that $\mathbf{V_{L_{V_1}}}$ and $\mathbf{V_{L_{V_2}}}$ are merged and used as one complete validation set.

Comparison to alternative methods

We compare DCI to the baselines LP, LR, LR+N2V, RNCC, PLEM, and PLEM+N2V. Figure 2 reports the results. The mean and standard errors of the BAE measure are plotted on each dataset. We use the same train/test splits in each model.

It's important to note that amazon_DVD_20000 and amazon_Music_64500 both have about 50/50 class distributions, while the rest of the datasets are imbalanced. In these cases, the performance gap between DCI and competing methods is significant and clear. Amazon_DVD_7500 is the only dataset where DCI does not outperform PLEM+N2V on the majority of label proportions. However, DCI does outperform it when the training size is high enough. This is possibly due to small data, large class imbalance, and/or low network density. These may not be problems for PLEM+N2V since node2vec explores more of the network besides just neighbors. However, on all other datasets DCI outperforms other state-of-the art methods for most if not all training/test splits, especially when it is not sparsely labeled.

It's interesting to see that the RNCC implementation does not outperform PLEM for most datasets. This indicates that the algorithmic decisions for DCI allow DCI to significantly outperform RNCC.

Because of the size of the Patents data, for efficiency we perform DCI-10 and RNCC-10, which only uses a subset of at most 10 neighbors (randomly selected). For DCI-10, we

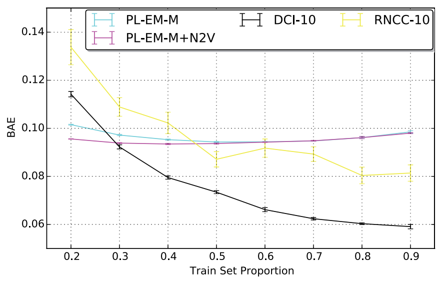


Figure 3: DCI-10 compared to alternatives RNCC, PLEM, PLEM+N2V on Patents dataset.

also only run for 10 collective iterations, while keeping the rest of RNCC's learning parameters the same. We compare DCI-10 to PLEM and PLEM+N2V. See Figure 3 for the results. Note that we did not include the more naive methods in the plot, since their BAE values were too high and masked the performance of the other methods. LR has ≈ 0.4 BAE and LR+N2V has ≈ 0.19 BAE for all proportions. For this network, it is interesting that with only 10 hidden units and truncated backpropagation, DCI-10 eventually outperforms PLEM and PLEM+N2V. We expect further improvement if all neighbors are used.

In summary, our evaluation show that, across seven network datasets, DCI resulted in an up to a 12% reduction in error compared to the best alternative (PLEM+N2V). Moreover, DCI achieved an average of 25% reduction in error over all methods, all datasets, and all label proportions. These results demonstrate the impact of our proposed method for improving collective inference in large-scale networks.

Comparison to DCI variants

We evaluate several variants of our proposed DCI method on smaller datasets to assess initial algorithmic choices. All other variants of DCI do not perform any label distribution corrections and use the whole validation set for early stopping instead of separating out a portion for $\mathbf{V_{L_{V_2}}}$. We test on all but the largest dataset in order to compute personalized page rank vectors for each node, which is computationally intensive on large datasets.

- DCI-WSB: Learned without data augmentations or cross entropy balancing (WSB)
- DCI-ST: WSB version learned with stacking predictions instead of using predicted class labels
- DCI-A: Instead of random ordering, WSB version learned by ordering the top 10 neighbors in *ascending* order with respect to personalized page rank for each given node where the restart vector always restarts back to the given node (Jeh and Widom 2002)
- DCI-D: Instead of random ordering, WSB version learned by ordering the top 10 neighbors in *descending* order with respect to personalized page rank
- DCI-10-WSB: Uses only a subset of at most 10 neighbors (randomly selected) rather than the full set.
- DRI-WSB: Learned without label distribution corrections.

Figure 4 shows a comparison between DCI, DCI-WSB, DCI-ST, DCI-A, DCI-D, DCI-10-WSB, and DRI-WSB. DRI-WSB performs worst in all evaluations compared to all collective methods, which indicates that collective inference is improving performance. In all datasets, specifying an order and running DCI-A or DCI-D result in about the same or worse performance than all other collective methods, which suggests that a page rank does not improve and sometimes degrades performance.

