12-2016

Deep collective inference

John A. Moore
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By John A Moore

Entitled
DEEP COLLECTIVE INFERENCE

For the degree of Master of Science

Is approved by the final examining committee:

Jennifer Neville
Chair

Bruno Ribeiro

Dan Goldwasser

Mark Daniel Ward

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Jennifer Neville

Approved by: Sunil Prabhakar / William J Gorman 12/6/2016

Head of the Departmental Graduate Program Date

DEEP COLLECTIVE INFERENCE

A Thesis

Submitted to the Faculty

of

Purdue University

by

John A. Moore

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

December 2016

Purdue University

West Lafayette, Indiana

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Moore, John A. MS, Purdue University, December 2016. Deep Collective Inference. Major Professor: Jennifer Neville.

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.

1. INTRODUCTION

Collective inference is widely used to improve classification in network datasets (see e.g., [1]). 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., [2]) and apply the model using collective classification (see e.g., [3]) 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 [4–8]. 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 [9], node embeddings [10, 11]). However, when these methods have been applied for classification (e.g., link prediction [12], graph classification [13], node classification [14]), 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., [15]; [16]; [17]). 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 [18] 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 [19] 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 [20], network embedding [14], and RNCC [18]. 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 [21] where class probabilities are used as features instead.

2. BACKGROUND AND RELATED WORK

We define a graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$ where $\mathbf{v_i} \in \mathbf{V}$ 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 $\mathbf{v_i}$ and $\mathbf{v_j}$, otherwise there is not. Let $\mathcal{N}_i$ correspond to the set of neighbors of $\mathbf{v_i}$, that is $\mathcal{N}_i = \{\mathbf{v_j} \mid e_{ij} \in \mathbf{E}\}$. Let $\mathbf{F}, \mathbf{Y}$ be the feature and label set over the nodes, respectively. Each $\mathbf{v_i} \in \mathbf{V}$ 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 $\{0, 1\}$. Each prediction also takes values $\hat{y}_i \in \{0, 1\}$ and is obtained from probabilistic models by selecting the class label with highest probability for v_i. Let $\mathbf{V_U}, \mathbf{V_L}$ refer to nodes that are unlabeled and labeled, respectively.

2.1 Relational Machine Learning

Relational machine learning (RML) methods seek to jointly model user labels given their attributes and relational structure [2]. In particular, for our given problem, semi-supervised learning (SSL) RML approaches (e.g., [20]), have been developed for partially-labeled networks. These method utilize the full network and simultaneously estimate the parameters of the model while making predictions for the class labels of unlabeled nodes. Collective classification methods (e.g., [3]) iteratively update the predictions of unlabeled nodes and then use these predictions in features of neighboring nodes.

The relational SSL approach of [20] comprises a templated model that ties the parameters across a set of local component conditional models. A number of rela-

tional local conditional models exist, and most can be viewed relational extensions to common independent models (e.g., relational naive Bayes [1] or relational logistic regression [22]).

The most intuitive and naive examples of component conditional models are *Relational Logistic Regression* (RLR; [22]) and *Relational Naive Bayes* (RNB; [1]). Each of the two exemplify the two main ways that RML handle variable length neighbors. RLR summarizes neighbors by computing features such as the proportions for each label and total degree. RLR can also incorporate neighbor features by avereraging over these feature vectors. Aggregation features thus depend on the researcher's subjective choice of aggregation feature(s). The second way to handle varying neighbors is to incorporate them in a product of probabilities.

More formally, let node $\mathbf{v_i}$ correspond to the node in consideration. In Relational Naive Bayes, we let:

$$\begin{aligned} P(Y_i|\mathbf{F}) &= P(Y_i|\mathbf{f}_i, \mathcal{N}_i) \propto P(\mathbf{f}_i, \mathcal{N}_i|Y_i)P(Y) \\ &= P(\mathbf{f}_i|Y_i)P(Y) \prod_{j \in \mathcal{N}_i} P(Y_j|Y_i) \end{aligned}$$

by the conditional independence assumption.

Hence, one only needs to estimate attribute vector given label probability, $P(\mathbf{f}_i|Y_i)$, prior probability $P(Y)$, and neighbor's corresponding probabilities $P(Y_j|Y_i)$.

To perform relational learning for a partially labeled network, a natural extension of basic RML methods utilizes the unlabeled data to make better predictions within the network. The most common form utilizes *expectation maximization* (EM). EM iteratively updates the parameter estimates by utilizing the expected values of the unlabeled examples to relearn the parameters and can be divided into two basic steps: an **E-Step** that uses collective classification and an **M-Step** that optimizes the parameters given the predicted labels.

Pseudolikelihood Expectation Maximization ([20]) is currently one of best performing relational EM methods. One can further improve the method by incorporat-

ing a Maximum Entropy constraint in the inference step to produce better calibrated probability estimates. Maximum Entropy coupled with Pseudolikelihood Expectation Maximization ($PLEM$) is the current best performing SSL RML method for large-scale partially-labeled networks [20].

2.2 Neural Networks

Multilayer Perceptrons (MLPs) or vanilla neural networks are simply compositions of non-linear and linear functions [23]. Let $\mathbf{x}$ be a fixed length vector input, W^i weights, $\mathbf{b_i}$ bias terms, act_i a non-linear differentiable activation function, and l corresponds to the last layer.

Then an MLP consists of a set of activation functions that produce continuous output zs such as:

Consider $\mathbf{z_1} = act_1(\mathbf{b_1} + W^1\mathbf{x})$,

$\mathbf{z_2} = act_2(\mathbf{b_2} + W^2\mathbf{z_1})$,

...

$\mathbf{z_l} = act_l(\mathbf{b_l} + W^l\mathbf{z_{l-1}})$.

The output of these hidden units are transformed into a probability distribution using softmax.

$\hat{y} = softmax(\mathbf{b_{l+1}} + W^{l+1}z_l)$,

where $softmax(\mathbf{z})_j = \frac{e^{z_j}}{\sum e^{z_i}}$ for the jth entry in vector $\mathbf{z}$.

Or

For binary labels,

$\hat{y} = logistic(\mathbf{b_{l+1}} + W^{l+1}z_l)$,

where $logistic(z) = \frac{1}{1+e^{-z}}$.

The logistic function is used for binary labels such that weights and bias terms are restricted to $b_{l+1} \in R^{1\times 1}$, $W^{l+1} \in R^{1\times w}$, $z_l \in R^{w\times 1}$, w is hidden node size.

To learn MLPs, various objective functions can be used such as Cross Entropy. This is combined with backpropagation, which differentiates the objective function

and subsequently applies the chain rule to backpropagate until the input of the neural network is reached.

$CrossEntropy(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{|\mathbf{V_{L_s}}|} y_i log(\hat{y_i}) + (1 - y_i)(log(1 - \hat{y_i}))$

$\mathbf{V_{L_s}}$ is any subset of $\mathbf{V_L}$ or $\mathbf{V_{L_s}} \subset \mathbf{V_L}$

We sometimes call MLPs, Deep Neural Networks if l is large.

2.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) have been used extensively in sequence prediction problems. The vanilla RNN can be thought of as an unfolded feedforward neural network through time where recurrent edges that share weights exist at each time step. Each node at time t receives input from the current element in the sequence, $\mathbf{x^{(t)}}$, and also the network's previous state $\mathbf{h^{(t-1)}}$. For prediction, the output $\hat{\mathbf{y}}^{\mathbf{(t)}}$ at each timestep t is calculated given current hidden node values $\mathbf{h^{(t)}}$. Since all previous inputs, $\mathbf{x^{(0)}}, ..., \mathbf{x^{(t-1)}}$ are encoded in $\mathbf{h^{(1)}}, ..., \mathbf{h^{(t)}}$ via recurrent connections, all previous inputs influence the prediction $\hat{\mathbf{y}}^{\mathbf{(t)}}$. We can rewrite the hidden states:

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

Or

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

for binary label classification as described in the previous section. Here σ is the continuous, differentiable activation function, and in our work we used the standard hyperbolic tangent (tanh) function. W^{hx} is the matrix of weights between input and hidden layer, W^{hh} is the matrix of recurrent weights between hidden to hidden layers, while W^{yh} is the matrix of weights between hidden to output layers. $\mathbf{b_h}$ and $\mathbf{b_y}$ are bias parameters that allow every node to learn offset parameters. In Figure 2.1, we

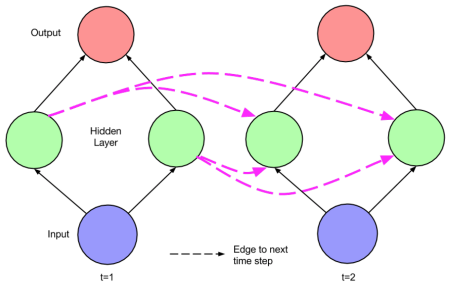


Fig. 2.1.: RNN example for a sequence of two inputs from [4].

show an RNN example with a sequence of 2 inputs. As you can see, previous hidden units and inputs feed into the next layers to produce outputs.

