Embedding Big Data in Graph Convolutional Networks

Gerasimos Palaiopanos
Information Technologies Institute
CERTH
Thessaloniki, Greece
gerasimos@iti.gr

Panagiotis Stalidis
Information Technologies Institute
CERTH
Thessaloniki, Greece
stalidis@iti.gr

Nicholas Vretos
Information Technologies Institute
CERTH
Thessaloniki, Greece
vretos@iti.gr

Theodoros Semertzidis
Information Technologies Institute
CERTH
Thessaloniki, Greece
theosem@iti.gr

Petros Daras
Information Technologies Institute
CERTH
Thessaloniki, Greece
daras@iti.gr

***Abstract*—Deep learning architectures and Convolutional Neural Networks (CNNs) have made a significant impact in learning embeddings of high-dimensional datasets. In some cases, and especially in the case of high-dimensional graph data, the interlinkage of data points may be hard to model.**

Previous approaches in applying the convolution function on graphs, namely the Graph Convolutional Networks (GCNs), presented neural networks architectures that encode information of individual nodes along with their connectivity. Nonetheless, these methods face the same issues as in traditional graph-based machine learning techniques i.e. the requirement of full matrix computations. This requirement bounds the applicability of the GCNs on the available computational resources. In this paper, the following assumption is evaluated: the training of a GCN with multiple subsets of the full data matrix is possible and converges to the full data matrix training scores, thus lifting the aforementioned limitation.

Following this outcome, different subset selection methodologies are also examined to evaluate the impact of the learning curriculum in the performance of the trained model in small as well as very large scale graph datasets.

***Index Terms*—Graph Convolutional Networks, Big Data, Large scale graphs, Node embeddings, Semi-supervised classification**

I. Introduction

During the last years, the ubiquity of data that can be structured as networks has boosted the research in social graph analytics. Graphs have been used to denote information in diverse areas such as communication networks, biology, linguistics and social sciences. Modelling the interactions between entities as graphs has enabled researchers understand the various network systems in a systematic manner.

Graph analytic methods can be used for numerous categories of problems such as node classification ([1]), graph classification, link prediction, node clustering, node recommendation and visualization. The problem of node classification concerns the use of the already labelled nodes of a network and its topology for the purpose of predicting a label for an unclassified node of this network.

As far as the network topology features are concerned, they can be either directly or indirectly extracted from the adjacency matrix of the network and therefore their computation is bounded from the graph size, irrespective of the category of the classification method. This paper introduces the idea that even when the network undergoes analysis with multiple subsets of the aforementioned input matrix, the graph embeddings that are extracted can be as meaningful as if the whole matrix were used.

In order to test this assumption the work of [2] is extended by training the Graph Convolutional Network (GCN) defined therein on subgraphs extracted from the original graph. The results validate that similar classification accuracy can be achieved in the same computational resources as the other methods, at a cost in the number of training epochs. This allows for the processing of large graphs (e.g., in the order of millions of nodes) as input, providing a scalable approach in the node classification task. Additionally, by applying several methodologies for formulating subgraphs of better quality, it is demonstrated that a considerable improvement occurs when specific community detection algorithms are used in the input network. In this case the training accuracy of the GCN outperforms all previous state-of-the-art methods.

The remainder of the paper is as follows: In section II a short description of related works is presented. Therefore, in Section III the problem statement and the proposed method are detailed. A thorough experimental evaluation of the proposed method is illustrated in Section IV while Section V concludes the paper.

II. Related Work

The problem under investigation emerges from the fields of graph embedding and node classification. As stated in [1], the task of node embedding concerns the mapping of each node in a new space preserving a proximity measure. The proximity of nodes can be represented by the similarity of their corresponding vectors. Deep learning (DL) based graph embeddings can be applied either on the whole of a graph or on the nodes of the graph.

The problem includes feature extraction from nodes for the subsequent classifying operation on them, such as in [3] and [4]. The relevant solutions and algorithms aim to label nodes based on the information they encode. The input can be either grid-like data (image, a video frame, sequence of video frames, text) or interconnected raw data (graphs). Another leading category of techniques is employment of random walks to propagate the labels like for instance in [5] and [6].

