SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

ABSTRACT

We present a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. We motivate the choice of our convolutional architecture via a localized first-order approximation of spectral graph convolutions. Our model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. In a number of experiments on citation networks and on a knowledge graph dataset we demonstrate that our approach outperforms related methods by a significant margin.

1 INTRODUCTION

We consider the problem of classifying nodes (such as documents) in a graph (such as a citation network), where labels are only available for a small subset of nodes. This problem can be framed as graph-based semi-supervised learning, where label information is smoothed over the graph via some form of explicit graph-based regularization (Zhu et al., 2003; Zhou et al., 2004; Belkin et al., 2006; Weston et al., 2012), e.g. by using a graph Laplacian regularization term in the loss function:

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}} , \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2 = f(X)^\top \Delta f(X) . \tag{1}$$

Here, $\mathcal{L}_0$ denotes the supervised loss w.r.t. the labeled part of the graph, $f(\cdot)$ can be a neural network-like differentiable function, λ is a weighing factor and X is a matrix of node feature vectors X_i. $\Delta = D - A$ denotes the unnormalized graph Laplacian of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$, an adjacency matrix $A \in \mathbb{R}^{N \times N}$ (binary or weighted) and a degree matrix $D_{ii} = \sum_j A_{ij}$. The formulation of Eq. 1 relies on the assumption that connected nodes in the graph are likely to share the same label. This assumption, however, might restrict modeling capacity, as graph edges need not necessarily encode node similarity, but could contain additional information.

In this work, we encode the graph structure directly using a neural network model $f(X, A)$ and train on a supervised target $\mathcal{L}_0$ for all nodes with labels, thereby avoiding explicit graph-based regularization in the loss function. Conditioning $f(\cdot)$ on the adjacency matrix of the graph will allow the model to distribute gradient information from the supervised loss $\mathcal{L}_0$ and will enable it to learn representations of nodes both with and without labels.

Our contributions are two-fold. Firstly, we introduce a localized and well-behaved propagation rule for graph convolutional neural networks (Bruna et al., 2014; Duvenaud et al., 2015; Niepert et al., 2016; Defferrard et al., 2016) and show how it can be motivated from a first-order approximation of spectral convolutions on graphs (Hammond et al., 2011). Secondly, we show how this form of a graph convolutional neural network can be used for fast and scalable semi-supervised classification of nodes in a graph. Experiments on a number of datasets demonstrate that our model compares favorably both in classification accuracy and efficiency (measured in wall-clock time) against state-of-the-art methods for semi-supervised learning.

2 FAST APPROXIMATE CONVOLUTIONS ON GRAPHS

In this section, we provide theoretical motivation for a specific graph-based neural network model $f(X, A)$ that we will use in the rest of this paper. We choose this model to be a multi-layer Graph Convolutional Network (GCN) with the following propagation rule:

$$H^l = \sigma\Big(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l-1}W^l\Big) \,. \tag{2}$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and W^l is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function, such as the $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$. $H^l \in \mathbb{R}^{N\times D}$ is the matrix of activations in the l-th layer; $H^0 = X$.

In the following, we show that the form of this propagation rule can be motivated via a first-order approximation of localized spectral filters on graphs (Hammond et al., 2011; Defferrard et al., 2016). We consider spectral convolutions on graphs defined as the multiplication of a signal $x \in \mathbb{R}^N$ (a scalar for every node) with a filter $g_\theta = \mathrm{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^N$ in the Fourier domain, i.e.:

$$g_\theta \star x = U g_\theta U^\top x \,, \tag{3}$$

where U is the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = U\Lambda U^\top$, with a diagonal matrix of its eigenvalues Λ and $U^\top x$ being the graph Fourier transform of x. We can understand g_θ as a function of the eigenvalues of L, i.e. $g_\theta(\Lambda)$. Evaluating Eq. 3 is computationally expensive, as multiplication with the eigenvector matrix U is $\mathcal{O}(N^2)$. Furthermore, computing the eigendecomposition of L in the first place might be prohibitively expensive for large graphs. To circumvent this problem, it was suggested in Hammond et al. (2011) that $g_\theta(\Lambda)$ can be well-approximated by a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K^{th} order:

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \,, \tag{4}$$

with a rescaled $\tilde{\Lambda} = \frac{2}{\lambda_{\text{max}}}\Lambda - I_N$. λ_{max} denotes the largest eigenvalue of L. $\theta' \in \mathbb{R}^K$ is now a vector of Chebyshev coefficients. The Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. The reader is referred to Hammond et al. (2011) and Defferrard et al. (2016) for an in-depth discussion of this approximation.

