2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Network Classification Using Adjacency Matrix Embeddings and Deep Learning

Ke Wu
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180
Email: kevinwu1024@yahoo.com

Philip Watters
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180
Email: watterspm@gmail.com

Malik Magdon-Ismail
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180
Email: magdon@gmail.com

***Abstract*—We study a natural problem: Given a small piece of a large parent network, is it possible to identify the parent network? We approach this problem from two perspectives. First, using several "sophisticated" or "classical" network features that have been developed over decades of social network study. These features measure aggregate properties of the network and have been found to take on distinctive values for different types of network, at the *large scale*. By using these classical features within a standard machine learning framework, we show that one can identify large parent networks from small (even 8-node) subgraphs. Second, we present a novel adjacency matrix embedding technique which converts the small piece of the network into an image and, within a deep learning framework, we are able to obtain prediction accuracies upward of 80%, which is comparable to or slightly better than the performance from classical features.**

Our approach provides a new tool for topology-based prediction which may be of interest in other network settings. Our approach is plug and play, and can be used by non-domain experts. It is an appealing alternative to the often arduous task of creating domain specific features using domain expertise.

1. Introduction

It is well known that large networks in practice are of different types. These types can often be distinguished by measuring aggregate properties (like average node-degree or clustering coefficient). These aggregate properties usually equilibrate to distinct values for different networks as the networks get large. It is probably no surprise that the Wikipedia and Facebook networks have very different clustering coefficients. What happens when you look at very small pieces of a network, for example as few as 8 node subgraphs? We address this question from the machine learning perspective.

Given a small piece of a large parent network, is it possible to identify the parent network?

Mathematically, given two networks G_1 and G_2, and small subgraphs S_1 and S_2, can one learn a target function f which maps $S_1 \mapsto G_1$ and $S_2 \mapsto G_2$. If so, then not only are the the parent networks distinguishable on the large, aggregate scale, but they are also distinguishable at the micro scale. We show that this is indeed the case, the a variety of popular networks are distinguishable from the practical scale.

Problem Formulation. From K parent networks $G_1, \ldots, G_K$, viewed as K classes, is sampled a data set of N training data points $(S_1, y_1) \cdots (S_N, y_N)$, where $y_n \in \{1, \ldots, K\}$ and S_n is a subgraph of G_{y_n} (this is a standard multiclass learning from data setup, see for example [1]). The subgraphs S_n can be of different sizes. The output of the learning process is a hypothesis $g : S \mapsto \{1, \ldots, K\}$. The out of sample performance of g is a measure of how distinguishable the networks $G_1, \ldots, G_K$ are at the local scale. Figure 1 shows the the flow chart of our method.

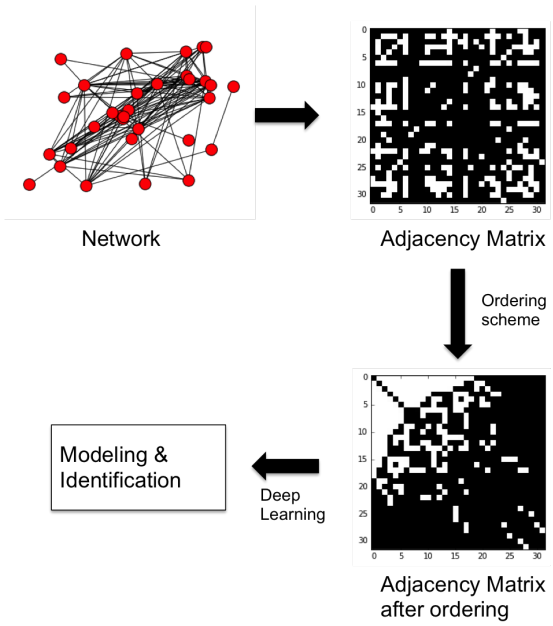


Figure 1: Work flow of the current study. A network is first represented as an adjacency matrix, and then the matrix is ordered using our ordering scheme. Finally, the ordered adjacency matrix is used as input features for deep learning modeling.