Since RNNs can be unfolded into a feedforward network, one can simply apply the backpropagation algorithm to an unfolded recurrent network through time (BPTT). However, this approach suffers from the exploding and vanishing gradients problem when backpropagating error through many timesteps. Since recurrent weights are shared across time steps, the effect of an input at time τ on the label $\hat{\mathbf{y}}^{(\mathbf{t})}$, where $t > \tau$ either explodes or approaches zero, exponentially fast with respect to $t-\tau$. Therefore, gradients during backpropagation either explode or vanish as well [4]. There have been many approaches to addressing this problem. However, the approach that has gained the most recent attention are Long Short Term Memory (LSTM) networks [19].

LSTM networks overcome the exploding and vanishing gradients problem by carefully designing the hidden units such that gradients do not explode or vanish. LSTMs can therefore be regarded as RNNs except for these modifications that allow backpropagation to behave well. The name long-short term memory is derived from the fact that an LSTM has the ability to remember important long-term and important short-term information. That is intuitively, the LSTM can decide, which pieces of information are important to remember for the short-term and which are important for the long-term. Hidden units in LSTMs are referred to as memory cells, and are

modified to have an input node $g^{(t)}$, an input gate $i^{(t)}$, a forget gate $f^{(t)}$, output gates $o^{(t)}$, and internal state $s^{(t)}$. Current LSTMs have the corresponding update equations:

$$
\begin{aligned}
g^{(t)} &= tanh(W^{gx}\mathbf{x^{(t)}} + W^{gh}\mathbf{h^{(t-1)}} + \mathbf{b_g}) \\
i^{(t)} &= sigmoid(W^{ix}\mathbf{x^{(t)}} + W^{ih}\mathbf{h^{(t-1)}} + \mathbf{b_i}) \\
f^{(t)} &= sigmoid(W^{fx}\mathbf{x^{(t)}} + W^{fh}\mathbf{h^{(t-1)}} + \mathbf{b_f}) \\
o^{(t)} &= sigmoid(W^{ox}\mathbf{x^{(t)}} + W^{oh}\mathbf{h^{(t-1)}} + \mathbf{b_o}) \\
s^{(t)} &= g^{(t)} \odot i^{(i)} + s^{(t-1)} \odot f^{(t)} \\
h^{(t)} &= tanh(s^{(t)} \odot o^{(t)})
\end{aligned}
$$

where $\odot$ is pointwise multiplication.

The input node $g^{(t)}$, takes input and the previous hidden layer in the standard way. Tanh is used here, but other activation functions can be used as well. The internal state $s^{(t)}$ consists of a self-connected recurrent edge with fixed unit weight. This allows error to flow in backpropagation through time steps easily and solves the problem of vanishing or exploding gradients. The input gate $i^{(t)}$, helps to modulate how much of the input that we should utilize since it is pointwise multiplied by $g^{(t)}$ when calculating $s^{(t)}$. If $i^{(t)}$ consists of 0s, then we completely disregard the current input. If it consists of 1s, we utilize the whole current input. Similarly, forget gates $f^{(t)}$, allow us to forget unneeded past internal state. Lastly, the output gate $o^{(t)}$ allows us to remember important information when calculating the next hidden state $h^{(t)}$. The standard softmax can then be applied to obtain predictions.

Many have noted that recent breakthroughs in sequence prediction for various problems have been due to LSTMs, not RNNs [5]. Currently LSTMs are viewed as the start of the art in speech translation [6], machine translation [25], image captioning [7], and question answering systems [8]. All of these problems consist of sequence problems and LSTMs have performed very well on each of these.

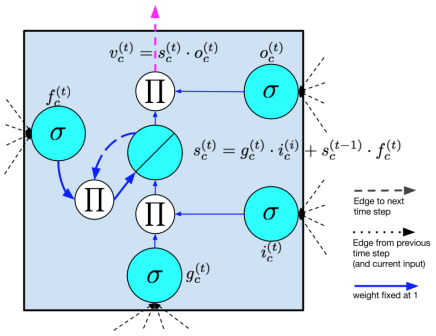


Fig. 2.2.: LSTM memory cell from [24].

2.4 Related Work

The majority of the domains where RNNs have been applied successfully involve examples that are structured as vectors, sequences, or matrices. Due to the heterogeneous structure of graph data, research involving neural networks for graph data has only recently begun to be explored.

There has been some recent work on using RNNs for graph clustering [9], labeling graphs [13], and link prediction [12]. 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 as features when learning predictive models for node prediction. Line [11] 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) [26] generates a network embedding via an auto-encoder architecture where 1st and 2nd order neighbors are used in their objective function. DeepWalk [10] uses random walks and a skip-gram based approach. Node2Vec [14] 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 di-

rectly learn to optimize class label predictions (i.e., the embedding is unsupervised). Rather, one hopes that the embeddings learned are useful for classification. Furthermore, they do not consider node attributes in their models. However, a typical application of node embeddings for node classification is to simply use the learned embeddings and node attributes in a standard classification method. Other applications of embeddings involve graph clustering [9], graph prediction [13], and link prediction [12].

The Graph Neural Network (GNN; [27]) is a specially designed recurrent neural network for graph data. The model assumes there are node attributes and optionally specifies attributes for edges and edge types. The model takes a whole graph as input and naturally propagates information about each node in each hidden state. More formally, the update for a hidden state at time k for each node $\mathbf{v_i}$ is denoted as: $\mathbf{h_i^{(k)}} = f^*(\mathbf{a_i}, \mathbf{a_{CO(i)}}, \mathbf{a_{NBR(i)}}, \mathbf{h_{NBR(i)}^{(k-1)}})$, where $\mathbf{a_i}$ is the attribute for $\mathbf{v_i}$, $\mathbf{a_{CO(i)}}$ refers to the attributes of the edges incident on $\mathbf{v_i}$, $\mathbf{a_{NBR(i)}}$ refers to the attributes of neighbors connected to $\mathbf{v_i}$ by an edge, and $\mathbf{h_{NBR(i)}^{(k-1)}}$ is the previous hidden states for $\mathbf{v_i}$'s neighbors. Unfortunately, GNNs are infeasible to apply to very large networks on a GPU since the whole graph must be used as input into the GNN at once. Therefore, GNN's will have memory issues when run on a GPU (as of right now GPUs have max 12gb). The time complexity of this model is $O(|E| + |V|)$, which is relatively fast. Furthermore, the GNN cannot be directly applied to the node classification problem since the graph is partially labeled and training examples for GNNs consist of entire labeled graphs.

Another line of work related to our proposed methods is the Search Convolutional Neural Network (SCNN) [28]. This neural network extends the idea of convolution to graphs and subsequently searches over H hops from a given node. SCNNs have a computational complexity of $O(N^2F)$, where F is the size of a feature vector. While SCNNs can be used for relational classification, they currently do not employ any kind of collective inference. Future work could incorporate the DCI approach to collective

learning and inference into the SCNN model to ascertain if the predictions and class labels of neighbors can further improve classification performance.

The work most closely related to ours is the Recurrent Neural Collective Classification (RNCC) method [18]. 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 an LSTM structure, for collective classification. However, there are a number of key algorithmic differences between the two methods. We outline these below. See section 3.1 for a more detailed description of DCI.

- *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 [29], while DCI uses the standard LSTM backpropgation through time algorithm [4].

In our experimental evaluation (see Section 4.5.1), we compare directly to RNCC and also investigate the impact of a number of these algorithmic decisions in ablation studies.

One final area of related work is the use of data augmentation to improve learning in neural networks. The use of data augmentation to produce similar and noisy images in image classification [30] motivates some aspects of our DCI algorithm. In image classification, there is often not enough training data for classification algorithms.

Images may be represented by large $n \times m$ matrices composed of pixel values. If every pixel value is used as features in a learning algorithm, then the curse of dimensionality takes effect and there is often not enough training examples for consumption by the learning algorithm [31]. Therefore, researchers perform data augmentation methods, which takes as input pixel matrices and produces a similar image but with noise added in some way. [30] modifies pixel value intensities and generate images with translations and horizontal reflections. In DCI, we will use a similar approach to generate additional training examples when learning from networks with skewed label distributions.

3. DEEP COLLECTIVE INFERENCE

In this work, 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.

