Inductive–Transductive Learning with Graph Neural Networks

Alberto Rossi[1,2(✉)], Matteo Tiezzi[1(✉)], Giovanna Maria Dimitri[3], Monica Bianchini[1], Marco Maggini[1], and Franco Scarselli[1(✉)]

[1] Department of Information Engineering and Mathematics, University of Siena, Siena, Italy
{mtiezzi,monica,maggini,franco}@diism.unisi.it
[2] Department of Information Engineering, University of Florence, Florence, Italy
alberto.rossi@unifi.it
[3] Department of Computer Science, Computer Laboratory, University of Cambridge, Cambridge, UK
gmd43@cam.ac.uk
http://sailab.diism.unisi.it

Abstract. Graphs are a natural choice to encode data in many real–world applications. In fact, a graph can describe a given pattern as a complex structure made up of parts (the nodes) and relationships between them (the edges). Despite their rich representational power, most of machine learning approaches cannot deal directly with inputs encoded by graphs. Indeed, Graph Neural Networks (GNNs) have been devised as an extension of recursive models, able to process general graphs, possibly undirected and cyclic. In particular, GNNs can be trained to approximate all the "practically useful" functions on the graph space, based on the classical inductive learning approach, realized within the supervised framework. However, the information encoded in the edges can actually be used in a more refined way, to switch from inductive to transductive learning. In this paper, we present an inductive–transductive learning scheme based on GNNs. The proposed approach is evaluated both on artificial and real–world datasets showing promising results. The recently released GNN software, based on the Tensorflow library, is made available for interested users.

Keywords: Graph Neural Networks · Transductive learning
Graph representations

1 Introduction

Graphs are a rich structured model that can be exploited to encode data from many different domains, which range from bioinformatics [1,2] to neuroscience [3], and social networks [4]. Despite the simplicity of the concepts at the basis of the definition of a graph, the possibility to encode complex data as a set of parts, i.e. the graph nodes, and a set of relationships between these parts, i.e.

L. Pancioni et al. (Eds.): ANNPR 2018, LNAI 11081, pp. 201–212, 2018.
https://doi.org/10.1007/978-3-319-99978-4_16

the graph edges, allows a good compromise between the need of a compact data representation and the preservation of most of the original input information. This model is quite natural in many of the aforementioned applications and the encoding is often straightforward. As an example, the Web can be naturally seen and represented as a graph, with nodes corresponding to web pages (and storing their content), and edges standing for the hyper-links between them [5].

Many classical machine learning approaches assume to deal with *flat* data, encoded, for instance, as real valued vectors. Hence, complex data, such as graphs, need to be transformed into these simpler encodings—typically with some sort of graph traversal—often loosing useful information [6]. In this way, the input to the machine learning tool is no more the original graph but instead a linearized representation, in which the topological and structural information is usually encoded in an unnatural way, that may hinder the learning process itself. Moreover, the natural variability within graphs requires artificial solutions. For instance, the mapping of a graph to a real valued vector can be implemented by concatenating the features stored in each node, following an order derived from the connection topology. However, this approach has many drawbacks. First, in order to have a fixed dimensionality for the vector, all the input graphs should have the same number of nodes or, at least, a maximum cardinality must be chosen for this set, filling the vector elements with padding values when the graph has a lower number of nodes. Second, the encoding of the topology by the position of the node inside the vector is not well defined for any category of graphs. Indeed, if for Directed Ordered Acyclic Graphs (DOAGs) such a topological order is uniquely defined, this does not hold for generic cyclic graphs, where the mapping between nodes and related elements occupying particular positions is arbitrary.

