Bachelor Research Project: Graph Machine Learning

Jannick Stuby
Department of Computer Science
University of Stuttgart
Stuttgart, BW 70565
st160888@stud.uni-stuttgart.de

Christian Staib
Department of Computer Science
University of Stuttgart
Stuttgart, BW 70565
staib.christian@hotmail.de

Lukas Zeil
Department of Computer Science
University of Stuttgart
Stuttgart, BW 70565
st161467@stud.uni-stuttgart.de

Abstract

We present our research on graph machine learning that we have made as part of the bachelor research project. We started our project examining DeepChem and MoleculeNet but later decided to focus on graph machine learning in general. During our project we had two main goals: to learn about graph machine learning in general and implement some graph machine learning algorithms ourselves. Therefore our report is structured in two parts: first we give an overview of graph machine learning and then we present our own implementations of some graph machine learning algorithms.

1 Introduction

1.1 Problem Domain

Neural Networks are thriving in the modern society with software ranging from facial recognition to language translation augmenting humans day-to-day. In this aspect, Machine Learning (ML) for chemical sciences gains more and more value and therefore is the focus of an increasing amount of research. In consequence, an increasing amount of data regarding molecules are gathered and labeled, multiplying the amount of information that can be used to train ML models to help chemical discovery and molecule analysis to aid chemical discovery for drug development and molecule investigation.

To aid in the improvement of ML models, benchmarks have been developed. A benchmark in a ML setting, is a site where models are compared against an accumulation of datasets, where the datasets are usually split into fixed training- and testing sets. The benchmark around which this project will be largely revolving is MoleculeNet. They tried a standardized approach for splitting their data, by creating more ways in which these datasets can be split, that can be applied to every dataset they have.

As the MoleculeNet paper was published in 2018, we try to find and use methods already used in other context but not in the paper. We will set up a dataset-to-metric model for each benchmark using new tools in four areas: splitters, featurizers, deep learning, and transfer learning. These results will be compared against the best-known results from MoleculeNet. While we may not be able to enhance the best-known results, we will formulate our accumulated knowledge for future exploration.

Preprint. Under review.

1.2 Definitions

A graph is a mathematical structure that consists of a set of vertices V and a set of edges E. Similarly a attributed graph is a mathematical structure that consists of a set of vertices and a set of edges, where each vertex and edge has an attribute, e.g. $A(V \bigcup E) \rightarrow \Sigma^*$. While this basic definition does not specify any conditions on the attributes, it is possible to limit the attributes to a certain set of values. Some graph machine learning methods require the attributes to be of certain types or to provide certain functionality.

1.3 MoleculeNet

TODO: add information on MoleculeNet and DeepChem framework

2 Preliminary

As we started our project examining DeepChem, we first looked into handcrafted graph to vector methods.

Later we looked into graph machine learning.

2.1 Open Graph Benchmark

Open Graph Benchmark(OGB) [4] is a framework for benchmarking machine learning on graphs, providing large-scale, real world datasets as well as an easy to use software framework for loading datasets and evaluating model performance. For comparing model performances, OGB provides leaderboards for different datasets and challenges, split into the sections Node Property Prediction, Link Property Prediction and Graph Property Prediction, as well as a large-scale challenge [3] leaderboard, which aids comparison of performance on very large graphs.

Our main focus will be Graph Property Prediction, mainly on the datasets ogb-molhiv and ogb-molpcba. Both these datasets are derived from MoleculeNet and respectively provide a large amount of molecules for molecular property prediction. Molecules are represented by graphs, where atoms are the nodes and corresponding chemical bonds are represented by the edges.

add input node features (9-dimensional), perhaps table?

For ogb-molhiv the model is evaluated using ROC-AUC, for ogb-molpcba Average Precision is used.

We additionally used ogb-molfreesolv as a smaller dataset which is also derived from MoleculeNet. FreeSolv contains SMILES strings of molecules from the Free Solvation Database and consists of "experimental and calculated hydration free energy of small molecules in water" . The model is evaluated using Root Mean Square Error.

add citation see moleculeNet paper

We decided to use OGB as a benchmarking framework instead of the MoleculeNet benchmark results, because OGB submissions are much more versatile and active, containing results from many research teams with the latest submission for ogb-molhiv dating to May 2022, contrary to the MoleculeNet results which were provided by the authors of the paper only and are dated to January 2018.

2.2 Graph Embedding

TODO: Add information on graph embedding relevant to problem domain

2.2.1 doc2vec

TODO: explain doc2vec according to [5] approach with respect to problem domain, possibly move to another section (paragraph/word embedding)

2.2.2 graph2vec

TODO: explain graph2vec [7] approach with respect to problem domain

2.3 Graph Transformers

TODO: add information for Graph Transformers [11] [6] [1] [2] and their usage for the problem domain and OGB leaderboards

With the success of Transformers [9] for NLP problems , Dwivedi et al. introduced Graph Transformers as a generalization of Transformers usable for arbitrary graphs [2]

remove this section and merge with Graphormer Section/replace by general Transformer

cite success information

2.3.1 Graphormer

As an adaption of Transformers [9] for graph representation learning, Graphormer [10] was presented by Ying et al., providing a performant Transformer model, excelling in various graph representation learning tasks.