IEEE/ACM ASONAM 2016, August 18-21, 2016, San Francisco, CA, USA
978-1-5090-2846-7/16/$31.00

Our Contributions. We make two contributions. The first is an approach toward solving the machine learning problem of predicting the parent network from an observed small subgraph. We present two solutions of comparable prediction performance but of a distinct nature: 1) We feed standard classical features that have been developed by network scientists over decades into a "simple" machine learning framework (decision trees – random forests– and linear models – logistic regression). The rationale here is that if you have sophisticated, well constructed features developed by domain experts, then you do not need sophisticated machine learning technology to get good out-of-sample performance. 2) We develop a novel topological feature based on embedding the adjacency matrix of the topology into an image. To do this it is necessary to "normalize" the embedding so that it can accommodate subgraphs of different sizes. Our novel feature can be applied to any network in any domain, where as classical features created for a specific domain may not be transferable to another domain. Our image feature is informative (as visual inspection indicates) yet it is "raw" in the sense that to get good performance it is necessary to capture the essential properties of the topology into higher-level features. Hence we apply a more sophisticated machine learning framework, deep Learning, which is able to capture those higher-level features in internal layers of the network. Our experiments reveal that our novel image feature together with deep learning is as good or better than the classical features. Our image embedding approach does not require domain expertise, and so it is "plug and play", and thus an appealing alternative to the often arduous task of creating domain specific features using domain expertise.

Our second main contribution is to apply our two machine learning frameworks to parent classification from a small subgraph. Our results show upwards of 70% prediction accuracy with as few as 16-node subgraphs, and upwards of 80% prediction accuracy with 32-node subgraphs in a 5-class problem. These results are well above random performance which is 20%. The conclusion: networks are distinguishible at the local scale.

Related Work. To our knowledge, our specific problem has not been studied explicitly: to identify a parent network from a small subgraph. Our work addresses the issue of whether networks are structurally similar at a local level. Similar questions have been addressed at a more global level [2]. Network classification in a more traditional setting has received some attention: the goal is to distinguish different types of network (eg protein function) from topological information. Naturally graph-based approaches are typical, and there are two main classical methods for network/graph classification. One is using kernel methods [3], [4], [5], [6], [7], [8]. The idea is to compute the similarity (kernel function, $K(S_1, S_2)$) between two graphs S_1 and S_2. The value of $K(S_1, S_2)$ can be used for kernel based method such as SVM [9]. While this method has achieved great success in many problems, it requires a good choice of the Kernel function K, which is often not straightforward. Scalability of Kernel approaches is also a concern since the computation time for popular kernel functions for graphs with d nodes is $O(d^3)$ to $O(d^6)$ (on sparse graph the time could reduce to $O(d^2)$ [10]). Further, deploying the kernel for prediction requires $O(n^2)$ time. Another approach is to compute topological features and represent a graph as a vector of features. This approach has been used in graph classification [11], social network analysis [12], and other application areas like cheminformatics [13]. The "classical" feature based approach is popular because many simple topological features can be easily computed, such as number of triangles and average node degree. However, sophisticated features can be difficult to design and their importance can only be verified on a case by case basis. Moreover, developing a "complete set" of features is to be an "endless" process. For each new application, new features may need to be designed. Some approaches compute similarities using common subgraphs as obtained by fragment searching algorithms (e.g. gSpan [14]) or use the counts of important fragments as features [15]. This idea is often called graph-fingerprint analysis in some application domains like cheminformatics. Such approaches will fail if specific fragments appearing in the test set do not appear in training.

Our adjacency matrix image embedding has two advantages. First, the computation time can be maintained as $O(nd^3)$ and GPU computing technology can be applied to accelerate the process. Second, the adjacency matrix contains all the information of a network (it is a "lossless" representation), analogous to the pixels in pictures and signals in audios. The challenge then becomes two fold: 1) there are factorial number of ways to present an adjacency matrix to a classifier, which means one must align the adjacency matrices to preserve intrinsic topological properties. Mueller et al. [16] compared several ordering algorithms for visualization purpose and discover that BFS can provide some benefits for real-world graphs. Here we develop a BFS-based ordering algorithm using the node connectivity as the main factor to rank the nodes. Second, it is necessary to use more sophisticated machine learning techniques that are capable of learning higher-level feature representations from the raw adjacency matrix. That this is possible will become evident from the clear patterns in our image embeddings for different types of networks. We did compare different ordering schemes and our BFS-based ordering performs best.

