Graph Neural Networks and the Vertex k-center Problem

Elia Chudov, Harsha Venkatesh, German Bautista

May 2020

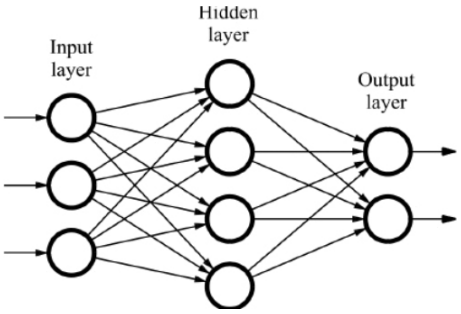
Abstract

In this project we implement two algorithms for solving the k-center problem on graph structured data, an exact algorithm based on minimum dominating sets and a greedy approximation algorithm. We then transition to a Graph Neural Network approach to this problem. To do so, we train a graph attention network first proposed in *Graph Attention Networks* by Petar Velickovic, et. al on Erdos-Renyi graphs with random edge weights to discern k-centers and k-clusters. In doing so, we develop a new label-isomorphic approach to generating feature vectors that encodes intuitive structure of a given graph. Finally, we conclude with an evaluation of our efforts and future applications of such a GNN.

1 Introduction

1.1 Neural Network

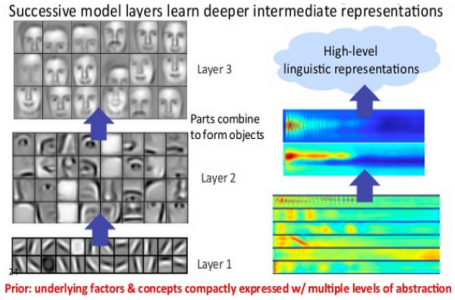
In the field of computer science, a neural network is essentially a system of neurons or nodes in which a set of data inputs is given and through a series of forward computations involving weights and thresholds, a final output is computed and the weights are "trained" to better predict or match the actual output. Data inputs can be measurements, sound frequencies, images, and even graphs. Two primary uses of neural networks is to find data clusters and to classify data. When a network attempts to find clusters i.e groups that share some kind of feature, the data is raw. The network has no prior labels for the data. Examples of labels include: 'cat', 'spam', 'jazz music'. Throughout iterations, the network learns by finding correlations amongst the data freely without labels. This is called unsupervised learning. Some uses can be finding similarities in the genome sequences of different mammals or note differences in grade distributions of a course for different sections. Clustering can also pinpoint any outliers in data such as account behavior or weather forecast. On the other hand, if the data requires a prior set of labels then we call this supervised learning. Some applications are identifying individuals or objects in group pictures, matching a blood sample to blood type, sending spam e-mails to a folder, and predicting the parent distribution of sample points.



Neural Networks have come a long way since they were first initialized in 1944 with University of Chicago researchers Warren McCullough and Walter Pitts. They had the notion that the neurons, synapses, and signal processing in the human brain could act like a computer. In contrast to the brain, a computer is a deterministic machine with finite memory so there is some limitation on computers when solving complex random tasks or doing top-down processes such as recognizing images or sounds. This task is simple for a human since we can both look at specific details and gather the general picture or main message easily. Surprisingly, some of the methods used to adjust weights in certain neural networks mimicked those of the brain.

The notion that neuroscientists, computer scientists, and mathematicians could try to give a computer a 'brain' using new mathematical methods expanded computers' versatility and efficiency through learning by experience: hence artificial intelligence (AI) came into play. Unfortunately, due to the lack of computational power as assessed in the book *Perceptrons* in 1969 by mathematicians Marvin Minsky and Seymour Papert, neural networks came

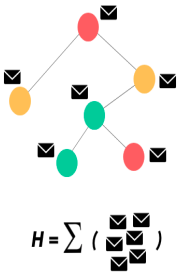
to a temporary halt. In the 1980's however, computers became more efficient and computational power increased. "Researchers had developed algorithms for modifying neural nets' weights and thresholds that were efficient enough for networks with more than one layer, removing many of the limitations identified by Minsky and Papert. The field enjoyed a renaissance" (Hardesty)



1.2 Graph Neural Networks

Moreover, graphs can be used as input data for a neural network. A Graph Neural Network (GNN) is a type of neural network that takes in the set of vertices from a graph G, an adjacency matrix to see how vertices are connected, and a matrix of vector attributes for each node. Note that a graph's adjacency matrix is not unique since vertex labels can be rearranged and still represent the graph's structure. We can associate each vertex in a graph with vector of features such as cost, color, a measurement, etc. GNN's can be used to find approximations for graph theory problems such as the traveling salesman problem (TSP), vertex matching, etc. as well as convolutional problems such as social network classification, image recognition etc. This is all because, as we know, graphs are unique in the sense that neural networks can make use of the edge classification in order to solve problems that traditional deep learning methods may not do efficiently.

