Neural Graph Learning: Training Neural Networks Using Graphs

Thang D. Bui*
University of Cambridge
Cambridge, UK
tdb40@cam.ac.uk

Sujith Ravi
Google Research
Mountain View, California, US
sravi@google.com

Vivek Ramavajjala
Google Research
Mountain View, California, US
vramavaj@google.com

ABSTRACT

Label propagation is a powerful and flexible semi-supervised learning technique on graphs. Neural networks, on the other hand, have proven track records in many supervised learning tasks. In this work, we propose a training framework with a graph-regularised objective, namely *Neural Graph Machines*, that can combine the power of neural networks and label propagation. This work generalises previous literature on graph-augmented training of neural networks, enabling it to be applied to multiple neural architectures (Feed-forward NNs, CNNs and LSTM RNNs) and a wide range of graphs. The new objective allows the neural networks to harness both labeled and unlabeled data by: (a) allowing the network to train using labeled data as in the supervised setting, (b) biasing the network to learn similar hidden representations for neighboring nodes on a graph, in the same vein as label propagation. Such architectures with the proposed objective can be trained efficiently using stochastic gradient descent and scaled to large graphs, with a runtime that is linear in the number of edges. The proposed joint training approach convincingly outperforms many existing methods on a wide range of tasks (multi-label classification on social graphs, news categorization, document classification and semantic intent classification), with multiple forms of graph inputs (including graphs with and without node-level features) and using different types of neural networks.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Semi-supervised learning settings*;

KEYWORDS

semi-supervised learning, neural network, graph

ACM Reference Format:
Thang D. Bui, Sujith Ravi, and Vivek Ramavajjala. 2018. Neural Graph Learning: Training Neural Networks Using Graphs. In *WSDM 2018: WSDM 2018: The Eleventh ACM International Conference on Web Search and Data Mining , February 5–9, 2018, Marina Del Rey, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3159652.3159731

*This work was done during an internship at Google.

WSDM 2018, February 5–9, 2018, Marina Del Rey, CA, USA

ACM ISBN 978-1-4503-5581-0/18/02.
https://doi.org/10.1145/3159652.3159731

1 INTRODUCTION

Semi-supervised learning is a powerful machine learning paradigm that can improve the prediction performance compared to techniques that use only labeled data, by leveraging a large amount of unlabeled data. The need of semi-supervised learning arises in many problems in computer vision, natural language processing or social networks, in which getting labeled datapoints is expensive or unlabeled data is abundant and readily available.

There exist a plethora of semi-supervised learning methods. The simplest one uses bootstrapping techniques to generate pseudo-labels for unlabeled data generated from a system trained on labeled data. However, this suffers from label error feedbacks [13]. In a similar vein, autoencoder based methods often need to rely on a two-stage approach: train an autoencoder using unlabeled data to generate an embedding mapping, and use the learnt embeddings for prediction. In practice, this procedure is often costly and inaccurate. Another example is transductive SVMs [8], which is too computationally expensive to be used for large datasets. Methods that are based on generative models and amortized variational inference [10] can work well for images and videos, but it is not immediately clear on how to extend such techniques to handle sparse and multi-modal inputs or graphs over the inputs.

In contrast to the methods above, graph-based techniques such as label propagation [4, 23] often provide a versatile, scalable, and yet effective solution to a wide range of problems. These methods construct a smooth graph over the unlabeled and labeled data. Graphs are also often a natural way to describe the relationships between nodes, such as similarities between embeddings, phrases or images, or connections between entities on the web or relations in a social network. Edges in the graph connect semantically similar nodes or datapoints, and if present, edge weights reflect how strong such similarities are. By providing a set of labeled nodes, such techniques iteratively refine the node labels by aggregating information from neighbours and propagate these labels to the nodes' neighbours. In practice, these methods often converge quickly and can be scaled to large datasets with a large label space [15]. We build upon the principle behind label propagation for our method.

Another key motivation of our work is the recent advances in neural networks and their performance on a wide variety of supervised learning tasks such as image and speech recognition or sequence-to-sequence learning [7, 12, 17]. Such results are however conditioned on training very large networks on large datasets, which may need millions of labeled training input-output pairs. This begs the question: can we harness previous state-of-the-art semi-supervised learning techniques, to jointly train neural networks using limited labeled data and unlabeled data to improve its performance?