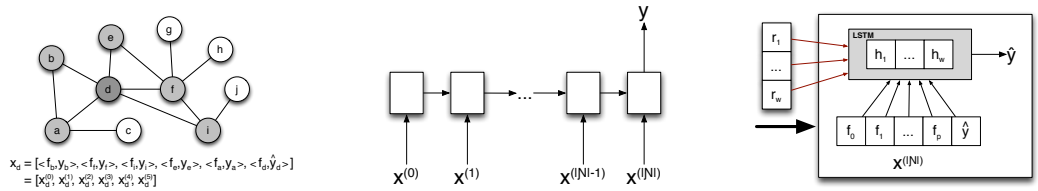
3.1 Problem Definition and Input Specification

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 $\{\mathbf{f}_j \,|\, v_j \in \mathcal{N}_i\}$. We introduce 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:

$$\begin{aligned} \mathbf{x}_i &= [\mathbf{f}_{j_1}, \mathbf{f}_{j_2}, ..., \mathbf{f}_{j_{|\mathcal{N}_i|}}, \mathbf{f}_i] \\ &= [\mathbf{x}_i^{(0)}, \mathbf{x}_i^{(1)}, ..., \mathbf{x}_i^{(|\mathcal{N}_i|)}] \end{aligned} \tag{3.1}$$

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



(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.

Fig. 3.1.: Illustration of model structure.

feature vector sequences: $\mathbf{X_L} = \{\mathbf{x}_i | \forall v_i \in \mathbf{V}_L\}$. 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.

3.2 RNN for Collective Inference

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

$$\begin{aligned} \mathbf{x}_i &= \big[[\mathbf{f}_{j_1}, y_{j_1}], [\mathbf{f}_{j_2}, y_{j_2}], ..., [\mathbf{f}_i, \hat{y}_i]\big] \\ &= [\mathbf{x}_i^{(0)}, \mathbf{x}_i^{(1)}, ..., \mathbf{x}_i^{(|\mathcal{N}_i|)}] \end{aligned} \tag{3.2}$$

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 3.1a, 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 3.1b and 3.1c. 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 3.1b represents a local RNN, with parameters $\mathbf{W}$ that are shared across each the replication in the sequence. Figure 3.1c 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 $\hat{y}$.

3.3 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.

3.3.1 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 [30]. 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

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

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

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

counts0 = countClasses($\mathbf{Y_L}$, 0)
counts1 = countClasses($\mathbf{Y_L}$, 1)
smallLabel = 0
if counts0>counts1 **then**
    smallLabel = 1
end if
classDiff = | counts0 − counts1 |
Let $\mathbf{X_S} \subseteq \mathbf{X_L}$ be the set of sequences that have smallLabel
initialize **newX** to list of size=classDiff+1
$j = 0$
while j < length(**newX**) **do**
    j+= 2
    n_1, n_2 = Select two integers at random $\in [0, \text{length}(\mathbf{X_S}) - 1]$
    $\mathbf{x}_{t_1}$ = copy($\mathbf{X_S}[n_1]$), $\mathbf{x}_{t_2}$ = copy($\mathbf{X_S}[n_2]$)
    numSwaps = min(length($\mathbf{x}_{t_1}$)/2, length($\mathbf{x}_{t_2}$)/2)
    t1Idxs = sample numSwap integers $\in [0, \text{length}(\mathbf{x}_{t_1}) - 1]$
    t2Idxs = sample numSwap integers $\in [0, \text{length}(\mathbf{x}_{t_2}) - 1]$
    for `each` (t1idx, t2idx) `in` (t1Idxs, t2Idxs) **do**
        $\text{swap}\left(\mathbf{x}_{t_1}^{(t1\text{idx})}, \mathbf{x}_{t_2}^{(t2\text{idx})}\right)$
    end for
    Append $\mathbf{x}_{t_1}$, $\mathbf{x}_{t_2}$ to **newX**
end while
$\mathbf{X_{ret}}$ = Sample (counts0+counts1) sequences randomly from **newX** + $\mathbf{X_L}$
return $\mathbf{X_{ret}}$

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.

3.3.2 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:

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

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.

3.4 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 $\mathbf{V_{L_T}}$,

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: $\quad$ $\mathbf{X}$ = 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: $\quad$ **if** $\mathbf{t_c} != \mathbf{0}$ **then**
14: $\quad\quad$ Set $\mathbf{W}_{t_c} = \mathbf{W}_{t_c-1}^{trained}$
15: $\quad$ **end if**
16: $\quad$ Form $\tilde{\mathbf{X}}$ by constructing $\mathbf{x}_i$ from $\mathbf{V_{L_T}}$, $\mathbf{F}$, $\mathbf{Y}$, $\hat{\mathbf{Y}}^{t_c}$
17: $\quad$ Form $\tilde{\mathbf{X}}_{\mathbf{val}}$ similarly except with $\mathbf{V_{L_{V_1}}}$
18: $\quad$ **if** performSwap **then**
19: $\quad\quad$ $\tilde{\mathbf{X}}$ = SwapAug($\tilde{\mathbf{X}}$, $\mathbf{Y}_L$)
20: $\quad$ **end if**
21: $\quad$ Let RNN, $\mathbf{W}_{t_c}^{trained}$ = RNNTrain($\tilde{\mathbf{X}}$, $\tilde{\mathbf{X}}_{\mathbf{val}}$, $\mathbf{Y}_L$, maxItr, performSwap)
22: $\quad$ Use the RNN to predict $\hat{y}_i^{t_c+1}$ for each $v_i \in \mathbf{V}_U$ to form $\hat{\mathbf{Y}}^{t_c+1}$
23: $\quad$ ScoresV1[$\mathbf{t_c}$] = Calculate loss on $\mathbf{V_{L_{V_1}}}$
24: $\quad$ ScoresV2[$\mathbf{t_c}$] = Calculate loss on $\mathbf{V_{L_{V_2}}}$
25: $\quad$ $\mathbf{t_c}$++
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[$t_{best}$]

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: $\quad$ **return** DCI_S_RNN, $\hat{\mathbf{Y}}_{\mathbf{S}}^{t_{best}}$
6: **else**
7: $\quad$ **return** DCI_B_RNN, $\hat{\mathbf{Y}}_{\mathbf{B}}^{t_{best}}$
8: **end if**

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 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, $\mathbf{V}_{\mathbf{L}_{\mathbf{V_1}}}$, or until `maxItr` is reached. We return the RNN with predictions that performed best in terms of BAE on the validation set $\mathbf{V}_{\mathbf{L}_{\mathbf{V_1}}}$ (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.

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.

3.5 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. In the above analysis, we assume that the number of iterations for collective inference is constant and the number of epochs over training data is also constant. In practice, the constants are relatively large; however, GPUs help to parallelize and reduce these constants.

4. MAIN EMPIRICAL EVALUATIONS

4.1 Data

We employ seven datasets in our evaluations, all from [20]. Table 4.1 reports the number nodes, edges, and positive label proportion of each dataset.

The Facebook dataset is a snapshot of the Purdue University Facebook network. Users have two attributes: religious views, and gender, and a class label: political views. The labels are political views, while religious views and gender are features. This is the smallest network. The network consists of 5906 nodes and 73394 edges.

The Internet Movie Database (IMDB) is a movie dataset where we predict if a movie will have a gross revenue $\geq$ \$50 million. 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 co-purchase data gathered by [32]. 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. This network has about 50/50 label distribution. The network consists of 16,118 nodes and 75,596 edges.

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. This network has about 50/50 label distribution. This network is the second largest and consists of 56,891 nodes and 272,544 edges.

Table 4.1.: Datasets

Dataset	$\|V\|$	$\|E\|$	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

Amazon Music 7500 is the same as Amazon Music 64500 except it uses the threshold for class labels is salesrank < 7500.

The Patents citation network 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. Edges are created through citations. We consider the "Computers" classification task where patents are predicted to be filed in "Primary Category 2" (a computer related category) or not. This network is by far our largest network with 881,187 nodes and 5,302,712 edges.

4.2 Evaluation Methodology

We use Balanced Absolute Error (BAE) as our error statistic, which normalizes error across classes, and measures the absolute error of a classifier C. Let κ be the set of label classes and V is a node set. BAE is defined as:

$BAE_C = \frac{\sum_{y \in \kappa} err_C(y,V)}{|\kappa|}$ where $err_C(y,V) = \frac{\sum_{\mathbf{v_i} \in \mathbf{V_U}} P_C(\mathbf{y_i} \neq y) I(\mathbf{y_i}=y)}{\sum_{\mathbf{v_i} \in \mathbf{V}} I(\mathbf{y_i}=y)}$