Going back to our definition of a convolution of a signal x with a filter $g_{\theta'}$, we now have:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \,, \tag{5}$$

with $\tilde{L} = \frac{2}{\lambda_{\text{max}}}L - I_N$; as can easily be verified by noticing that $(U\Lambda U^\top)^k = U\Lambda^k U^\top$. Note that this expression is now K-localized since it is a K^{th}-order polynomial in the Laplacian, i.e. it depends only on nodes that are at maximum K steps away from the central node (K^{th}-order neighborhood). The complexity of evaluating Eq. 5 is now $\mathcal{O}(|\mathcal{E}|)$, i.e. linear in the number of edges. Defferrard et al. (2016) use this K-localized convolution to define a convolutional neural network on graphs.

In this work, we now suggest to approximate Eq. 5 further by keeping only terms up to order $k = 1$. Our reasoning here is as follows: as we intend to stack multiple layers of parameterized graph convolutions followed by non-linearities, we expect that a per-layer convolution operation that is linear with respect to the adjacency matrix increases modeling capacity while keeping the computational complexity comparable to a single graph convolution with $k > 1$. We further approximate $\lambda_{\text{max}} \approx 2$, as we can expect that neural network parameters will adapt to this change in scale during training.

With these approximations, Eq. 5 simplifies to:

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}}AD^{-\frac{1}{2}}x \,, \tag{6}$$

with two free parameters θ'_0 and θ'_1. Eq. 6 can be understood as a localized convolution operation with a parameterized filter that acts only on the immediate neighborhood of a node. The filter parameters can be shared over the whole graph. Successive application of filters of this form then effectively convolve the k^{th}-order neighborhood of a node, where k is the number of successive filtering operations or convolutional layers in the neural network model.

In practice, it can be beneficial to constrain the number of parameters further, thereby reducing the number of operations (such as matrix multiplications) per layer. We therefore write:

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x \,, \tag{7}$$

with a single parameter $\theta = \theta'_0 = -\theta'_1$. Note that $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ now has eigenvalues in the range $[0, 2]$. Repeated application of this operator can therefore lead to numerical instabilities and exploding/vanishing gradients when used in a deep neural network model. To alleviate this problem, we introduce the following *renormalization trick*: $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, with $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

We can generalize this definition to a signal $X \in \mathbb{R}^{N \times C}$ with C input channels (i.e. a C-dimensional feature vector for every node) and F filters or feature maps as follows:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \,, \tag{8}$$

where $\Theta \in \mathbb{R}^{C \times F}$ is now a matrix of filter parameters and $Y \in \mathbb{R}^{N \times F}$ is the convolved signal matrix. This filtering operation has complexity $\mathcal{O}(|\mathcal{E}|FC)$, as $\tilde{A}X$ can be efficiently implemented as a product of a sparse matrix with a dense matrix.

3 Semi-Supervised Node Classification

Having introduced a simple, yet flexible model $f(X, A)$ for efficient information propagation on graphs, we can return to the problem of semi-supervised node classification. As outlined in the introduction, we can relax certain assumptions typically made in graph-based semi-supervised learning by conditioning our model $f(X, A)$ both on the data X and on the adjacency matrix A of the underlying graph structure. We expect this setting to be especially powerful in scenarios where the adjacency matrix contains information not present in the data X, such as citation links between documents in a citation network or relations in a knowledge graph. The overall model, a multi-layer GCN for semi-supervised learning, is schematically depicted in Figure 1.