Convolutional Neural Networks (CNN) have proven popular for graph embedding in the area of geometric deep learning. [1] distinguishes the methods in two groups. The first contains CNNs that work in Euclidean domains and reformat input graphs to fit it. For example, [7] perform a node-ordering creating a sequence and then uses the CNN to learn a neighbourhood representation. In the second group, the deep neural models operate in non-Euclidean domains (a leading example is graphs). Each approach builds a convolutional filter which is applied in the input graph and can be either spectral ([8], [9], [10]) or spatial, i.e., a neighborhood matching ([11], [2], [12]). [13] provides an explanatory list of the geometric aspect of this research.

In [2] the authors provide a solution scalable in the number of edges. The current paper extends this classification system in the case of million scale (order) of nodes managing to have it as input in the GPU and achieving the same or higher accuracy, as well.

As for the challenge of preprocessing arbitrarily large datasets, handling big datasets as inputs usually entails a computational cost incremental to the training time or is impossible in common desktop machines. Our proposed strategy permits the use of large data in such machines keeping at the same time the whole of the information of the input and giving results nearly in the same training time as the existing methods.

Considering the arbitrary connectivity of dataset entities versus the features on nodes, since the proposed approach fragments the input to leverage the connectivity of entities (nodes), the input dataset does not need to present any structural pattern.

III. Proposed Method

The methodology extended in this work tackles the following graph-based learning problem: for each node i of a graph G, the prediction of a label y_i from a predefined set of labels. In order to make this prediction one uses the available features x_i defined for each node in the form of observed numerical attributes and the weighted edges of the graph, usually represented by an adjacency matrix A. The main assumption is that, when predicting the output y_i for node i, the attributes and connectivity of nearby nodes provide useful side-information or additional context for the training of the above CNN.

The requirement of processing datasets (particularly the corresponding adjacency matrix) as large as million of entities in the hardware resources of a common PC cannot be satisfied using the existing deep learning models.

While, in theory, feeding the model with all possible subgraphs would at some point include the entirety of the network information, this is not feasible, since the number of these increases exponentially with the number of nodes in the graph. To bypass this, the current approach uses a smaller number of subgraphs in the form of submatrices of the adjacency matrix. It is shown experimentally that, given a fixed number of nodes as size, it suffices to use the minimum number of subgraphs of this size in order to approximate the accuracy rate of different approaches, as long as every node is included in some subgraph and the computation remains efficient.

The methodology is based on the work of [2], where the authors introduce a layer-wise propagation rule that allows for neural networks to operate directly on graphs. This is motivated from first-order approximation of spectral graph convolutions, and in their work they demonstrate how a multi-layer Graph Convolutional Network model can be used for embedding network structure information.

The GCN model uses the following layer-wise propagation rule:

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

where $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections, $\tilde{D}$ is the degree (diagonal) matrix $\tilde{D} = \sum_j \tilde{A}_{ij}$ and is positive semidefinite, $W^{(l)}$ is a layer-specific trainable weight matrix and $\sigma(\cdot)$ denotes an activation function, such as ReLU. Finally, $H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the l^{th} layer.

One of the major restrictions of this method, especially for the use of GPUs, where memory resources are limited, is the size of the adjacency matrix in equation (1). When the number of nodes N is very large, as is usually the case in social network graphs, this makes the operation prohibitive.

Nevertheless, following the equation (1), each node embedding includes information about the number and quality of all the neighbors of the node (also called "first order neighbors"). Recursively, for the l^{th} layer, the embedding information contains the information embedded in the $(l-1)^{th}$ layer.

For the graph $G = (V, E)$ and a predefined dimensionality d, $d << |V|$, the aforementioned problem of node embedding is to convert each node into a d-dimensional vector, so that nodes that are "close" in the graph are denoted by similar vectors. The graph property is quantified by various measure functions that capture the proximity of nodes ([1]).

Since the embedding receptive field is the k^{th} order neighbors ([1]), one only needs to include the k^{th} order neighborhood subgraph and can remove the irrelevant information during the calculation of the embedding.