Graphormer achieves such good results by introducing three types of encodings to incorporate structural information of graphs in the feature vectors of nodes and edges.

To capture information about the importance of a node in a given network, Graphormer uses a measure called node centrality by utilizing degree centrality of a given node. This is done by introducing two new vectors for each node, one for the degree of ingoing edges and one for the outgoing ones. Those vectors are learnable scalars which are added to the node features, so node importance and semantic correlation are respected during self attention.

The second encoding, called spatial encoding, is used to capture positional information for each node with respect to the network. This is achieved by using the distance of the shortest path between connected nodes and, once again, using a learnable scalar which is added as bias in the self-attention module. By doing that, the model can attend to a more relevant subset of the network according to the task. An example given in the paper is the possibility to increase attention to nodes surrounding a given node instead of nodes which lay further away.

add more information about learnable scalars, add general transforer structure to explain queries and keys

Finally Graphormer introduces a method to include edge information in the model by adding an additional bias term to the attention module. This bias is created by averaging the dot-products of edge features and a learnable scalar of the edges describing the shortest path between each two connected nodes.

perhaps add equations according to Transformer and Graphormer paper

2.3.2 GraphGPS

TODO: explain GPS framework [8], recipes and focus on our uses

2.3.3 Heterogeneous Interpolation on Graphs

TODO: briefly explain and focus on our possible usage

3 Our approach

This is just a stub text to prevent the section from being empty.

In order to further strengthen our understanding, we decided to implement some graph machine learning methods ourselves. We had two ideas in mind: try something really simple and see if it works, and try something more complex and see if it works. For the really simple method, we used random walk based methods.

3.1 Walk based graph embeddings

One of the primary challenges in graph machine learning, as opposed to text-based machine learning, is the multidimensionality of graphs. Each node can have varying numbers of neighbors and different features and feature types. We considered the possibility of using graph walks to represent a graph. These walks would produce two-dimensional data that we could embed using methods to embed text.

Our main point of research was the embedding of whole graphs, as the a similar method to embed nodes already exists (node2vec). We also focused on the dataset 'ogbg-molhiv'.

3.1.1 walk to vector

A walk on graph is a unambiguously sequence of vertices and edges, where each vertex is connected to the next vertex by an edge. Given a graph g, we can write a walk as list of nodes, e.g. $(0, 2, 3)$ is a walk that start from the node with the id 0, goes to 2 and end at 3. For one graph commonly there a many different possible walks, therefore we can create a set of walks w that all walk on the same graph.

Given a set of graphs $G = \{g_i \mid i \in 0 \dots n\}$, where n is the number of graphs, we can generate a set of sets of walks $W = \{\ w_i \mid i \in 0 \dots n\}$. Now we create a set W' where for each walk we substitute the node id with the features of the node. If we view each walk as sentence we get a set of documents (multiple sentences). These documents are feed into a text embedding method.

Algorithm 1: basic idea of our walk based embedding

```
def get_vectors(graphs)
    documents = [ ]
    forall graph in graphs do
        this_graphs_walks = generate_walks(graph)
        this_graphs_sentences = generate_sentences(this_graphs_walks)
        documents.append(this_graphs_sentences)
    end
    model = Text_Embedding_Model()
    model.fit(documents)
    return model.get_document_vectors()
```

3.1.2 Generate the random walks

We focused on two ways to generate walks: random walks and shortest paths.

Generating all pair shortest paths of a graph guarantees to include all information of that graph.

Random walks don't guarantee that, but offer other usefull options. If the random walks methods favours

References

[1] Deng Cai and Wai Lam. *Graph Transformer for Graph-to-Sequence Learning*. 2019. DOI: 10.48550/arXiv.1911.07470.

[2] Vijay Prakash Dwivedi and Xavier Bresson. *A Generalization of Transformer Networks to Graphs*. 2021. DOI: 10.48550/arXiv.2012.09699.

[3] Weihua Hu et al. *OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs*. 2021. DOI: 10.48550/arXiv.2103.09430.

[4] Weihua Hu et al. *Open Graph Benchmark: Datasets for Machine Learning on Graphs*. 2021. DOI: 10.48550/arXiv.2005.00687.

[5] Quoc V. Le and Tomas Mikolov. *Distributed Representations of Sentences and Documents*. 2014. DOI: 10.48550/arXiv.1405.4053.

[6] Yuan Li et al. *Graph Transformer*. 2019. URL: https://openreview.net/forum?id=HJei-2RcK7.

[7] Annamalai Narayanan et al. *graph2vec: Learning Distributed Representations of Graphs*. 2017. DOI: 10.48550/arXiv.1707.05005.

[8] Ladislav Rampášek et al. *Recipe for a General, Powerful, Scalable Graph Transformer*. 2023. DOI: 10.48550/arXiv.2205.12454.

[9] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/arXiv.1706.03762.

[10] Chengxuan Ying et al. *Do Transformers Really Perform Bad for Graph Representation?* 2021. DOI: 10.48550/arXiv.2106.05234.

[11] Seongjun Yun et al. *Graph Transformer Networks*. 2020. DOI: 10.48550/arXiv.1911.06455.