ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2.
Available from http://www.i6doc.com/en/.

Linear Graph Convolutional Networks

Nicolò Navarin, Wolfgang Erb, Luca Pasa and Alessandro Sperduti *

University of Padua - Department of Mathematics "Tullio Levi-Civita"
via Trieste 63, 35121, Padua - Italy

Abstract. Many neural networks for graphs are based on the graph convolution operator, proposed more than a decade ago. Since then, many alternative definitions have been proposed, that tend to add complexity (and non-linearity) to the model. In this paper, we follow the opposite direction by proposing a linear graph convolution operator. Despite its simplicity, we show that our convolution operator is more theoretically grounded than many proposals in literature, and shows improved predictive performance.

1 Introduction

In the last few years, there has been an increasing interest in machine learning models able to deal with graph-structured data, including kernel methods [1] and neural networks. The idea of Graph Neural Networks (GNNs) is to define a neural architecture that follows the topology of the graph. Then a transformation is performed from the neurons corresponding to a vertex and its neighborhood to a new hidden representation, that is associated to the same vertex (possibly in another layer of the network). This transformation depends on some parameters, that may be shared among all the vertices, obtaining Graph Convolutional Networks (GCNs). All these models share the intuition that non-linearities are essential to obtain methods with high accuracy. Recently, [2] put this concept into discussion, showing that removing the non-linearities from a popular GCN model actually did not impact much on the resulting predictive performance.

In this paper, we take a further step in this direction: we start from the theoretical foundations of graph convolution (GC), i.e. graph spectral filters, and define a theoretically grounded linear graph convolution layer. We show that the resulting Linear GCN actually performs better than many approaches in literature, at least on the semi-supervised node classification tasks we considered, while being very fast to compute.

2 Background and Related Works

Let $G = (V, E, \mathbf{X})$ be a graph, where $V = \{v_0, \ldots, v_{n-1}\}$ denotes the set of vertices (or nodes) of the graph, $E \subseteq V \times V$ is the set of edges, and $\mathbf{X} \in \mathbb{R}^{n \times c}$ is a multivariate signal on the graph nodes with the i-th row representing the attributes of v_i. We define $\mathbf{A} \in \mathbb{R}^{n \times n}$ as the adjacency matrix of the graph, with elements $a_{ij} = 1 \iff (v_i, v_j) \in E$. Note that the method presented in this

*This work has been supported by the University of Padova, Department of Mathematics, DEEPer project.

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

paper can be applied also in the case of weighted edges, i.e. when $a_{ij} \geq 0$. Let also $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the degree matrix where the diagonal elements are defined as $d_{ii} = \sum_j a_{ij}$, and $\mathbf{L}$ the normalized graph laplacian defined by $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{I}$ is the identity matrix. In this paper, we deal with the problem of semi-supervised learning on the nodes of the graph G, and we focus on neural networks models. The first definition of neural network for graphs has been proposed in [3]. More recent models have been proposed in [4, 5]. Both works are based on an idea that has been re-branded later as *graph convolution* or *neural message passing*. The derivation of the graph convolution operator originates from graph spectral filtering [6].

For a fixed graph G, let $\mathbf{x} : V \to \mathbb{R}$ be a signal on the nodes V of the graph G, i.e. a function that associates a real value to each node of V. We will represent such a signal as a vector $\mathbf{x} \in \mathbb{R}^n$. The graph convolution operator is usually defined in terms of the graph Fourier transform, using an analogy to classical Fourier analysis in which the convolution of two signals is calculated as the pointwise product of their Fourier transforms. The graph Fourier transform on G is given in terms of the eigenvalue decomposition $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\intercal$ of the graph Laplacian $\mathbf{L}$. The rows $\{\mathbf{u}_0, \ldots, \mathbf{u}_{n-1}\}$ of the matrix $\mathbf{U}$ form a basis of orthonormal eigenvectors of the Laplacian $\mathbf{L}$ and provide a Fourier basis for the signals on G. For a signal $\mathbf{x}$, the Fourier transform of $\mathbf{x}$ is then defined as $\hat{\mathbf{x}} = \mathbf{U}^\intercal\mathbf{x}$. Further, the inverse Fourier transform given by $\mathbf{U}$ enables us to recover the signal $\mathbf{x}$ as $\mathbf{x} = \mathbf{U}\hat{\mathbf{x}}$. Using the graph Fourier transform to switch between spatial and spectral domains, the graph convolution between a filter $\mathbf{f}$ and a signal $\mathbf{x}$ is defined as:

$$\mathbf{f} *_G \mathbf{x} := \mathbf{U}\left(\hat{\mathbf{f}} \odot \hat{\mathbf{x}}\right) = \mathbf{U}\hat{\mathbf{F}}\mathbf{U}^\intercal\mathbf{x}. \tag{1}$$