We are agnostic to the specific learning method used, so we compare results from logistic regression, random forest [17] and unsupervised pre-trained deep networks [18], [19].

2. Data and Method

2.1. Classical Graph Features

As a benchmark against adjacency matrix feature, we used 16 classical graph features, which are well known in network studies. Since we uses different sizes of networks, to avoid obvious infeasibility of "volumetric" features, features highly related to the size of a network are not used.

Table 1 summarize the meaning of the features used in this study. Detailed information can be found in the documentations of the *networkx* package [20].

TABLE 1: Classical Features.

Name	Description
Transitivity	fraction of all possible triangles
Clustering Coefficient	fraction of friends who are friends
Node Connectivity	average min-node-cut over node-pairs
Edge Connectivity	average min-edge-cut over node-pairs
Eccentricity	averaged diameter from each node
Diameter	largest eccentricity value
Radius	smallest eccentricity value
Shortest Path Length	average all pairs shortest path (APSP)
Av. degree	averaged node-degree
Number of Endpoints	number of degree-1 nodes
Av. Closeness Centrality	averaged reciprocal of APSP
% Central	% nodes with minimum eccentricity
Edge Density	fraction of realized edges
Neighbor Degree	average neighbor-degree over nodes
First Eigenvalue	largest graph-eigenvalue
Second Eigenvalue	second largest graph-eigenvalue

2.2. Adjacency Matrix Embedding

In addition to classic methods, we took a different approach to identify these subgraphs without utilizing any domain knowledge. The idea is to convert each graph into an image and then use classification techniques which have worked well on images in the past to classify the graphs. The conversion from graph to image makes use of the graph's adjacency matrix, a method of ordering the nodes in the graph, and a rescale algorithm.

Ordering Nodes Adjacency matrix is used as a base image for our data transformation process. In order for the adjacency matrix to be "informative", we used the following algorithm to perform the ordering. The ordering starts with the node with the highest degree, tie broken by the node with the largest k-neighborhood for k=2, then k=3, and so on. If a tie cannot be broken, we pick randomly. Once this node is decided, the next node in the ordering is the node with the shortest path to the first already ordered node, tie broken by the shortest path to the second already ordered node, and so on. If a tie still exist after the shortest path length comparisons, the node with highest degree will be picked. If a tie is yet unbroken after considering degree and the size of each node's neighborhood, we pick randomly (in symmetric structures, some nodes are equivalent essentially). This order starts by prioritizing the most connected nodes, then prioritizes locality over high degree.

Two important properties of this algorithm is:

1) Nodes with the same parent must be adjacent in the adjacent matrix.
2) Parent P and its first child C are separated in the adjacency matrix by a bounded range of $[D_{PP}, D_{PP} + \sum D_{Pcousin}]$, ordered by D_P, where D_P, D_{PP} and $D_{Pcousin}$ are the degrees of P, P's parent and P's cousins.

Figure 2 shows an example of the ordering on a small network. Based on our observation, the two properties of this algorithm results in two behaviors: (1) nodes in a cluster are also clustered in the adjacency matrix; (2) important nodes (with more neighbors) are ordered relatively higher than unimportant ones. The second behavior may help in denoising since a extra single node will not change the overall structure and visualization of the ordered adjacency matrix dramatically.

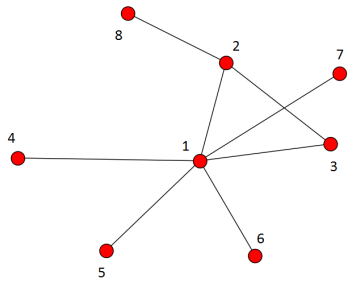
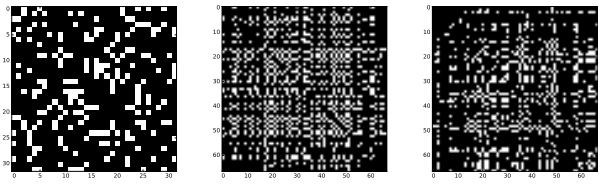
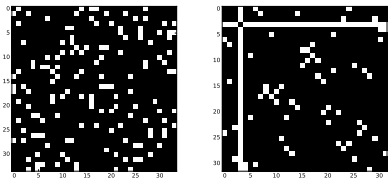


Figure 2: Illustration of our BFS-based ordering scheme for a 8-node network. The order of each node is denoted in number and clustered nodes are visited first.