Contributions: We propose a discriminative training objective for neural networks with graph augmentation, that can be trained with stochastic gradient descent and efficiently scaled to large graphs. The new objective has a regularization term for generic neural network architectures that enforces similarity between nodes in the graphs, which is inspired by the objective function of label propagation. In particular, we show that:

- Graph-augmented neural network training can work for a wide range of neural networks, such as feed-forward, convolutional and recurrent networks. Additionally, this technique can be used in both inductive and transductive settings. It also helps learning in low-sample regime (small number of labeled nodes), which cannot be handled by vanilla neural network training.
- The framework can handle multiple forms of graphs, either naturally given or constructed based on embeddings and knowledge bases.
- As a by-product, our proposed framework provides a simple technique to finding smaller and faster neural networks that offer competitive performance with larger and slower non graph-augmented alternatives (see section 4.2).

We experimentally show that the proposed training framework outperforms state-of-the-art or perform favourably on a variety of prediction tasks and datasets, involving text features and/or graph inputs and on many different neural network architectures (see section 4).

The paper is organized as follows: we first review some background and literature, and relate them to our approach in section 2; we then detail the training objective and its properties in section 3; and finally we validate our approach on a range of experiments in section 4.

2 BACKGROUND AND RELATED WORKS

In this section, we will lay out the groundwork for our proposed training objective in section 3.

2.1 Neural network learning

Neural networks are a class of non-linear mapping from inputs to outputs and comprised of multiple layers that can potentially learn useful representations for predicting the outputs. We will view various models such as feed-forward neural networks, recurrent neural networks and convolutional networks under the same umbrella. Given a set of N training input-output pairs $\{x_n, y_n\}_{n=1}^N$, such neural networks are often trained by performing maximum likelihood learning, that is, tuning their parameters so that the networks' outputs are close to the ground truth under some criterion,

$$C_{\text{NN}}(\theta) = \sum_n c(g_\theta(x_n), y_n), \tag{1}$$

where $g_\theta(\cdot)$ denotes the overall mapping, parameterized by θ, and $c(\cdot)$ denotes a loss function such as l-2 for regression or cross entropy for classification. The cost function c and the mapping g are typically differentiable w.r.t θ, which facilitates optimisation via gradient descent. Importantly, this can be scaled to a large number of training instances by employing stochastic training using minibatches of data. However, it is not clear how unlabeled data, if available, can be treated using this objective, or if extra information about the training set, such as relational structures can be used.

2.2 Graph-based semi-supervised learning

In this section, we provide a concise introduction to graph-based semi-supervised learning using *label propagation* and its training objective. Suppose we are given a graph $G = (V, E, W)$ where V is the set of nodes, E the set of edges and W the edge weight matrix. Let V_l, V_u be the labeled and unlabeled nodes in the graph. The goal is to predict a soft assignment of labels for each node in the graph, $\hat{Y}$, given the training label distribution for the seed nodes, Y. Mathematically, label propagation performs minimization of the following convex objective function, for L labels,

$$\begin{aligned} C_{\text{LP}}(\hat{Y}) = \mu_1 \sum_{v \in V_l} \left\| \hat{Y}_v - Y_v \right\|_2^2 \\ + \mu_2 \sum_{v \in V, u \in \mathcal{N}(v)} w_{u,v} \left\| \hat{Y}_v - \hat{Y}_u \right\|_2^2 \\ + \mu_3 \sum_{v \in V} \left\| \hat{Y}_v - U \right\|_2^2, \end{aligned} \tag{2}$$

subject to $\sum_{l=1}^L \hat{Y}_{vl} = 1$, where $\mathcal{N}(v)$ is the neighbour node set of the node v, and U is the prior distribution over all labels, $w_{u,v}$ is the edge weight between nodes u and v, and μ_1, μ_2, and μ_3 are hyperparameters that balance the contribution of individual terms in the objective. The terms in the objective function above encourage that: (a) the label distribution of seed nodes should be close to the ground truth, (b) the label distribution of neighbouring nodes should be similar, and, (c) if relevant, the label distribution should stay close to our prior belief. This objective function can be solved efficiently using iterative methods such as the Jacobi procedure. That is, in each step, each node aggregates the label distributions from its neighbours and adjusts its own distribution, which is then repeated until convergence. In practice, the iterative updates can be done in parallel or in a distributed fashion which then allows large graphs with a large number of nodes and labels to be trained efficiently. [4] and [15] provide good surveys on the topic for interested readers.

