YouTube Link Prediction using Graph Neural Nets

David Gong
Georgia Institute of Technology
Atlanta, GA, USA
davidcgong@gatech.edu

Reagan Kan
Georgia Institute of Technology
Atlanta, GA, USA
rkan3@gatech.edu

Sreyans Sipani
Georgia Institute of Technology
Atlanta, GA, USA
ssipani6@gatech.edu

Kwok Kuin Ek Jeremy
Georgia Institute of Technology
Atlanta, GA, USA
kkwok9@gatech.edu

Abstract

In this paper, we explored the possibilities of applying a graph neural network (GNN) to produce related YouTube videos for newly generated videos using link prediction. We conducted representation learning for our dataset by applying graph embedding techniques such as node2vec and spectral embedding. These new inputs from the embeddings feed to a GNN for calculations. We also investigated a novel method of combining both SEAL and spectral embedding for link prediction. The results and analyses indicate an alternative approach for video recommendation tasks, which is both scalable and accurate. We find that GNNs can be further explored for building video recommendation systems.

1. Introduction

On YouTube, hours of content are being uploaded every second [7]. This generates a rich network of new data that is being added to the back-end servers at any given time. In the context of this paper, we will think of this network as a graph. This is important because each video, which represents a node, will need to generate directed edges to other nodes (other videos) in order for there to be recommended videos. We consider the following problem.

1. Given a set of nodes and a query node, utilize link prediction / GNN to accurately predict related IDs for the query node.

The results gathered from this project introduce an alternative other than deep neural networks for YouTube video recommendation. Further research into this topic from previous papers introduced practical applications for social networks, leading us to believe that the same interpretations could be said for video based platforms like YouTube and other various, cutting-edge applications such as cancer detection. We now discuss papers that relate to the research at hand.

2. Related works

Currently, the approach that YouTube utilizes for recommending videos is detailed by the paper ”Deep Neural Networks for YouTube Recommendations” which was written by Paul Covington, Jay Adams, and Emre Sargin in 2016. The first layer consists of candidate generation, in which a deep learning neural network determines hundreds of related content. The second and final layer consists of utilizing user activity, video characteristics, and other candidate traits to further filter the related content by ranking them — providing dozens of personalized recommendations for the user based on the video that is viewed [7].

As far as the limits of current practice, this seems to be a generally accurate and encompassing approach to recommend videos to users. However, one can make the argument that as the number of videos grows, so does the training time due to the model being all encompassing. Being able to think about the collection of videos as a network allows us to investigate and compare how a graph neural network and other graph-dependent techniques such as SEAL can be applied rather than a deep neural network. While not investigated in this paper, we also believe that a varied implementation of the PageRank algorithm could also lead to positive results.

Other papers such as *The link prediction problem for social networks* by David Liben-Nowell and Jon Kleinberg in 2013 indicate various link prediction algorithms (Adamic-Adar, Jaccard’s coefficient, common neighbors, etc.) that have been used, along with performance comparisons. [5]

We similarly compare these within the paper, and then lastly compare the results for our GNN specifically for our problem.

3. Dataset

For this study, we used the data set here [2], which was crawled in 2007 and 2008. According to Cheng, Dale, and Liu, they considered all YouTube videos to form a directed graph with each other. Each video represents a node within the graph, and for each video, if there is a video b in the related video list of a video a, then there is a directed edge from the node a to node b. There can only be 20 connections at once for our dataset, but the main purpose of us utilizing this dataset is so that we can quickly generate embeddings and clustering tasks within the context of utilizing GNNs. In addition, there are 3 depths of video data labelled for each of the dates within the data set. We chose the .txt file which had video data with 3 depths for our analyses, specifically from the .zip file named 080327.zip. The crawled data is labelled in the following table as can also be seen in [2]:

video ID	`string`
uploader	`string`
age	`integer`
category	`string`
length	`int`
views	`int`
rate	`float`
ratings	`float`
comments	`int`
related IDs	`string`

Some further analysis on the graph structure shows that the maximum degree of a node is close to 300 and there are only a few vertices with degree in the order 10^2. We did not exclude any of the variables for our deep learning neural net. Our output variable is the related IDs field.