Here, we reformulated the Hadamard product $\hat{\mathbf{f}} \odot \hat{\mathbf{x}}$ in matrix-vector notation as $\hat{\mathbf{f}} \odot \hat{\mathbf{x}} = \hat{\mathbf{F}}\hat{\mathbf{x}}$, by applying the diagonal matrix $\hat{\mathbf{F}} = \mathrm{diag}(\hat{\mathbf{f}})$. The diagonal matrix $\hat{\mathbf{F}}$ and, thus, the spectral filter $\mathbf{f}$ can be designed in various different ways. The simplest way would be to define $\mathbf{F}_\Theta$ as a non-parametric filter. However, such a filter grows in size with the data, and it is not well suited for learning. A better option is to use a polynomial parametrization based on powers of the spectral matrix $\mathbf{\Lambda}$, such as $\hat{\mathbf{F}}_\Theta = \sum_{i=0}^{k} \theta_i \mathbf{\Lambda}^i$. This filter has $k+1$ parameters to learn, and it is spatially exactly K-localized. One of the main advantages of this filter is that we can formulate the convolution explicitly in the graph domain:

$$\mathbf{f}_\Theta *_G \mathbf{x} = \mathbf{U}\hat{\mathbf{F}}_\Theta\mathbf{U}^\intercal\mathbf{x} = \sum_{i=0}^{k} \theta_i \mathbf{U}\mathbf{\Lambda}^i\mathbf{U}^\intercal\mathbf{x} = \sum_{i=0}^{k} \theta_i \mathbf{L}^i\mathbf{x}. \tag{2}$$

Real-world problems typically involve graphs with millions of nodes: in this case, it is prohibitive to calculate the eigendecomposition of $\mathbf{L}$ and a filter of the form in eq. (2) has clear advantages compared to a spectral filter given in the form of eq. (1).

The parametrization of the polynomial filter in eq. (2) is given in the monomial basis. Alternatively, [6] proposes to use Chebyshev polynomials as a poly-

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2.
Available from http://www.i6doc.com/en/.

nomial basis to improve the numerical stability. In [7], the authors propose to fix the order $K = 1$ in eq. (2) to obtain a linear first order filter for each graph convolutional layer in a neural network. These simple convolutions can then be stacked in order to improve the discriminatory power of the resulting network. Generalizing the definition to a signal $\mathbf{X} \in \mathbb{R}^{n \times c}$ and using m filters, [7] derives the following definition for the k-th GCN graph convolutional layer:

$$\mathbf{H}^{(k)} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(k-1)}\mathbf{\Theta}), \tag{3}$$

where $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{D}}$ are the renormalized adjacency and degree matrices (see [7]), $\mathbf{\Theta} \in \mathbb{R}^{c \times m}$ (m being the number of neurons in the k-th layer), $\mathbf{H}^{(0)} = \mathbf{X}$ and σ is a nonlinear activation function, typically ReLU (for multi-class classification in the last layer ReLu is usually replaced by the *softmax* function).

In the last few years several models inspired by the graph convolution have been proposed. Graph Attention Networks [8] exploit a different convolution operator based on self-attention. Fast GCN [9] uses node sampling to define a fast convolution operator, suited for the inductive setting. Graph Invariant Networks (GIN) [10] and [11] adopt a more powerful graph convolution operator. LNet and AdaLNet [12] exploit filters learned on an approximation of the Laplacian matrix. Deep Graph InfoMax (DGI) [13] trains a GCN in an unsupervised setting to obtain general node embeddings. GNN with ARMA filters (ARMA) [14] defines an ARMA filter for graph convolution.

2.1 Simple Graph Convolution

In [15], a simplification of the graph convolution operator in eq. (3) is proposed, dubbed Simple Graph Convolution (SGC). The idea is that perhaps the nonlinear operator introduced by GCNs is not essential. However, stacking multiple GC layers has an important effect on the locality of the learned filters in that, after k GC layers, the hidden representation of a vertex considers information coming from the vertices up to distance k, i.e. the filters on the k-th layer are k-localized. Let us rewrite $\mathbf{H}^{(k)} = \mathbf{S}\mathbf{H}^{(k-1)}\mathbf{\Theta}^{(k)}$, where $\mathbf{S} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$. If we stack k such layers without any non-linearity, and apply a *softmax* classifier at the end, the output after k hidden layers is:

$$\mathbf{Y} = softmax(\mathbf{S}^k\mathbf{X}\mathbf{\Theta}). \tag{4}$$

This is possible because the SGC model is linear, therefore we can apply the reparametrization $\mathbf{\Theta} = \mathbf{\Theta^{(0)}} \cdots \mathbf{\Theta^{(k)}}$, where $\mathbf{\Theta^{(0)}} \in \mathbb{R}^{c \times m_0}$, $\mathbf{\Theta^{(i)}} \in \mathbb{R}^{m_{i-1} \times m_i}$, $\forall i \in [1 \dots k-1]$ and, $\mathbf{\Theta^{(k)}} \in \mathbb{R}^{m_{k-1} \times m}$. The great advantage of this model is a reduced number of parameters compared to classical graph convolution. Moreover, $\mathbf{S}^k$ can be computed only once, with a dramatic speedup compared to GCNs.

The SGC formulation has an interesting interpretation. We can think about having a fixed feature extractor / representation for the graph ($\bar{\mathbf{X}} = \mathbf{S}^k\mathbf{X}$) and a simple multinomial logistic regression applied to it ($\mathbf{Y} = softmax(\bar{\mathbf{X}}\mathbf{\Theta})$). The training of the model reduces to training a standard softmax classifier.

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2.
Available from http://www.i6doc.com/en/.

3 Linear Graph Convolutions

In this section we present the main contribution of this paper, namely the definition of a Linear Graph Convolution layer.

Let us consider the SGC formulation in eq. (4). We can notice that the output depends just on the k-th exponentiation of $\mathbf{S}$. This is in contrast with previous proposals, and with approximation theory, that suggests to consider all monomials up to a maximum order, e.g. polynomial filters, such as the one in eq. (2). We propose an alternative definition for a Linear Graph Convolution operator (LGC), that we define as:

$$\mathbf{Y} = softmax(\sum_{i=0}^{k} \alpha_i \mathbf{S}^i \mathbf{X}\mathbf{\Theta}). \tag{5}$$

Note that the difference compared to SGC is the introduction of skip connections from each layer to the last one, that is a merge layer implementing the sum operator, followed by a softmax activation. Moreover, we further limit the number of parameters of our model, forcing $\mathbf{\Theta} = \mathbf{\Theta}^{(i)}$ for all $0 \leq i \leq k$, and including a single multiplicative weight for each component of the sum. In this way, the network has a simple way to downweight (or to put to zero) the contributions of higher order exponentials of $\mathbf{S}$, making the hyperparameter k controlling the number of layers easy to tune (the higher the better).

It is possible to define the graph Fourier transform in terms of its (normalized) adjacency matrix, instead of its Laplacian [16]. In this setting, the graph convolution becomes:

$$\mathbf{f}_\Theta *_G \mathbf{x} = \mathbf{U}\hat{\mathbf{F}}_\Theta \mathbf{U}^\intercal \mathbf{x} = \sum_{i=0}^{k} \alpha_i (\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^i \mathbf{x}.$$

Also the definition of Linear Graph Convolution remains similar to eq. (3), using $\mathbf{A}$ and $\mathbf{D}$ instead of $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{D}}$. Even though we observe similar results, from preliminary results the latter formulation seems more robust. Our final model formulation, that we refer to as Linear Graph Convolutional Network (LGCN), is then:

$$\mathbf{Y} = softmax(\sum_{i=0}^{k} \alpha_i (\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^i \mathbf{X}\mathbf{\Theta}). \tag{6}$$

4 Experiments

In this section, we present our numerical experiments in the setting of semi-supervised node classification. Concerning our proposed method, we use a single Linear Graph Convolution layer (detailed in Section 3), followed by a softmax activation function as reported in eq. (6). We solve the resulting optimization problem with the Adam algorithm (a variant of stochastic gradient descent with momentum and adaptive learning rate). We used early stopping (with the patience set to 100) and model checkpoint, monitoring the loss on the validation

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

Method/Dataset	Cora	Citesser	PubMed
GCN [7]	81.4±0.4	70.9±0.5	79.0±0.4
GAT [8]	83.3±0.7	72.6±0.6	78.5±0.3
FastGCN [9]	79.8±0.3	68.8±0.6	77.4±0.3
GIN [10]	77.6±1.1	66.1±0.9	77.0±1.2
LNet [12]	80.2±3.0	67.3±0.5	78.3±0.6
AdaLNet [12]	81.9±1.9	70.6±0.8	77.8±0.7
DGI [13]	82.5±0.7	71.6±0.7	78.4±0.7
ARMA [14]	83.4±0.6	72.5 ±0.4	78.9 ±0.3
SGC [2]	81.0±0.0	71.9±0.1	78.9±0.0
LGCN (k)	**85.0**±0.3 (10)	**72.9**±0.3 (5)	**80.2**±0.4 (10)