There are many variants of label propagation that could be viewed as optimising modified versions of eq. (2), and in essence balancing the *smoothness* constraint and the *fitting* constraint [22]. For example, manifold regularization [3] replaces the label distribution $\hat{Y}$ by a Reproducing Kernel Hilbert Space mapping from input features. Similarly, [18] also employs such mapping but uses a feed-forward neural network instead. Both methods can be classified as inductive learning algorithms; whereas the original label propagation algorithm is transductive [19].

These aforementioned methods are closest to our proposed approach; however, there are key differences. Our work generalizes previously proposed frameworks for graph-augmented training of neural networks (e.g., [18]) and extends it to new settings, for example, when there is only graph input and no features are available. Unlike the previous works, we show that the graph augmented training method can work with multiple neural network architectures (Feed-forward NNs, CNNs, RNNs) and on multiple prediction tasks and datasets using *natural* as well as *constructed* graphs. The

experiment results (see section 4) clearly validate the effectiveness of this method in all these different settings, in both inductive and transductive learning paradigms. Besides the methodology, our study also presents an important contribution towards assessing the effectiveness of graph combined neural networks as a generic training mechanism for different architectures and problems, which was not well studied in previous work.

More recently, graph embedding techniques have been used to create node embedding that encode local structures of the graph and the provided node labels [14, 19]. These techniques target learning better node representations to be used for other tasks such as node classification. In this work, we aim to directly learn better predictive models from the graph. We compare our method to these two-stage (embedding + classifier) techniques in several experiments in section 4

Our work is also different and orthogonal to recent works on using neural networks *on* graphs, for example: [5, 11] employ spectral graph convolution to create a neural-network like classifier. However, these approaches requires many approximations to arrive at a practical implementation. Here, we advocate a training objective that *uses* graphs to augment neural network learning, and works with many forms of graphs and with any type of neural network.

3 NEURAL GRAPH MACHINES

In this section, we devise a discriminative training objective for neural networks, that is inspired bnormy the label propagation objective and uses both labeled and unlabeled data, and can be trained by stochastic gradient descent.

First, we take a close look at the two objective functions discussed in section 2. The label propagation objective (eq. (2)) ensures the predicted label distributions of neighbouring nodes to be similar, while those of labeled nodes to be close to the ground truth. For example: if a *cat* image and a *dog* image are strongly connected in a graph, and if the *cat* node is labeled as *animal*, the predicted probability of the *dog* node being *animal* is also high. In contrast, the neural network training objective (eq. (1)) only takes into account the labeled instances, and ensure correct predictions on the training set. As a consequence, a neural network trained on the *cat* image alone will not make an accurate prediction on the *dog* image.

Such shortcoming of neural network training can be rectified by biasing the network using prior knowledge about the relationship between instances in the dataset. In particular, for the domains we are interested in, training instances (either labeled or unlabeled) that are connected in a graph, for example, *dog* and *cat* in the above example, should have similar predictions. This can be done by encouraging neighboring data points to have a similar hidden representation learnt by a neural network, resulting in a modified objective function for training neural network architectures using both labeled and unlabeled datapoints. We call architectures trained using this objective *Neural Graph Machines* (NGM), and schematically illustrate the concept in figure 1. The proposed objective function is a weighted sum of the neural network cost and the label propagation cost as follows,

$$\begin{aligned} C_{\text{NGM}}(\theta) = \alpha_1 & \sum_{(u,v)\in\mathcal{E}_{LL}} w_{uv} d(h_\theta(x_u), h_\theta(x_v)) \\ & + \alpha_2 \sum_{(u,v)\in\mathcal{E}_{LU}} w_{uv} d(h_\theta(x_u), h_\theta(x_v)) \\ & + \alpha_3 \sum_{(u,v)\in\mathcal{E}_{UU}} w_{uv} d(h_\theta(x_u), h_\theta(x_v) \\ & + \sum_{n=1}^{V_l} c(g_\theta(x_n), y_n), \end{aligned} \tag{3}$$

where $\mathcal{E}_{LL}$, $\mathcal{E}_{LU}$, and $\mathcal{E}_{UU}$ are sets of labeled-labeled, labeled-unlabeled and unlabeled-unlabeled edges correspondingly, $h(\cdot)$ represents the hidden representations of the inputs produced by the neural network, and $d(\cdot)$ is a distance metric, and $\{\alpha_1, \alpha_2, \alpha_3\}$ are hyperparameters. Note that we have separated the terms based on the edge types, as these can affect the training differently.