Arranging the data in subgraphs leads to the use of submatrices S_i of the adjacency matrix A:

S_i is the adjacency matrix of the undirected subgraph $\mathcal{G}_i$ of $\mathcal{G}$, with K nodes and $\tilde{S}_i$ each version with added self-connections in the form of I_K, i.e. the identity matrix of

TABLE I
STATISTICS FOR THE CORA[14], NELL[2] AND SOCIAL SPAMMER[15] DATASETS.

Dataset	Type	Nodes	Edges	Classes	Features	Label rate[a]
Cora	Citation network	2,708	5,429	7	1,433	0.052
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001
Social Spammer	Social network	5,607,447	858,247,099	2	4	0.05

[a]Rate of known labels in the semi-supervised classification experiments

Algorithm 1 Training using subgraphs

```
Input: Graph $\mathcal{G}(V, E)$
    Set of subgraphs $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, ..., \mathcal{G}_N$ of $\mathcal{G}$, corresponding
    adjacency matrices $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, ..., \mathcal{S}_N$ and corresponding
    feature matrices $X_1, X_2.X_3, ..., X_N$.
Initialization: Added self-connections to matrices:
    $\tilde{\mathcal{S}_1}, \tilde{\mathcal{S}_2}, \tilde{\mathcal{S}_3}, ..., \tilde{\mathcal{S}_N}$
1: for i=1 to N do
2:     $H^0 = X_i$
3:     $\tilde{D}_{mm} = \sum_n (\tilde{\mathcal{S}_i})_{mn}$
4:     for l=0 to LAYERS - 1 do
5:         $H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{S}_i \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$
6:     end for
7: end for
8: return trained CNN
```

this subgraph, $\tilde{D}_{mm} = \sum_n (\tilde{\mathcal{S}_i})_{mn}$ following appropiately the equation (1).

In order to better arrange the nodes into subgraphs, the experiments focused on a curriculum strategy (following ideas from [16]) that aids the learning process capture the structural information of the input: the input graphs were partitioned into communities. To this end, community detection algorithms were employed, in order to partition the vertex set into groups with high concentrations of edges in the interior of each group and low concentrations (loose connectivity) between every pair of groups. These communities were used to obtain samples as input to our network.

In real-world graphs (as our datasets) the distribution of the edges is not only globally, but also locally inhomogeneous with high concentrations of edges within special groups of vertices and low concentrations between these groups. As stated in [17], "communities", also called clusters or modules, are those groups of nodes into which a graph (network) is divided, with dense connections internally and sparser connections to the rest of the groups. To some extent, they can be considered as separate entities with their own autonomy. So, as far as the current work is concerned, they are used within the context of training because it makes sense to evaluate them independently of the graph as a whole and are expected to share common properties in terms of classification (e.g., to have the same label).

A first choice of a community detection algorithm is the "Label propagation" version, where vertices are initially given unique labels (e.g., their vertex labels). At each iteration, a sweep over all vertices, in random sequential order, is performed: each vertex takes the label shared by the majority of its neighbors. If there is no unique majority, one of the majority labels is picked at random. In this way, labels propagate across the graph: most labels will disappear, others will dominate. The process reaches convergence when each vertex has the majority label of its neighbors. Communities are defined as groups of vertices having identical labels at convergence.

The employment of the modularity criterion [17] gives better partitioning and a greedy approach has been introduced by Blondel et al. for the general case of weighted graphs known as the "Louvain" method. Initially, all vertices of the graph are put in different communities. The first step consists of a sequential sweep over all vertices. Given a vertex i, one computes the gain in weighted modularity coming from putting i in the community of its neighbor j and picks the community of the neighbor that yields the largest increase, as long as it is positive. At the end of the sweep, one obtains the first level partition. In the second step communities are replaced by supervertices and two supervertices are connected if there is at least an edge between vertices of the corresponding communities. In this case, the weight of the edge between the supervertices is the sum of the weights of the edges between the represented communities at the lower level. The two steps of the algorithm are then repeated, yielding new hierarchical levels and supergraphs. At some iteration, modularity cannot increase any more, and the algorithm stops. This more credible output is at the expense of higher time cost in comparison to the previous method: $O(|V|log|V|)$.

