Attributed network embedding

Given a dataset of small networks with labeled nodes we aim to learn an embedding that well-separates the networks into a positive and negative class. We assume that the classification depends on the presence or absence of small network motifs, or subgraphs.

We model the embedding function with a neural network architecture on the basis of repeated short random walks on each network. We seek to understand what aspects of the network input our model focuses on to make its decision, i.e. we include model interpretability as one of our main design goals.

Data

We consider the classification of molecules---small networks of atoms---as either active or inactive w.r.t. some target as one goal of this project. We represent each molecule as a network where each node has a discrete label, with multiple nodes potentially sharing a label (there may be more than one C atom in a molecule, for instance). Each molecule has an associated binary label placing it into a positive or negative group. We assume the molecules are small, containing roughly 30-100 nodes each. Example datasets include these.

Synthetic data

In addition to real-world datasets, a clean artificial setting containing purely synthetic data might be a useful start. One approach is to consider the classification of molecules into positive or negative groups can be done purely on the basis of the molecule containing a small fragment or subgraph containing 3-10 nodes. We first create one (or more) such network motifs, or specific small labeled subgraphs. Each positive example must contain the network motif, or some small subset of them. The negative class then either doesn't contain any of those motifs, or contains its own separate motifs.

Model

A couple of challenges arise in modeling this problem, especially when using a neural network approach. These include the inputs not being of a constant size, the representation of the nodes being invariant to their ordering, and that we seek an interpretable model that we can query as to how and why it makes its decisions.

Implementation guidance

We will implement this project using TensorFlow, with Keras serving as a high-level API. If this project is your first time using either, I highly recommend going through a simple tutorial and

implementing a basic MNIST (handwritten digit) classifier. E.g., this one, but using tensorflow instead of theano.

In addition, when first starting out and prototyping ideas, I highly recommend using Jupyter notebooks, which offer a matlab style working environment where you can interleave code, the output of running that code, figures, commentary, etc. This makes for easily reproducible and self-documenting experiments.

In short: Install python 2.7 (the anaconda distribution is used by other members of the lab), jupyter, tensorflow, and keras. Follow along with a simple tutorial to tackle the MNIST problem, implementing your solution in a jupyter notebook. From there, you'll be at a good baseline to tackle the described problem.

Baseline model of differential fingerprints

This paper[0] learns an embedding, or feature vector, for each node label. Given a new network to classify, we iterate through every node and sum up all feature vectors in that node's neighborhood (obtained by repeated random walk with restart). This summed vector is fed through a single dense neural network layer which outputs a new representation of that node. This is repeated for a couple of layers, where the new node representation is used instead of the node label embedding. See Algorithm 2 in the paper for exact details.

Implementation guidance

Keras provides an embedding layer that takes as input a positive integer (e.g. discrete node label) and outputs a vector of a given size corresponding to the r_is in Algorithm 2 in the paper. Line 9 of that Algorithm corresponds to a dense layer, and Line 10 a softmax operation. Note that the output of line 9 replaces the embedded node features of a node *a* with its L-hop feature representation, which the next layer use instead of the output of a embedding layer; that is, this is how the the output of one layer goes as input to the next to make a deep neural network.

DeepWalk-esque model

The DeepWalk paper [1] presented a method to learn node embeddings in an unsupervised manner by building upon the success of Word2Vec [2], which learned word embeddings. The DeepWalk idea is to generate 'sentences' from a network by random walks, where each visited node is a 'word'. The model is trained to predict a word from its context (or, a node from its neighborhood), or vice-versa. This idea was further generalized and improved with Node2Vec [3].

In the context of this project, we could see how initializing (or simply replacing and holding constant) the otherwise randomly-initialized embedding layer from the baseline model with a DeepWalk-esque embedding would affect the classification performance. This method is also

not constrained to graphs with few discrete node labels, as is the case for molecules, which allows us to tackle other similar problems.

Implementation guidance

A convenient Python implementation of Node2Vec exists on github to learn node embeddings.

First steps

A good suggested work order would be to get a feel for the keras/tensorflow framework by completing one or more tutorials (MNIST is a good start). Concurrently, read and understand the differential fingerprints for molecules paper [0]. The very first big milestone we want to hit is to implement that algorithm (Alg 2 in the paper) with a dataset.

References

[0] Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." *Advances in neural information processing systems*. 2015.
[1] Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014.
[2] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
[3] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016.

Questions?

Reach out to Haraldur: hth@cs.ucsb.edu