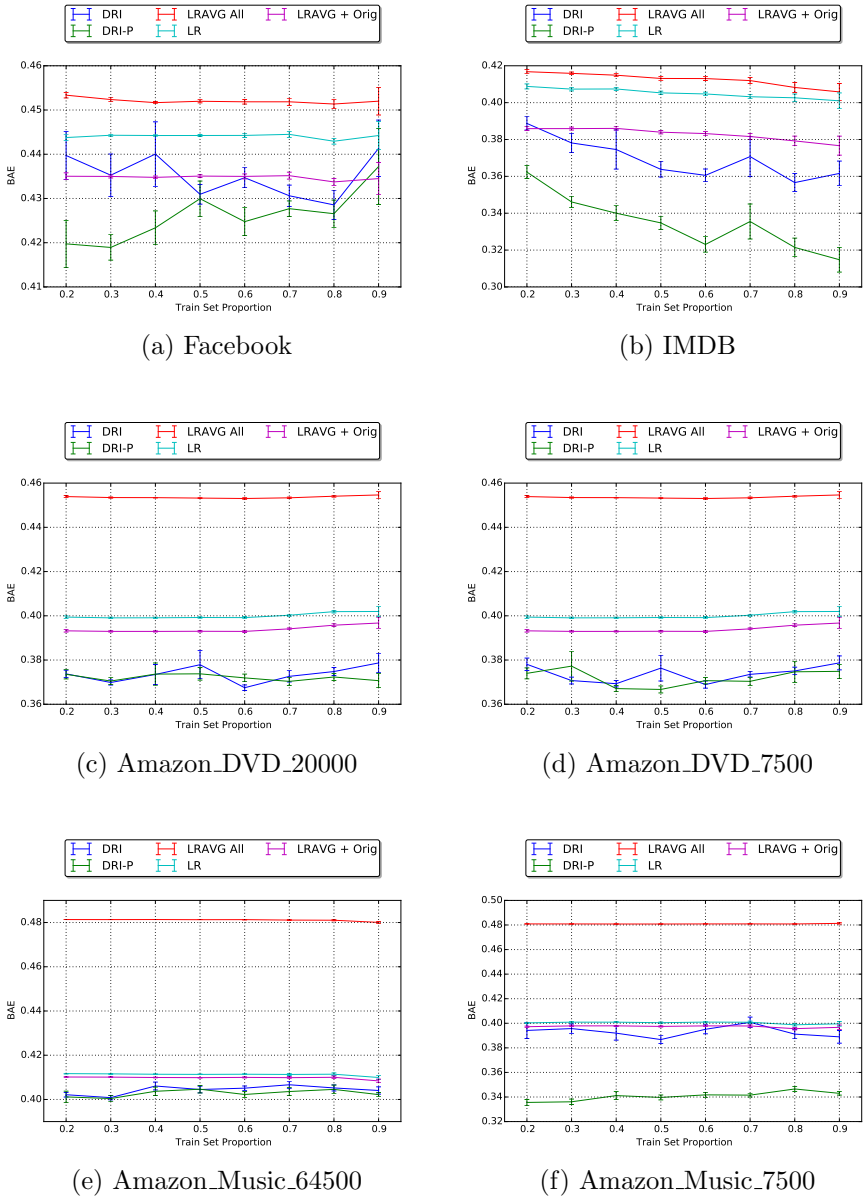
We also calculate reduction in overall error in section 4.5.1. Let $reduction(m_i) = \frac{1}{|methods|} \sum_{j=0}^{|methods|} gain(m_i, m_j)$ for method m_i, which is DCI in our case. Furthermore let $gain(m_i, m_j) = \frac{1}{|datasets|} \sum_{k=0}^{|datasets|} gain(m_j, m_i, d_k)$ where $gain(m_i, m_j)$ shows the overall gain of method m_i over m_j. Finally let
$gain(m_j, m_i, d_k) = \frac{1}{|X_{bae}|} \sum_{x \in X_{bae}} \frac{bae_{x_i} - bae_{x_j}}{bae_{x_i}}$ where $gain(m_j, m_i, d_k)$ shows the average gain given dataset, d_k. bae_{x_i} and bae_{x_j} are the bae values at point x for each method m_i and m_j. X_{bae} is the set of points tested on the plots.

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 [20]. For our implementation, we use Theano under the library known as Blocks [33]. All evaluations are performed using three computer clusters with 20 Xeon cores each and memory ranging from 64gb-256gb ram.

4.3 DRI Variants

We first evaluate on variants of DRI vs variants of logistic regression (LR). We do so to show even DRI can outperform simple baselines. This also verifies that the non-linear way of summarizing neighborhoods is better than simply averaging neighbor vectors and use this as input to Logistic Regression. Each of the DRI models have the same hyperparameters and learning parameters. Weights of LSTMs are initialized 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. Because of the size of the Patents data, for efficiency we perform DCI-10, which only uses a subset of at most 10 neighbors (randomly selected).

- DRI-WSB: DRI learned without data augmentations or cross entropy balancing
- DRI-S: DRI learned with swapping



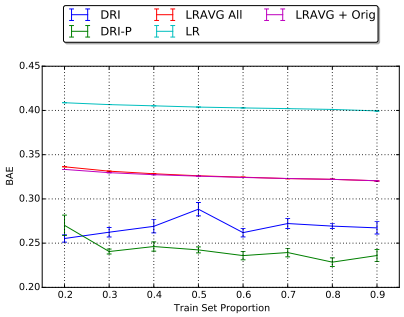
(a) Facebook (b) IMDB

(c) Amazon_DVD_20000 (d) Amazon_DVD_7500

(e) Amazon_Music_64500 (f) Amazon_Music_7500

Fig. 4.1.: DRI compared to its variants on Facebook, IMDB, Amazon DVD, and Amazon Music datasets.

- DRI-B: DRI learned with balancing the cross entropy objective function
- LR: An independent logistic regression baseline that uses only node features (no relational data) to predict the label optimized using L2 regularization.



(a) Patents

Fig. 4.2.: DRI compared to its variants on the Patents dataset.

- LRAVG-A: Recall that $\mathbf{x}_i = [\mathbf{f}_{j_1}, \mathbf{f}_{j_2}, ..., \mathbf{f}_{j_{|\mathcal{N}_i|}}, \mathbf{f}_i]$

  For each training example for LR, we simply average the attribute vectors, ie $\mathbf{x}_{i_{avg}} = \frac{f_i + \sum_k^{|\mathcal{N}_i|} \mathbf{f}_{j_k}}{|\mathcal{N}_i|+1}$

- LRAVG-O: Averaging all neighbor attribute vectors in $\mathbf{X}$ and concatenating the original node's attributes as input, ie $\mathbf{x}_{i_{avg_c}} = [\frac{\sum_k^{|\mathcal{N}_i|} \mathbf{f}_{j_k}}{|\mathcal{N}_i|}, f_i]$

Figures 4.1 and Figure 4.2 show a comparison between DRI-WS, DRI-P, LR, LRAVG-A, and LRAVG-O. DRI-P tends to perform the best or equal to DRI-WS. In the balanced datasets (Amazon_DVD_20000 and Amazon_Music_64500), both DRI-WS and DRI-P. However, on imbalanced and smaller datasets (facebook, IMDB, Amazon_DVD_7500), DRI-WS performs better or has around the same performance as the Logistic Regression alternatives. Both DRI-WS and DRI-P have clear gains when compared to on the biggest datasets (Amazon_Music data and Patents), with DRI-P having the best performance. These experiments illustrate the effectiveness of using a non-linear way of aggregating inputs or RNNs.

4.4 DCI and Alternatives

We compare our proposed method DCI to the following baseline and state-of-the-art alternative methods:

- LP: A label propagation baseline that uses a weighted vote of the predicted/true labels of neighbors to make predictions [1].
- LR: Logistic Regression, same as in section 4.3
- LR+N2V: LR but utilizing new attributes by concatenating node2vec [14] features of size 128 with default hyperparameters and the original attribute vectors of each node.
- PLEM: The MaxEntInf adjusted semi-supervised relational learning method from [20], 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 [18]. 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 [18]. RNCC is learned with exact same hyperparameters and learning settings as 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.

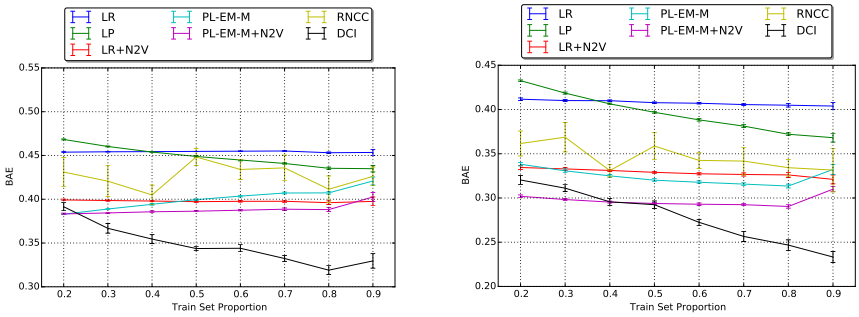
4.5 Main Results

The mean and standard errors of the BAE measure are plotted on each dataset. We plot results for the same train/test splits in each model.

