Comp 6730 Advanced Database Systems Project

GNN Implementation

Chris Winsor

12-December-2021

The project investigates GNN (Graph Neural Network) by way of two graph datasets: CORA (a citation network where nodes are documents and edges are citations), and ENZYMES (a collection of graphs representing the protein structure of enzymes).

The toolset is PyG (PyTorch Geometric), an extension to PyTorch with methods for deep learning on graphs. GraphSage, GCN, and GAT are sub-classes within PyG we will use.

Our compute platform is Google Colab. To explore model structure and parameters we use PyGym.

Benchmarking: OGB (Open Graph Benchmark) which is like Kaggle but run by Stanford and focuses on graph datasets - see <http://ogb.stanford.edu/> and <https://arxiv.org/pdf/2005.00687.pdf>

## Graph Neural Networks and Embeddings

GNNs are different from traditional ML in both goal and approach. The goal of traditional machine learning is to map a set of attributes to some target class or measure. The dataset can be visualized as a table with attributes and class as the columns, and each row being a sample.

A Graph is a collection of nodes and edges where nodes represent things and edges represent relationships. The definition is very general - edges can be directed or undirected, the graph structure can be cyclic or acyclic, nodes and edges can include attributes.

This general structure means graphs are useful in representing many real-world problems. Some examples include drive time planning <reference>, social network relationships <reference>, <example, example>.

Message Passing is a technique that addresses the irregular structure characteristic of graphs. Where a traditional dataset (think image) has regular structure on which the neural network can rely, a graph has an irregular structure. Thus it is necessary to establish a consistent "compute framework" upon which the neural network can be based. Typically this is a reduction formulated from a node's immediate neighbors, and this is called "message passing".

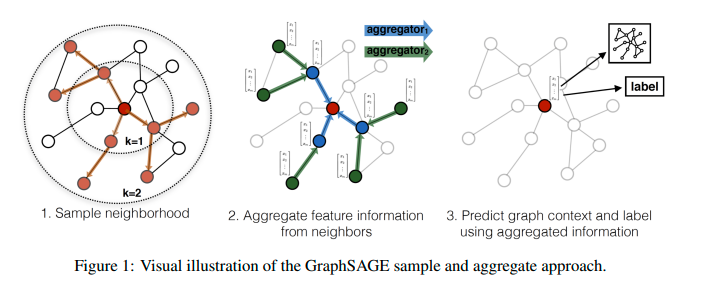
Embeddings are a key component of a GNN. An embedding is a vector associated with each node. The key characteristic is that given two nodes that are similar, the dot product of their embeddings is high. This characteristic is used to summarize attributes from nodes, edges and network structure, into a representation that is convenient for the intended use. An example is visualizing the relationship between music genres for a selection of songs. <picture here>. Other uses would be recommending purchase based on past purchases.

## GraphSAGE

GraphSAGE is introduced by Hamilton, Ying, Leskovec at <https://arxiv.org/pdf/1706.02216.pdf>. "SAGE" is a pseudonym of "sample and aggregate" and is indicative of the steps in the procedure.

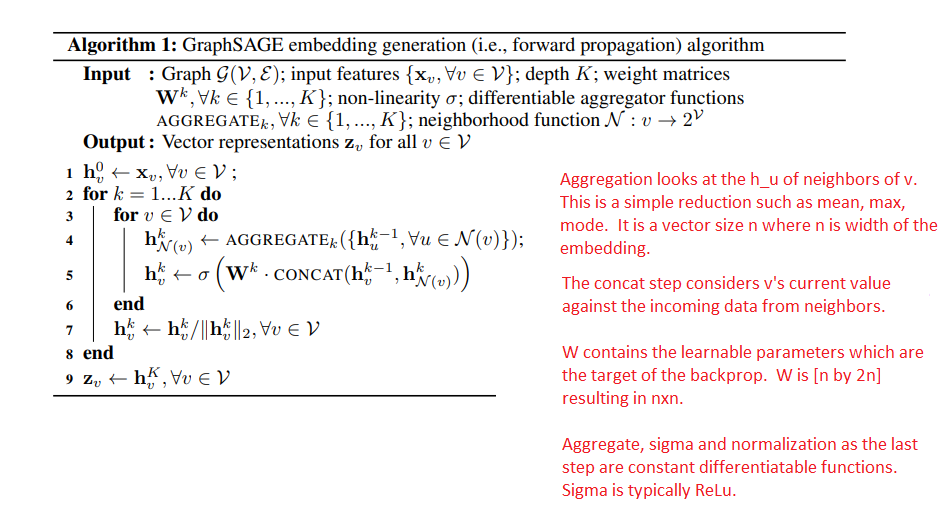
GraphSAGE uses an inductive approach when generating embeddings and this distinguishes it from earlier work. In the inductive technique a model is established that characterizes the local neighborhood of a node. That model is then used to create embeddings. This differs from the transductive approach of earlier work which creates the embedding for each node directly from that node's neighbors without the intermediate model.