Figure 3 and Figure 4 shows the adjacency matrices of classical networks (bipartite, clusters growth, random growth, torus and wheel) with random ordering and our ordering schemes. It is clear that our ordering scheme can reveal important structures of networks. Cluster growth and random growth networks are generated with two connected clusters.



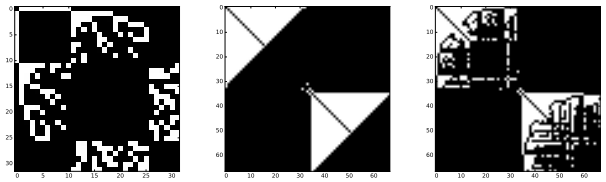
bipartite clusters growth random growth



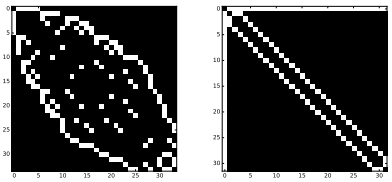
torus wheel

Figure 3: Adjacency matrices ordered randomly for five classic networks (32 nodes).

Rescale The classification methods we used required that the input feature vectors be of the same length for all



bipartite clusters growth random growth



torus wheel

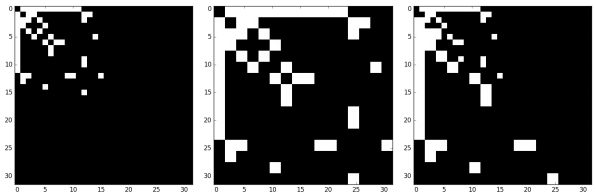
Figure 4: Adjacency matrices ordered by our ordering scheme for five classic networks (32 nodes).

examples. In order to accommodate networks as different sizes, we scaled all of the graphs in the data set to one size of matrix, and used the resulting set of image matrices for the classification task.

Two methods for rescaling have been used: image resizing and padding. In order to resize each ordered adjacency matrix, "pixels" of the adjacency matrix are linear-mapped to the final resized image. Padding is even simpler, the matrix is padding with zeros with the original matrix kept in the left top corner. This two methods represent two assumptions on the how the similarity maintained during the rescaling process. Resizing assumes that the structure of a network is similar to its subnetwork in a relative sense that the subnetwork has the same overall characteristics as the larger network, while padding assumes that a randomly sampled subnetwork's structure is similar to the "important" part of a larger network existed in that class. We also studied the "combined" representation, where the adjacency matrices from the two methods are combined to one matrix where the upper triangle (with diagonal) is the padded matrix and the lower triangle is the resized one. Figure 5 shows an example of "wikipedia" treated by the rescale methods. One can compare them to Figure 7.

2.3. Machine Learning Methods

We ran two off-the-shelf classifiers from the *sklearn* library [21], Logistic Regression (LR) and Random Forest (RF) classifiers to serve as benchmarks for our deep learning methods on the image data. Logistic regression is a quite robust linear method and can often provide good result with little computation resources, according to our experience. Random forest, a classic ensemble method of decision trees, is a highly non-linear method with very good generality in many applications. One important feature about random forest is that the prediction power increases asymptotically to optimal with the number of trees used, so a convergence



Padding Resizing Combined

Figure 5: The adjacency matrix of a 16-node network rescaled to 32 by 32 using resizing, padding and combination of the two methods. For network with the rescale ratio of 1.0, the matrices will be identical.

test on the "out of bag error" (close to cross-validation error in principle) can be used to decide how many trees to use. We believe this two methods are good representatives of the classical methods.