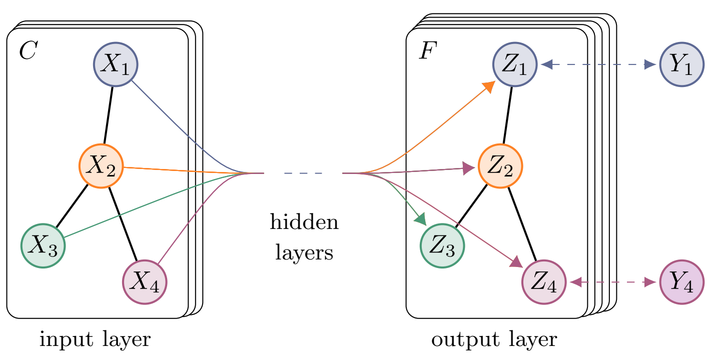

Figure 1: Schematic depiction of multi-layer Graph Convolutional Network (GCN) for semi-supervised learning with C input channels and F feature maps in the output layer. The graph structure (edges as black lines) is shared over layers, labels are denoted by Y_i.

3.1 Example

We consider a two-layer GCN for semi-supervised node classification on a graph with a symmetric adjacency matrix A (binary or weighted). We first calculate $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ in a pre-processing step. Our forward model then takes the simple form:

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A} \ \mathrm{ReLU}\Big(\hat{A} X W^{(0)} \Big) W^{(1)} \Big) \,. \tag{9}$$

Here, $W^{(0)} \in \mathbb{R}^{C\times H}$ is a input-to-hidden weight matrix for a hidden layer with H feature maps. $W^{(1)} \in \mathbb{R}^{H\times F}$ is a hidden-to-output weight matrix. The softmax activation function, defined as $\mathrm{softmax}(x_i) = \frac{1}{\mathcal{Z}}\exp(x_i)$ with $\mathcal{Z} = \sum_i \exp(x_i)$, is applied row-wise. For semi-supervised multi-class classification, we then evaluate the cross-entropy error over all labeled examples:

$$\mathcal{L} = -\sum_{l\in\mathcal{Y}_L}\sum_{f=1}^{F} Y_{lf}\ln Z_{lf}\,, \tag{10}$$

where $\mathcal{Y}_L$ is the set of node indices that have labels.

The neural network weights $W^{(0)}$ and $W^{(1)}$ are trained using gradient descent. In this work, we perform batch gradient descent using the full dataset for every training iteration, which is a viable option as long as datasets fit in memory. Using a sparse representation for A, memory requirement is $\mathcal{O}(|\mathcal{E}|)$, i.e. linear in the number of edges. Stochasticity in the training process is introduced via dropout (Srivastava et al., 2014). We leave memory-efficient extensions with mini-batch stochastic gradient descent for future work.

3.2 IMPLEMENTATION

In practice, we make use of TensorFlow (Abadi et al., 2015) for an efficient GPU-based implementation of Eq. 9 using sparse-dense matrix multiplications. The computational complexity of evaluating Eq. 9 is then $\mathcal{O}(|\mathcal{E}|CHF)$, i.e. linear in the number of graph edges.

4 RELATED WORK

Our model draws inspiration both from the field of graph-based semi-supervised learning and from recent work on neural networks that operate on graphs. In what follows, we provide a brief overview on related work in both fields.

4.1 GRAPH-BASED SEMI-SUPERVISED LEARNING

A large number of approaches for semi-supervised learning using graph representations have been proposed in the recent years, most of which fall into two broad categories: methods that use some form of explicit graph Laplacian regularization and graph embedding-based approaches.

Prominent examples for graph Laplacian regularization include label propagation (Zhu et al., 2003), manifold regularization (Belkin et al., 2006) and deep semi-supervised embedding (Weston et al., 2012). These approaches have been extended with ideas from spectral graph theory (Shuman et al., 2011; Ekambaram et al., 2013).

Recently, attention has shifted to models that learn graph embeddings with methods inspired by the skip-gram model (Mikolov et al., 2013). DeepWalk (Perozzi et al., 2014) learns embeddings via the prediction of the local neighborhood of nodes, sampled from random walks on the graph. LINE (Tang et al., 2015) and node2vec (Grover & Leskovec, 2016) extend DeepWalk with more sophisticated random walk or breadth-first search schemes. For all these methods, however, a multi-step pipeline including random walk generation and semi-supervised training is required where each step has to be optimized separately. Planetoid (Yang et al., 2016) was recently introduced to alleviate this shortcoming by injecting label information in the process of learning embeddings, thereby removing one step from the aforementioned pipeline.