The Graph Neural Network Model (GNN), which was introduced in [6], is able to process graphs directly, without the need of a preprocessing step and without any limitation on the graph type. GNNs are supervised architectures, designed as an extension to Recursive Neural Networks [17–19] and Markov Random Chain Models. The original GNN model is based on the classical inductive learning scheme, where a training set is used to adapt a parametric model. Actually, inductive learning assumes the existence of some rules, that can be implemented by the model, allowing us to classify a pattern given its properties. In this framework, GNNs have been successfully used in different applications, from the classification of Web pages (in Spam or Non–Spam) to the prediction of chemical properties of drug molecules [7].

On the other hand, transductive learning adopts a more direct approach, by which a pattern is classified according to its relationships with the examples available in the training set. In this case, the training patterns are used directly in the classification procedure, without adapting a parametric model, and even without relying on the existence of classification rules and pattern features. In the standard inductive approach, GNNs exclusively employ the parameters learnt during the training procedure. Vice versa, in the transductive approach, the

available targets are added to the node labels, and they are directly diffused through the graph in the classification phase.

In this paper, we present a mixed transductive–inductive GNN model that exhibits characteristics common to both the learning frameworks. This model is evaluated on synthetic (clique detection and subgraph matching) and real (traffic flow prediction and web–spam prediction) problems, involving structured inputs encoded with graphs. In particular, we exploit a new implementation of GNNs based on the TensorFlow platform [8].

The paper is organized as follows. In the next section, the GNN model and the related learning algorithms are briefly sketched. Then, in Sect. 3 the transductive approach for GNNs is described. Section 4 presents the experimental settings and reports the obtained results. Finally Sect. 5 draws some conclusions and delineates future perspectives.

2 The Graph Neural Network Model

A graph G is defined as a pair $G = (V, E)$, where V represents the finite set of *nodes* and $E \subseteq V \times V$ denotes the set of *edges*. An edge is identified by the unordered pair of nodes it connects, i.e. $e = (a, b)$, $e \in E$ and $a, b \in V$. In the case in which an asymmetric relationship must be encoded, the pair of nodes that define an edge must be considered as ordered, so as (a, b) and (b, a) represent different connections. In this case, it is preferable to use the term *arc*, while the corresponding graph will be referred as *directed*. The GNN model has been devised to deal with either directed or undirected graphs. Both edges and nodes can be enriched by attributes that are collected into a *label*. In the following we will assume that labels are vectors of predefined dimensionality (eventually different for nodes and edges) that encode features describing each individual node (f.i. average color, area, shape factors for nodes representing homogeneous regions in an image) and each edge (f.i. the distance between the barycenters of two adjacent regions and the length of the common boundary), respectively.

Graph Neural Networks are supervised neural network architectures, able to face classification and regression tasks, where inputs are encoded as graphs [6]. The computation is driven by the input graph topology, which guides the network unfolding. The computational scheme is based on a diffusion mechanism, by which the GNN updates the state vector at each node as a function of the node label, and of the informative contribution of its neighborhood (edge labels and states of the neighboring nodes), as defined by the input graph topology. The state is supposed to summarize the information relevant to the task to be learnt for each node and, given the diffusion process, it will finally take into account the whole information attached to the input graph. Afterwards, the state is used to compute the node output, f.i. the node class or a target property.

More formally, let $x_n \in \mathbb{R}^s$ and $o_n \in \mathbb{R}^m$ be the state and the output at node n, respectively, being f_w the state transition function that drives the diffusion process, while g_w represents the output function. Then, the computation locally performed at each node during the diffusion process can be described by the following equation:

$$x_n = \sum_{(n,v)\in E} f_w(l_n, l_{(n,v)}, x_v, l_v) \tag{1}$$

$$o_n = g_w(x_n, l_n) \tag{2}$$

where $l_n \in \mathbb{R}^q$ and $l_{(n,v)} \in \mathbb{R}^p$ are the labels attached to n and (n, v), respectively. As previously stated, the computation considers the neighborhood of n, defined by its edges $(n, v) \in E$. In particular for each neighbor node v, the state x_v and the label l_v are used in the computation (Fig. 1). The summation in Eq. 1 allows us to deal with any number of neighbors without the need of specifying a particular position for each of them.