It is not so clear to determine which is best among DCI-ST and DCI-WSB. Therefore, we summarize the gain by calculating reduction in error over all training set proportions. Overall there is small gain of 1.6% comparing DCI-WSB to DCI-ST. This shows a small improvement when we use class label predictions compared to stacking.

We also perform DCI-WSB with randomly initializing predictions for the first iteration of collective inference (DCI-WSB-R) and notice these results do not show a clear winner in the plots. Therefore, we perform DCI with swapping (DCI-S), DCI with balancing (DCI-B), and their random initialization variants (DCI-S-R, DCI-B-R). Overall there is a small 0.82% gain when we initialize with predictions from DRI (DCI, DCI-S, and DCI-B) compared to using random initial predictions from the prior (DCI-R, DCI-S-R, and DCI-B-R). DCI-WSB has a 0.06% gain over DCI-WSB-R. DCI-S has a 1.78% gain over DCI-S-R. DCI-B has a 1.04% gain over DCI-B-R.

Discussion and Conclusion

Recurrent neural networks have recently produced impressive performance gains but have also been traditionally used for sequential problems where order is generally important and well-suited for structured inputs such as vectors or matrices. In this work, we provided an end-to-end learning framework by using RNNs for collective classification as opposed to a two-stop process of finding a node embedding, then using this representation in another model. *Deep Collective Inference* (DCI) is developed for semi-supervised learning in partially labeled networks. We proposed a data augmentation scheme and a Balanced Cross Entropy objective to balance the classes, which improves performance.

We conducted experiments across seven network datasets with varying levels of label availability and class proportions. We compare to other state-of-the-art methods in relational learning, node embeddings, and RNNs. Our results are clearly superior to other methods except in sparsely labeled networks. DCI provides up to a 12% reduction in error compared to the best state-of-the-art alternative (PLEM+N2V) and a 25% reduction in error on average—over the six competing methods, across all label proportions.

Acknowledgements

This research is supported by NSF under contract numbers IIS-1149789, IIS-1302172, CCF-0939370. We thank Joseph J. Pfeiffer III for helpful suggestions that motivated this work. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

References

Chung, J.; Gülçehre, Ç.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR* abs/1412.3555.

Getoor, L., and Taskar, B., eds. 2007. *Introduction to Statistical Relational Learning*. MIT Press.

Graves, A., and Jaitly, N. 2014. Towards end-to-end speech recognition with recurrent neural networks. In Jebara, T., and Xing, E. P., eds., *ICML'14*, 1764–1772. JMLR Workshop and Conference Proceedings.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD'16*.

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

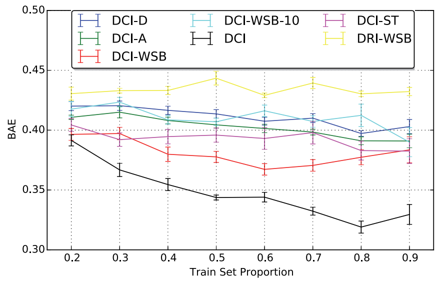
Jeh, G., and Widom, J. 2002. Scaling personalized web search. In *WWW'12*, 271–279. ACM Press.

Kou, Z., and Cohen, W. W. 2007. Stacked graphical models for efficient inference in markov random fields. In *SDM*, 533–538. SIAM.

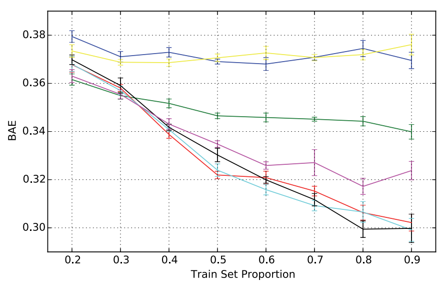
Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS'12*.

Leskovec, J.; Adamic, L. A.; and Huberman, B. A. 2007. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)* 1(1):5.

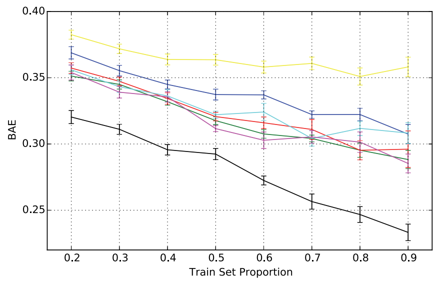
Li, X.; Du, N.; Li, H.; Li, K.; Gao, J.; and Zhang, A. 2014. A deep learning approach to link prediction in dynamic networks. In *SDM*, volume 14, 289–297. SIAM.