Our framework is general so that one can plug in either the hidden representations at any intermediate layer or the estimated soft label vector at the final layer. However, similar to any neural network regularisation scheme, it is not obvious what strategy works best in general. For example, forcing bottom layers (closer to the inputs) to be similar would have a stronger regularisation effect, and vice versa. In practice, we choose an l-1 or l-2 distance metric for $d(\cdot)$, and $h(x)$ to be the last hidden layer of the neural network, or a cross-entropy cost for the predicted label vector.

3.1 Connections to previous methods

The graph-dependent α hyperparameters control the balance of the contributions of different edge types. When $\{\alpha_i = 0\}_{i=1}^{3}$, the proposed objective ignores the similarity constraint and becomes a supervised-only neural network objective as in eq. (1). When only $\alpha_1 \neq 0$, the training cost has an additional term for labeled nodes, that acts as a regularizer. When $g_\theta(x) = h_\theta(x) = \hat{y}$, where $\hat{y}$ is the label distribution, the individual cost functions (c and d) are squared l-2 norm, and the objective is trained using $\hat{y}$ directly instead of θ, we arrive at the label propagation objective in eq. (2). Therefore, the proposed objective could be thought of as a *non-linear* version of the label propagation objective, and a *graph-regularized* version of the neural network training objective.

3.2 Network inputs and graph construction

Similar to graph-based label propagation, the choice of the input graphs is critical, to correctly bias the neural network's prediction. Depending on the type of the graphs and nodes in the graph, they can be readily available to use such as social networks or protein linking networks, or they can be constructed (a) using generic graphs such as Knowledge Bases, that consists of relationship links between entities, (b) using embeddings learnt by an unsupervised learning technique, or, (c) using sparse feature representations for each vertex. Additionally, the proposed training objective can be easily modified for directed graphs.

We have discussed using node-level features as inputs to the neural network. In the absence of such inputs, our training scheme can still be deployed using input features derived from the graph

