Friend recommendation system

Varga Krisztián

December 14, 2024

1 Introduction

The goal of this project was to develop a personalized friend recommendation system by using Graph Neural Networks (GNNs). We chose to analyse data from Twitter (currently known as X) to suggest meaningful connections based on user profiles and interactions. The project was developed in Python version 3.9.13 using PyTorch [1] and PyTorch Geometric [2] as our ML framework.

2 The data

The Twitter dataset was first introduced in [3] for the task of discovering the social circles of social network users given their ego network, that is, the connections between their immediate neighbours. As such, the data is split into 973 sets of five files, each set made up of the (1) edges and (2) circles in the users ego network graph, the (3) feature vector of the user (followed hashtags and users, given as a series of zeroes and ones), the (4) features of their friends, and the (5) feature names corresponding to the previous two files. We are mostly interested in processing the last three, since we are also given the combined graph as a separate list of 1.8 million directed edges.

As with most projects of this kind, a considerable amount of time was spent preparing the data to allow for better model training. Since each ego network had a separate feature space, we had to combine them into one. Even after some heuristic reductions, like removing case-sensitivity, scrapping special characters from the end of feature names, and dropping the distinction between hashtags and at-s, the feature space remained at a massive 155 thousand individual features. Way too big for our 71 thousand remaining nodes to track individually. Our later models wouldn't be able to process subgraphs of even a fraction of this size in memory. But since each node only interacts with a very small percentage of features (the longest being 195 connections), we settled to represent these interactions as a bag-of-words, with each node only storing the indices of their features, with multiplicities carried over from the previous reductions.

This representation turned out to be not ideal in it's current form, and is currently the biggest problem with the project. One solution was to have features represented as sparse tensors, freeing up much needed space, but since many methods are not available for sparse tensors in PyTorch Geometric, we decided against that. Another possible solution would have been to massively reduce the amount of throughput in terms of graph size for training batches, which was considered at one point, but was ultimately scrapped due to time being a key limiting factor in project development. The 71 thousand individual feature vectors are still saved (as sparse tensors) if we decide to use them anyway.

3 Methods of training

In the previous section we outlined the general design choices regarding the dataset. In this section we discuss the models used to accomplish our link prediction task on said data. We will go in order of complexity.

3.1 Baseline models

We chose two simple models, which do not use the graph structure, as a baseline to compare our more advanced models to.

The first of these is based on the heuristic that people tend to follow people similar to them (i.e. form echo chambers). Therefore we only added an embedding layer, followed by the vector dot product of the resulting embeddings. The embedding ought to place features that have a strong connection close to each other, which the dot product uses to find similar nodes. Since edges are directed, the main problem with this approach is it's symmetry: it can not distinguish between an edge and it's reverse. One can easily see, for example, how this

would be problematic for people with many followers (like celebrities), who don't follow many people in return.

Our next model is the gold standard in baselines: A pair of fully connected (linear) layers joined by a ReLU and dropout layer. This was of course also preceded by an embedding layer, as before. The rationale behind this model is the Universal Approximation Theorem [4], which roughly states that any neural network with at least one hidden layer can approximate any continuous function, given enough neurons. Of course whether or not one person wants to follow another is not a well-defined continuous function, but neither are most other ML tasks.

Due to time constraints neither of these models had their hyperparameters rigorously fine-tuned (namely: optimizer, learning rate, learning rate scheduler, embedding dimension and hidden dimension), nor did we extensively train them, since we expect limited insight from their performance.

3.2 GNN models

The following section is where we introduce our GNNs. There was supposed to be more to this, but the aforementioned time constraints kept us from fully realizing some of our designs.

Further models were to use the node2vec embedding technique [5] in addition to the previously mentioned bag-of-words feature embedding. In contrast to the latter, node2vec embedding was pretrained on the training data, which is, in hindsight, something the feature embedding could have benefitted from as well.

The last model to be conceived begins yet again with a feature embedding layer, but this time we also apply the node2vec embedding, and combine these with a linear layer. We then pass to two graph convolutional (GCN) layers [6] with a ReLU and dropout layer, as with the fully connected network in the previous section. In our setup the flow of the GCN layers points from the target to the source node, because in real life, the people followed are more likely to influence their followers, than the followers are to influence them (hence the term influencer).

At this point CUDA decided to inexplicably throw error messages at the GCN layers, sometimes after a few dozen batches. This prompted the development to halt, and the developer to finally go to sleep...

4 Evaluation

Since the deadline was approaching rapidly, there was no real reason to continue working on developing new models, when even the previous ones aren't even done training. Instead of model evaluation, we will take this section to evaluate our work and design choices in detail.

The biggest design error by far was starting the project on the last week, betting on the relative simplicity of the task. It was not, in fact, simple, with much time spent waiting for the large dataset to compute, only to eventually run into an OOM error, or an erroneous output caused by carelessness at first, and exhaustion later during the week.

We already touched on the flaws with the data processing, but to briefly summarize: Since we wanted to keep using the bag-of-words with embedding, we should have set up a proper pretrained embedding we can then apply to each model, like we had with the node2vec embedding. On the other hand, we could have just accepted not being able to learn from big graphs, and made a dataloader for our purposes. Or we could have simply implemented GCNs and similar methods for sparse tensors. Easy enough!

Much of the later problems can be traced back to these earlier two. As they say: "You can't build a house on sand." and "You can't build Rome in a day."

5 Conclusion

"What is the main result of your project?"
Start your homework in time, and sleep plenty.

References

[1] J. Ansel, E. Yang, H. He, *et al.*, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*, ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. [Online]. Available: https://pytorch.org/assets/pytorch2-2.pdf.

[2] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[3] J. Leskovec and J. Mcauley, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/7a614fd06c325499f1680b9896beedeb-Paper.pdf.

[4] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[5] A. Grover and J. Leskovec, *Node2vec: Scalable feature learning for networks*, 2016. arXiv: 1607.00653 [cs.SI]. [Online]. Available: https://arxiv.org/abs/1607.00653.

[6] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, 2017. arXiv: 1609.02907 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1609.02907.