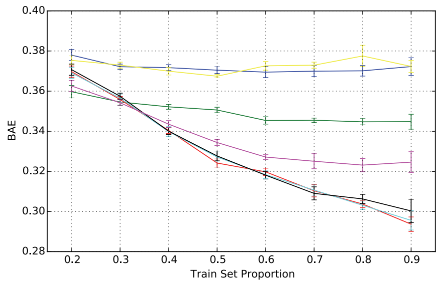
(a) Facebook



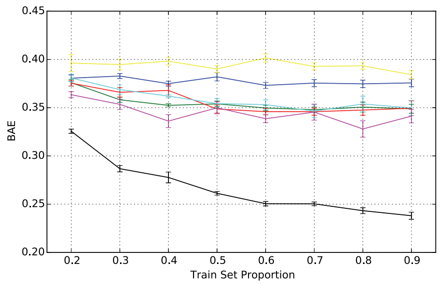
(b) Amazon_DVD_7500



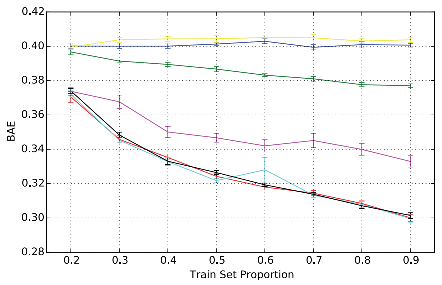
(c) IMDB



(d) Amazon_DVD_20000



(e) Amazon_Music_7500



(f) Amazon_Music_64500

Figure 4: DCI compared to its variants on Facebook, IMDB, and Amazon datasets

Lin, F., and Cohen, W. W. 2010. Semi-supervised classification of network data using very few labels. In *ASONAM*, 192–199.

Lipton, Z. C.; Berkowitz, J.; and Elkan, C. 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. *ArXiv e-prints*.

Macskassy, S. A., and Provost, F. 2007. Classification in networked data: A toolkit and a univariate case study. *The Journal of Machine Learning Research* 8:935–983.

McDowell, L., and Aha, D. 2012. Semi-supervised collective classification via hybrid label regularization. *arXiv preprint arXiv:1206.6467*.

Monner, D., and Reggia, J. A. 2012. A generalized lstm-like training algorithm for second-order recurrent neural networks. *Neural Netw.* 25:70–83.

Monner, D. D., and Reggia, J. A. 2013. Recurrent neural collective classification. *IEEE Transactions on Neural Networks and Learning Systems* 24(12):1932–1943.

Neville, J., and Jensen, D. 2007. Relational Dependency Networks. *JMLR* 8:653–692.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *KDD'14*, 701–710. ACM.

Pfeiffer III, J. J.; Neville, J.; and Bennett, P. N. 2015. Overcoming relational learning biases to accurately predict preferences in large scale networks. WWW '15.

Ren, M.; Kiros, R.; and Zemel, R. S. 2015. Image question answering: A visual semantic embedding model and a new dataset. *CoRR* abs/1505.02074.

Richardson, M., and Domingos, P. 2006. Markov Logic Networks. *Mach. Learn.* 62(1-2):107–136.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The graph neural network model. *Trans. Neur. Netw.* 20(1):61–80.

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

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *WWW'15*, 1067–1077.

Taskar, B.; Abbeel, P.; and Koller, D. 2002. Discriminative probabilistic models for relational data. In *UAI*, 485–492.

Tian, F.; Gao, B.; Cui, Q.; Chen, E.; and Liu, T.-Y. 2014. Learning deep representations for graph clustering. In *AAAI*, 1293–1299.

van Merriënboer, B.; Bahdanau, D.; Dumoulin, V.; Serdyuk, D.; Warde-Farley, D.; Chorowski, J.; and Bengio, Y. 2015. Blocks and fuel: Frameworks for deep learning. *CoRR* abs/1506.00619.

Vinyals, O.; Toshev, A.; Bengio, S.; and Erhan, D. 2014. Show and tell: A neural image caption generator. *CoRR* abs/1411.4555.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. KDD'16, 1225–1234.

Xiang, R., and Neville, J. 2008. Pseudolikelihood em for within-network relational learning. In *ICDM*, 1103–1108.

Yanardag, P., and Vishwanathan, S. 2015. Deep graph kernels. In *KDD'15*, 1365–1374. ACM.