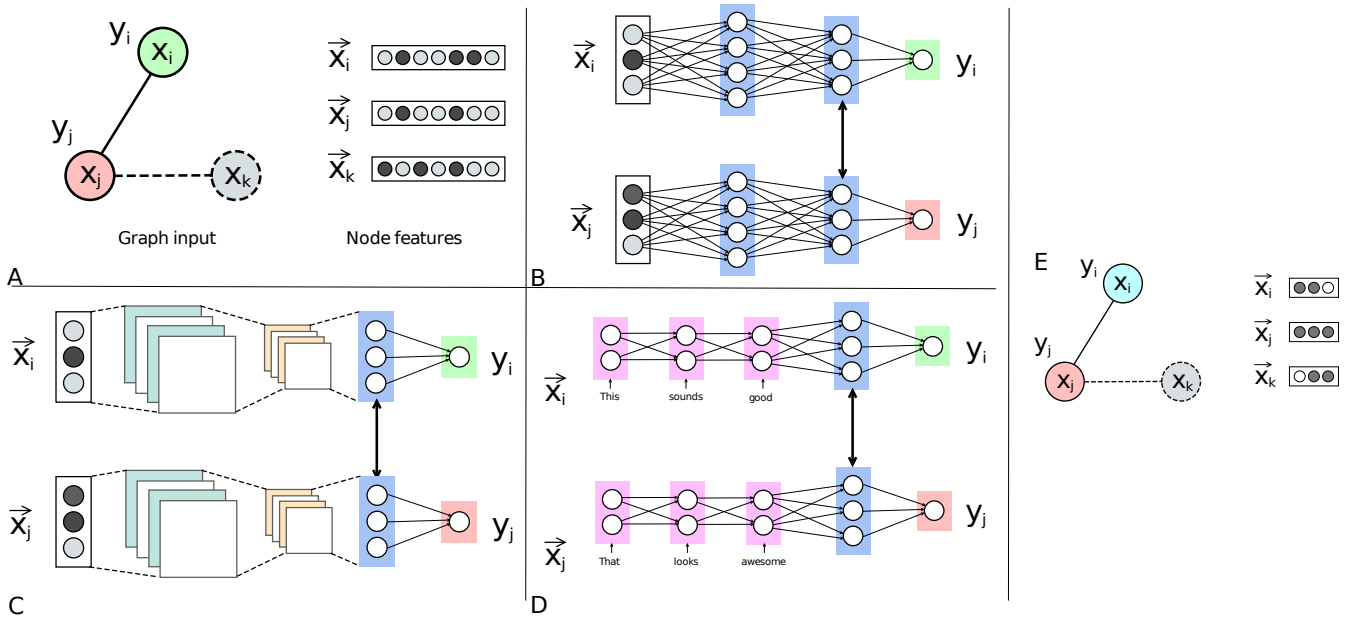

Figure 1: A: An example of a graph and feature inputs. In this case, there are two labeled nodes (x_i, x_j) and one unlabeled node (x_k), and two edges. The feature vectors, one for each node, are used as neural network inputs. B, C and D: Illustration of Neural Graph Machine for feed-forward, convolution and recurrent networks respectively: the training flow ensures the neural net to make accurate node-level predictions and biases the hidden representations/embeddings of neighbouring nodes to be similar. In this example, we force h_i and h_j to be similar as there is an edge connecting x_i and x_j nodes. E: Illustration of how we can construct inputs to the neural network using the adjacency matrix. In this example, we have three nodes and two edges. The feature vector created for each node *(shown on the right)* **has 1's at its index and indices of nodes that it's adjacent to.**

itself. We show in figure 1 and in experiments that the neighbourhood information such as rows in the adjacency matrix are simple to construct, yet powerful inputs to the network. These features can also be combined with existing features.

When the number of graph nodes is high, this construction can have a high complexity and result in a large number of input features. This can be avoided by several ways: (i) clustering the nodes and using the cluster assignments and similarities, (ii) learning an embedding function of nodes [14], or (iii) sampling the neighbourhood/context [19]. In practice, we observe that the input space can be bounded by a constant, even for massive graphs, with efficient scalable methods like unsupervised propagation (i.e., propagating node identity labels across the graph and selecting ones with highest support as input features to neural graph machines).

3.3 Optimization

The proposed objective function in eq. (3) has several summations over the labeled points and edges, and can be equivalently written as follows,

$$\begin{aligned} C_{\text{NGM}}(\theta) = &\sum_{(u,v)\in\mathcal{E}_{LL}} [\alpha_1 w_{uv} d(h_\theta(x_u), h_\theta(x_v)) + c_{uv}] \\ &+ \sum_{(u,v)\in\mathcal{E}_{LU}} [\alpha_2 w_{uv} d(h_\theta(x_u), h_\theta(x_v)) + c_u] \\ &+ \sum_{(u,v)\in\mathcal{E}_{UU}} \alpha_3 w_{uv} d(h_\theta(x_u), h_\theta(x_v)), \end{aligned} \tag{4}$$

where

$$c_{uv} = \frac{1}{|u|} c(g_\theta(x_u), y_u) + \frac{1}{|v|} c(g_\theta(x_v), y_v); \quad c_u = \frac{1}{|u|} c(g_\theta(x_u), y_u),$$

$|u|$ and $|v|$ are the number of edges incident to vertices u and v, respectively. The objective in its new form enables stochastic training to be deployed by sampling edges. In particular, in each training iteration, we use a minibatch of edges and obtain the stochastic gradients of the objective. To further reduce noise and speedup learning, we sample edges within a neighbourhood region, that is to make sure some sampled edges have shared end nodes.

3.4 Complexity

The complexity of each epoch in training using eq. (4) is $O(M)$ where $M = |\mathcal{E}|$ is the number of edges in the graph. In the case where there is a large number of unlabeled-unlabeled edges, they potentially do not help learning and could be ignored, leading to a lower complexity. One strategy to include them is self-training, that is to grow seeds or labeled nodes as we train the networks. We experimentally demonstrate this technique in section 4.4. Predictions at inference time can be made at the same cost as that of vanilla neural networks.

4 EXPERIMENTS

In this section, we provide several experiments showing the efficacy of the proposed training objective on a wide range of tasks, datasets and network architectures. All the experiments are done using a

TensorFlow implementation [1]. Models were trained using multiple runs, each experiment was run for a fixed number of time steps and batch size (details described in each section). The observed variance across runs wrt accuracy was small, around ±0.1% Note that we did not perform any cross-validation to select the regularisation loss or which hidden layers to compare. As such, we expect even better results than ones presented below if there is a more careful selection in place.