IV. EXPERIMENTS AND RESULTS

In order to prove our assumptions, the first experiments were carried out for semi-supervised document classification in a citations network dataset, and spammer classification in a social network graph dataset, all used in a fully and semi-supervised manner.

A. Datasets

The Cora dataset [14] consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

NELL is a dataset extracted from the knowledge graph introduced in [18]. The pre-processed version [19] of the data is used from [20]. A knowledge graph is a set of entities connected with directed, labeled edges (relations). As described in [19], representing by e_1, r, e_2 the head entity, the relation and the tail entity correspondingly, the knowledge-base formulation of the relation is the tuple (e_1, r, e_2). A graph is constructed using the entities as nodes. Each entity is represented as a node and for each relation r two more nodes, denoted as r_1 and r_2, are added in the graph. Finally the edges of the graph are created by linking each pair of nodes (e_1, r_1), (e_2, r_2) for each tuple (e_1, r, e_2). Entity nodes are described by sparse feature vectors. The number of features in NELL were extended by assigning a unique one-hot representation for every relation node, effectively resulting in a 61,278-dimensional sparse feature vector per node. The semi-supervised task here considers the extreme case of only a single labeled example per class in the training set. The binary, symmetric adjacency matrix from this graph was constructed by setting entries $A_{ij} = 1$, if one or more edges were present between nodes i and j.

Both of these datasets are fairly small, in the sense that the entire model and training data can fit in a modern GPU for the training procedure. Nonetheless, the benefit of being able to train the model using subgraphs of the network becomes evident in larger datasets such as social networks.

The Social Spammer dataset [15] is a social network graph dataset originally published for the task of identifying the spammer users based on their relational and non-relational features. It contains 5.6 million users of the Tagged.com social network website and 858 million links between them. Each user has 4 features and is manually labeled as ”spammer” or ”not spammer”. Each link represents an action between two users and includes a timestamp and one of 7 anonymized types of link.

From the 4 user features, the *userId* is used as the node index and there is a user feature vector created by concatenating the one-hot vectors of *sex* and *ageGroup* with the *timePassedValidation* features. From the relations, the temporal (*day* and *time*) information and the *relation* type are discarded and there is a weighted undirected graph created. For the training set 4.5 million users are randomly selected and 0.5 million users are used for validation purposes. The testing set is comprised by the remaining 607 thousand users.

With this dataset, there were experiments implemented both in a fully and a semi-supervised manner. For the semi-supervised experiments 95% of the training set nodes was randomly selected and their label was deleted. The dataset statistics are summarized in Table I.

B. Experimental setup

The graph convolution layer from the Keras implementation [20] is used and the same experimental setup as in [2] is followed. Therefore a 5 layer model is employed: an input layer and alternating Dropout and GCN layers.

For the Cora dataset, the first GCN layer uses 16 hidden units and the second GCN layer has 7 hidden units. The activation function of the first layer is ReLU and a constraint of L2 regularization of $5 \cdot 10^{-4}$ is applied on the kernel weights. Both the Dropout layers mask 0.5 of the input features. The second GCN layer is the last layer of the model, so the activation function applied is the softmax. For the training process an Adam solver with a learning rate of 0.001 is used. The loss function is categorical cross-entropy.

The node features are pre-processed by applying standard normalization and the localpool renormalization trick from [2] is used for the neighborhood features. The model is trained for 500 epochs.

For the NELL dataset 64 hidden units are used in the first GCN layer (instead of 16), the dropout is reduced from 0.5 to 0.1 and the L2 constraint is changed from $5e^{-4}$ to $1e^{-5}$.

On the other hand, the Social Spammer dataset is much larger, so a larger model with more parameters is needed. Thus a model with 3 GCN layers and a fully connected classification layer is employed. As previously, before each layer a dropout of 0.1 is applied but no weight regularization was perfomed. Here the Adam solver is applied, too, but with the default Keras learning rate of 0.01.

