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/.

Deep Recurrent Graph Neural Networks

Luca Pasa, Nicolò Navarin and Alessandro Sperduti *

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

Abstract. Graph Neural Networks (GNN) show good results in classification and regression on graphs, notwithstanding most GNN models use a limited depth. In fact, they are composed of only a few stacked graph convolutional layers. One reason for this is the number of parameters growing with the number of GNN layers. In this paper, we show how using a recurrent graph convolution layer can help in building deeper GNNs, without increasing the complexity of the training phase, while improving on the predictive performances. We also analyze how the depth of the model influences the final result.

1 Introduction

In a broad range of real-world Machine Learning applications, representing examples as fixed-size vectors leads to a loss of information (e.g. predicting the toxicity of a molecule). In some of these problems, representing examples as graphs is more natural, therefore availability of Machine Learning methods capable of dealing with such structured data becomes crucial. Among other models [1], neural networks for graphs (GNNs) have been proposed in [2]. More recently, [3] proposed the idea that has been re-branded later as *graph convolution*, and [4] defined a recurrent neural network for graphs. Many recent works defining graph convolutional networks (GCNs) extend the idea in [3], e.g. [5, 6]. In [7], authors show that removing the non-linearities from an existing models actually doesn't significantly hurt the predictive performance. [8] extended the work in [4], exploiting Gated Recurrent Units (GRUs). The model is tested on basic reasoning tasks over graphs, and on program verification. Other approaches have been studied as well, such as deep graph echo state networks [9] and deep generative models for graphs [10]. However, unlike deep networks in other domains, existing GNN models do not seem to benefit much from increased depth, arguably because of a fast increase in the number of parameters, which easily leads to overfitting.

In this paper, we propose a novel, simple, recurrent GNN model that uses a recurrent graph convolutional layer. This particular layer, inspired by Recurrent Neural Networks, allows to increase the depth of the model, while the number of weights in the convolutional stage of the GNN does not change. In order to evaluate our approach, we focus on graph classification tasks. We use five widely adopted bioinformatics datasets, that allow us to compare the proposed method with exiting state-of-the-art GNN models. The proposed model obtains

*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/.

promising results (that in three out of five cases it outperforms the current state-of-the-art), even in its simplest formulation that adopts a linear activation function.

2 Background

Given a graph $G = (V, E, L)$, 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 connecting the nodes, and $L = \{\mathbf{l}_0, \ldots, \mathbf{l}_{n-1}\}$, with $\mathbf{l}_i \in \mathbb{R}^s$ is the set of node attributes (features) of each vertex $v_i \in V$. Let $\mathcal{N}(v)$ be the set of nodes adjacent to v. A Graph Neural Network (GNN) is a model that exploits the structure of the graph and the information embedded in feature vectors of each node in order to learn a representation $\mathbf{h}_v \in \mathbb{R}^m$ for each vertex $v \in V$. In modern GNNs models, the computation of $\mathbf{h}_v$ can be divided in two main steps: *aggregate* and *combine*. We can define aggregation and combination by using two functions, $\mathcal{A}$ and $\mathcal{C}$, respectively: $\mathbf{h}_v = \mathcal{C}(\mathbf{l}_v, \mathcal{A}(\{\mathbf{l}_u : u \in \mathcal{N}(v)\}))$.
It is possible to extend the range of the considered neighborhood by iteratively performing aggregation and combination for k iterations. In this way, we obtain a hidden representation $\mathbf{h}_v^{(k)}$ of the node v that contains information about the structure and the neighbors that are at distance k from v:

$$\mathbf{h}_v^{(i)} = \mathcal{C}(\mathbf{h}_v^{(i-1)}, \mathcal{A}(\{\mathbf{h}_u^{(i-1)} : u \in \mathcal{N}(v)\})),\ i \in [0, \ldots, k],$$
$$\mathbf{h}_v^{(0)} = \mathbf{l}_v, \quad \mathbf{h}_v^{(i)} \in \mathbb{R}^{m_i}, \text{where } m_i \text{ is the size of convolutional layer } i.$$