The success of deep learning technique on image and audio classification encourages us to apply it on this problem. We refer to [1, e-Chapter 7] for the basics of multilayer neural networks. A "deep" neural network is defined as a classical neural network with more than one internal layers. In each layer (ℓ), the output of the previous layer ($\ell-1$) is transformed into the output of the layer ℓ. The function implemented by layer ℓ is

$$\mathbf{x}^{(\ell)} = \text{activation}(\mathrm{W}^{(\ell)}\mathbf{x}^{(\ell-1)}) \tag{1}$$

where $\mathbf{x}^{(\ell)}$ is the output of layer ℓ, and the matrix of weights $\mathrm{W}^{(\ell)}$ ($\mathrm{W}^{(\ell)}$ maps the data from layer $\ell-1$ to the data in ℓ). The parameters $\{\mathrm{W}^{(1)}, \mathrm{W}^{(2)}, \ldots, \mathrm{W}^{(L)}\}$ are learned from the input data.

Multi-layer (deep) neural network achieves great success thanks to a breakthrough in the methods used to train the network [18], [19], [22]. Instead of training the whole network simultaneously, each layer is trained to "reconstruct" the input from the previous layer. This pre-training process can mitigate the local minima issues (which are very common in the classical full network training for high dimension data) and provide better regularization [23]. Generally speaking, a deep network trained in this way has the potential to learn hierarchical representations/features [23], [24] from fundamental input features.

In practice, once all the weights have been learned in pre-training, the system is "specialized" by fine tuning all the weights (supervised training on the whole network) using a few iterations of backpropagation. This process is illustrated in Figure 6.

In this study, we applies the "stacked denoising autoencoder" proposed by Vincent et al [19]. The basic idea of this algorithm is that instead of using the original input to reconstruct itself, a "corrupted" input is used to reconstruct the real input. Corruption rate (the probability of a input is flipped between 1 and 0) quantifies the level of corruption, which adds extra regularization effect to the model training.

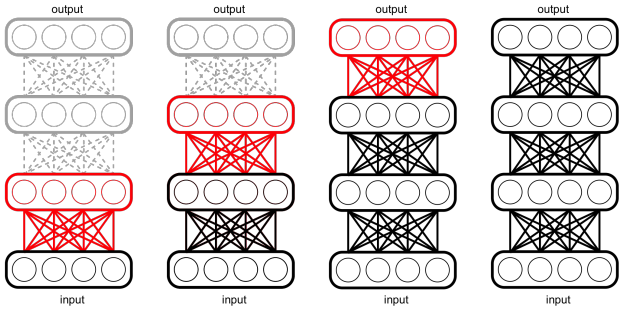


Figure 6: Layer-by-layer greedy deep learning algorithm.

The deep learning algorithm uses a symmetric autoencoder setting, where the encoder and decoder uses the same weight matrix:

$$X_{recon} = g(W^T g(W^T X + b_1) + b_2) \quad (2)$$

where g is the logistic activation function. The learning rate for autoencoder is 0.001 and for fine-tuning is 0.1. 10% of the training data is used as a validation set to determine when to stop the training process. An acceptance threshold of 0.995 is used with an increase ratio of 3 (after one valid decrease of the validation error rate, the "patience" increases by 3 times. The epoch limit is 10 for pretaining and 400 for fine-tuning. Cross-entropy cost is used for pretraining and fine-tuning. The algorithm is implemented using the *Theano* package [25].

2.4. Data

We obtained five real world network graphs from the Stanford Network Analysis Project (SNAP) [26]. The five graph domains we focused on were Social Networks (Facebook), Citation Networks (HEP-PH), Web graphs, Road networks (PA roadNet), and Wikipedia networks (wikipedia).

1000 subgraphs were from each of the graph types (5 types) for 3 subgraph sizes (32 nodes, 16 nodes, and 8 nodes). Overall we have 5000 data for each subgraph size with 32 nodes, 16 nodes, and 8 nodes. A random walk algorithm was used to extract a N-node subnetwork. The first node was selected randomly and added to the set of currently used nodes. For the next $N-1$ iterations after the first node, we consider all edges with one node in the set and one node outside the set. We pick one of these edges randomly and add the node incident to that edge which wasn't in the set. After all of the nodes are selected, we add the subgraph induced by these nodes from the original graph to our data set to maintain the correct connectivity. This algorithm simulates the process that a real human learner sampling a network naively.

Subgraphs with insufficient nodes will be removed from the data set, which could happen because some of the subgraphs have less than N connected nodes. Table 2 shows the number of data actually in use for each subgraph size. 10% data from each size is used for external test.