Since the number of spammers is much smaller than the number of non-spammers, the ratio of spammer to non-spammer users is calculated and the loss function was modified to give higher weight to the under-represented class in order to have a more balanced learning. For the semi-supervised experiments, the loss is computed only from the nodes that have a “known” target label.

As noted in [20], the initialization and dropout schemes of Keras and Tensorflow (used in the original paper) are different, so the reported accuracy of the original GCN model is different than the one reported by this paper.

C. Subgraph creation

In order to split the original graphs to smaller subgraphs of the same size (K nodes) that would fit in our GPU, a random split methodology is initially used. To avoid dividing by zero while calculating the loss, at least one node with a known label in each subgraph is added. The rest of the nodes are randomly picked so that all nodes belong to exactly one subgraph. The creation of random, same-sized subgraphs is repeated in each training epoch.

This procedure of fragmentation is very likely to create subgraphs with nodes that have very sparse connectivity. This phenomenon leads to subgraphs that practically do not contain any of the inherent connectivity features of the original network. Even though it is assumed that after numerous training epochs an adequate number of densely connected subgraphs will be shown to the model allowing for the extraction of the inherent network features, a smarter way of creating the subgraphs may lead to faster training times.

To first test this theory the “Label Propagation” [21] algorithm for community detection in large-scale networks is applied, using the “iGraph” [22] library. This algorithm works by initially assigning a unique community label to every node. The community label to be assigned in the node in the next iteration is determined by the nodes’ neighbors. It is assumed that each node in the network chooses to join the community to which the maximum number of its neighbors belong, with ties broken uniformly randomly. As the labels propagate, densely connected groups of nodes quickly reach a consensus on a unique label. This label propagation continues until the are no more changes in the node labels. At the end of the propagation process, nodes having the same labels are grouped together as one community.

By design, each vertex has more neighbors in its community than in any other community. The time complexity of each iteration of the algorithm is $O(|E|)$ and the number of iterations to convergence appears independent of the graph size, or growing very slowly with it, so the technique is really fast for sparse graphs.

When the Label Propagation method is applied in the Social Spammer dataset, the outcome is a very small number of communities containing the majority of the nodes and the remainder of the nodes are assigned to singleton communities. This community detection is deemed very poor for the purposes of this paper and therefore the Louvain [23] algorithm from the iGraph [22] library was then applied.

This method consists of repeated application of two steps. The algorithm is initialized with each node in its own community. The first step is a “greedy” assignment of nodes to communities, favoring local optimization of modularity. For each node in the graph the modularity change is computed for each of the node’s neighbors. If none of these modularity changes is positive, the node remains in its current community. If some of the modularity changes are positive, the node is removed from its current community and moved into the community where the modularity change is most positive. This process is repeated for each node until one pass through all nodes yields no community assignment changes.

The second step is the definition of a new coarse-grained network, based on the communities found in the first step. In this network, the newly discovered communities are the nodes. The relationship weight between the nodes representing two communities is the sum of the relationship weights between the lower-level nodes of each community.

These two steps are repeated until no further modularity-increasing reassignments of communities are possible.

For both methods of community detection that are used, the subgraphs are created firstly by randomly picking a community. Then the subgraph was filled with members of this community up to the preset size. If the community was exhausted, members of the next community were used. Consecutive subgraphs were created in the same manner, continuing with the already selected community.

In the experiments subgraph sizes of 50%, 12.5% and 4% of the dataset size were tested.

D. Metrics

For evaluating the performance of the model in this setup of feeding the graph data in the form of subgraphs, two metrics were employed: Accuracy and AUROC.

$Accuracy$ is the proportion of correct predictions among the total predictions made and is calculated by:

$$\text{Accuracy} = \frac{\text{True}_{c_1} + \text{True}_{c_2} + ... + \text{True}_{c_n}}{N} \tag{2}$$

where $True_{c_n}$ is the number of correctly predicted nodes of the n^{th} class, while N is the total number of evaluated nodes.

While $accuracy$ is a favored way to measure the ability of each method to correctly classify a node throughout the bibliography, when dealing with data that have class imbalance, it is not very informative: by always predicting the most dominant class, the scores will be very high.

