WALKRNN: READING STORIES FROM PROPERTY GRAPHS

A PREPRINT

Deborah Tylor
Tylor Data Services, LLC
dtylor@tylordata.com

Mirco A. Mannucci
HoloMathics, LLC
mirco@holomathics.com

Joseph Haaga
Georgia Institute of Technology
jhaaga3@gatech.edu

October 19, 2019

ABSTRACT

WalkRNN, the approach described herein, leverages research in learning continuous representations for nodes in networks, layers in features captured in property graph attributes and labels, and uses Deep Learning language modeling via Recurrent Neural Networks to read the grammar of an enriched property graph. We then demonstrate translating this learned graph literacy into actionable knowledge through graph classification tasks.

***K*eywords** Deep Learning · Graph Mining · Random Walk · Language Model · Graph Language · RNN · GraphWave

1 Introduction

Graph provides a flexible data modeling and storage structure that can represent real-life data, which rarely fits neatly into a fixed structure paradigm or repeatable method of analysis. Graph heterogeneity, the interplay of local and global contexts, have in the past been difficult to express at once with repeatable analytical processes. Because of this challenge, graph applications historically were mostly limited to presenting this information in small networks that a human could visually inspect while reasoning over the 'story' and meaning. This approach failed to contemplate many property sub-graphs of a large graph in an automated fashion, and therefore limited the ability to conduct timely top-down analysis across the entire population of graph data.

Deep Learning is an ideal tool to help mine large graphs' latent patterns and hidden knowledge. In the past, applying wide-scope machine learning algorithms to graphs has been difficult. Methods often reduced the degrees of freedom by fixing the structure in a repeatable pattern, such as looking at individual nodes within their immediate neighborhoods so that the data could then be consumed by classification algorithms. The rich information captured by the local graph topology could be lost with such simplifications, making it difficult to derive local sub-structures, latent communities and larger structural concepts in the graph.

Our approach benefits from existing tools in the space of network machine learning and overlays a property graph with learned network attributes, thus preserving the interplay of domain-specific content with learned structural attributes in the overall the graph. We then apply flexible deep learning language modeling methods to mine the stories of its sub-graphs.

2 Background: Machine Learning and Vector Embeddings applied to graph

Describing all of the active research in the space of machine learning applied to graph is outside of the scope of this paper; however this referenced paper [3] presents a good survey of the growing number of methods developed in the area of learned network representations. We cover two paths that will be leveraged in our proposal.

2.1 Vector representations of graphs

Research in the past few years has made great strides in a class of approaches that learn, with unsupervised techniques, continuous feature representations for nodes in networks. Some of these approaches learn features that are sensitive to the local neighborhood of a node and can also be compared against the global population.

The goal of these approaches are varied and can be organized into three categories: *structural*, *informational*, and *community membership*. With these representations (stored in a suitable vector space), nodes can be analyzed in terms of their *structural* roles in their local network, the *informational* context of graph attributes, and the *communities* they belong to. Jure Leskovec and others at Stanford have contributed research and performing algorithms in this space, and we have taken advantage of two approaches, **GraphWave** and **Node2Vec** and its random walk generator.

GraphWave [4] provides an effective way to capture the structural role of nodes, such that *"nodes residing in different parts of a graph can have similar structural roles within their local network topology."* GraphWave leverages Graph Signal Processing, in particular wavelets, to discover diffusion patterns in graphs. By studying the 'heat' propagation starting at a particular node, GraphWave can recover and encode the structural setup of that node.

The Node2vec approach contributes a flexible neighborhood sampling approach which builds on its predecessor pioneer **DeepWalk** [6]. The Node2vec paper [1] gives a background of research that uses the Random Walk method to represent a network as a 'document' that is sensitive to edge weights and local topology of the starting node's surrounding nodes. Most importantly for our application, Node2vec random walks captures the edge traversal, which can contain property and type information important to understanding the neighborhood context. Both DeepWalk and Node2vec feed their walks into the SkipGram Word2Vec [2] algorithm, which is based on the hypothesis that words in similar linguistic contexts tend to have similar meaning. Applied to a graph, the net output is a learned vertex representation, capturing both structural roles in local sub-networks as well as community membership.