4.1 Multi-label Classification of Nodes on Graphs

We first demonstrate our approach using a multi-label classification problem on nodes in a relationship graph. In particular, the *BlogCatalog* dataset [2], a network of social relationships between bloggers is considered. This graph has 10,312 nodes, 333,983 edges and 39 labels per node, which represent the bloggers, their social connections and the bloggers' interests, respectively. Following previous approaches in the literature [2, 6], we train and make predictions using multiple one-vs-rest classifiers.

Since there are no provided features for each node, we use the rows of the adjacency matrix as input features, as discussed in section 3.2 in the main text. Feed-forward neural networks (FFNNs) with one hidden layer of 50 units are employed to map the constructed inputs to the node labels. As we use the test set to construct the graph and augment the training objective, the training in this experiment is transductive. Critically, to combat the unbalanced training set, we employ weighted sampling during training, i.e. making sure each minibatch has both positive and negative examples. In this experiment, we fix α_i to be equal, and experiment with $\alpha = 0.1$ and use the l-2 metric to compute the distance d between the hidden representations of the networks. In addition, we create a range of train/test splits by varying the number of training points being presented to the networks.

We compare our method (NGM-FFNN) against a two-stage approach that first uses node2vec [6] to generate node embeddings and then uses a linear one-vs-rest classifier for classification. The methods are evaluated using two metrics Macro F1 and Micro F1. The average results for different train/test splits using our method and the baseline are included in table 1. In addition, we compare NGM-FFNN with a non-augmented FFNN in which $\alpha = 0$, i.e. no edge information is used during training. We observe that the graph-augmented training scheme performs better (6% relative improvement on Macro F1 when the training set size is 20% and 50% of the dataset) or comparatively (when the training size is 80%) compared to the vanilla neural networks trained with no edge information. Both methods significantly outperform the approach that uses node embeddings and linear classifiers. We observe the same improvement over node2vec on the Micro F1 metric and NGM-FFNN is comparable to vanilla FFNN ($\alpha = 0$) but outperforms other methods on the recall metric.

These results demonstrate that using the graph itself as direct inputs to the neural network and letting the network figure out a non-linear mapping directly from the raw graph is more effective than the two-stage approach considered. More importantly, the results also show that using the graph information improves the performance in the limited data regime (for example: when training set is only 20% or 50% of the dataset).

[1]These results are different compared to [6], since we treat the classifiers (one per label) independently. Both methods shown here use the exact same setting and training/test data splits.

Table 1: Macro F1 results for BlogCatalog dataset averaged over 10 random splits. The higher is better. Graph regularized neural networks outperform node2vec embedding and a linear classifer in all training size settings.

|Train| / |Dataset|	NGM-FFNN	node2vec[1]
20%	**0.191**	0.168
50%	**0.242**	0.174
80%	**0.262**	0.177

4.2 Text Classification using Character-level CNNs

We evaluate the proposed objective function on a multi-class text classification task using a character-level convolutional neural network (CNN). We use the AG news dataset from [21], where the task is to classify a news article into one of 4 categories. Each category has 30,000 examples for training and 1,900 examples for testing. In addition to the train and test sets, there are 111,469 examples that are treated as unlabeled examples.

As there is no provided graph structure linking the articles, we create such a graph based on the embeddings of the articles. We restrict the graph construction to only the train set and the unlabeled examples and keep the test set only for evaluation. We use the Google News word2vec corpus to calculate the average embedding for each news article and use the cosine similarity of document embeddings as a similarity metric. Each node is restricted to have a maximum of 5 neighbors.

We construct the CNN in the same way as [21] and pick their competitive *"small CNN"* as our baseline for a more reasonable comparison to our set-up. Our approach employs the same network, but with significantly smaller number of convolutional layers and layer sizes, as shown in table 2.

Table 2: Settings of CNNs for the text classification experiment, including the number of convolutional layers and their sizes. The baseline model is the *small CNN* from [21] and is significantly larger than our model.

Setting	Baseline	Our "tiny CNN"
# of conv. layers	6	3
Frame size in conv. layers	256	32
# of FC layers	3	3
Hidden units in FC layers	1024	256

The networks are trained with the same hyper-parameters as reported in [21]. We observed that the model converged within 20 epochs (the model loss did not change much) and hence used this as a stopping criterion for this task. Experiments also showed that

running the network for longer also did not change the qualitative performance. We use the cross entropy loss on the final outputs of the network, that is $d = \text{cross\_entropy}(g(x_u), g(x_v))$, to compute the distance between nodes on an edge. In addition, we also experiment with a data augmentation technique using an English thesaurus, as done in [21].