$AUROC$ (Area Under Curve - Receiver Operating Characteristic) is the calculated area under a receiver operating characteristic (ROC) curve and is calculated by ploting parametrically $TPR(m)$ versus $FPR(m)$ with m being the probability threshold used to classify correctly a prediction for each class.

$$AUROC = \int_{\infty}^{-\infty} \big(TPR(m) - FPR'(m)\big)\, dm \tag{3}$$

The TPR (True Positive Rate) and FPR (False Positive Rate) are defined as:

$$TPR = \frac{TP}{TP + FN} \tag{4}$$

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

where TP is the number of correctly classified samples that belong to the class, TN is the number of correctly classified samples that do not belong to the class, FP is the number of misclassified samples that belong to the class and FN is the number of misclassified samples that do not belong to the class.

The area under the curve summarizes the performance of a classifier for all possible thresholds and is equal to the probability that the model has higher confidence for a randomly chosen correctly classified node than for a randomly chosen misclassified node, when using normalized units [24].

E. Evaluation of using subgraphs

For the first task of evaluating the use of random subgraphs during the training process of the GCN model, the performance results are summarized in table II. Reported numbers denote classification accuracy in percentages. The Planetoid and GCN original method results are taken from [2]. The baseline result is from a two layer MLP model following the same configuration as the GCN model. The “GCN whole input” result is the accuracy score that is achieved in the run of the GCN model without actually creating any subgraphs (the input is the whole of the graph adjacency matrix).

As observed in the results, when the graph is split into random subgraphs, the GCN model attains classification accuracy comparable to the GCN’s accuracy with the whole of the input (deteriorating only 1-2 percentile units).

As expected, because of the randomness in the subgraph sampling process, when the number of created subgraphs increases, the model cannot generate as good embeddings in the same amount of training epochs.

It would be expected theoretically that all possible permutations of the nodes would be needed, in the form of subgraphs,

TABLE II
SUMMARY OF RESULTS FOR DIFFERENT SIZES OF RANDOM SUBGRAPHS IN TERMS OF CLASSIFICATION ACCURACY

Method	Cora	NELL
Baseline	61.10	39.41
Planetoid	75.70	61.90
GCN original method	81.50**	66.00
GCN Whole input	81.20	43.60*
GCN 2 Subgraphs	79.30	41.68
GCN 8 Subgraphs	78.35	39.41
GCN 25 Subgraphs	60.20	33.54

*According to [20] due to the differences in the initialization method and dropout implementations of Keras and Tensorflow, the difference in the reported accuracy is expected.
**[2]

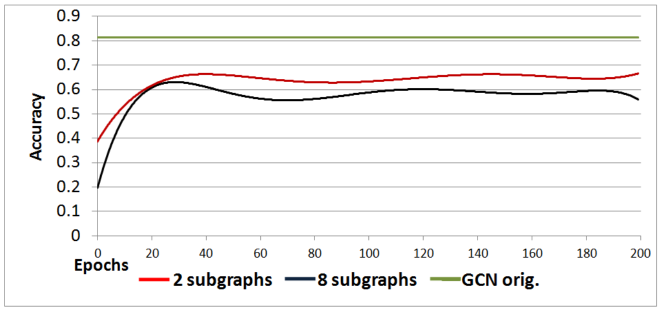

Fig. 1. Evolution of accuracy during training of the GCN model on the “Cora” dataset for the whole input (green), for 2 subgraphs (red) and 8 subgraphs (blue).

for the topology information to be properly encoded in the embeddings. However, the proximity of the above accuracy results demonstrate that the CNN is able to learn the graph information without accounting for all possible subgraphs (Fig. 1).

F. Evaluation of node classification in big data

The above observation prompted the application of the GCN model over data that normally would not fit in the memory using the original model for a node classification task.

For a dataset similar to “Social Spammer”, the original GCN model requires $|V|^2$ parameters (i.e., in the scale of 10^{13}) and thus is inapplicable in the available resources. On the contrary, the proposed solution overcomes this restriction by drastically reducing the input data dimensionality down to the scale of 10^5 parameters. Furthermore, despite the need of a number of iterations to loop through the subgraphs, the overall training procedure is within acceptable times (i.e., scale of few hours).