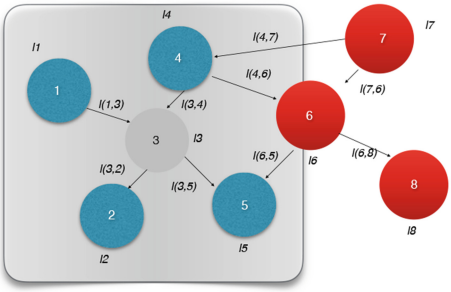

Fig. 1. The neighborhood of node 3. The state x_3 depends on the node neighborhood as $x_3 = f_w(l_3, l_{(3,1)}, x_1, l_1) + f_w(l_3, l_{(3,2)}, x_2, l_2) + f_w(l_3, l_{(3,4)}, x_4, l_4) + f_w(l_3, l_{(3,5)}, x_5, l_5)$.

Equation 1, replicated on all the nodes in the graph, defines a system of non–linear equations in the unknowns $x_n, n \in V$. The solution can be computed by the Jacobi iterative procedure as

$$x_n(t+1) = \sum_{(n,v)\in E} f_w(l_n, l_{(n,v)}, x_v(t), l_v) \tag{3}$$

that implements the diffusion process for the state computation. If the state transition function f_w is a contraction mapping, the Banach Theorem guarantees that the iterative procedure (Eq. 3) converges to a unique solution [6]. In practice, the required iterations can be limited to a maximum number.

Both f_w and g_w can be implemented by simple multilayer perceptrons (MLPs), with a unique hidden layer. The computation of Eq. 3 represents the unfolding of the so called *encoding network* (Fig. 2), where f_w and g_w are computed for each node. Basically, at each node in the graph, there is a replica of the MLP realizing f_w. Each unit stores the state at time t, i.e. $x_n(t)$. The set of states stored in all the nodes at time t are then used to compute the states at time $t + 1$. The module g_w is also applied at each node for calculating the output, but only after the state computation has converged.

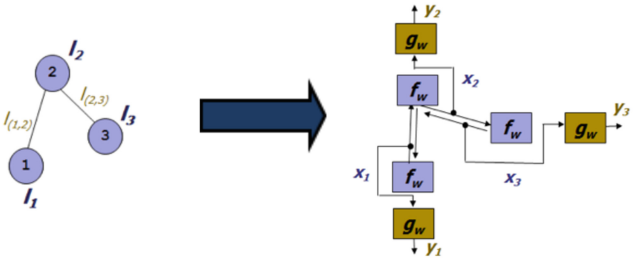

Fig. 2. Construction of the encoding network corresponding to a given input graph (from left to right). The processing units f_w and g_w are replicated for each node and connected following the graph topology.

During training, the network weights are adapted to reduce the error between the network outputs and the expected targets on the set of supervised nodes. The gradient computation is performed following the error Backpropagation scheme on the unfolding network (see [21] for more details).

3 Transductive Learning with GNNs

In the *inductive* learning approach, a model I_w is learnt by adjusting its weights w based on a set of labeled data, namely the training set [9]. Each example is processed independently of the others, but the overall statistics allow the learning algorithm to induce a general model to solve the task. Model prediction is based only on the features describing each different input object. Once the model is learnt, new unseen inputs can be processed one at a time to compute the model output (f.i. the predicted class of the pattern).

Instead, in the *transductive* framework, the algorithm is designed to exploit both labeled and unlabeled examples, taking advantage from relationships between different samples, such as, for instance, some kind of spatial regularization in the feature space (e.g. manifold regularization). The relationships among data can be exploited either in the learning or in the prediction phase, or in both of them. Basically, the prediction on the unlabeled data is obtained by propagating the information available for the "near" examples, through the given relationships between them. For instance, if n is an example at test time, then the targets available in its neighborhood may be exploited, together with the local features of n, as inputs to compute the transduced output [9]. This approach is especially useful and natural when only a small set of labeled data, that comes from an unknown stochastic process, is available. Indeed, a small sample cannot be statistically relevant for inducing a general predictive rule [10] based only on local features.