4.5.1 Comparison to Other Methods

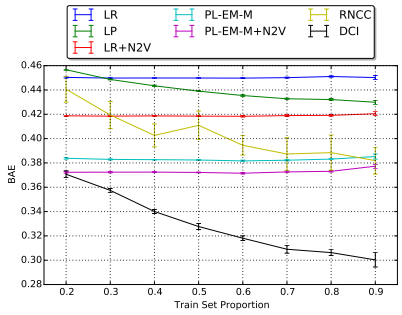
We compare DCI to the baselines LP, LR, LR+N2V, RNCC, PLEM, and PLEM+N2V. Figure 5.1 reports the results. The mean and standard errors of the BAE measure are plotted on each dataset. We use the same train/test splits in each model.

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.

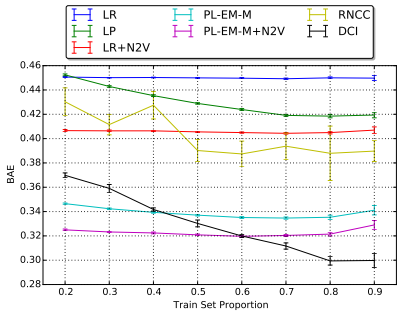


(a) Facebook

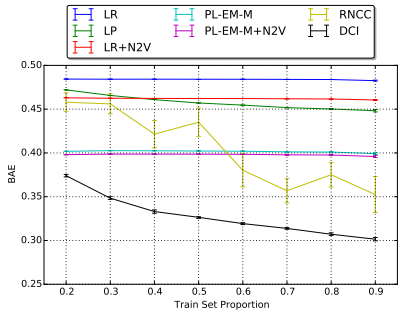
(b) IMDB



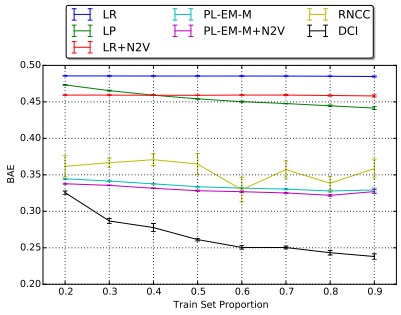
(c) Amazon_DVD_20000



(d) Amazon_DVD_7500



(e) Amazon_Music_64500



(f) Amazon_Music_7500

Fig. 4.3.: DCI compared to its alternatives LP, LR, LR+N2V, RNCC, PLEM, and PLEM+N2V.

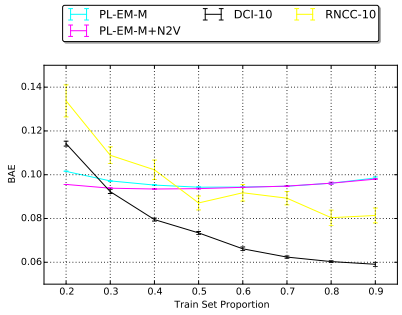


Fig. 4.4.: DCI-10 compared to its alternatives PLEM+N2V and PLEM on Patents dataset.

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.

Again, 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 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 4.4 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.

5. DCI VARIANTS EVALUATIONS

In this set of evaluations we compare variants of our proposed methods. The same evaluation methodology is used from Section *Evaluation Methodology*. We split our evaluations into two categories: Data Oriented Variants and Algorithmic Oriented Variants. We study how different variants affect performance and how this motivates our decisions for the DCI algorithm. Not every combination of decisions can be explored easily. Therefore, once we find an inferior decision, we tend not to consider it in later evaluations. For example, we find that page rank orders don't help empirically; therefore, we do not consider this in combination with swapping.

5.1 Data Oriented Variants

5.1.1 DCI Variants on Smaller Datasets

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. (ie $O(|V|^2 + |V||E|)$).

- 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

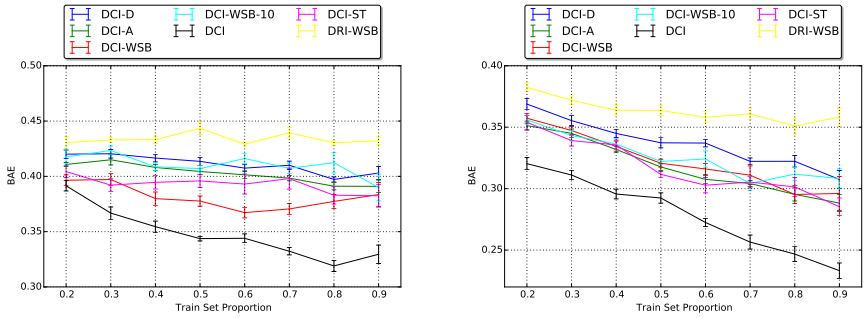
- DCI-A: WSB version learned by ordering the neighbors in *ascending* order with respect to personalized page rank instead of random ordering
- DCI-D: WSB version learned by ordering the neighbors in *descending* order with respect to personalized page rank instead of random ordering
- DCI-10-WSB: Uses only a subset of at most 10 neighbors (randomly selected) rather than the full set.
- DRI-WSB: Learned without label distribution corrections.

Recall that Page Rank satisfies the discrete-time Markov Process ([34]) where $z^{(t+1)} = \alpha P z^{(t)} + (1-\alpha)c$, where P is the transition matrix and c is a vector representing teleportation probabilities. To utilize page rank we run a Personalized Page Rank ([35]) for every node in the network. The transitions equations are now for each $\mathbf{v_i}$, $z_i^{(t+1)} = \alpha P z_i^{(t)} + (1-\alpha)c_i$, where $c_{ii} = 1$ and $c_{ij} = 0 \forall j \neq i$. Setting c_i this way, intuitively, enables PPR to find nodes that are "close" to $\mathbf{v_i}$. For DCI-A, we simply order the top (highest PR valued) neighbor nodes according to page rank in ascending order to determine neighbor order. For DCI-D, we do the same except order in descending order. Note that the last feature vector in each sequence still corresponds to the feature of the target node itself as $\mathbf{f}_i$.

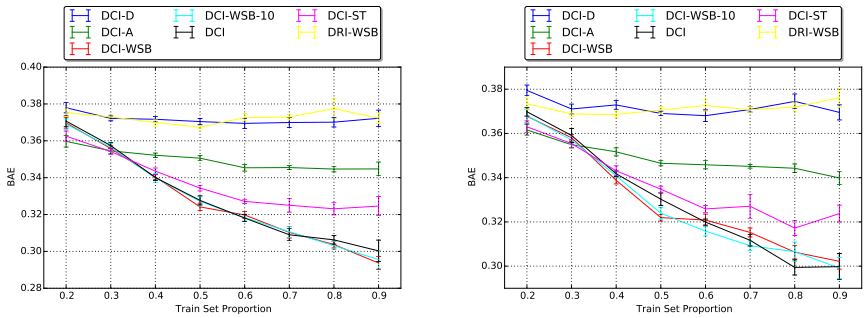
Figurea 5.1 show a comparison between DCI, DCI-WSB, DCI-ST, DCI-A, DCI-D, DCI-10-WSB, and DRI-WSB. DRI-WSB performs worst in all evaluations compared to all collective methods, which indicates that collective inference is improving performance. In all datasets, specifying an order and running DCI-A or 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 important to notice that it is not so clear to determine which is best among DCI-ST, and DCI-WSB. Therefore, we summarize results by reduction in error.

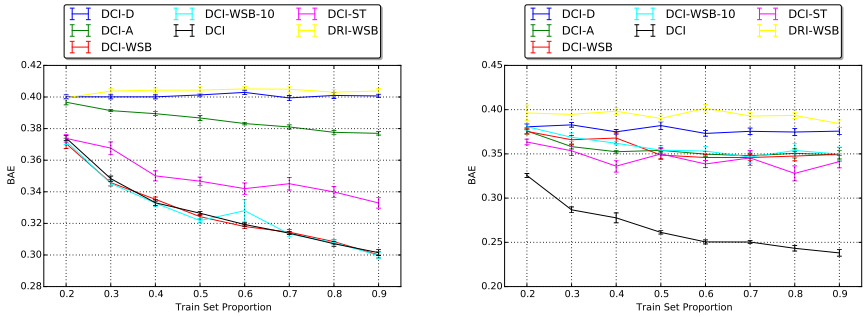
$gain(m_{DCI-WSB}, m_{DCI-ST}) = 0.0162357$ or about a 1.6% gain of DCI-WSB over DCI-ST.



(a) Facebook (b) IMDB



(c) Amazon_DVD_20000 (d) Amazon_DVD_7500



(e) Amazon_Music_64500 (f) Amazon_Music_7500

Fig. 5.1.: DCI compared to its variants on Facebook, IMDB, and Amazon datasets

5.2 Algorithmic Variants

5.2.1 Swapping vs Objective Function Balancing

We evaluate our two main variants or label distribution correction methods of DCI, swapping (DCI-S) and balanced cross entropy (DCI-B).