4.2 NEURAL NETWORKS ON GRAPHS

Neural networks that operate on graphs have previously been introduced in Gori et al. (2005); Scarselli et al. (2009) as a form of recurrent neural network. Their framework requires the repeated application of contraction maps as propagation functions until node representations reach a stable fixed point. This restriction was later alleviated in Li et al. (2016) by introducing modern practices for recurrent neural network training to the original graph neural network framework.

Duvenaud et al. (2015) introduced a convolution-like propagation rule on graphs and methods for graph-level classification. Their approach requires to learn node degree-specific weight matrices which does not scale to large graphs with wide node degree distributions.

A related approach to semi-supervised node classification with a graph-based neural network was recently introduced in Atwood & Towsley (2016). Their model differs in that they integrate local graph information (up to a pre-chosen neighborhood size) in a single graph convolution-like layer, followed by fully-connected neural network layers. They report $\mathcal{O}(N^2)$ complexity, limiting the range of possible applications.

Another framework for convolutional neural networks on graphs was introduced in Niepert et al. (2016). Their approach converts graphs locally into sequences that are fed into a conventional 1D convolutional neural network, which requires to define a node ordering in a pre-processing step.

Our method essentially builds on spectral graph convolutional neural networks, introduced in Bruna et al. (2014) and later extended by Defferrard et al. (2016) with fast localized convolutions (Hammond et al., 2011).

5 EXPERIMENTS

We test our model in a number of experiments: semi-supervised document classification in citation networks, semi-supervised entity classification in a bipartite graph extracted from a knowledge graph, an evaluation of various graph propagation models and a run-time analysis on random graphs.

5.1 DATASETS

We closely follow the experimental setup in Yang et al. (2016). Dataset statistics are summarized in Table 1. In the citation network datasets—Citeseer, Cora and Pubmed (Sen et al., 2008)—nodes are documents and edges are citation links. Label rate denotes the number of labeled nodes that are used for training divided by the total number of nodes in each dataset. NELL (Carlson et al., 2010; Yang et al., 2016) is a bipartite graph dataset extracted from a knowledge graph with 55,864 relation nodes and 9,891 entity nodes.

Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Citation networks We consider three citation network datasets: Citeseer, Cora and Pubmed (Sen et al., 2008). The datasets contain sparse bag-of-words feature vectors for each document and a list of citation links between documents. We treat the citation links as (symmetric) edges and construct a binary, symmetric adjacency matrix A. Each document has a class label. For training, we only use 20 labels per class, but all feature vectors.

NELL NELL is a dataset extracted by Yang et al. (2016) from the knowledge base introduced in (Carlson et al., 2010). A knowledge graph is a set of entities connected with directed, labeled edges (relations). Yang et al. (2016) assign separate relation nodes r_1 and r_2 for each entity pair (e_1, r, e_2) as (e_1, r_1) and (e_2, r_2). Entity nodes are described by sparse feature vectors. We extend the number of features in NELL by assigning a unique one-hot representation for every relation node, effectively resulting in a 61,278-dim sparse feature vector per node.

The dataset was preprocessed as described in Yang et al. (2016) so that only entities of 105 out of the 240 classes are kept. The semi-supervised task here considers the extreme case of only a single labeled example per class in the training set.

The original dataset contains multiple edges between two nodes for a substantial number of node pairs. We construct a binary, symmetric adjacency matrix from this graph by setting entries $A_{ij} = 1$, if one or more edges are present between nodes i and j.

