1 Problem Context

One of the central problems to graph mining and learning is generating representative embeddings for the nodes in a graph. Such embeddings have proven to be extremely useful in various identification and classification tasks. Examples of their use range from web recommendation systems to understanding protein interaction. The goal in most cases, is to make these embeddings low-dimensional while still capturing the high-dimensional information about a node and its role in the graph. In recent years, various models have been proposed to generate such embeddings. The focus of this report as well as that of the remainder of the project, is on one such model called a Graph Convolution Network. Specifically, I will be focusing on Inductive Graph Convolution Networks. These models are called inductive because they can embed nodes for dynamic graphs. This is an improvement from transductive models that require the graph to be static, as many real-world problems deal with dynamic graphs to which nodes are constantly being added or are being removed (e.g., Reddit posts, Facebook friend networks etc.). The current state of the art in inductive GCNs is based on the core ideas expressed in GraphSAGE (Hamilton et al., 2017), with various developments such as (Ying et al., 2018) and (Chen et al., 2018a) building on these core ideas. In this report, I explain the details of my implementation of an inductive model based on the same core ideas of GraphSAGE. I also explore some of the drawbacks of GraphSAGE as well as those of a few subsequent models that built on it and propose a plan that will overcome these drawbacks.

2 GraphSAGE

GraphSAGE is an inductive GCN model that learns embeddings for nodes in a graph using a sample and aggregate approach. Every node samples a fixed size neighborhood and learns an embedding based on the aggregated embedding of the neighbors it samples. The final embeddings are a function of the initial features of the nodes as well as their topological structure in the graph. Figure 1 below illustrates this process. A core idea of this embedding process is that GraphSAGE is not directly learning node embeddings. Instead, it learns a set of neural networks that learn how to aggregate information from a node’s neighborhood. These neural networks are called aggregators. In addition to the aggregators, GraphSAGE also learns a set of output layers that generate the final node embeddings.

Algorithm 1 GraphSAGE forward propagation

Input: Graph \( G(V, E) \); input features \( \{x_v, \forall v \in V\} \);
Output: Vertex embedding \( E_v \) for each \( v \in V \)

\[ h^0_v \leftarrow x_v, \forall v \in V \]

for \( k = 1 \ldots K \) do

\[ h^k_{N(v)} \leftarrow \text{AGGREGATE}k(\{h^{k-1}_u, \forall u \in N(v)\}) \]

\[ h^k_v \leftarrow \text{DENSE}k(\text{CONCAT}(h^{k-1}_v, h^k_{N(v)})) \]

end

\[ h^K_v \leftarrow \text{L2Norm}(h^K_v) \]

end

\[ E_v \leftarrow h^K_v, \forall v \in V \]

Algorithm 1 describes the forward pass of GraphSAGE. The input to the algorithm is a graph \( G \) and the input features of the nodes in the graph. These features serve as the initial embeddings of the
nodes. At each iteration of the outer loop, a node is aggregating information from its k-hop neighbors. Thus each iteration of the outer loop, corresponds to a search depth, with the value of k denoting the depth.

At each depth, we have a separate aggregator function to aggregate information from nodes at that depth. Thus, for any K, the model learns K different aggregators. Upon aggregation, a node’s current embedding is concatenated with its neighborhood’s aggregated embedding and passed through a dense layer to get an updated embedding. At every search depth, we have one such dense layer to generate an updated node embedding. Thus the model has K such output layers, with the Kth layer outputting the final node embedding. The algorithm also exposes why GraphSAGE is an inductive model. The learnt aggregation functions can be used on any new node that is added to the graph. All it requires is the new node’s initial feature vector and it’s neighborhood in the graph.

3 Baseline Model

To implement an inductive baseline model based on GraphSAGE, I made the following key design choices:

- I chose to use a max-pooling aggregation layer to aggregate neighbor embeddings for a node. A max-pooling aggregator is merely a dense layer followed by element-wise max-pooling. (Hamilton et al., 2017) experiment with mean-pooling, max-pooling and LSTM based aggregators. There were no additional benefits to using the LSTM based aggregator over the pooling based aggregators but they were 2x slower to train.
- I chose K = 2, meaning that every node aggregates information from nodes that are at most 2 hops away in the graph.
- GraphSAGE, in its general form, works in supervised and unsupervised settings. I chose to only implement a supervised model as my goal was to explore the drawbacks of the model as it pertains to neighborhood sampling and computational efficiency.

To train the parameters of the model in a supervised setting, I used the final embedding generated by the model for each node to predict the label of each node and minimized a cross-entropy loss based on the node’s actual label.

4 Experiments

4.1 Dataset

The dataset I used to train my model in a supervised setting was a Reddit data set available at (Leskovec and Krevl, 2014). The graph consists of 232,965 nodes with an average degree of 492. The dataset also includes features for each of the nodes in the graph as well as labels for each of the nodes. I used 70% of the dataset for training and the remaining 30% for validation.