The previous evaluation included both a fully-supervised and a semi-supervised setup. For this task the Social Spammer dataset was used and the results are presented in table III. The classification accuracy is presented as well as the AUROC for the model with 3 embedding layers and a fully connected classification layer.

The baseline method used in this paper is a multi-layer perceptron (MLP) with 4 layers of 256 hidden units. Moreover, the current work is also compared with the work of *Fakhraei et al* ([25]) and specifically their experiments that include graph-based features, for fair comparison.

Since the size of the embedding is one of the most important aspects of the model, various sizes are tested and it was concluded that neither 16 nor 64 hidden units are enough to properly encode the relevant information. On the other hand, using 1024 hidden units allows for more trainable parameters than the dataset can train. Finally 256 hidden units were chosen. It is observed that the fully supervised classifier gives AUROC of 0.8134. This is slightly higher than 0.8072 provided by the MLP. These two results approximate *Fakhraei et al*’s classifier ([25]) of graph-based features (AUROC score: 0.817).

Hence these scores lead to the deduction that the random selection of subgraphs brings the GCN to a similar classifying capability as the MLP, which ignores the graph connectivity completely. It follows that a more comprehensive administering of this structure may enhance the classifier efficacy. Thereupon, the next step is to switch to community detection algorithms which partition the graph into communities.

G. Evaluation of the sampling method

As discussed in section III, this work explores the benefits of a more sophisticated arrangement of the nodes in subgraphs (than simple random selection). The methodology used is to perform community detection on the graph in order to detect groups of nodes that are relevant in the network topology. Then, using these communities as the focus point of the selection process, an algorithm is devised to create subgraphs that are contained entirely or in their biggest part in the communities.

The output of the Label Propagation algorithm is not ideal: there are fewer than 10 communities containing more than 80% of the nodes, while the remaining 20% of the nodes were assigned to singleton communities (containing a single member). Despite this uneven composition, the GCN model is able to create better embeddings than the random-sampling case, a fact that is depicted by the improved accuracy and AUROC scores (0.8185 in comparison to the previous 0.8134 AUROC metric for the supervised case).

Using Louvain algorithm as a sampling method yields a more balanced subgraph composition leading to a raise in AUROC (0.8187). The accuracy reaches 92.11% outperforming all previous methods.

In Fig. 2 there is the validation accuracy visualised at each training epoch. Comparing the training process of the three sampling methods, it is shown that the Louvain sampling not only produces better classification but also learns faster that both the Random and Label Propagation sampling methods (higher convergence rate).

V. Conclusions and Future Work

This paper presented a method that permits semi-supervised node classification in large datasets using graph community

TABLE III
FULLY AND SEMI-SUPERVISED NODE CLASSIFICATION USING GRAPH STRUCTURE FEATURES

Method	**Fully-supervised**		**Semi-supervised**	
	Accuracy (%)	AUROC	Accuracy (%)	AUROC
Baseline method	81.97	0.8072	70.49	0.6421
Fakhraei et al [25]	-	0.8170	-	-
Random S-GCN 16	81.52	0.7927	75.50	0.7080
Random S-GCN 64	82.33	0.8115	79.13	0.7516
Random S-GCN 256	85.59	0.8134	87.94	0.7777
Random S-GCN 1024	84.31	0.8120	77.68	0.7719
LabelPropagation S-GCN 256	85.95	0.8185	91.13	0.7802
Louvain S-GCN 256	**92.11**	**0.8187**	**93.25**	**0.8023**

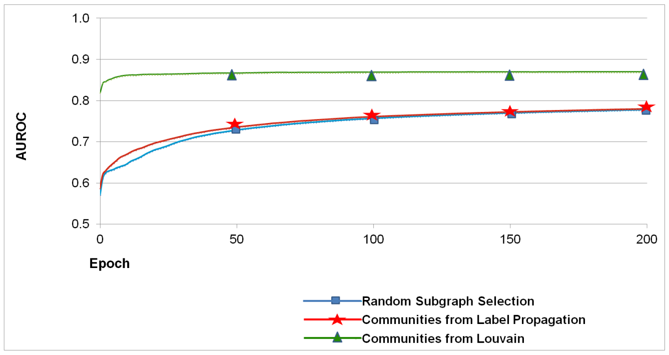

Fig. 2. Training accuracy over epochs for the semi-supervised classification task on the Social Spammer dataset using 3 different sampling methods.