- DCI-WSB: Learned without data augmentations or cross entropy balancing (WSB)
- DCI-S: Learned only with data augmentations
- DCI-B: Learned only with balanced cross entropy
- DCI: The main DCI method which uses both data augmentations and balanced cross entropy

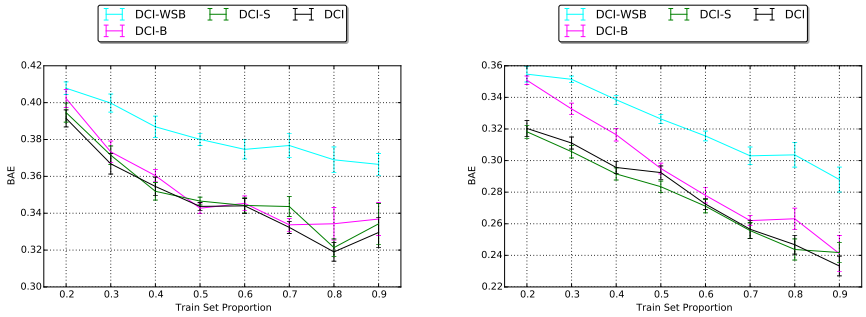
DCI and DCI-WSB are plotted for reference as well. For smaller datasts IMDB, swapping tends to perform better, while for Amazon_Music_7500, balanced cross entropy tends to perform better. For all other datasets, there doesn't seem to be noticeable differences between the two. Figures 5.2 and 5.3 show a comparison of DCI-S and DCI-B.

5.2.2 DCI vs RNCC Variants

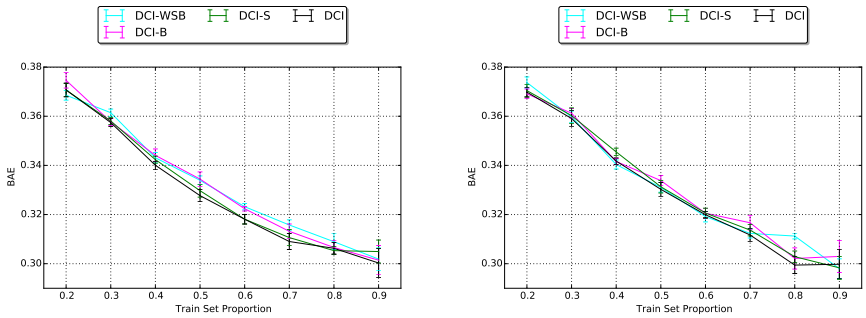
We compare DCI with label distribution correction methods to RNCC with the same methods.

- DCI-WSB: Learned without data augmentations or cross entropy balancing (WSB)
- DCI-S: Learned only with data augmentations
- DCI-B: Learned only with balanced cross entropy
- RNCC: The main RNCC method
- RNCC-S: Learned with data augmentations
- RNCC-B: Learned with balanced cross entropy

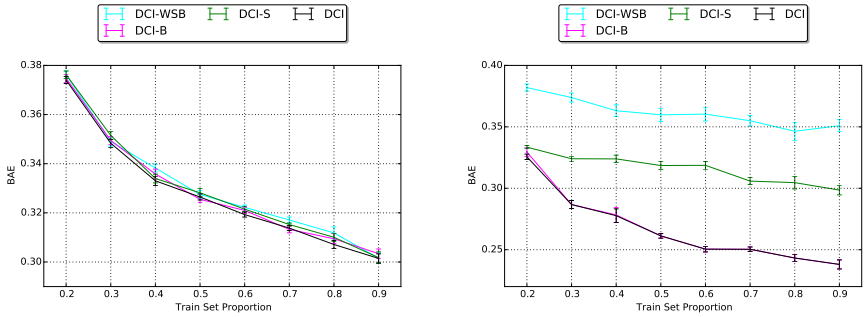
The RNCC variants tend to perform worse and sometimes the label distribution correction methods don't improve performance significantly. Figure 5.4 show a comparisons of the DCI and RNCC variants.



(a) Facebook (b) IMDB



(c) Amazon_DVD_20000 (d) Amazon_DVD_7500

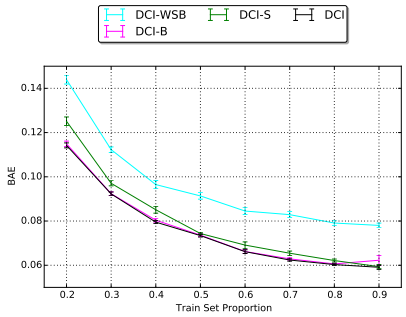


(e) Amazon_Music_64500 (f) Amazon_Music_7500

Fig. 5.2.: DCI-S and DCI-B compared to its variants on Facebook, IMDB, Amazon DVD, and Amazon Music datasets

5.2.3 DCI Initialization Variants

We evaluate random initialization variants of DCI to show that initialization predictions by DRI helps to improve performance. We evaluate the following variants



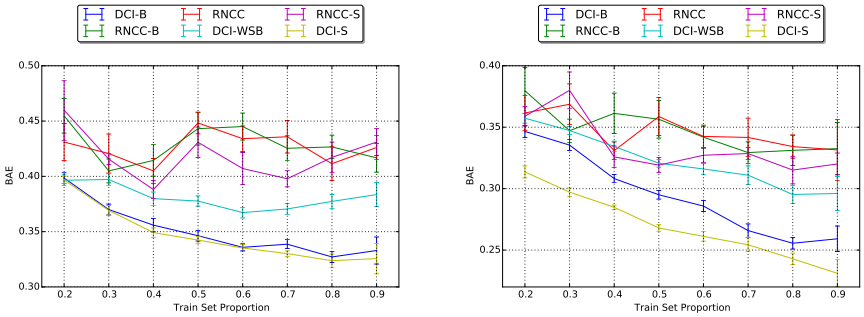
(a) Patents

Fig. 5.3.: DCI-S and DCI-B compared to its variants on the Patents dataset

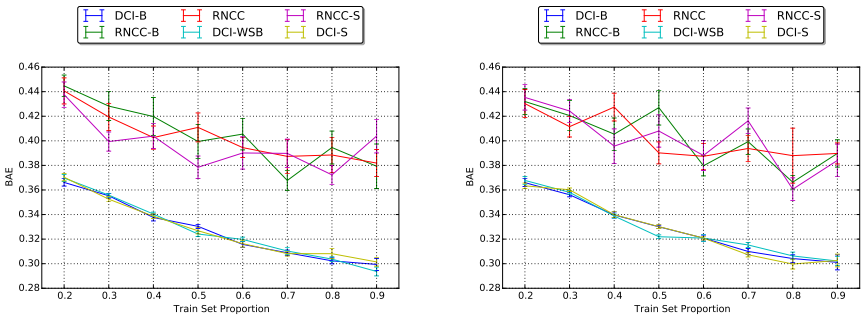
- DCI-WSB: Learned without data augmentations or cross entropy balancing (WSB)
- DCI-R: WSB version that uses randomly initialized prediction for the unlabeled set (to compare to default of making initial predictions with DRI)
- DCI-S: Learned only with data augmentations
- DCI-S-R: Swapping version with randomly initialized prediction for the unlabeled set
- DCI-B: Learned only with balanced cross entropy
- DCI-B-R: Balanced cross entropy version with randomly initialized prediction for the unlabeled set

Figure 5.5 show a comparison of random initialization vs DRI initializations.

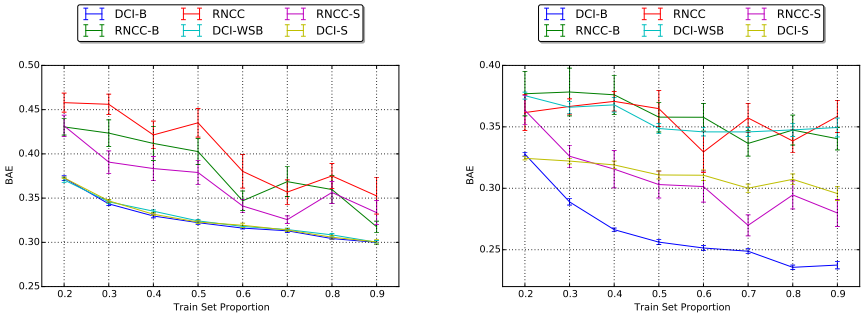
It is not so clear to determine if the random variants do as well. Therefore, we summarize results by reduction in error. $gain(m_{DCI-WSB}, m_{DCI-R}) = 0.0006495$ or about a 0.06% gain of DCI over DCI-R. $gain(m_{DCI-S}, m_{DCI-S-R}) = 0.0156055$ or about a 1.78% gain of DCI-S over DCI-S-R. $gain(m_{DCI-B}, m_{DCI-B-R}) = 0.0082923$ or about a 1.04% gain of DCI-B over DCI-B-R. The overall gain of these methods or average of these gains is 0.00818 or 0.82% gain. Therefore, we have considered the non-random initialization procedure as part of our main DCI algorithm.