2.2 Deep learning in graph analysis: CNN and RNN

Deep Learning has been prominent in the last few years, chiefly due to its successes in such diverse areas as NLP and image recognition. It thus comes as no surprise that a new area of research has cropped up, marrying graph analytics and deep learning (for a survey, see [10]).

Deep learning graph classification and other supervised machine learning tasks have emerged in the area of Convolutional Neural Networks (CNNs). The **DGCNN** team (2018 see [5]), for example, developed an architecture for using the output of graph kernel node vectorization, using Weisfeiler-Lehman subtree kernel [8]. With the node vectorization, DGCNN produces a fixed sorting order of nodes to allow algorithms designed for images to run over graphs. Our work was inspired by their approach of enriching a graph with structural roles and then performing deep learning on the results.

Moving beyond the neural embedding captured in SkipGram (used by DeepWalk and Node2Vec), neural NLP architectures such as RNN (LSTMs) provide the ability to model sequential ordering and mid range dependencies in text, e.g. a word in the beginning of a sentence affecting the end of a sentence [7]. Unlike "shallow" neural embeddings such as Word2Vec, RNNs can delineate non-linear surfaces which provide a better fit for complex graph relationships in property context and network structure.

In this paper, we explore the use of RNNs, as opposed to more commonly used CNNs, as an approach to graph classification. In this blog post [11], Karpathy discusses the effectiveness of RNNs, which is due in part to its ability to process sequences. Karpathy also makes the following general statements, *"If training vanilla neural nets is optimization over functions, training recurrent nets is optimization over programs"* and *"it is known that RNNs are Turing-Complete in the sense that they can simulate arbitrary programs (with proper weights)."*

We have also chosen RNN learners due to their flexibility in accepting varying inputs. Our approach incorporates node and edge properties which may vary in sparsity and complexity throughout the graph. For example, schematically a *person* node type may have four attributes by definition, whereas a *location* node type may be defined as having only one. Also individual nodes may have incomplete sets of data attributes, which depend on the sparsity of data observations.

As a result, flexibility of the input as well as its ability to learn arbitrary programs is essential for a general graph classification tool to take advantage of the rich, potentially 'messy' and in-homogeneous data. Karpathy describes CNNs as accepting a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output. Because CNN approaches rely on each node occupying a fixed amount of space, the limiting data input model doesn't lend itself as readily to the variable nature of real-life property graphs data.

3 WalkRNN - Language Model of structurally enriched graph

Property graph can contain contextual and rich information in properties, both on the nodes and edges, specific to the subject domain and data model. Graph also captures information about larger network organizations and how the parts interact to create the whole.

The goal of our approach is to learn the stories of sub-graphs and how these stories start to follow patterns in the global population. We accomplish this through a few simple steps. Our proposed pipeline, **WalkRNN**, mixes the two streams - vector representations and deep learning of graphs - which we have highlighted in the previous section.

WalkRNN first enriches an existing property graph with featurization such as the structural role of nodes and collapses categorical and continuous data in attributes and labels into repeated 'words'. The words may be globally present within the graph. In this featurization step, our pipeline utilizes GraphWave to identify structural roles of nodes across the graph and then augments nodes' lists of properties with a label denoting its role.

The next step involves translating the enriched property graph into a block of text. We use Node2Vec's implementation of biased random walks to create 'sentences' of nodes and edges, an approach which the Node2Vec paper notes is computationally efficient in terms of both space and time requirements. An implementation in Apache Spark is also available, which can further improve performance and throughput in converting graph to text.

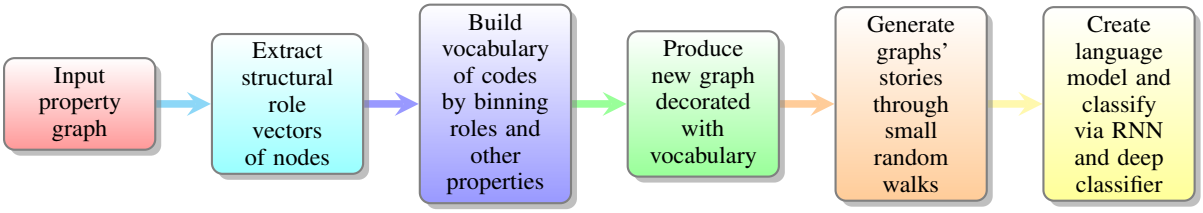
Every node spawns multiple random walks. For example, sub-structures such as triangles can lead to loops, or the walk can journey into new parts of the graph depending on the random decisions at each node. The random walks are centered at nodes which create small stories, i.e. phrases which include node roles and node and edge attributes. The stories preserve the order information in the walk. One may think of these initial steps as an attempt at generating a corpus of text with stories of the graph.