We thus obtain a deep GNN of k-layers. The choice of aggregation function $\mathcal{A}$ and combination function $\mathcal{C}$ defines the type of *Graph Convolution (GC)* adopted by the GNN. In [3], the first model that uses graph convolutions is proposed, while in [5] a widely adopted formulation is derived. In the last few years, several different GCs have been proposed [11, 12, 8, 13, 6, 14]. In this work we focus our attention on the graph convolution proposed in [15]: $\mathbf{h}_v^{(i)} = \mathcal{F}(\mathbf{W}_1^{(i)}\mathbf{h}_v^{(i-1)} + \sum_{u \in \mathcal{N}(v)} \mathbf{W}_2^{(i)}\mathbf{h}_u^{(i-1)})$, where $\mathbf{W}_1^{(i)}, \mathbf{W}_2^{(i)} \in \mathbb{R}^{m_i \times m_{i-1}}$ (with $m_0 = s$) are the network parameters, and $\mathcal{F}$ is the element-wise (usually, nonlinear) activation function. In node classification tasks $\mathbf{h}_v^{(k)}$ can be used as input for the output layer (that usually involves a *softmax* function), while for graph classification (the task that we consider in this work) a *readout* function is required to aggregate the computed hidden representations of all nodes in V into a graph-level representation before applying the output layer. Different strategies can be used to aggregate the hidden representations, from simple invariant functions like sum or average, to more complex computational modules that involve several feed-forward layers and nonlinear functions [16, 17, 18].

3 Deep Recurrent Graph Neural Networks

Many GNN models in literature (mentioned in Sections 1 and 2) exploit several GC layers. Each one of these layers has its own parameters, and this makes it

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/.

really complex to train a truly deep model. In fact, the number of learnable parameters grows fast as the number of layers increases. A high number of parameters makes the training phase heavy, and a huge quantity of training samples is necessary in order to obtain a good predictive performance (and to avoid overfitting). Therefore, inspired by the shared weights mechanism of Recurrent Neural Networks, we developed a GNN model that exploits a simple recurrent graph convolutional layer. It is important to point out that stacking k GC layers allows the hidden vectors $\mathbf{h}_v^{(k)}$ to embed information about the structure and the neighbors that are at most k-hop far from v. Recurrent weights make possible to maintain the number of parameters fixed, regardless of the number k of *unfolded* GC layers.

Our aim is to develop a simple model that allows us to study and analyze how a recurrent weight sharing mechanism influences the learning capability of the model. The proposed model uses a graph convolution, inspired by [15], where the weights are shared among layers and that can be formally defined as follows:

$$\mathbf{h}_v^{(0)} = \mathcal{F}(\mathbf{W}_1^{(l)}\mathbf{l}_v + \sum_{u \in \mathcal{N}(v)} \mathbf{W}_2^{(l)}\mathbf{l}_u),$$
$$\mathbf{h}_v^{(i)} = \mathcal{F}(\mathbf{W}_1\mathbf{h}_v^{(i-1)} + \sum_{u \in \mathcal{N}(v)} \mathbf{W}_2\mathbf{h}_u^{(i-1)}), \quad \forall i \in [1, \ldots, k-1],$$

where $\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)} \in \mathbb{R}^{m \times s}$, $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{m \times m}$, and $\mathcal{F}$ is the activation function. In the linear case $\mathcal{F}$ is the identity function (following the same intuition as [7]), while in the nonlinear version of the convolution we use the LeakyRelu activation function. We compute a graph-level representation for each GC layer using a pooling layer $\mathbf{s}_j, \forall j \in [0, \ldots, k-1]$ that exploits three different aggregation strategies over the whole set of nodes V: $\mathbf{s}_j = [avg(\{h_v^{(j)}, \forall v \in V\}) | max(\{h_v^{(j)}, \forall v \in V\}) | sum(\{h_v^{(j)}, \forall v \in V\})]$, where avg, max, and sum are three element-wise function that compute the average, the maximum value and the sum, respectively, of all the vectors in the given set, therefore, $\mathbf{s}_i \in \mathbb{R}^{3m}$. The readout part of the model is composed of three dense feed-forward layers. The input of this stage of the model is the concatenation of all $\mathbf{s}_i$ vectors computed at each layer: $\mathbf{y_0} = [\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{k-1}] \in R^{3km}$. We stack two fully connected layers and the output layer that, for a c-class classification task, are defined as follows:

$$\mathbf{y}_1 = ReLu(\mathbf{W}_{y_1}\mathbf{y}_0 + \mathbf{b}_{y_1}),$$
$$\mathbf{y}_2 = ReLu(\mathbf{W}_{y_2}\mathbf{y}_1 + \mathbf{b}_{y_2}),$$
$$\mathbf{o} = LogSoftmax(\mathbf{W}_o\mathbf{y}_2 + \mathbf{b}_o),$$

where $\mathbf{W}_{y_1} \in \mathbf{R}^{d_1 \times 3km}$, $\mathbf{y}_1$, $\mathbf{b}_{y_1} \in \mathbf{R}^{d_1}$, $\mathbf{W}_{y_2} \in \mathbf{R}^{d_2 \times d_1}$, $\mathbf{y}_2$, $\mathbf{b}_{y_2} \in \mathbf{R}^{d_2}$, $\mathbf{W}_o \in \mathbf{R}^{c \times d_2}$, and $\mathbf{o}, \mathbf{b}_o \in \mathbf{R}^c$.

4 Experimental Setup and Results

In this section, we report and analyze the results obtained by our model on graph classification tasks. The proposed Deep Recurrent Graph Neural Network

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/.

(DRGNN) is compared versus several state-of-the-art methods. Moreover, we analyze how using a higher number of recurrent GC layers influences the training phase and the classification capability of the model.

Datasets: We performed experiments on five bioinformatics datasets: MUTAG, PTC, NCI1, PROTEINS, and D&D (see [18] for details). These datasets model binary classification tasks over node-labeled graphs (representing chemical compounds or proteins).

Model Setup: Similarly to modern Neural Networks for sequential data, we applied layer normalization to the recurrent GC layer [19], to reduce the *covariate shift* effect. Moreover, it allows for a more stable training phase. In order to avoid divergence during training and to attenuate overfitting effects, we applied dropout on the last two layers. The dropout probability was fixed to 0.5 for all tests. We used the *Negative Log Likelihood* (NLL-Loss) loss function and the *Adam* optimizer. We performed a limited random search-based hyper-parameter tuning, therefore all the reported results may improve through a finer hyper-parameters validation phase. Moreover, since one of the main goals of this work is investigating the impact of using different values of k with a recurrent GC layer, we maintained most of the hyper-parameters fixed and varied only the value of k. For further details about experimental setup, it is possible to consult the publicly available code [1], that adopts the library *pytorch geometric* [20].

Result Analysis: We compare DRGNN with four state-of-the-art GNN architectures, in 10-fold cross-validation. We consider PSCN [21], Capsule Graph Networks (CapGCN) [14], DGCNN [16], and GIN [13] (we re-run the experiments using the validation set for model selection and, differently than [13], using only the node labels as input of the model). For DRGNN, the hyper-parameters were chosen according to a random search validation phase, by using accuracy as the reference metric. Table 1 shows that the DRGNN obtains state-of-the-art results on 3 out of 5 datasets (PTC, NCI1, and D&D), while in the others it obtains comparable performances with the other methods, but with a significantly lower standard deviation. In particular, the comparison with the Capsule GNN (that achieves the best results on PROTEINS dataset) is not completely fair, in that it is a meta-model, that embeds another model (DGCNN) as a capsule.

It is surprising that DRGNN achieves these results without using any non-linearity in the graph convolution (exploited in all the compared models). Indeed the comparison between Linear-DRGNN (L-DRGNN) and LeakyReLu-DRGNN (LR-DRGNN) shows that the use of a nonlinear function does not bring any advantage in almost all considered cases.

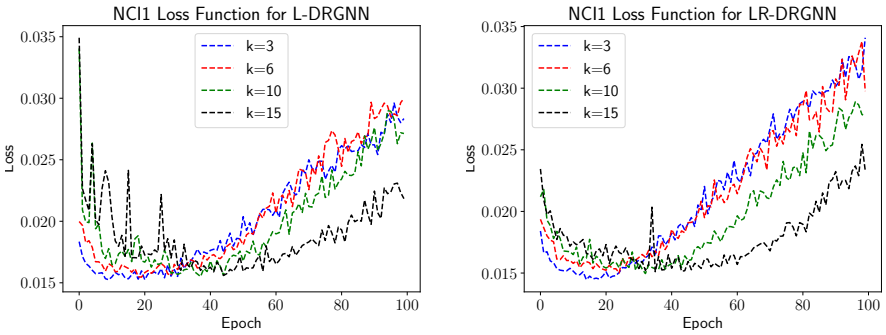
In Figure 1 we report the plots of the evolution of the loss function on the validation set during training for some values of k on the NCI1 dataset (other plots omitted for space limitations), for L-RDGNN (a) and LR-RDGNN (b). The starting point gets worse as the value of k increases. This behaviour is due to the higher depth of the unfolded network, that during the first epochs is far from being optimal. After 30-40 epochs, the values of the loss function are very close regardless of the value of k. In Figure 1b it is possible to notice that the

[1] `https://github.com/lpasa/RecurrentDGNN`

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/.