algorithms. The recently developed graph convolutional networks are based on spectral graph convolutions and thus prove inadequate when the size of the network adjacency matrix is large, which is the case when dealing with big data (millions of nodes). The idea of using subgraphs, in order to feed the graph in the model to be trained, allows to scale up to millions of entities (nodes).

As analyzed in [16], when the examples are not randomly presented but organized in a meaningful order which illustrates concepts in a gradually increasing quantity and complexity, the learning procedure is expected to improve. Indeed, as it was experimentally proven in this work, the more targeted selection of subgraphs results in node embeddings superior to embeddings created from random subgraphs. Furthermore, the training time required to reach a certain classification quality is reduced when using targeted subgraphs compared to random ones.

Thus, comparing a **no curriculum** setting (random arrangement) to the **curriculum** setting of Louvain algorithm, the information is enough for the neural network to learn the whole input embedding. The experiments presented with two community detection algorithms indicate that the quality of embeddings is directly influenced by the quality of subgraphs.

An interesting research direction to follow in the future is the evaluation of other graph data organization algorithms for subgraphs creation, such as [26]. The GCN model consists of multiple embedding layers where each layer learns a different level of the graph topology. Early layers of the model learn local features while late layers learn the global features of the graph. Taking this into consideration, another research approach would be the evaluation of residual connections, e.g., as the ones that appear in ResNet [27] architectures.

ACKNOWLEDGMENTS

The work presented in this paper was supported by the European Commission under contract H2020-780355 FANDANGO.

REFERENCES

[1] Hongyun Cai, Vincent W Zheng, and Kevin Chang. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[2] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[3] Smriti Bhagat, Graham Cormode, and Irina Rozenbaum. Applying link-based classification to label blogs. In *Advances in Web Mining and Web Usage Analysis*, pages 97–117. Springer, 2009.

[4] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 496–503, 2003.

[5] Arik Azran. The rendezvous algorithm: Multiclass semi-supervised learning with markov random walks. In *Proceedings of the 24th international conference on Machine learning*, pages 49–56. ACM, 2007.

[6] Shumeet Baluja, Rohan Seth, D Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the 17th international conference on World Wide Web*, pages 895–904. ACM, 2008.

[7] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[8] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[9] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[11] Chao Zhang, Keyang Zhang, Quan Yuan, Haoruo Peng, Yu Zheng, Tim Hanratty, Shaowen Wang, and Jiawei Han. Regions, periods, activities: Uncovering urban dynamics via cross-modal representation learning. In *Proceedings of the 26th International Conference on World Wide Web*, pages 361–370. International World Wide Web Conferences Steering Committee, 2017.

[12] 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, 2009.

[13] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[14] LINQS. Cora dataset. In *https://linqs.soe.ucsc.edu/data/*, 2015.

[15] Shobeir Fakhraei, James Foulds, Madhusudana Shashanka, and Lise Getoor. Social spammer dataset. In *https://linqs-data.soe.ucsc.edu/public/social_spammer/*, KDD '15, pages 1769–1778. ACM, 2015. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2788606.

[16] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[17] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

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

[19] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[20] Thomas Kipf website. https://github.com/tkipf/keras-gcn. *Github*, 2016.

[21] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76 (3):036106, 2007.

[22] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL https://igraph.org/python/.

[23] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[24] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[25] Shobeir Fakhraei, James Foulds, Madhusudana Shashanka, and Lise Getoor. Collective spammer detection in evolving multi-relational social networks. In *Proceedings of the 21th acm sigkdd international conference on knowledge discovery and data mining*, pages 1769–1778. ACM, 2015.

[26] Jun Jin Choong, Xin Liu, and Tsuyoshi Murata. Learning community structure with variational autoencoder. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 69–78. IEEE, 2018.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.