We compare the "tiny CNN" trained using the proposed objective function with the baseline using the accuracy on the test set in table 3. Our approach outperforms the baseline by provides a 1.8% absolute and 2.1% relative improvement in accuracy, despite using a much smaller network. In addition, our model with graph augmentation trains much faster and produces results on par or better than the performance of a significantly larger network, *"large CNN"* [21], which has an accuracy of 87.18 without using a thesaurus, and 86.61 with the thesaurus.

Table 3: Results for news article categorization using character-level CNNs. Our method gives better predictive accuracy, despite using a much smaller CNN compared to the "small CNN" baseline from [21]‡.

Network	Accuracy %
Baseline‡	84.35
Baseline with thesaurus augmentation‡	85.20
Our "tiny" CNN	85.07
Our "tiny" CNN with NGM	**86.90**

4.3 Semantic Intent Classification using LSTM RNNs

We compare the performance of our approach for training RNN sequence models (LSTM) for a semantic intent classification task as described in the recent work on SmartReply [9] for automatically generating short email responses. One of the underlying tasks in SmartReply is to discover and map short response messages to semantic intent clusters.[2] We choose 20 intent classes and created a dataset comprised of 5,483 samples (3,832 for training, 560 for validation and 1,091 for testing). Each sample instance corresponds to a short response message text paired with a semantic intent category that was manually verified by human annotators. For example, *"That sounds awesome!"* and *"Sounds fabulous"* belong to the *sounds good* intent cluster.

We construct a sparse graph in a similar manner as the news categorization task using word2vec embeddings over the message text and computing similarity to generate a response message graph with fixed node degree (k=10). We use l-2 for the distance metric $d(\cdot)$ and choose α based on the development set.

We run the experiments for a fixed number of time steps and pick the best results on the development set. A multilayer LSTM architecture (2 layers, 100 dimensions) is used for the RNN sequence model. The LSTM model and its NGM variant are also compared against other baseline systems—*Random* baseline ranks the intent categories randomly and *Frequency* baseline ranks them in order of their frequency in the training corpus. To evaluate the intent prediction quality of different approaches, for each test instance, we compute the rank of the actual intent category rank_i with respect to the ranking produced by the method and use this to calculate the Mean Reciprocal Rank: $\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{rank}_i}$ We show in table 4 that LSTM RNNs with our proposed graph-augmented training objective function outperform standard baselines by achieving a better MRR.

[2]For details regarding SmartReply and how the semantic intent clusters are generated, refer [9].

Table 4: Results for Semantic Intent Classification using graph-augmented LSTM RNNs and baselines. Higher MRR is better.

Model	Mean Reciprocal Rank (MRR)
Random	0.175
Frequency	0.258
LSTM	0.276
NGM-LSTM	**0.284**

4.4 Low-supervision Document Classification

Finally, we compare our method on a task with very limited supervision—the PubMed document classification problem [16]. The task is to classify each document into one of 3 classes, with each document being described by a TF-IDF weighted word vector. The graph is available as a citation network: two documents are connected to each other if one cites the other. The graph has 19,717 nodes and 44,338 edges, with each class having 20 seed nodes and 1000 test nodes. In our experiments we exclude the test nodes from the graph entirely, training only on the labeled and unlabeled nodes.

We train a feed-forward neural network (FFNN) with two hidden layers with 250 and 100 neurons, using the l-2 distance metric on the last hidden layer. The NGM-FFNN model is trained with $\alpha_i = 0.2$, while the baseline FFNN is trained with $\alpha_i = 0$ (i.e., a supervised-only model). We use self-training to train the model, starting with just the 60 seed nodes (20 per class) as training data. The amount of training data is iteratively increased by assigning labels to the immediate neighbors of the labeled nodes and retraining the model. For the self-trained NGM-FFNN model, this strategy results in incrementally growing the neighborhood and thereby, *LL* and *LU* edges in equation 4 objective.

We compare the final NGM-FFNN model against the FFNN baseline and other techniques reported in [19] including the Planetoid models [19], semi-supervised embedding [18], manifold regression [3], transductive SVM [8], label propagation [24], graph embeddings [14] and a linear softmax model. Full results are included in table 5. The results show that the NGM model (without any tuning) outperforms many baselines including FFNN, semi-supervised embedding, manifold regularization and Planetoid-G/Planetoid-T, and compares favorably to Planetoid-I. Most importantly, this result demonstrates the graph augmentation scheme can lead to better regularised neural networks, especially in low sample regime (20 samples per class in this case). We believe that with tuning, NGM accuracy can be improved even further.