TABLE 2: Data set used for each subnetwork size

Size	Citation	Facebook	Roadnet	Web	Wiki
8	995	1000	998	992	999
16	995	1000	999	980	1000
32	998	1000	1000	988	1000

Figure 7 shows typical adjacency matrices of the subnetworks used in this study. Compared to Figure 4, the real world networks can be seen as containing components of classical networks. For example, the adjacency matrix of facebook is similar to cluster growth and random growth. Road net is similar to torus. Wikipedia, citation and web are similar to the random growth (one cluster).

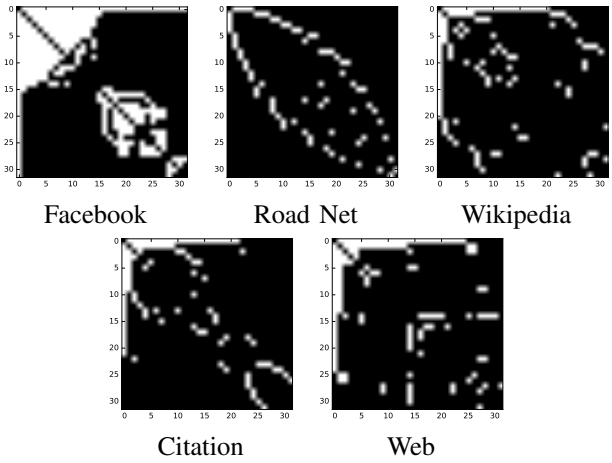


Figure 7: Examples of the ordered adjacency matrices for the five types (32 nodes).

3. Results

3.1. Classic Features

Five sets of data are used for experiment. The first three each consist of subgraphs with the same number of nodes, either 8, 16, or 32 nodes. The fourth and fifth data sets are the combined data sets put together. Logistic regression (LR) and random forest (RF) on each data set using classic features. For both methods, we used a fixed set of hyperparameters (For LR, the regularization constant was 1.0 and for RF, 500 trees were used after a convergence test on the number of trees). For each experiment, we report the test accuracy averaged over a 10-fold cross-validation (Table 3).

Two observations can be found there. First, as the number of nodes increases, the model quality increases significantly. It is clear that a too small network does not have informative "patterns" as a large network should have. Even though this observation seems trivial, it actually implies that

TABLE 3: Prediction accuracy using classic features with logistic regression and random forest.

Method	8x8	16x16	32x32	16 & 32	8 & 16 & 23
LR	0.548	0.706	0.830	0.753	0.664
RF	0.530	0.726	0.855	0.789	0.704

features based on localized fragments (as small as to eight nodes) may not help. A fragment, such as a triangle or a four-member cluster, may exist in any type of networks. However, the actual patterns that distinguish one network from the other may be represented by a more complicated delocalized structure, instead of localized fragments. This observation also motivates the usage of deep learning: Even though some classic features can capture similar information in a pre-defined way, deep learning may be able to "learn" better features automatically. Second, the prediction performance using the combined data is quite close to the average of those using individual data sets. The existence of other data sets does not seem to influence the prediction performance much, in either direction. A possible reason is that the features of the 8-node networks are too "vague" to either support or confuse classification tasks for other data sets. Deep learning using the classical features generate similar result as RF (data not shown).

We here list the top six most important features with their importance score for the "16 & 32" data set computed from random forest (random forest uses a permutation-type of sensitivity analysis to assess the feature importance) in Table 4. It seems that features describing the clustering effect are important.

TABLE 4: Top six important features used by random forest for 32x32 subnetworks.

Feature	Importance
assortativity	0.167
average path length	0.115
average shortest path	0.106
average clustering coefficient	0.087
largest magnitude eigenvalue of adjacency matrix	0.086
second largest magnitude eigenvalue of adjacency matrix	0.072

Figure 8 shows the prediction performance's change with the number of features for the same data set and method. For each number of features, 20 times of random sampling (without replacement) of f features are done from the total 16 features and the model is built with the same settings as previously mentioned but only predict one test set (instead of 10-fold cross-validation). The data point shows the median and the error bars represent the the maximum and minimum value. In some rare cases, a particular subset of features may provide a higher prediction accuracy compared to using all features. It is mainly due to the random nature of the algorithm and possibly also influenced by the choice of the test set. From the figure, we argue that to improve the prediction accuracy further, a new and more sophisticated feature must be used. However, how to find that feature is unknown.