(a) Facebook (b) IMDB



(c) Amazon_DVD_20000 (d) Amazon_DVD_7500

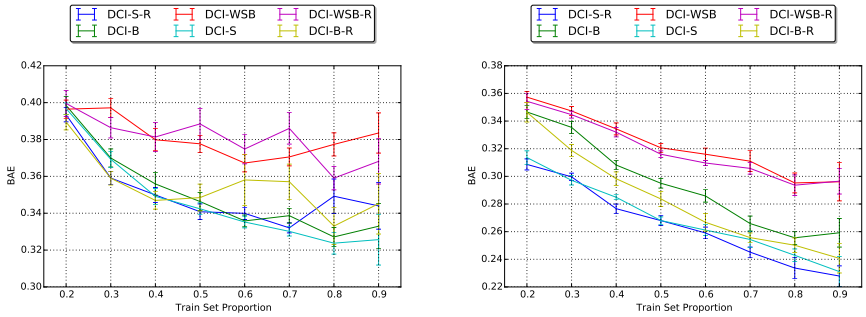


(e) Amazon_Music_64500 (f) Amazon_Music_7500

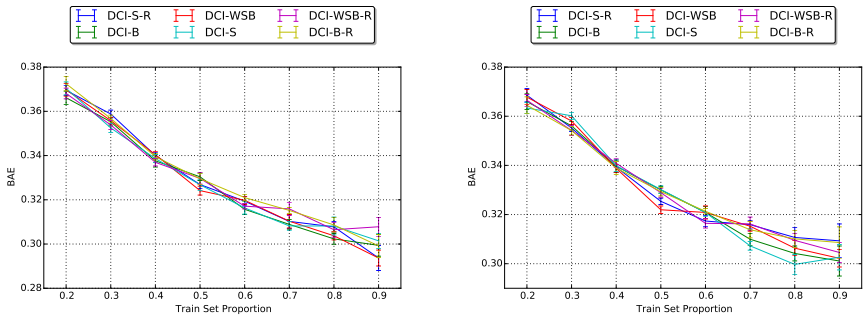
Fig. 5.4.: DCI variants compared to its variants as well as RNCC variants on Facebook, IMDB, Amazon DVD, and Amazon Music datasets

5.2.4 Utilizing Maximum Entropy

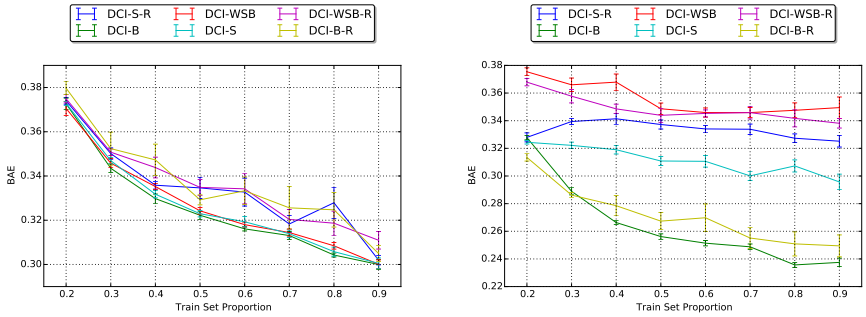
We evaluate utilizing the Maximum Entropy correction from [20]. Any version of DCI may have an appending MEF to denote the method utilized Maximum Entropy.



(a) Facebook (b) IMDB



(c) Amazon_DVD_20000 (d) Amazon_DVD_7500



(e) Amazon_Music_64500 (f) Amazon_Music_7500

Fig. 5.5.: DCI compared to its variants on Facebook, IMDB, and Amazon datasets

Given output probabilities for positive labels for each node, $Q_i(+)$, we calculate $z_i = log(\frac{Q_i(+)}{1-Q_i(+)})$, the pivot index, $\phi = argmin_r(\frac{\sum_{i=1}^{|} V_U|I(i \leq r)}{|V_U|} - P(-))^2$, and the updated probabilities $Q_i(+) := \sigma(z_i - z_\phi)$, where $P(-)$ is the proportion of negative labels, σ is the sigmoid function, and z_ϕ corresponds to the pivot index when z is sorted

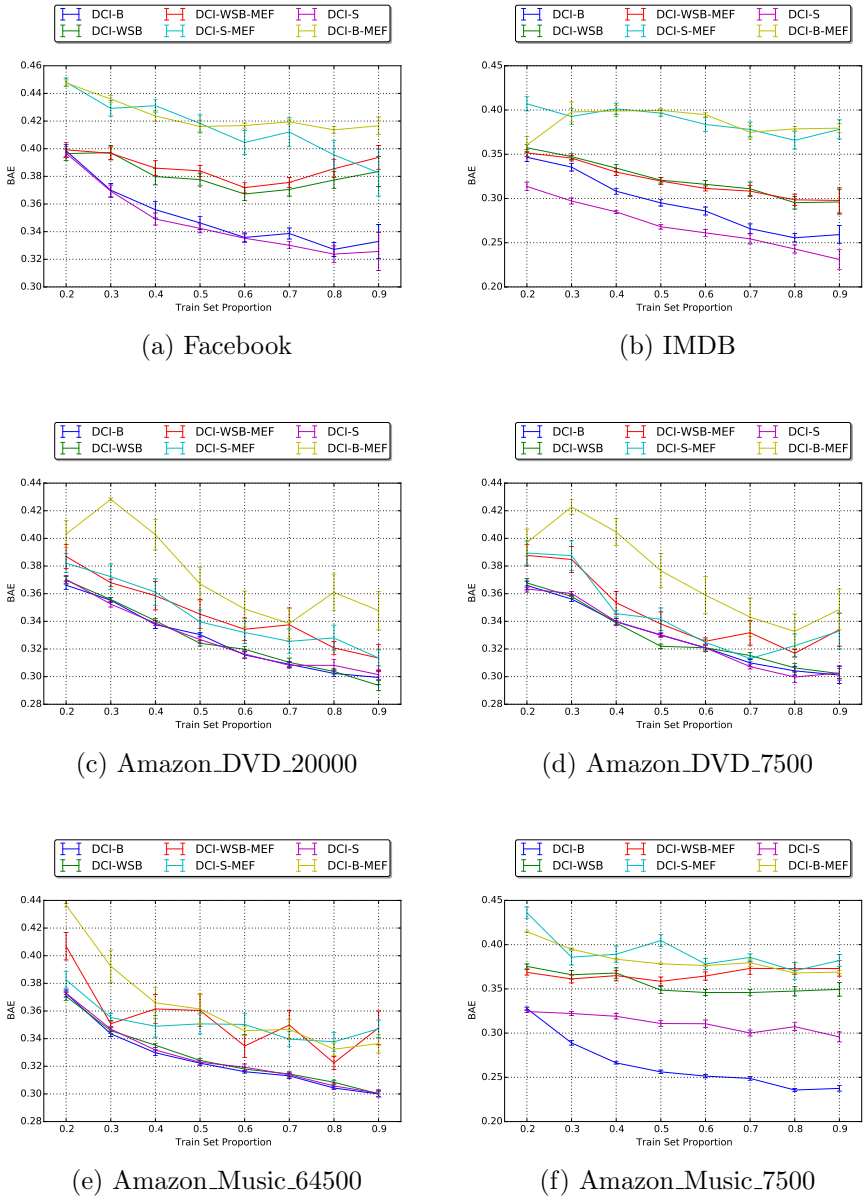
in ascending order. MaxEntInf allows for better calibrated probabilities in [20]. We compare and explore whether Maximum Entropy can improve DCI methods.

- DCI-WSB: Learned without data augmentations or cross entropy balancing (WSB)
- DCI-S: Learned only with data augmentations
- DCI-B: Learned only with balanced cross entropy
- DCI-WSB-MEF: Learned without data augmentations or cross entropy balancing (WSB) but performing the maximum entropy constraint on the final output probabilities.
- DCI-S-MEF: Learned only with data augmentations but performing the maximum entropy constraint on the final output probabilities.
- DCI-B-MEF: Learned only with balanced cross entropy but performing the maximum entropy constraint on the final output probabilities.

It appears as though DCI-WSB-MEF has some improvement over DCI-WSB on IMDB, Amazon_Music_7500, and Patents. However, DCI-WSB-MEF does not outperform DCI-S and DCI-B. In most cases, the MEF version of DCI-S and DCI-B appear to do worse. This is probably because DCI-S and DCI-B already perform a label distribution "correction." Figures 5.6 and 5.7 show a comparison of DCI variants and MEF variants.