The next main step is to learn a language model based on the graph's corpus. We employ RNNs since their memory retention capability proves effective in understanding phrases beyond a bag of words by including ordering. The resulting graph language model is our first output in deep learning. We use the fast.ai library's text parsers and language model. This approach passes the documents of random walks into an RNN (AWD LSTM) which retains memory of past words, such as the node structural role or property categorical value, in sequence. The RNN then learns meaning behind the words and their relation in sequence so that the graph relationships between nodes and their local context are learned by the computer. Patterns held across the separate components are globally learned, and words that are often shared across components and patterns begin to emerge in the relationships of words.

The final step is to improve on the core language model and use it for classification (or other tasks) to prove its efficacy in domain-specific tasks. For every component, our code randomly selects walks and concatenates them together into a 'paragraph' of cuts through the sub-graph, tuned to be long enough to cover the width of the graph and to capture the gist of the component. Finally, these paragraphs per component are passed into an RNN classifier using transfer learning from the pre-trained RNN language model and predicts the label for the component.

3.1 The WalkRNN pipeline

The following diagram highlights the entire pipeline:



The pipeline begins by feeding a property graph as input. It then applies GraphWave to extract a dictionary of roles for nodes. The next step turns the calculated roles into codes, for instance a structural hub becomes hub-node and appends them as properties to nodes. Similarly the pipeline transforms existing node and edge properties into codes, or feature words, by clustering. At this point, the graph is ready for random walks of fixed length starting at various nodes per component. [1]

[1] This step can be adjusted by setting the number of hops in the graph. Essentially the larger the number of hops, or walk length, the longer the sentences will become. We can also decide either to sub-sample the set of starting nodes or perform it on all nodes in the graph. The trade-off is of course between speed and thoroughness; the larger the corpus generated with respect to the size of the graph, the more we will expect to know about its stories.

3.2 Implementation

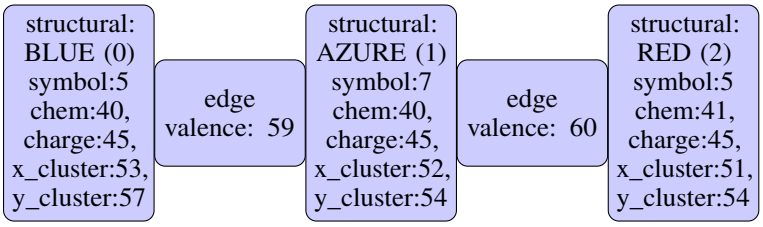
Below is a short description of the open source code bases we used to create WalkRNN. For the first step, given an input property graph, we automate an open source, python implementation of GraphWave [14]. We use the algorithm output to decorate nodes with binned structural role information.

Figure 1 is a visualization of an example component from the AIDS dataset, colored with the calculated GraphWave-based bins. In this example, we can see that the leaf nodes have been colored differently than the interior nodes which tend to vary depending on degree. The code also performs K Means clustering on quantitative, non-categorical variables, parameterized by the number of clusters. The vocabulary of structural roles and categorical and clustered quantitative node and edge properties are then one-hot encoded. A transformed graph is generated, enriched with the new vocabulary.

In the next step our code uses the open source, python Node2Vec [15] walk simulation code to generate random walks from every node in the graph. The number of walks per node and the length of the walk are adjusted manually depending on the average size of the graph. The Node2Vec implementation outputs the edges traversed (DeepWalk only includes node ids), which our code puts to use. It builds a sentence of both node and edge information as it randomly walks.

Figure 1: Sample component in AIDS kernel dataset with Graphwave based structural role colors

Here below, we can see an example section of a sentence obtained by performing a random walk within the subgraph in the previous figure. We have binned the structural roles into the colors BLUE, AZURE, and RED.