Random graphs We simulate random graph datasets of various sizes for experiments where we measure training time per epoch. For a dataset with N nodes we create a random graph assigning $2N$ edges uniformly at random. We take the identity matrix I_N as input feature matrix X, thereby implicitly taking a featureless approach where the model is only informed about the identity of each node, specified by a unique one-hot vector. In these experiments, we omit regularization (i.e. no dropout and no L2 regularization on the weights) and create dummy labels $Y_i = 1$ for each node. In each training epoch, we perform a forward pass on the full dataset, evaluate the cross-entropy error between the model prediction and the label *for every node* and update weights using Adam (Kingma & Ba, 2014). We measure and report the average wall-clock time in seconds per epoch for 100 training epochs. We compare results on a GPU and on a CPU-only implementation in TensorFlow (Abadi et al., 2015)[1].

5.2 Experimental Set-Up

Unless otherwise noted, we train a two-layer GCN as described in Section 3.1 and evaluate prediction accuracy on a test set of 1000 labeled examples. We choose the same dataset splits as in Yang et al. (2016) with an additional validation set of 500 labeled examples for hyperparameter optimization (dropout rate for all layers, L2 regularization factor for the first GCN layer and number of hidden units). We do not use the validation set labels for training.

For the citation network datasets, we optimize hyperparameters on Cora only and use the same set of parameters for Citeseer and Pubmed. We train all models for a maximum of 200 epochs (training iterations) using Adam (Kingma & Ba, 2014) with a learning rate of 0.01 and early stopping with a window size of 10, i.e. we stop training if the validation loss does not decrease for 10 consecutive epochs. We initialize weights using the initialization described in Glorot & Bengio (2010) and accordingly (row-)normalize input feature vectors.

5.3 Baselines

We compare against the same baseline methods as in Yang et al. (2016), i.e. label propagation (LP) (Zhu et al., 2003), semi-supervised embedding (SemiEmb) (Weston et al., 2012), manifold regularization (ManiReg) (Belkin et al., 2006) and skip-gram based graph embeddings (DeepWalk) (Perozzi et al., 2014). We omit TSVM (Joachims, 1999), as it does not scale to the large number of classes in one of our datasets. We further compare against Planetoid (Yang et al., 2016), where we always choose their best-performing model variant (transductive vs. inductive) as a baseline.

6 Results

6.1 Semi-Supervised Node Classification

Results for the citation network datasets—Citeseer, Cora and Pubmed—and for the knowledge graph dataset NELL are summarized in Table 2.

Reported numbers denote mean classification accuracy in percent. Results for baseline methods are taken from the Planetoid paper (Yang et al., 2016). Planetoid* denotes the best model for the respective dataset out of the variants presented in their paper.

We further report wall-clock training time in seconds until convergence (in brackets) for our method (incl. evaluation of validation error) and for Planetoid. For the latter, we used an implementation provided by the authors[2] and trained on the same hardware (with GPU) as our GCN model. We trained and tested our model on the same dataset splits as in (Yang et al., 2016) and report mean

[1]Hardware used in experiments: 16-core Intel® Xeon® CPU E5-2640 v3 @ 2.60GHz, GeForce® GTX TITAN X

[2]`https://github.com/kimiyoung/planetoid`

Table 2: Summary of results in terms of classification accuracy in percent. See text for details.

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [24]	59.6	59.0	71.1	26.7
LP [27]	45.3	68.0	63.0	26.5
DeepWalk [18]	43.2	67.2	65.3	58.1
Planetoid* [25]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

accuracy of 100 runs with random weight initializations. We used the following sets of hyper-parameters for Citeseer, Cora and Pubmed: 0.5 (dropout rate), $5 \cdot 10^{-4}$ (L2 regularization) and 16 (number of hidden units); and for NELL: 0.1 (dropout rate), $1 \cdot 10^{-5}$ (L2 regularization) and 64 (number of hidden units)

In addition, we report performance of our model on 10 randomly drawn dataset splits of the same size as in (Yang et al., 2016), denoted by GCN (rand. splits). Here, we report mean and standard error of prediction accuracy on the test set split in percent.

6.2 Evaluation of Propagation Model

We compare different variants of our proposed per-layer propagation model on the citation network datasets. We follow the experimental set-up described in the previous section. Results are summarized in Table 3. The propagation model of our original GCN model is denoted by *renormalization trick* (in bold). In all other cases, the propagation model of both neural network layers is replaced with the model specified under *propagation model*.

Table 3: Propagation model evaluation. See text for details.