In summary, Given a graph, we first convert the nodes to recurrent units and the edges to feed-forward neural networks. Then we perform Neighbourhood Aggregation (Message Passing, if that sounds better) for all nodes n number of times. Then we sum over the embedding vectors of all nodes to get graph representation H.



1.3 k-Center Problem

"The vertex k-center problem is a classical NP-Hard problem in computer science. It was first proposed by Hakimi in 1964.[3] Formally, the vertex k-center problem consists in: given a complete undirected graph G = (V,E) in a metric space, and a positive integer k, find a subset C of V such that the cardinality of C is less than or equal to k and we choose a vertex v in V such that the distance from v to its nearest center C is minimized" (Wikipedia). The vertex 1-center problem is essentially finding a vertex v such that the distances between v and every other vertex is minimized. In this project we tackle the vertex k-center problem for certain classes of graphs using a Graph Neural Network (GNN).

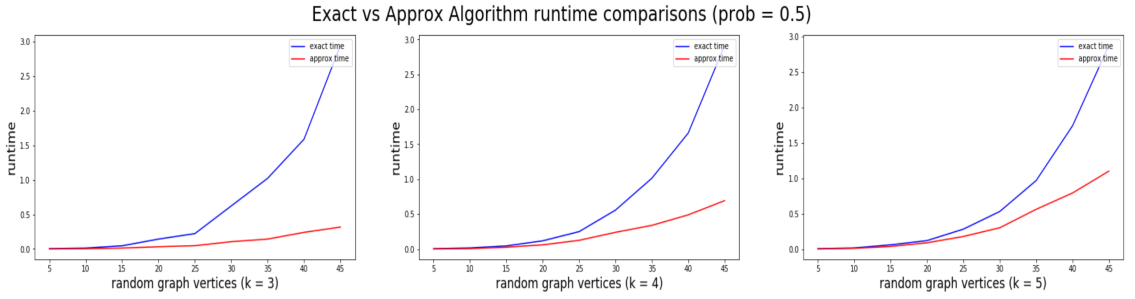
The k-center problem is NP-Hard, formally, "A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder." (Wolfram) Nevertheless, the k-center problem is worth researching due to its immediate applications in planning and logistics. Currently, there are several approximation algorithms such as Sh, HS, Gon, and most recently in 2017, the CDS algorithm which is based off of the Sh and Gon algorithms and runs $O(n^2)$. Although the problem is traditionally viewed in the context of planar graphs (i.e. a graph representing roads between various buildings in a city), its solution has applications to non-

planar graphs as well. A direct application could be determining which friend within a social network is the "largest influencer" i.e one who is most closely connected to everyone.

2 Process of developing the GNN in relation to vertex k-center problem

As with any Neural Network, good data is essential for training and implementation for the final result. Here, as we did not have preloaded datasets to train on, we decided to create all facets of what we needed from scratch. In relation to the k-center problem, our first task was to introduce an algorithm such that given a graph, it would calculate a single center. Developing on that, our research into the vertex k-center problem provided us various avenues to explore algorithms that compute the k-center of a graph network, ranging from computationally quick and decently accurate to longer computational time with much better accuracy. We implemented both an exact algorithm using the dominating set approach and a greedy approximation algorithm.

In regard to the exact algorithm with the dominating set approach, we based it off a plethora of papers and resources which detailed its efficiency over the other algorithms. (Robič). A dominating set is essentially a subset of the vertices of the graph such that every vertex not in the set is adjacent to a vertex in the set. We have explained the exact method of solving it as well as the greedy approach in the jupyter notebook of our project.



2.1 Choice of training data

Keeping in mind our goal for the GNN, we had broad ideas of implementing the GNN on a variety of graphs that would provide useful and efficient analysis to real- world situations depending on the graphs we picked. Early on in our course, we were introduced to a number of different graph types and their innate quirks, such as Grid graphs, plethora of random graphs, planar and non-planar graphs, each of which can be fitted to model some real -world aspect. In the situations we looked at, a classic viable option were to use results on Grid graphs for approximate results on block-style cities, or use different random graphs to model social networks, community interactions etc. Although the choice of training graph type is fairly arbitrary since this is an abstract problem, we decided to train our initial model on random GNP (Erdős-Rényi) graph with random edge weights, n=50, and p=0.4.