The code creates a mapping of one hot encoded numbers to the graph transformed vocabulary. The non-overlapping class values, such as in the example above, are written in a string of numbers. For a given node, the structural, symbol,

component	label	text
1	0	0 5 41 45 53 57 59 0 5 41 45 52 54 59 0 5 41 4...
2	1	0 7 40 45 52 54 59 2 5 41 45 51 54 59 2 5 41 4...
3	1	2 5 41 45 50 54 59 2 5 41 45 51 54 59 2 7 40 4...
4	1	0 5 41 45 51 57 60 0 5 41 45 53 57 60 0 5 41 4...
5	0	2 6 41 45 52 54 59 2 5 41 45 53 57 59 0 7 40 4...

Figure 2: Randoms walks over components prepared for input to be used in RNN classifier

chemical, charge, and x and y clusters are concatenated into a series of 5 numbers followed by an edge valence number

and then another node's series of five numbers and continues with this pattern. Figure 2 shows the actual text of concatenated walks for each component which is passed into the classifier.

Finally, for both the language learner and for the classifier, we have used fast.ai [16], which is built on top of pytorch. Fast.ai has "out of the box" support for text language models, including AWD LSTM RNNs, and incorporates deep learning best practices. The first step involves training the RNN language model off random walk sentences and then using this pre-trained model in an RNN classifier to predict component labels from paragraphs of concatenated walks.

4 Experimental Validation

In order to validate the WalkRNN approach and efficacy, we performed experiments on benchmark datasets and compared to representatives of state-of-the-art in graph classification. We compare to a deep learning approach and a highly performant application of graph embeddings. The code and data used to perform these experiments are available at *https://github.com/dtylor/WalkRNN*.

4.1 Experiments on graph kernel datasets

The graph kernel data sets and accompanying stats used for these experiments were downloaded from a website provided by the TU Dortmund Dept of Computer Science [13]. The data sets share a common format, which facilitates experimentation with graph classification across multiple domains. Each data set is broken into multiple components, each with a class label.

4.2 WalkRNN accuracy in graph classification tests

Our results below are compared to the DGCNN and HSGE [12] papers to illustrate how a language model can be used successfully to classify graphs. In cases where rich information is stored in graph properties (e.g. node and edge attributes and labels), our approach produces strong results.

The effectiveness of this method really shines for certain types of graphs, and when additional feature information is present in attributes, the approach often achieves greater than 90 percent accuracy in classifying the test datasets. The results below are dependent on parameters - such as dropout, learning rate, number of neural network hidden layers, random walk length, number of structural GraphWave words. Repeated runs and manual adjustments were required to fine-tune results. The D&D data set only includes node labels and edges (ie no attributes or edge labels), and the power in enriching the graph story with properties is not fully realized. MUTAG was less stable in training as there were fewer examples (only 188 components in total). Our code learns how to read the various graph domains from scratch and then learns how to predict the class label for each graph (e.g. AIDS or not AIDS). The results in Table 1 show our accuracy at predicting test graphs.

Table 1: Comparison of WalkRNN to DGCNN and HSGE Accuracy

Dataset	AIDS	D&D	MUTAG	PROTEINS
Classes	2	2	2	2
Node Labels	+	+	+	+
Edge Labels	+	-	+	-
Node Attr. (Dim)	+(4)	-	-	+(1)
Edge Attr. (Dim)	-	-	-	-
Nodes (avg)	15.69	284.32	17.93	39.06
Edges (avg)	16.2	715.66	19.79	72.82
Graphs	2000	1178	188	1113
WalkRNN	98.5	**92**	89.5	**96.4**
HSGE	**98.74±0.21**	80.3	**92.8**	76.6
DGCNN		79.4±0.9	85.8±1.7	75.5±0.9

Figure 3: Comparison

Typically our test sets were comprised of 10% random sample of the entire set of components, training 72% and validation 18%. The only exception was in the case of the larger D&D data set in which the training set was 30.6% of the components, validation 7.6% and testing 4.2%.

5 Future Directions

In this last section we highlight a few themes for future direction, each expanding the reaches of WalkRNN.