Dataset (#Graphs)	**MUTAG** (188)	**PTC** (344)	**NCI1** (4110)	**PROTEINS** (1113)	**D&D** (1178)
PSCN[21]	**88.95±4.37**	60.00 ±4.82	76.34 ±1.68	75.00 ±2.51	76.27 ±2.64
CapGCN[14]	86.67 ±6.88	-	78.35 ±1.55	**76.28±3.63**	75.38 ±4.17
DGCNN[16]	85.83 ±1.74	58.59 ±2.47	74.44 ±0.47	75.54 ±0.94	79.37 ±0.69
GIN[13]	84.68 ±1.82	57.80 ±0.86	71.14 ±1.71	71.89 ±1.41	72.60 ±2.14
L-DRGNN (m - k)	87.09 ±1.66 (80-15)	**61.54±1.30** (50-3)	**83.07±0.35** (150-6)	75.62 ±0.52 (100-15)	**79.62±0.22** (100-3)
LR-DRGNN (m - k)	87.64 ±2.09 (80-15)	60.54 ±2.76 (50-3)	83.05 ±0.09 (150-10)	75.36 ±0.58 (100-10)	78.24 ±0.96 (100-3)

Table 1: Accuracy comparison among DRGNN and *state-of-the-art* models.



(a) (b)

Fig. 1: Loss function curves ($k = \{3, 6, 10, 15\}$) for *(a)* linear activation function, and *(b)* LeakyReLU activation function, obtained for the NCI1 dataset.

use of LeakyRelu activation makes the training phase more stable. After epoch 40, the trend of the loss function suggests that deeper networks show a more stable behavior. In fact, with higher k, the loss function tends to diverge later.

5 Conclusions

In this paper, we proposed a simple Recurrent Deep GNN model. The model exploits the weight sharing technique for the convolutional layers. This allows to define a model with arbitrary depth while avoiding the substantial increase in the number of parameters, typical of deep GNNs. The experimental results showed that this simple model can achieve comparable, and some times better, accuracy than the state-of-the-art models proposed in the literature, without using any non-linearity in the convolutional layers. Moreover, we showed the effect of increasing the depth of the unfolded network on the loss function. In the future, we plan to study how the diameters of the input graphs could influence the choice of the "optimal" value for the depth of the unfolded network. We

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/.

also plan to extend our model by adding more complex graph-level pooling techniques, inspired by the methods proposed in [16, 17]. Moreover, we plan to extend the empirical comparison by considering the model proposed in [8].

References

[1] Giovanni Da San Martino, Nicolò Navarin, and Alessandro Sperduti. Tree-Based Kernel for Graphs With Continuous Attributes. *IEEE Transactions on Neural Networks and Learning Systems*, 29(7):3270—-3276, 2018.

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

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

[4] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

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

[6] Dinh V. Tran, Nicolo Navarin, and Alessandro Sperduti. On Filter Size in Graph Convolutional Networks. In *IEEE SSCI*, pages 1534–1541, Bengaluru, India, nov 2018. IEEE.

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

[8] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. In *ICLR*, 2016.

[9] Claudio Gallicchio and Alessio Micheli. Fast and Deep Graph Neural Networks. In *AAAI*, 2019.

[10] Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual Graph Markov Model: A Deep and Generative Approach to Graph Processing. In *ICML*, 2018.

[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3844–3852, 2016.

[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[13] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, 2019.

[14] Zhang Xinyi and Lihui Chen. Capsule graph neural network. In *ICLR*, 2019.

[15] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, volume 33, pages 4602–4609, 2019.

[16] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[17] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[18] Nicolò Navarin, Dinh Van Tran, and Alessandro Sperduti. Universal Readout for Graph Convolutional Neural Networks. In *IJCNN*, Budapest, Hungary, 2019.

[19] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[20] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[21] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016.