4.2 Experiment Framework

The main goal of the experiments I ran was to estimate how important the neighborhood sample size was to generating representative embeddings. Ideally, I would have liked to test the sensitivity of K as well, but even at K = 2 and small sample sizes,
the network takes close to 10 hours to train for 10 epochs on a single NVIDIA Tesla P100 GPU. As a result of the long training time, my experiments didn’t involve training for all 10 epochs as (Hamilton et al., 2017) suggests, but training for 3 and analyzing changing trends with variation in the neighborhood sample size. The hyperparameters used in my model were similar to those the authors of GraphSAGE used with the exception of the batch size which I set to 128 instead of 512.

4.3 Results

Since I wasn’t running the training to completion due to long training times, my primary mode of analyzing the effect of varying neighborhood sample size was to observe the average training loss decay over the epochs as well as a validation micro-F1 score at the end of 3 epochs. My results are summarized below.

<table>
<thead>
<tr>
<th>S</th>
<th>Loss decay</th>
<th>Val F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8.1%</td>
<td>0.27</td>
</tr>
<tr>
<td>15</td>
<td>13.5%</td>
<td>0.41</td>
</tr>
<tr>
<td>20</td>
<td>18.2%</td>
<td>0.46</td>
</tr>
</tbody>
</table>

The loss decay is computed as the percentage drop from the highest loss in the first epoch to the lowest loss in the 3rd epoch, while the validation F1 is computed on half the validation set.

4.4 Analysis

The general trend is in support of including a larger fraction of a node’s neighborhood during aggregation. The authors of GraphSAGE also report that the result of increasing neighborhood size is a rise in F1 score. However, they mention that increasing the neighborhood size beyond a certain point leads to diminishing returns while leading to near exponential increases in training time. This is true in general. However, the threshold beyond which diminishing returns starts to take effect is graph dependent. Particularly, it is dependent on the average degree of the nodes in the graph as well as the quality of the initial node features. The dataset used in GraphSAGE, to determine the sensitivity of the model to neighborhood sample size, is a citation graph with an average degree of 9. On that graph, the point of diminishing returns starts at around a sample size of 30. The reddit graph however has an average degree of 492. Hence, it is extremely likely that in a graph with an average degree of 9, that a sample size of 30 is more than adequate. It is unlikely however that the same number holds for graphs with larger average degrees. This is especially true when the nodes have sparse initial features. In that case, the topological structure and the structural role of nodes in a graph becomes extremely important. Here, comprehensive aggregation of neighborhood information is paramount.

5 Drawbacks

The main drawback of GraphSAGE that I would like to address in my final project is small neighborhood sample sizes. It is difficult to make an unequivocal decision about the neighborhood sample size, since the optimal size with regards to the accuracy vs training time trade off could be very different for different graphs. Unfortunately, in the current framework, using larger neighborhood sample sizes as well as using larger K values leads to prohibitive increases in training time. Moreover, including a large sample size for moderately large batch sizes places a huge strain on the GPU memory system. There have been improvements to stochastic methods of generating neighborhoods. (Ying et al., 2018) and (Chen et al., 2018b) use slightly different formulations of a technique called importance sampling, while (Chen et al., 2018a) uses a control variate based algorithm to effectively select neighborhoods. I would like to approach this problem from a different perspective and explore the possibility of a systems level solution to this problem.

6 Future plan

My plan is to implement a distributed GCN model. In this framework, the graph would be split across the nodes of a distributed system and each node would perform a GraphSAGE-like algorithm on it’s subgraph. A few points to note about this approach.

- From the perspective of each node in the distributed system, the graph it operates on is an independent graph. Moreover, each of these subgraphs is of a significantly smaller size than the original graph.
- Since the subgraph sizes are smaller, every vertex in a subgraph would be able to include
a large fraction of its neighborhood (that belongs in the same subgraph) during computation. As a result, this framework would allow the inclusion of large neighborhood sample sizes.

- The aggregation step of the algorithm would require communication between the different nodes of the distributed system. This is necessary as not all sampled neighbors of a vertex will necessarily be on the same computational node.

- The number of iterations for each epoch would be much fewer than my current implementation, as each node would be updating the embeddings in its subgraph in parallel. This is similar to parallelizing batch processing, except each subgraph would also be split into batches.

Again, the goal is still to achieve the optimal tradeoff between accuracy and speed. The main problem to solve is then to come up with an algorithm to distribute the graph across the nodes of the distributed system. Similar to (Gao et al., 2018) in terms of creating subgraphs, this model is different in that it will use all the subgraphs that it creates. While this model affords a lot of parallelism, it also introduces the additional cost of communication overhead. Hence, the distribution scheme will account for this as well.

References