Figure 1 from the paper illustrates the procedure. First a node is selected and a sample of its neighbors within K hops is taken. An aggregation is then performed to give the embedding (label) for the node. This aggregation function is the inductive model. An implementation of the GraphSAGE uses a multi-layer neural network is used to learn the aggregation function.



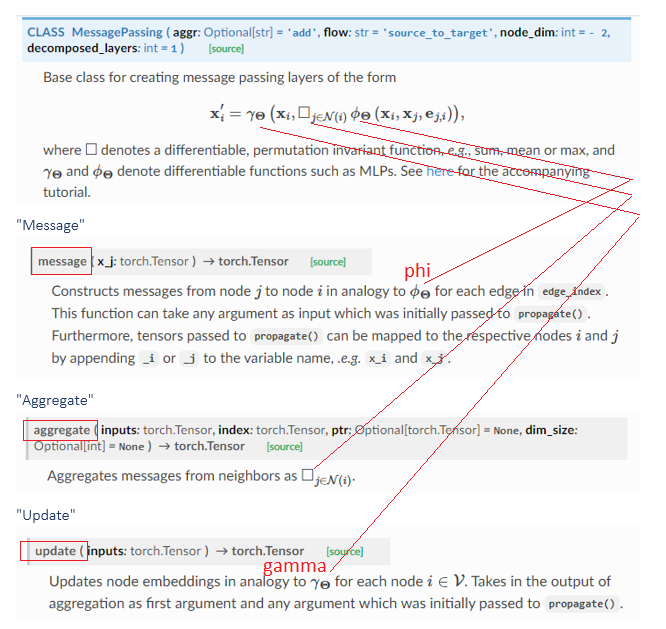
## The Algorithm

Section 3.2 identifies the forward propagation algorithm (the method by which embeddings are generated).



## PyG Implementation

To implement we map the theory to the Torch toolset. pyg\_nn.MessagePassing provides the mechanics. The forward() method (established as fully abstract in the parent torch.nn.Module) is defined in MessagePassing as calling message(), aggregate(), and update(). Our task is to implement those methods to implement the algorithm of GraphSAGE.



## Code

# References

GAT:

* /examples in <https://github.com/pyg-team/pytorch_geometric.git>
* GAT is based on the paper [“Graph Attention Networks”](https://arxiv.org/abs/1710.10903) paper. GAT uses GATConv operator for message passing.
* A good explanation at: <https://towardsdatascience.com/a-comprehensive-case-study-of-graphsage-algorithm-with-hands-on-experience-using-pytorchgeometric-6fc631ab1067>
* GCN is based on the paper [“Semi-supervised Classification with Graph Convolutional Networks”](https://arxiv.org/abs/1609.02907). GCN uses GCNConv for message passing.
* GraphSAGE is based on the paper [“Inductive Representation Learning on Large Graphs”](https://arxiv.org/abs/1706.02216) paper. GraphSAGE uses SAGEConv for message passing.
* PyTorch Geometric: [**https://pytorch-geometric.readthedocs.io/en/latest/notes/colabs.html**](https://pytorch-geometric.readthedocs.io/en/latest/notes/colabs.html)