5.1 Other ways to add nodes structural roles

We have used GraphWave to enrich the graph with labels of node structural roles. There are other ways to enrich nodes' property lists with structural attributes. For instance, we could use standard graph algorithms to assess the centrality degree of all nodes, or their in-betweeness. Notice that this information concern global roles of nodes, not just local information. It may help build intuition to think of our graph as a social network. The random walks around a starting point will tell us what kind of relations the target individual has with his neighbors, but the augmented label can tells us "this fellow is a big global influencer." Given our test sets were broken into small, disjoint components, these global indicators were less important for classification accuracy.

5.2 Adding edges structural roles

In our presentation, we have simply augmented the graph (and thereby the generated language) with nodes' structural roles. The edges roles can also be detected [9]. In some scenarios, this additional step dramatically enriches the graph language and its expressiveness. Nodes can be seen as standing for nouns or named entities, edges are verbs or actions, and the edges attributes are verb qualifiers, metaphorically adverbs.

5.3 WalkRNN and dynamic property graphs

Thus far, we have learned and applied the graph language model from a single property graph. Imagine that such a graph evolves over time; in some scenarios the underlying language model may change (for instance, new properties become available), whereas in others the language stays the same. In both scenarios, it will be intriguing to read the stories of the graphs as they unfold over time.

5.4 Learning a hyper-graph language model

Our method also extends to hyper-graphs in a straightforward way. Given a property hyper-graph, we can simulate it as a property graph where some nodes stand for the hyper-edges (call them connector nodes). Once we have generated the new graph, we apply WalkRNN. From the graph language standpoint, this translates to adding collective entities to our grammar (e.g. a Facebook group would be such an entity).

References

[1] A. Grover, J. Leskovec. *node2vec: Scalable Feature Learning for Networks.* ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2016.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean. *Efficient estimation of word representations in vector space.* In *ICLR*, 2013.

[3] Daokun Zhang, Jie Yin, Xingquan Zhu, Chengqi Zhang *Network Representation Learning: A Survey* Retrieved from https://arxiv.org/pdf/1801.05852.pdf, 2018.

[4] Claire Donnat and Marinka Zitnik and David Hallac and Jure Leskovec *Spectral Graph Wavelets for Structural Role Similarity in Networks* In *CoRR*, 2017

[5] Muhan Zhang, Zhicheng Cui, Marion Neumann, Yixin Chen *An End-to-End Deep Learning Architecture for Graph Classification* Retrieved from https://www.cse.wustl.edu/ ychen/public/DGCNN.pdf

[6] B. Perozzi, R. Al-Rfou, and S. Skiena, *DeepWalk: Online learning of social representations* in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 701–710.

[7] Retrieved from https://towardsdatascience.com/beyond-word-embeddings-part-2-word-vectors-nlp-modeling-from-bow-to-bert-4ebd4711d0ec

[8] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. *Weisfeiler-lehman graph kernels*. Journal of Machine Learning Research, 12:2539–2561, 2011.

Weisfeiler-lehman graph kernels. Journal of Machine Learning Research, 12:2539–2561, 2011.

[9] B. Perozzi, R. Al-Rfou, and S. Skiena, in *http://ryanrossi.com/pubs/ahmed-et-al-pakdd17-preprint.pdf*

[10] Ziwei Zhang, Peng Cui, Wenwu Zhu Deep Learning on Graphs: A Survey *https://arxiv.org/abs/1812.04202*

[11] Karpathy, Andrej The Unreasonable Effectiveness of Recurrent Neural Networks *http://karpathy.github.io/2015/05/21/rnn-effectiveness/*

[12] Anjan Dutta1, Pau Riba1, Josep Llados, Alicia Fornes Hierarchical Stochastic Graphlet Embedding for Graph-based Pattern Recognition *https://arxiv.org/pdf/1807.02839.pdf*

[13] Kristian Kersting and Nils M. Kriege and Christopher Morris and Petra Mutzel and Marion Neumann, Benchmark Data Sets for Graph Kernels, 2016 *http://graphkernels.cs.tu-dortmund.de*

[14] Anonymous, Open Source for GraphWave *https://github.com/snap-stanford/graphwave*

[15] Grover, Aditya and Leskovec, Jure, Open Source for Node2Vec *https://github.com/aditya-grover/node2vec*

[16] Howard, Jeremy, Thomas, Rachel, and Gugger, Sylvain, fast.ai *https://docs.fast.ai/index.html*