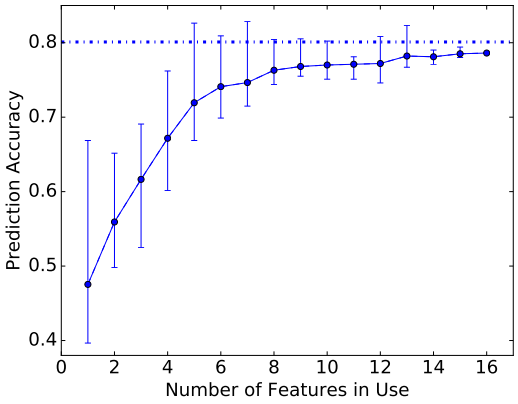


Figure 8: Prediction performance of random forest with different choice of classical features on the combined data sets of 16x16 and 32x32. The data point is the median and error bars shows the max and min values. The dotted line is the prediction accuracy of our method.

3.2. Adjacency Matrix Feature using Classic Methods

To evaluate the feasibility of the ordering scheme proposed in this study, other three schemes were used for comparison. The performances are listed in Table 5. Random ordering is the most natural way of represent an adjacency matrix. Degree ordering is to order the nodes by the connectivity degree with random tie-breaking. PageRank [27] is used widely in web network to assess the importance ("fame") of a webpage, this algorithm results in high rankings of pages with large degree and their neighbors.

TABLE 5: Logistic Regression and Random Forest Classifiers using the adjacency matrix with various ordering strategies for various sizes of data sets. LR is Logistic Regression. RF is Random Forest.

Ordering	Method	8x8	16x16	32x32
BFS-Based	LR	0.542	0.705	0.780
BFS-Based	RF	0.518	0.698	0.789
Random	LR	0.400	0.458	0.490
Random	RF	0.468	0.548	0.604
Degree	LR	0.536	0.675	0.768
Degree	RF	0.522	0.681	0.777
PageRank	LR	0.525	0.663	0.771
PageRank	RF	0.527	0.654	0.743

It is clear that our ordering scheme actually performs the best compared to other ordering schemes. This comparison verifies our hypothesis that connectivity and locality are both important in normalizing the networks. While connectivity

TABLE 6: Deep learning with adjacency matrix. A non-zero corruption rate is used for denoising auto-encoders.

Corruption Rate	8x8	16x16	32x32	16 & 32 (padding)	16 & 32 (resizing)	16 & 32 (Combined)
0.0	0.557	0.735	0.820	0.804	0.804	0.796
0.2	0.527	0.728	0.800	0.793	0.799	0.801
0.5	0.540	0.718	0.823	0.799	0.789	0.802

can be evaluated in many ways, it is impossible to maintain a perfect locality (connected pairs stays adjacent in adjacency matrix) in a 2D matrix. As we mentioned in this paper, our algorithm is able to maintain a "bounded locality", which guarantees that connected nodes are not separately "too far" in the matrix. However, it should be noted that it is unknown whether an "optimal" ordering scheme does exist, neither do we know exactly how to the define the optimality.

3.3. Deep Learning with Adjacency Matrix Feature

Deep learning with a suitable network structure is able to achieve even slightly better prediction accuracy than the classical models using classical features. The neural network used for each input dimension is listed in Table 7, following a pattern of $d^2 - 30d - 15d - 2.5d - 5$, where d is the size of the input adjacency matrix. The neural network structure is designed following a typical pattern that the network first "expand" to capture the structural complexity and then gradually reduced to provide further regularization effect. Table 6 lists the performance of deep learning using the same data sets as for the classical methods.

TABLE 7: Neural network structure used for each data set (bias nodes are used but not shown)

Data set	Network Structure
8x8	64-240-120-20-5
16x16	256-480-240-40-5
32x32 & Combined	1024-960-480-80-5

Comparing Table 6 and Table 3, deep learning using the adjacency matrix performs better than classical methods when (1) networks are small and (2) difference sizes of networks are combined. The overall performances between these two algorithms are comparable. The corruption rate does not seem to influence much.