Most of the transductive approaches, available in literature, are based on graphs (see e.g. [11,12]). In recent years, these methods have been widely applied and implemented in many domains, thanks to their capability of being

adaptable to different real–world applications, such as natural language processing, surveillance, graph reconstruction and ECG classification [9].

Being capable of implementing functions on graphs, GNNs can be employed either with a pure inductive approach or with a mixture of the transductive and the inductive schemes. Given an input graph $G = (V, E)$, the set of nodes V can be split into the set $S \subset V$ of supervised nodes and the set of unsupervised nodes $U \subset V$ ($S \cap U = \emptyset$). When a pure inductive approach is used, the GNN network is given as input one (or more) instance of the graph to be learnt (e.g. the Web graph) and the targets for the supervised nodes in S, that are used only to learn the GNN parameters. The trained GNN can then be exploited to process both the original graph(s) in the learning set, to compute the output predictions for the unlabeled nodes in U, and to process unseen graphs without supervised nodes. For example, when considering a Web Spam problem in which the input is a Web graph, the class of a given page is computed by the GNN considering only the node features (e.g. the page contents) and its context in the whole graph. The labels available at the nodes in S are not considered in the computation. Basically it is assumed that the learning process was able to embed the classification rules in the trained model.

However, it should be noted that also in this case the GNN exploits the topology of the relationships among the nodes through the diffusion process used to compute the states, as defined by Eq. 3. Both the nodes in S and U are involved in this computation, but no information on the targets of the nodes in S is exploited. In this sense, we cannot consider this scheme as a proper transduction, since at test time only the node features and its context in the data manifold affect the result of the computation.

In inductive–transductive GNNs, we assume to enrich the node features with the target label such that it is explicitly exploited in the diffusion process, yielding a direct transductive contribution. The way in which targets are diffused and contribute to the final outputs is learnt from examples. We assume that the learning set contains partially supervised graphs. For each graph, we split the set of supervised nodes S into two disjoint subsets: the set of nodes used to compute the loss L and the set of transductive nodes T. For the nodes in T the available target is concatenated to the input feature vectors, whereas for the nodes in L and in U a special `null` target is used (f.i. a vector of zeros). This setting corresponds to a transduction case in which only the targets on the nodes in T are available. Given a graph in the learning set, different training examples can be generated by different splits of S into L and T. The splits can be randomly generated. The nodes in L are used to define the training loss. This way the GNN learns how to exploit the features of the nodes in V, the topology of the relationships in E and the transductive targets in T to approximate the output targets for the nodes in L. During the test phase, the set of supervised nodes S is not split and all the targets are added as features for the corresponding nodes. As before, the features for the nodes in U are obtained by concatenating the original features with the `null` label, so that the trained model computes the outputs on the nodes in U exploiting also the learnt transductive process.

Notice that, during training, it is important that L and T have no intersection, otherwise the GNN would easily learn to produce the correct output at a node by propagating directly the added target feature. The criterion used to generate the sets L and U should try to approximate the actual distribution of these two sets of nodes in the test phase.

4 Experimental Evaluation

In this section, we describe the overall methodology applied to evaluate the proposed GNN transductive–inductive scheme. In particular, we describe the datasets, synthetic and real, the setup for the experiments, and finally we present the results.

4.1 Datasets

The evaluation of the inductive–transductive approach for GNNs has been performed on two synthetic datasets. The first one for subgraph matching, the other one for clique detection. Moreover, we tested the model on real–world benchmarks, i.e. the WEBSPAM–UK2006 dataset[1] and the traffic–flow graph of England[2].