Since the problem boils down to classification, where the intended output would either be a list of k-centers or k-clusters, we have two possible approaches: 1) Train the GNN on the list of centers and have it determine which points are most likely to be centers, 2) Train the GNN on the clustering around each center. Both of these methods have their own drawbacks. Approach 1 uses relatively sparse data since the quantity of centers is typically sparse relative to the order of a graph and is best implemented with exact centers, and approach 2 does not determine the actual centers in the graph; instead, it determines a probable clustering. We elected to try both approaches and see which gives us more interpretable results.

2.2 Feature Generation

One large recurring problem with neural networks is appropriately determining and tuning features. In typical applications, this is done by massaging real world data into floating point vectors that describe specific attributes of an input (examples of this are usually concrete, like age, bmi, etc.) In our case, however, we are dealing with purely abstract data that has no real world attributes. The only core property that we care about is the connectivity and structure of our graph. We encode this in a feature vector $\overrightarrow{\mathbf{h}}$ as follows where $\overrightarrow{h_i}$ represents the a feature vector for node i. For any arbitrary graph, we want to create an order isomorphic way of describing how far away nodes

are within a graph. Since the direct mapping to a feature vector for node i where $\overrightarrow{h_i} = \begin{bmatrix} dist(i,1) \\ dist(i,2) \\ . \\ . \\ dist(i,n) \end{bmatrix}$ only works for identically labeled graphs, we instead denote a feature vector as follows: Consider an arbitrary connected n-node graph G. The k-dimensional feature vector (k is an arbitrary dimension somewhat analogous to desired accuracy) is $\overrightarrow{h_i} = \begin{bmatrix} h_{i,1} \\ h_{i,2} \\ . \\ . \\ h_{i,k} \end{bmatrix} \in \mathbb{R}^k$ where $h_{i,j} = |\{dist(i,j) | dist(i,j)$ is in the j^{th} equal interval of $[0, Diameter(G)]\}|$ An intuitive interpretation of this is counting how many vertices are within specific distance intervals from vertex i, where interval size and quantity is determined based on how fine the intended 'resolution' is. This is, unfortunately, a fairly expensive process as it requires computing distances between every pair of vertices, but it is the only way we could figure out how to generate an accurate and descriptive feature set.

2.3 Building and training the GNN

Our GNN is an implementation of the Graph Attention Network developed in the paper *Graph Attention Networks* by Petar Velickovic, et. al. Although the derivations of the essential equations are quite tedious, the equations that govern the training and behavior of the GNN are intuitively straightforward. Our GNN is built upon multiple *graph attention layers*. Each layer maps a set of node features $\overrightarrow{\mathbf{h}} = \{\overrightarrow{h_1}, \overrightarrow{h_2}, ...\overrightarrow{h_N}\}$, $\overrightarrow{h_i} \in \mathbb{R}^F$ where N is the number of nodes and F is the number of features per node. This layer outputs a new set of features (with a possibly different dimension F') $\overrightarrow{\mathbf{h'}} = \{\overrightarrow{h_1'}, \overrightarrow{h_2'}, ...\overrightarrow{h_N'}\}$, $\overrightarrow{h_i'} \in \mathbb{R}^{F'}$. So that we obtain sufficient expressive power with our GNN, we create a weight matrix $\mathbf{W} \in \mathbb{R}^{F' \times F}$ that is shared across every node. Then, we compute attention coefficients (e_{ij}) with a shared mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} -> \mathbb{R}$ where

$$e_{ij} = a(\mathbf{W}h_i, \mathbf{W}h_j)$$

represents the the importance of node j's features to node i. In our GNN, a is a feedforward single layer neural network that applies LeakyReLU activation. To make these coefficients comparable, we pass them through a softmax function to yield:

$$\alpha_{ij} = softmax(e_{ij})$$

Following the original paper, we also implement multiple attention heads to improve comprehension slightly. This yields an output feature vector:

$$h_i^{(l+1)} = ||_{k=1}^{K} \sigma(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^k W^k h_j^{(l)})$$

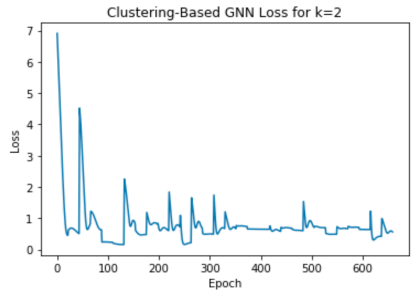
where σ is a nonlinearity function and $||$ represents concatenation. At the final layer, instead of concatenation we average the values, yielding the following modified equation:

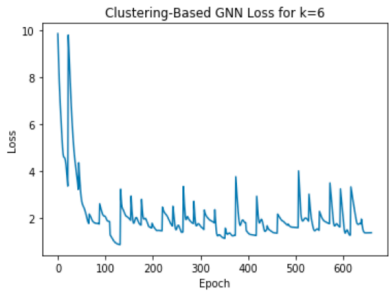
$$h_i^{(l+1)} = \sigma(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^k W^k h_j^{(l)})$$

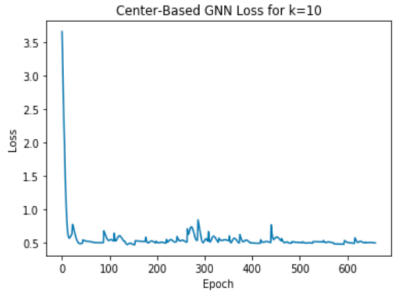
We used PyTorch's implementation of these equations to create our own attention layers and aggregate that into a GNN. After initializing our desired training graphs and precomputing the necessary features, we apply the standard techniques of neural networks to determine the appropriate weights. After the values are initialized at random, we employ gradient descent to iteratively converge to accurate parameters. Since this is a classification problem at heart, we used the negative log likelihood loss function prebuilt in the PyTorch neural network library. This general construction works for both of our intended classification approaches; the only attribute that changes is the labels that we train on and the output dimension of the network.

2.4 Results

Put bluntly, our GNN was an ineffective approach to solving the k-center problem. Although the loss itself was fairly good and consistently converged regardless of the approach (see below), the GNN was inconsistent with presenting identifiable clusters or centers on graphs it had never seen before.

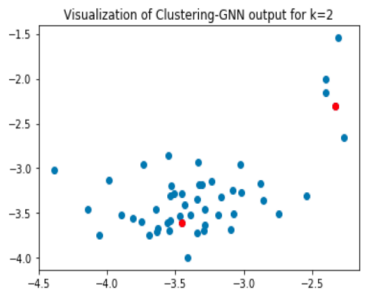


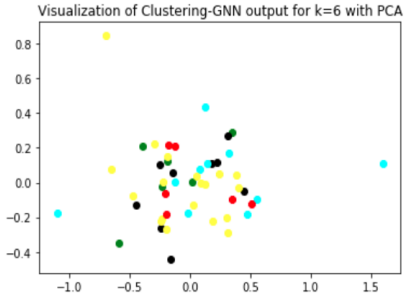




Clustering Classification Approach

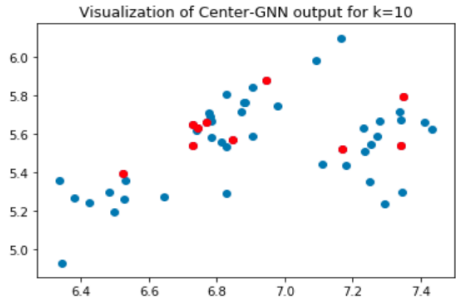
For the clustering approach, the GNN was fairly effective for k=2, clearly distinguishing two main clusters of the inputted graph. In the included figure, the two primary clusters are fairly visible and distinct, with their centers marked in red. However, this did not always produce consistent outputs and the clusters became more difficult to distinguish as k increased. This was partially due to the increase in dimensionality; however, even with conventional dimensionality reduction techniques like PCA (Principle Component Analysis) and TSNE (T-Distributed Stochastic Neighbor Embedding), we were unable to consistently distinguish clusters for a GNN trained for k = 6. The data appears random and less interpretable as k increases; the figures below demonstrate this misbehaviour.





Center Classification Approach

This approach was slightly more effective at interpreting the structure of the graph. We suspect that the output clustering represents vertices of the graph with similar 'centrality.' Although we were unable to quite discern this behavior, the clustering within the output itself was ineffective for solving the k-center problem, as it didn't return k clusters within the graph, but instead returned clusters of points with similar centrality. As such, we were unable to determine which points would be effective centers from the output of this GNN. However, this would be an interesting avenue to pursue given more time (since we saw this behavior only at the end of the project) and a modified problem statement.