Table 1: Accuracy comparison of our proposed method (LGCN) against different state-of-the-art methods in literature on three node classification datasets.

set. We set the maximum number of epochs to 500. Moreover, we validated the hyper-parameters of our model using the accuracy on the validation set. We validated the locality k in the set $\{3, 5, 10, 20, 50\}$. Note that, unlike many competing methods, our model does not need to specify a number of neurons of the hidden projections, since the model uses only one parameters matrix $\Theta \in \mathbb{R}^{c \times m}$, where m is the output size. We then validated the learning rate in $\{0.01, 0.02, 0.05\}$ and the weight decay (corresponding to l_2 regularization) in $\{5e^{-4}, 5e^{-3}, 5e^{-2}, e^{-2}\}$. We report the average and standard deviation over 10 runs of our method. We consider three widely adopted datasets of node classification following the same setting as [7]: Cora, Citeseer, Pubmed. Each dataset is a graph, where nodes represent documents and node features are sparse bag-of-words feature vectors.

We compare our model to many state-of-the-art approaches presented in Section 2. We use the training/test/validation splits from [7]. In Table 1 we report the results of our comparison. We can see that our proposed model, though not nearly as complex as many competing ones, shows the best predictive performance in all the three considered datasets. Compared to other neural models, our approach seems to be very stable in that it tends not to overfit on the validation set, i.e. the validation loss decreases with the test loss. Moreover, it seems pretty robust to the choice of its only hyper-parameter k, inasmuch values higher than the optimal one do not significantly degrade the performance (not shown for lack of space).

A note about computational times. Since our proposed model is linear (and $(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^k\mathbf{x}$ can be pre-computed similarly to $\mathbf{S}^k\mathbf{x}$ for SGC, see Section 2.1), its computational requirements are comparable to the ones of SGC, that are orders of magnitude faster with respect to the other (non-linear) approaches in literature. To give an idea of the computational times, on the machine we adopted for our experiments (equipped with 2 Intel®Xeon®CPU E5-2630L v3 and a GPU Nvidia Tesla V100) our approach takes in total 18 seconds for a single run with $k = 20$ on the Cora dataset. Most of the time is spent on reading the dataset, saving the models on disk and on pre-computing the matrix

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

exponentiation, since each epoch of training takes only 0.01 seconds. With $k = 50$ it takes 43 seconds in total, with 0.04 seconds per epoch.

5 Conclusions and Future Works

We proposed a linear graph convolution layer, that is defined as a weighted sum of monomial filters in the graph spectral domain. We showed that, in the considered datasets, such a simple approach outperforms all the (more complex) graph convolution definitions in literature, while being orders of magnitude faster than most of them. In the future, we plan: *i)* to extend our experimental comparison to other datasets and to the graph classification setting; *ii)* to explore more complex convolutional filters such as exponential filters.

References

[1] Nicoló Navarin and Alessandro Sperduti. Approximated neighbours minhash graph node kernel. In *ESANN*, pages 281–286, 2017.

[2] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying Graph Convolutional Networks. *ICML*, feb 2019.

[3] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Networks*, 8(3):714–735, 1997.

[4] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[5] F. Scarselli, M. Gori, Ah Chung Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Neural Information Processing Systems (NIPS)*, jun 2016.

[7] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, pages 1–14, 2017.

[8] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *ICLR*, 2018.

[9] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*, 2018.

[10] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *ICLR*, 2019.

[11] Dinh V. Tran, Nicolò Navarin, and Alessandro Sperduti. On Filter Size in Graph Convolutional Networks. In *IEEE SSCI*, Bengaluru, India, 2018.

[12] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S. Zemel. LanczosNet: Multi-Scale Deep Graph Convolutional Networks. In *ICLR*, 2019.

[13] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep Graph Infomax. In *ICLR*, 2019.

[14] Filippo Maria Bianchi, Daniele Grattarola, Cesare Alippi, and Lorenzo Livi. Graph Neural Networks with convolutional ARMA filters. *arXiv preprint*, jan 2019.

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

[16] Aliaksei Sandryhaila and José M. F. Moura. Discrete Signal Processing on Graphs. *IEEE Transactions on Signal Processing*, 61(7):1644–1656, apr 2013.