Subgraph Localization. Given a graph G, the subgraph matching problem consists in finding a subgraph S, of a given dimension, in G. In a more formal way, the task is that of learning a function τ, such that $\tau_S(G, n) = 1, n \in V$, when the node n belongs to the given subgraph S, and $\tau_S(G, n) = -1$, otherwise [15]. The problem of finding a given subgraph is common in many practical problems and corresponds, for instance, to finding a particular small molecule inside a greater compound [16]. An example of a subgraph structure is shown in Fig. 3. Our dataset is composed of 700 different graphs, each one having 30 nodes. Instead, the considered subgraphs contain 15 nodes.

Clique Localization. A clique is a complete graph [4], i.e. a graph in which each node is connected with all the others. In a network, overlapping cliques (i.e. cliques that share some nodes) are admitted. In a social network for example, cliques could represent friendship ties. In bioinformatics and computational biology, cliques could be used for identifying similarities between different molecules or for understanding protein–protein interactions [13]. Clique localization is a particular instance of the subgraph matching problem [14]. A clique example is shown in Fig. 4. In the experiments, we consider a dataset composed by 700 different graphs having 15 nodes each, where the dimension of the maximal clique is 7 nodes.

WEBSPAM–UK2006—The dataset has been collected by a web crawl based on the .uk domain [5]. The nodes of the network represent 11402 hosts, and more than 730775 edges (links) are present. Many sets of features are available,

[1] http://webspam.lip6.fr/wiki/pmwiki.php?n=Main.PhaseII.
[2] https://github.com/weijianzhang/EvolvingGraphDatasets/tree/master/traffic.

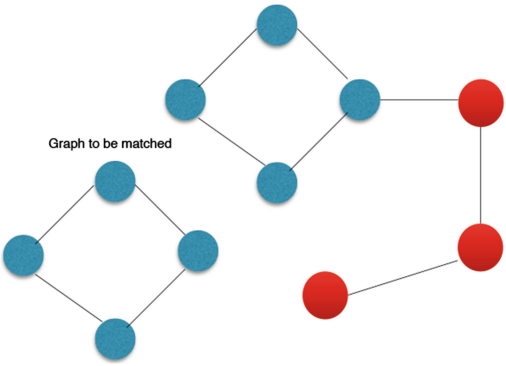

Fig. 3. An example of a subgraph matching problem, where the graph with the blue nodes is matched against the bigger graph. This task corresponds to finding the isomorphic function that maps the blue graph into the bigger one. (Color figure online)

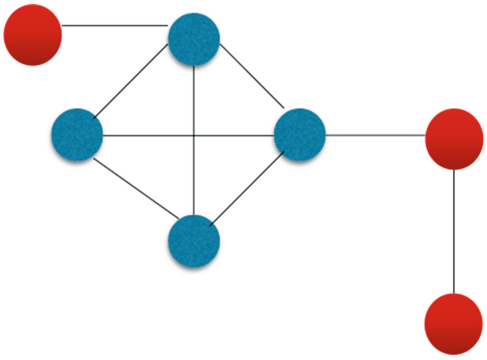

Fig. 4. An example of a graph containing a clique. The blue nodes represent a fully connected subgraph of dimension 4, whereas the red nodes do not belong to the clique. (Color figure online)

grouped into three categories: basic, link–based, and content–based. We consider only the first two categories, exploiting simple properties of the hosts, such as the number of pages and the length of their name, while in the link–based set we find also information on their in–degree, out–degree, PageRank, edge reciprocity, assortativity coefficient, TrustRank, Truncated PageRank, estimation of supporters, etc.