2.5 Limitations

There were several limitations that limited the accuracy of our approach and could be the cause of why our GNNs were not as effective as we would have liked. First, the problem itself is very abstract. As the nodes have no additional properties other than structure, we arbitrarily create more information in the form of feature vectors. This creates problems as computing these feature vectors is very expensive and limits our ability to expand to larger graphs while being able to test and tune parameters often. Second, the k-center problem has many 'good enough' approximate solutions. This makes it difficult for our GNN to distinguish the best centers. This could be a reason for why our

result data seems clustered and interspersed. Third, we were unable to quickly train and fix our model on large data sets. Due to the expensive creation process for quality data, we only trained on datasets of 30-40 graphs to debug and refine our code. As such, the GNN is less effective at inductive tasks with novel graphs that it has not seen before. Fourth, due to how long it took to create a working GNN, we only trained on Erdős-Rényi graphs. This limits the scope of our results significantly.

3 Conclusion

We decided that to tackle the k-center problem, it's best to simply use one of the already existing approximation algorithms. From a practicality standpoint, they are both easier to implement and are less computationally expensive; they require no training and massaging of parameters to get a workable model. Further, the accuracy of these approximation algorithms is acceptable enough and will often surpass the accuracy of a GNN solving the same problem because the GNN will never be able to realistically train on obtain perfect data (an exact algorithm takes too long as computation time balloons when input size increases). Knowing what we know now, this problem is of the wrong type to tackle with a GNN as it is both very abstract, requires extensive computations to make a workable solution, and also doesn't scale well to large graphs (100+ vertices).

3.1 Future Applications

While a GNN may not be the best approach to solving this problem, this does not mean that GNNs are impractical. They are very powerful on real world data with practical applications and reasons behind the structure of the graph. In our notebook, we include the original creators' implementation of a graph attention network on the cora dataset, demonstrating that the approach does, in fact, work on structured real world data. We believe that there can exist an approach to the k-center problem rooted in real world data that is better approximated with a GNN. A better problem would likely be in the context of social networks, where given a limited set of influencers, you try to predict which other nodes would be good candidates as influencers. Instead of purely using the structure of the graph, this question offers the opportunity to include additional information about individuals themselves that can contribute to their status as influencers. This is especially useful for advertisers looking to maximize sponsorships and find new untapped talent. This is just one of the many possible problems that is well suited to a GNN based approach.

References

"A Beginner's Guide to Neural Networks and Deep Learning." Pathmind, http://pathmind.com/wiki/neural-network. Accessed 7 May 2020.

Anand, Rishabh. "An Illustrated Guide to Graph Neural Networks." Medium, 30 Mar. 2020, https://medium.com/dair-ai/an-illustrated-guide-to-graph-neural-networks-d5564a551783.

Battaglia, Peter W., et al. "Relational Inductive Biases, Deep Learning, and Graph Networks." ArXiv:1806.01261 [Cs, Stat], Oct. 2018. arXiv.org, http://arxiv.org/abs/1806.01261.

Dmlc/Dgl. 2018. Distributed (Deep) Machine Learning Community, 2020. GitHub, https://github.com/dmlc/dgl.

"Explained: Neural Networks." MIT News, http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. Accessed 5 May 2020.

Garcia-Diaz, Jesus, et al. "Approximation Algorithms for the Vertex K-Center Problem: Survey and Experimental Evaluation." IEEE Access, vol. 7, 2019, pp. 109228–45. IEEE Xplore, doi:10.1109/ACCESS.2019.2933875.

Hong, Shanon. "An Introduction to Graph Neural Network(GNN) For Analysing Structured Data." Medium, 9 Mar. 2020, https://medium.com/datadriveninvestor/an-introduction-to-graph-neural-network-gnn-for-analysing-structured-data-afce79f4cfdc.

Knyazev, Boris. "Tutorial on Graph Neural Networks for Computer Vision and Beyond (Part 1)." Medium, 4 May 2020, https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d.

Robič, Borut, and Jurij Mihelič. "Solving the k-center problem efficiently with a dominating set algorithm." Journal of computing and information technology 13.3 (2005): 225-234.

Scarselli, F., et al. "The Graph Neural Network Model." IEEE Transactions on Neural Networks, vol. 20, no. 1, Jan. 2009, pp. 61–80. DOI.org (Crossref), doi:10.1109/TNN.2008.2005605.

Schapke, Joao. "An Introduction to Graph Neural Networks." Medium, 15 Feb. 2020, https://towardsdatascience.com/an-introduction-to-graph-neural-networks-e23dc7bdfba5.

"Vertex k-Center Problem." Wikipedia, 17 Apr. 2020. Wikipedia, https://en.wikipedia.org/w/index.php?title=Vertex_k-center_problemoldid=951600144

Weisstein, Eric W. NP-Hard Problem. https://mathworld.wolfram.com/NP-HardProblem.html. Accessed 8 May 2020.