Description	Propagation model	Citeseer	Cora	Pubmed
1^{st}-order model (Eq. 6)	$X\Theta_0 + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)	$(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X\Theta$	69.3	79.2	77.4
Renormalization trick (Eq. 8)	$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$	**70.3**	**81.5**	**79.0**
1^{st}-order term only (Eq. 6)	$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta$	68.7	80.5	77.8
Multi-layer perceptron	$X\Theta$	46.5	55.1	71.4

Reported numbers denote mean classification accuracy for 100 repeated runs with random weight matrix initializations. For the two-variable case (Eq. 6), we impose L2 regularization on both weight matrices of the first layer. The models denoted as 1^{st}*-order term only* and *multi-layer perceptron* (MLP) are included for comparison, they represent the 1^{st}- and 0^{th}-order terms in the original 1^{st}-*order model*, respectively.

6.3 Training Time per Epoch

Here, we report results for the mean training time per epoch (forward pass, cross-entropy calculation, backward pass) on simulated random graphs, measured in seconds wall-clock time. The experimental set-up follows the description from Section 5.1. Figure 2 summarizes the results.

7 Discussion

7.1 Semi-Supervised Model

In the experiments demonstrated here, our method for semi-supervised node classification outperforms all related methods by a significant margin. Methods based on graph-Laplacian regularization

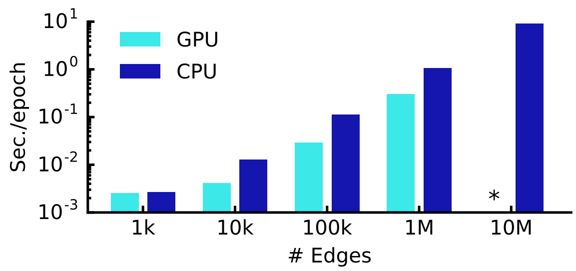

Figure 2: Experimental measurements of wall-clock time per epoch for simulated random graphs for a GPU and a CPU-only implementation. The GPU ran out of memory for the graph with 10 million edges.

(Zhu et al., 2003; Belkin et al., 2006; Weston et al., 2012) are most likely limited due to their assumption that edges encode mere similarity of nodes. Skip-gram based methods on the other hand are limited by the fact that they are based on a multi-step pipeline which is difficult to optimize. Our proposed model can overcome both limitations, while still comparing favorably in terms of efficiency (measured in wall-clock time) to related methods.

We should emphasize that we used a single set of hyperparameters for the citation network datasets (Citeseer, Cora and Pubmed). Other methods, such as Planetoid (Yang et al., 2016), typically do not generalize in such a way and require separate fine-tuning of hyperparameters.

We have further demonstrated that the proposed renormalized propagation model (Eq. 8) offers both improved efficiency (fewer parameters and operations, such as multiplication or addition) and better predictive performance compared to the naïve 1^{st}-order graph convolutional model (Eq. 6).

7.2 Limitations and Future Work

Here, we describe several limitations of our current model and outline how these might be overcome in future work.

Memory requirement In the current setup with full-batch gradient descent, memory requirement grows linearly in the size of the dataset. We have shown that for large graphs that do not fit in GPU memory, training on CPU can still be a viable option. Mini-batch stochastic gradient descent can alleviate this issue. The procedure of generating mini-batches, however, should take into account the number of layers in the GCN model, as the K^{th}-order neighborhood for a GCN with K layers has to be stored in memory for an exact procedure. For very large and densely connected graph datasets, further approximations might be necessary.

Limiting assumptions Through the approximations introduced in Section 2, we implicitly assume locality (dependence on the K^{th}-order neighborhood for a GCN with K layers) and equal importance of self-connections vs. edges to neighboring nodes. For some datasets, however, it might be beneficial to introduce a trade-off parameter λ in the definition of $\tilde{A}$:

$$\tilde{A} = A + \lambda I_N \,. \tag{11}$$

This parameter now plays a similar role as the trade-off parameter between supervised and unsupervised loss in the typical semi-supervised setting (see Eq. 1). Here, however, it can be learned via gradient descent.