Traffic–Flow Prediction—This task consists in the prediction of the traffic–flow over all motorways and 'A' roads, managed by the Highways Agency in England. The problem is formulated as an edge–regression problem, since the roads are encoded as the arcs of the graph and the nodes represent the crossroads. In this case, nodes are not labeled, whereas a set of features (a label) is attached to each edge. In particular, such features represent the journey times and speeds,

estimated using a combination of sources, including Automatic Number Plate Recognition (ANPR) cameras, in–vehicle Global Positioning Systems (GPS) and inductive loops installed on the road surface. Journey times are derived from real vehicle observations and computed using adjacent time periods or the same time period on different days. The data are collected every 15 min, based on a snapshot of the traffic at that time. The problem is that of predicting the traffic flow across a certain road. We focus on a single time–stamp, obtaining 1002 nodes and 2499 edges, representing the roads.

4.2 Experimental Setup

The main goal of this paper consists in providing a comparison between the transductive–inductive and the purely inductive learning frameworks. Hence, the reported results are not to be intended as the state of the art.

In the experiments, the available datasets were split into a training, a validation, and a test set, and different conditions were defined by varying the percentage of labeled nodes: these nodes are assigned to the set T and are not exploited in the performance evaluation. In fact, it is assumed that their output is given and they are only exploited for the transduction, thanks to the diffusion mechanism that characterizes the GNN model.

We evaluated all the models with five different percentages of labeled nodes: 0, 10, 20, 30, 50. In every task, we exploited a state function implemented by a feedforward neural network with two hyperbolic tangent layers, composed by 15 and 5 neurons, respectively. Consequently, the dimension of the state is 5. For the tasks of clique searching, subgraph and WebSpam detection, the output function consists of a single softmax layer. For the flow–traffic detection we employed a linear layer.

The learning procedure was based on a simple Gradient Descent Optimizer with learning rate of 10^{-3}, except for the WebSpam task, for which we used the Adam optimizer with the same learning rate, in order to speed–up the learning procedure. We set the threshold for the convergence of the state to 10^{-3}. This cut–off is used to stop the state update loop when the difference of the state vectors in two subsequent iterations is below this value.

Moreover, in the WebSpam problem, we used the softmax output of a simple MLP as node label, inspired by the work in [20]. This feedforward network is composed by two hyperbolic tangent layers and a softmax output layer. Their dimensions are 100, 20, and 2, respectively. We adopted the cross–entropy as the loss function for the classification problems, whereas we used the mean squared error function for the traffic–flow task, which is a regression problem.

In the comparisons, we considered also the learning time, since for some tasks (f.i. subgraph and clique detection) the differences in classification performances are not so evident when giving no time constraints. Hence, we set an appropriate maximum number of epochs for each problem. For the subgraph detection problem the limit was set to 20000 epochs, whereas we used 3000 as the number of epochs to train the GNN for the WebSpam and clique detection problems, and 5000 epochs for the traffic–flow prediction benchmark.

A full–batch learning has been used in all the tasks, meaning that we adapt the weights once for each epoch (i.e. after processing all the examples in the learning set). In the case of WebSpam, the learning set consists of the whole Web graph that must clearly be processed as a unique batch. In the case of synthetic datasets, we simply considered all the graphs as belonging to a bigger disconnected graph.

4.3 Results

Figure 5 shows the trend of the accuracy on the validation set for all the tasks during the learning process.

Table 1 reports the results for the addressed tasks, when varying the percentage of labeled nodes exploited in the transductive phase. The first column, corresponding to the value 0%, represents the purely inductive case.