Table 5: Results for document classification on the PubMed dataset using neural networks. The top results are taken from [19]. The bottom two rows are ours, with the NGM training outperforming all other baselines, except Planetoid-I. Please see text for relevant references.

Method	Accuracy
Linear + Softmax	0.698
Semi-supervised embedding	0.711
Manifold regularization	0.707
Transductive SVM	0.622
Label propagation	0.630
Graph embedding	0.653
Planetoid-I	0.772
Planetoid-G	0.664
Planetoid-T	0.757
Feed-forward NN	0.709
NGM-FFNN	0.759

5 CONCLUSIONS

We have revisited graph-augmentation training of neural networks and proposed Neural Graph Machines as a general framework for doing so. Its objective function encourages the neural networks to make accurate node-level predictions, as in vanilla neural network training, as well as constrains the networks to learn similar hidden representations for nodes connected by an edge in the graph. Importantly, the objective can be trained by stochastic gradient descent and scaled to large graphs.

We validated the efficacy of the graph-augmented objective on various tasks including bloggers' interest, text category and semantic intent classification problems, using a wide range of neural network architectures (FFNNs, CNNs and LSTM RNNs). The experimental results demonstrated that graph-augmented training almost always helps to find better neural networks that outperforms other techniques in predictive performance or even much smaller networks that are faster and easier to train. Additionally, the node-level input features can be combined with graph features as inputs to the neural networks. We showed that a neural network that simply takes the adjacency matrix of a graph and produces node labels, can perform better than a recently proposed two-stage approach using sophisticated graph embeddings and a linear classifier. Our framework also excels when the neural network is small, or when there is limited supervision available. We note that though overall complexity is linear in the number of edges in the graph, in practice NGM is more robust compared to the standard training method (without regularisation) and that it can converge to a better solution given a fixed time budget. We attribute this effect to the graph structure used for optimization within each mini-batch rather than individual training examples in baseline networks.

While our objective can be applied to multiple graphs which come from different domains, we have not fully explored this aspect and leave this as future work. We expect the domain-specific networks can interact with the graphs to determine the importance of each domain/graph source in prediction. We also did not explore using graph regularisation for different hidden layers of the neural networks; we expect this is key for the multi-graph transfer setting [20]. Another possible future extension is to use our objective on directed graphs, that is to control the direction of influence between nodes during training.

ACKNOWLEDGMENTS

The authors would like to thank the Google Expander team for insightful feedback.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[2] Nitin Agarwal, Huan Liu, Sudheendra Murthy, Arunabha Sen, and Xufei Wang. 2009. A Social Identity Approach to Identify Familiar Strangers in a Social Network.. In *3rd International AAAI Conference on Weblogs and Social Media (ICWSM09)*.

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

[4] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. 2006. Label propagation and quadratic criterion. In *Semi-supervised learning*, O Chapelle, B Scholkopf, and A Zien (Eds.). MIT Press, 193–216.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3837–3845.

[6] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. 855–864.

[7] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[8] Thorsten Joachims. 1999. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning*.

[9] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young, and Vivek Ramavajjala. 2016. Smart Reply: Automated Response Suggestion for Email. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[10] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. 2014. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*. 3581–3589.

[11] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907* (2016).

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

[13] Dong-Hyun Lee. 2013. Pseudo-Label: The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. In *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*.

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

[15] Sujith Ravi and Qiming Diao. 2016. Large Scale Distributed Semi-Supervised Learning Using Streaming Approximation. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. 519–528.

[16] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93.

[17] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*. 3104–3112.

[18] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. 2012. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*. Springer, 639–655.
[19] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proceedings of The 33rd International Conference on Machine Learning*. 40–48.
[20] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in Neural Information Processing Systems*. 3320–3328.
[21] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*. 649–657.
[22] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*. 321–328.
[23] Xiaojin Zhu and Zoubin Ghahramani. [n. d.]. *Learning from labeled and unlabeled data with label propagation*. Technical Report. School of Computer Science, Canegie Mellon University.
[24] X Zhu, Z Ghahramani, and J Lafferty. 2003. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning (ICML-2003) Volume 2*, Vol. 2. AIAA Press, 912–919.