5.2.5 Hidden State Propagation

In each collective inference step, we obtain the last hidden state for the LSTM for each node. We then experiment with propagating these hidden states in collective inference similar to how propagation of predictions is done. Each node has a hidden vector representation that is also concatenated to each attribute vector. Since hidden



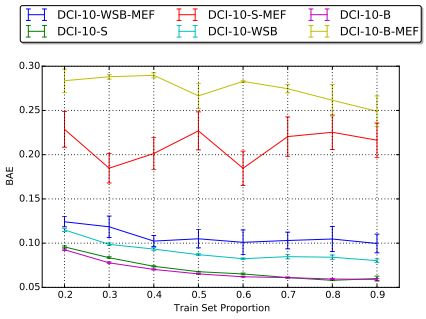
(a) Facebook (b) IMDB

(c) Amazon_DVD_20000 (d) Amazon_DVD_7500

(e) Amazon_Music_64500 (f) Amazon_Music_7500

Fig. 5.6.: DCI variants compared to the Maximum Entropy variants on Facebook, IMDB, Amazon DVD, and Amazon Music datasets

layer size is 10, the corresponding hidden state for each node in prediction is also of size 10. We evaluate two variants, DCI-W and DCI-WR. Both do not perform any label distribution correction mechanisms.



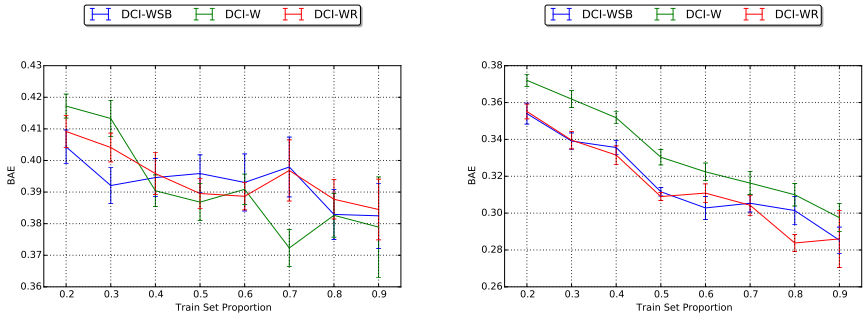
(a) Patents

Fig. 5.7.: DCI variants compared to the Maximum Entropy variants on the Patents dataset

- DCI-WSB: Learned without data augmentations or cross entropy balancing (WSB)
- DCI-W: Has initial hidden states for the first iteration of collective inference set to DRI's last hidden states.
- DCI-WR: Sets the first hidden states for each node to a vector of all 0's.

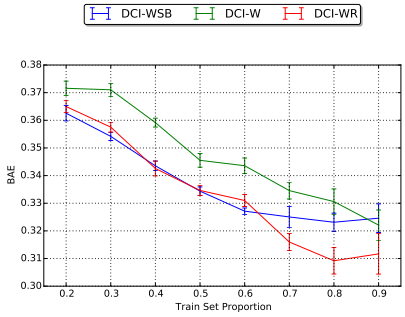
Figure 5.8 show a comparison of hidden state variants and DCI-WSB. The results indicate that DCI-WSB is significantly better than DCI-W; however, it is not so clear among DCI-WSB and DCI-WR. We calculate reduction in error here as well.

$gain(m_{DCI}, m_{DCI-W}) = 0.0333145$ or about a 3.3% gain of DCI over DCI-W. $gain(m_{DCI}, m_{DCI-WR}) = 0.0029219$ or about a 0.291% gain of DCI over DCI-WR.

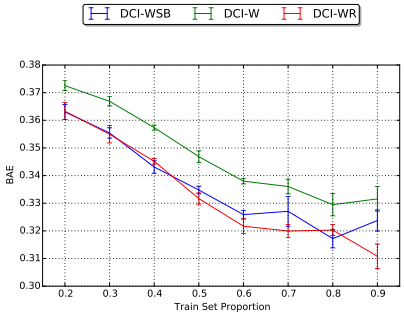


(a) Facebook

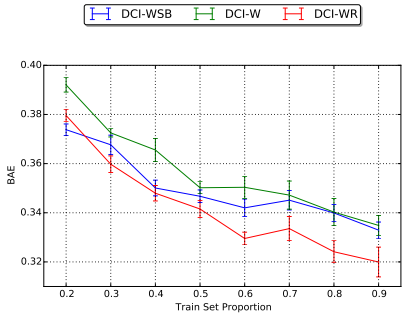
(b) IMDB



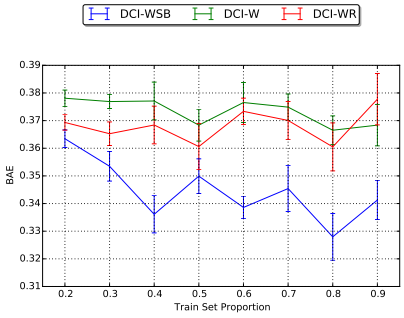
(c) Amazon_DVD_20000



(d) Amazon_DVD_7500



(e) Amazon_Music_64500



(f) Amazon_Music_7500

Fig. 5.8.: DCI variants compared to its hidden state propagation variants on Facebook, IMDB, and Amazon datasets

6. SUMMARY

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.

REFERENCES

REFERENCES

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

[2] L. Getoor and B. Taskar, Eds., *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[3] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *Artificial Intelligence Magazine*, vol. 29, no. 3, p. 93, 2008.

[4] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *Computing Research Repository*, vol. abs/1506.00019, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019

[5] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *Computing Research Repository*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[6] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks." in *Proceedings of the 31st International Conference on Machine Learning*, vol. 14, 2014, pp. 1764–1772.

[7] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *Computing Research Repository*, vol. abs/1411.4555, 2014. [Online]. Available: http://arxiv.org/abs/1411.4555

[8] M. Ren, R. Kiros, and R. S. Zemel, "Image question answering: A visual semantic embedding model and a new dataset," *Computing Research Repository*, vol. abs/1505.02074, 2015. [Online]. Available: http://arxiv.org/abs/1505.02074

[9] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering." in *Proceedings of the 28th Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, 2014, pp. 1293–1299.

[10] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 701–710.

[11] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 1067–1077.

[12] X. Li, N. Du, H. Li, K. Li, J. Gao, and A. Zhang, "A deep learning approach to link prediction in dynamic networks." in *Proceedings of the 2014 Society for Industrial and Applied Mathematics International Conference on Data Mining*, vol. 14, 2014, pp. 289–297. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611973440.33

[13] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21st International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1365–1374.

[14] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, 2016.

[15] B. Taskar, P. Abbeel, and D. Koller, "Discriminative probabilistic models for relational data," in *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, 2002, pp. 485–492.

[16] M. Richardson and P. Domingos, "Markov Logic Networks," *Machine Learning*, vol. 62, no. 1-2, pp. 107–136, Feb. 2006.

[17] J. Neville and D. Jensen, "Relational dependency networks," *Journal of Machine Learning Research*, vol. 8, pp. 653–692, May 2007.

[18] D. D. Monner and J. A. Reggia, "Recurrent neural collective classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 12, pp. 1932–1943, Dec 2013.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[20] J. J. Pfeiffer, III, J. Neville, and P. N. Bennett, "Overcoming relational learning biases to accurately predict preferences in large scale networks," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 853–863. [Online]. Available: http://doi.acm.org/10.1145/2736277.2741668

[21] Z. Kou and W. W. Cohen, "Stacked graphical models for efficient inference in markov random fields." in *In Proceedings of the 2007 Society for Industrial and Applied Mathematics International Conference on Data Mining*, 2007, pp. 533–538.

[22] S. M. Kazemi, D. Buchman, K. Kersting, S. Natarajan, and D. Poole, "Relational logistic regression," in *International Conference on Principles of Knowledge Representation and Reasoning*, 2014.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, 1986, pp. 318–362. [Online]. Available: http://dl.acm.org/citation.cfm?id=104279.104293

[24] F. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proceedings of the International Joint Conference on Neural networks*, vol. 3, 2000, pp. 189–194 vol.3.

[25] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Computing Research Repository*, vol. abs/1409.3215, 2014. [Online]. Available: http://arxiv.org/abs/1409.3215

[26] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1225–1234. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939753

[27] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1109/TNN.2008.2005605

[28] J. Atwood and D. Towsley, "Search-convolutional neural networks," *Computing Research Repository*, vol. abs/1511.02136, 2015. [Online]. Available: http://arxiv.org/abs/1511.02136

[29] D. Monner and J. A. Reggia, "A generalized LSTM-like training algorithm for second-order recurrent neural networks," *Neural Networks*, vol. 25, pp. 70–83, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2011.07.003

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[31] R. E. Bellman, *Dynamic Programming*. Dover Publications, Incorporated, 2003.

[32] J. Leskovec, "The dynamics of viral marketing," *Association for Computing Machinery Transactions on the Web*, p. 5, 2007.

[33] B. van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio, "Blocks and fuel: Frameworks for deep learning," *Computing Research Repository*, vol. abs/1506.00619, 2015. [Online]. Available: http://arxiv.org/abs/1506.00619

[34] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120. [Online]. Available: http://ilpubs.stanford.edu:8090/422/

[35] G. Jeh and J. Widom, "Scaling personalized web search," in *Proceedings of the 11th International Conference on World Wide Web*, 2002, pp. 271–279.