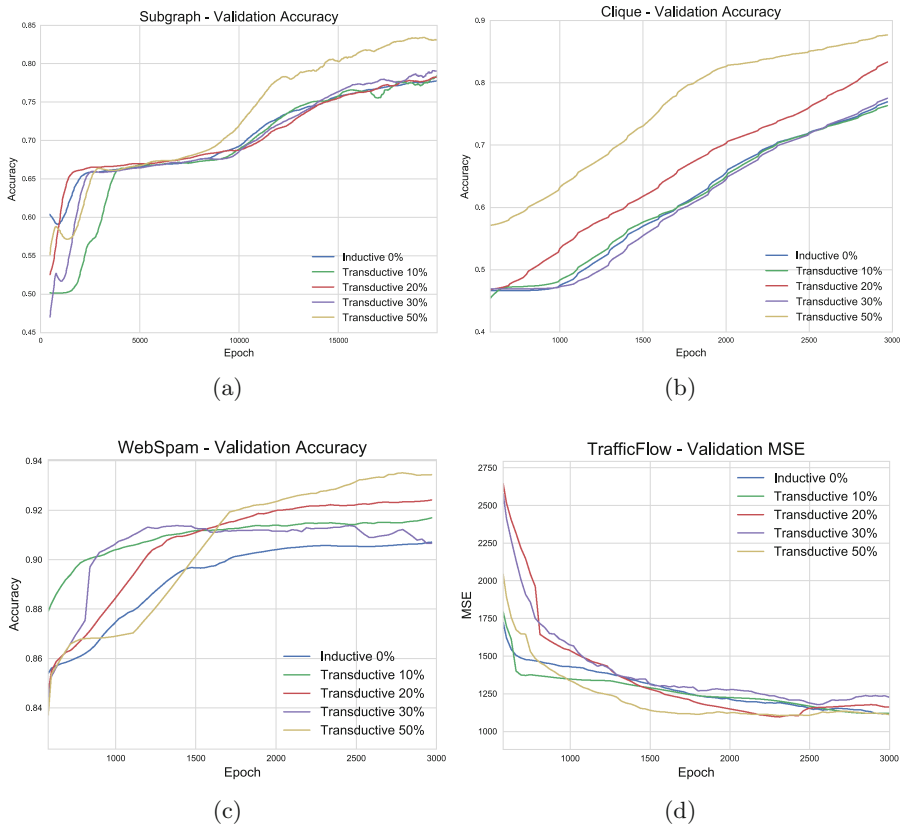

Fig. 5. Validation accuracy (or MSE) obtained varying the exploited percentage of labeled nodes, in the four different tasks.

Table 1. Mean accuracy over five runs (mean squared error for the traffic–flow benchmark), together with the standard deviation.

% of labels	0%		10%		20%		30%		50%	
Score	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
SubGraph	77.26	0.44	77.58	1.10	77.94	1.14	78.87	0.53	**83.12**	1.50
Clique	82.16	2.31	83.55	3.62	84.9	2.95	84.1	2.95	83.56	1.86
WebSpam	91.46	0.49	91.54	0.48	91.94	0.60	92.23	0.46	**92.49**	0.65
Traffic (MSE)	1123	232	1159	152	968	114	865	178	**811**	105

Transductive learning demonstrated its effectiveness on all the benchmarks. For some simple problems, like subgraph and clique detection, it is anyway difficult to obtain evident differences in absolute performance, for all the percentages of labeled nodes exploited in the transduction.

5 Conclusions

In this paper, we presented a transductive learning framework based on GNNs applied to graphs. We showed how this paradigm may improve the performances with an experimental evaluation both on synthetic and real–world problems, belonging to different domains.

Given the increasing amount of available structured data, it would be interesting to test these techniques in other application domains, ranging from mobile communications to the biomedical field, or to the large graphs provided by the online social networks, like Facebook and Twitter. It would be also of interest to investigate the properties of the diffusion process and the influence of a subset of labeled nodes over the neighbors, in order to have a deeper understanding of the GNN model.