4. Conclusion

We proposed a novel image embedding of adjacency matrices which can accommodate different sized graphs through padding or resizing (or a combination). Our feature or the "seasoned" classical features constructed by domain experts can be used as the basis of a machine learning approach to identifying parent graphs from small observed subgraphs. The surprising result is that even with just 8-node subgraphs, one can get 55% accuracy (well above random performance of 20%). Networks occurring in practice from social to citation to collaboration to physical *are distinguishable at the extremely local scale*.

Our results indicate that the classical feature approach, rich with domain expertise and the plug and play approach which uses our image embedding of the topology together with deep learning can perform comparably. This is extremely promising for the application of our image embedding to network domains where domain expert features may not be available. Our image embedding approach avoids a problem with the classical feature approach that becomes evident from Figure 8. If you choose too few features or the wrong features, the performance does not measure up to our "plug and play" image-embedding approach. In practice, one would never know whether new features are needed, a problem our near-lossless feature avoids.

There are several directions for further investigation. We see that the different orderings for the embedding can produce drastically different results. Is there a principled way to choose the ordering? It would be very interesting run our image embedding based machine learning approach to other areas where graph classification is used, such as cheminformatics. Within the deep learning framework, there are many popular approaches for pre-training the internal layers. We compared some of the popular ones based on auto-encoders.

Acknowledgement

This research was supported in part by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 and the Intelligence Advanced Research Projects Activity (IARPA) via Air Force Research Laboratory (AFRL) contract number FA8650-12-C-7202. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Army Research Laboratory, IARPA, AFRL, or the U.S. Government.

References

[1] Y. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data: A Short Course*. amlbook.com, 2012.

[2] A. Hashmi, F. Zaidi, A. Sallaberry, and T. Mehmood, "Are all social networks structurally similar?" in *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*. IEEE, 2012, pp. 310–314.

[3] H. Kashima and A. Inokuchi, "Kernels for graph classification," in *ICDM Workshop on Active Mining*, vol. 2002. Citeseer, 2002.

[4] T. Kudo, E. Maeda, and Y. Matsumoto, "An application of boosting to graph classification," in *Advances in neural information processing systems*, 2004, pp. 729–736.

[5] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *ICML*, vol. 3, 2003, pp. 321–328.

[6] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning Theory and Kernel Machines*. Springer, 2003, pp. 129–143.

[7] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs," in *Advances in Neural Information Processing Systems*, 2009, pp. 1660–1668.

[8] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Data Mining, Fifth IEEE International Conference on*. IEEE, 2005, pp. 8–pp.

[9] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[10] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *The Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.

[11] G. Li, M. Semerci, B. Yener, and M. J. Zaki, "Graph classification via topological and label attributes," in *Proceedings of the 9th international workshop on mining and learning with graphs (MLG), San Diego, USA*, vol. 2, 2011.

[12] E. Ferrara and G. Fiumara, "Topological features of online social networks," *arXiv preprint arXiv:1202.0331*, 2012.

[13] V. Sharma, R. Goswami, and A. Madan, "Eccentric connectivity index: A novel highly discriminating topological descriptor for structure-property and structure-activity studies," *Journal of chemical information and computer sciences*, vol. 37, no. 2, pp. 273–282, 1997.

[14] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. IEEE, 2002, pp. 721–724.

[15] N. Acosta-Mendoza, A. Gago-Alonso, and J. E. Medina-Pagola, "Frequent approximate subgraphs as features for graph-based image classification," *Knowledge-Based Systems*, vol. 27, pp. 381–392, 2012.

[16] C. Mueller, B. Martin, and A. Lumsdaine, "A comparison of vertex ordering algorithms for large graph visualization," in *Visualization, 2007. APVIS'07. 2007 6th International Asia-Pacific Symposium on*. IEEE, 2007, pp. 141–148.

[17] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[18] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *NIPS*, vol. 19, p. 153, 2007.

[19] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[20] D. A. Schult and P. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, vol. 2008, 2008, pp. 11–16.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[22] G. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[23] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *JMLR*, vol. 11, pp. 625–660, 2010.

[24] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proc. ICML*. ACM, 2009, pp. 609–616.

[25] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[26] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[27] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120.