Directed edges and edge features Our framework currently does not naturally support edge features and is limited to undirected graphs (weighted or unweighted). Results on NELL however show that it is possible to handle both directed edges and edge features by representing the original directed graph as an undirected bipartite graph with additional nodes that represent edges in the original graph (see Section 5.1 for details). Future work could overcome these limitations by naturally incorporating edge features and directed edges in the convolutional framework.

8 Conclusion

We have introduced an approach for semi-supervised node classification on graph-structured data using graph convolutional networks. Our model uses an efficient layer-wise propagation rule that is based on a first-order approximation of spectral convolutions on graphs. Experiments on a number of network datasets suggest that the proposed GCN model is capable of encoding both graph structure and node features in a way useful for semi-supervised classification. In this setting, our model outperforms several recently proposed methods by a significant margin, while being computationally efficient.

Acknowledgments

We would like to thank Christos Louizos, Taco Cohen, Zhilin Yang, Dave Herman, Pramod Sinha and Abdul-Saboor Sheikh for helpful discussions. This research was funded by SAP.

References

Abadi, Martín et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

Atwood, James and Towsley, Don. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2016.

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

Bruna, Joan, Zaremba, Wojciech, Szlam, Arthur, and LeCun, Yann. Spectral networks and locally connected networks on graphs. *ICLR*, 2014.

Carlson, Andrew, Betteridge, Justin, Kisiel, Bryan, Settles, Burr, Hruschka Jr, Estevam R, and Mitchell, Tom M. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, pp. 3, 2010.

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

Duvenaud, David K, Maclaurin, Dougal, Iparraguirre, Jorge, Bombarell, Rafael, Hirzel, Timothy, Aspuru-Guzik, Alán, and Adams, Ryan P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pp. 2224–2232, 2015.

Ekambaram, Venkatesan N, Fanti, Giulia, Ayazifar, Babak, and Ramchandran, Kannan. Wavelet-regularized graph semi-supervised learning. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pp. 423–426. IEEE, 2013.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pp. 249–256, 2010.

Gori, Marco, Monfardini, Gabriele, and Scarselli, Franco. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Grover, Aditya and Leskovec, Jure. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

Hammond, David K, Vandergheynst, Pierre, and Gribonval, Rémi. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

Joachims, Thorsten. Transductive inference for text classification using support vector machines. In *ICML*, volume 99, pp. 200–209, 1999.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Li, Yujia, Tarlow, Daniel, Brockschmidt, Marc, and Zemel, Richard. Gated graph sequence neural networks. *The International Conference on Learning Representations (ICLR)*, 2016.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Niepert, Mathias, Ahmed, Mohamed, and Kutzkov, Konstantin. Learning convolutional neural networks for graphs. *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

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

Scarselli, Franco, Gori, Marco, Tsoi, Ah Chung, Hagenbuchner, Markus, and Monfardini, Gabriele. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Sen, Prithviraj, Namata, Galileo Mark, Bilgic, Mustafa, Getoor, Lise, Gallagher, Brian, and Eliassi-Rad, Tina. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

Shuman, David I, Faraji, Mohammadjavad, and Vandergheynst, Pierre. Semi-supervised learning with spectral graph wavelets. In *Proceedings of the International Conference on Sampling Theory and Applications (SampTA)*, number EPFL-CONF-164765, 2011.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Tang, Jian, Qu, Meng, Wang, Mingzhe, Zhang, Ming, Yan, Jun, and Mei, Qiaozhu. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077. ACM, 2015.

Weston, Jason, Ratle, Frédéric, Mobahi, Hossein, and Collobert, Ronan. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pp. 639–655. Springer, 2012.

Yang, Zhilin, Cohen, William, and Salakhutdinov, Ruslan. Revisiting semi-supervised learning with graph embeddings. *ICML*, 2016.

Zhou, Dengyong, Bousquet, Olivier, Lal, Thomas Navin, Weston, Jason, and Schölkopf, Bernhard. Learning with local and global consistency. *Advances in neural information processing systems*, 16(16):321–328, 2004.

Zhu, Xiaojin, Ghahramani, Zoubin, Lafferty, John, et al. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, volume 3, pp. 912–919, 2003.