References

1. Grindrod, P., Kibble, M.: Review of uses of network and graph theory concepts within proteomics. Expert Rev. Proteomics **1**(2), 229–238 (2004)
2. Barabasi, A.-L., Oltvai, Z.N.: Network biology: understanding the cell's functional organization. Nature Rev. Genet. **5**, 101–113 (2004)
3. Sporns, O.: Graph theory methods for the analysis of neural connectivity patterns. In: Kötter, R. (ed.) Neuroscience Databases, pp. 171–185. Springer, Boston (2003). https://doi.org/10.1007/978-1-4615-1079-6_12
4. Newman, M.E.J.: Networks: An Introduction. Oxford University Press, Oxford (2010)
5. Belahcen, A., Bianchini, M., Scarselli, F.: Web spam detection using transductive–inductive graph neural networks. In: Bassis, S., Esposito, A., Morabito, F.C. (eds.) Advances in Neural Networks: Computational and Theoretical Issues. SIST, vol. 37, pp. 83–91. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18164-6_9

6. Scarselli, F., Gori, M., Tsoi, A.-C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Netw. **20**(1), 61–80 (2009)
7. Bianchini, M., Dimitri, G.M., Maggini, M., Scarselli, F.: Deep neural networks for structured data. In: Pedrycz, W., Chen, S.-M. (eds.) Computational Intelligence for Pattern Recognition. SCI, vol. 777, pp. 29–51. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89629-8_2
8. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. In: Proceedings of OSDI 2016, pp. 265–283 (2016)
9. Bianchini, M., Belahcen, A., Scarselli, F.: A comparative study of inductive and transductive learning with feedforward neural networks. In: Adorni, G., Cagnoni, S., Gori, M., Maratea, M. (eds.) AI*IA 2016. LNCS (LNAI), vol. 10037, pp. 283–293. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49130-1_21
10. Vapnik, V.N.: The Nature of Statistical Learning Theory. Statistics for Engineering and Information Science. Springer, New York (2013). https://doi.org/10.1007/978-1-4757-3264-1
11. Arnold, A., Nallapati, R., Cohen, W.W.: A comparative study of methods for transductive transfer learning. In: Proceedings of IEEE ICDMW 2007, pp. 77–82 (2007)
12. Zhou, D., Burges, C.J.C.: Spectral clustering and transductive learning with multiple views. In: Proceedings of ICML 2007, pp. 1159–1166. ACM (2007)
13. Bu, D., et al.: Topological structure analysis of the protein–protein interaction network in budding yeast. Nucl. Acids Res. **31**(9), 2443–2450 (2003)
14. Wood, D.R.: An algorithm for finding a maximum clique in a graph. Oper. Res. Lett. **21**(5), 211–217 (1997)
15. Bandinelli, N., Bianchini, M., Scarselli, F.: Learning long-term dependencies using layered graph neural networks. In: Proceedings of IEEE IJCNN 2010, pp. 1–8 (2010)
16. Hu, H., Yan, X., Huang, Y., Han, J., Zhou, X.J.: Mining coherent dense subgraphs across massive biological networks for functional discovery. Bioinformatics **21**(Suppl. 1), i213–i221 (2005)
17. Di Massa, V., Monfardini, G., Sarti, L., Scarselli, F., Maggini, M., Gori, M.: A comparison between recursive neural networks and graph neural networks. In: Proceedings of IEEE IJCNN 2006, pp. 778–785 (2006)
18. Bianchini, M., Maggini, M., Sarti, L.: Recursive neural networks and their applications to image processing. In: Hawkes, P.W. (ed.) Advances in Imaging and Electron Physics, vol. 140, pp. 1–60. Elsevier/Academic Press (2006)
19. Bianchini, M., Maggini, M., Sarti, L., Scarselli, F.: Recursive neural networks for processing graphs with labelled edges: theory and applications. Neural Netw. **18**(8), 1040–1050 (2005)
20. Scarselli, F., Tsoi, A.-C., Hagenbuchner, M., Di Noi, L.: Solving graph data issues using a layered architecture approach with applications to web spam detection. Neural Netw. **48**, 78–90 (2013)
21. Bianchini, M., Maggini, M.: Supervised neural network models for processing graphs. In: Bianchini, M., Maggini, M., Jain, L. (eds.) Handbook on Neural Information Processing, pp. 67–96. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36657-4_3