4. Approach

We first processed the data from the website by converting it to an adjacency matrix. In addition, due to computational constraints, we restricted the amount of connections that we used to train our model. For each node that we parsed per line, we grabbed all of its neighbors and repeated this same step for a limited amount of iterations.

After the adjacency matrix was created, we ran our embeddings and link prediction baseline algorithms on it.

In order to effectively utilize GNNs, we first needed to generate embeddings for the dataset to simplify our downstream deep learning neural net classification task. Excluding the action of utilizing feature learning in a separate module often leads to low performance and accuracy. We explored two different methods of graph embeddings, then compared separate link prediction methods for the problem.

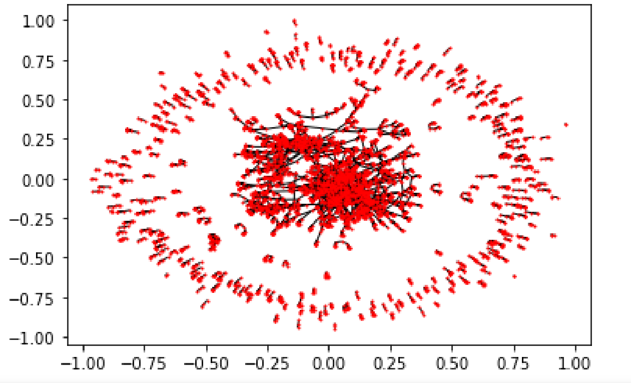


Figure 1. Visualization of the graph network for the extracted YouTube dataset.

As a novelty, we also combined spectral embedding with the SEAL (Subgraphs, Embeddings, and Attributes for Link Prediction) framework to combine subgraph evaluation and factoring in node separation distance.

4.1. node2vec

We used node2vec, which was introduced by Grover and Leskovec in 2016 [1]. node2vec is a semi-supervised algorithm that is used for representation learning on graphs, where continuous feature representations can be learned for the nodes - resulting in various uses for machine learning or deep learning tasks [1]. Nodes can be mapped to a low-dimensional space of features that maximises the likelihood of preserving network neighbour of node by using a Skip-gram model to networks to define a objective function over it's neighbours. We construct the neighbourhood using random walks and compute gradients for our objective function. Consider a walk (t, v), the use of edge $[(x, v)$ has the following probabilities. Where $p = \alpha(t, x)w_x v$.

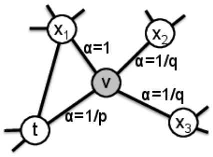


Figure 2. Visualization of the node2vec graph probabilities.

We use SGD for optimisation. This helps our case of predicting related videos not connected by an edge.

4.2. Spectral embedding

In addition to node2vec, we also utilized spectral embedding, specifically from the scikit-learn library. The process is as follows: [4]

- Data sample is projected on the first eigenvectors of the graph Laplacian.
- The adjacency matrix is used to compute a graph Laplacian that has been normalized which has a spectrum that can interpret the minimum number of cuts necessary to split the graph into comparably sized components.

According to scikit-learn's documentation, Laplacian Eigenmaps is the true algorithm that is implemented for spectral embedding.

4.3. Variational Graph Auto-Encoder

Here we pass the adjacency matrix and features through an auto-encoder to output a reconstructed adjacency matrix used for link prediction. We calculate the hidden layer using a graph convolution layer as follows -

$$H = ReLU(A * F * W_1)$$

where A is the adjacency matrix, F is the node features and W is the weights of the layer. We then use the hidden layer and the adjacency matrix to calculate the mean node embeddings and the standard deviation using graph convolution layers as well. Then the node embeddings are sampled using a normal distribution using the above 2 parameters.

$$\mu = A * H * W_2^{\mu}$$

$$\sigma = A * H * W_2^{\sigma}$$

$$Z \sim N(\mu, \sigma)$$

All the above layers use dropout. We finally reconstruct the adjacency matrix by computing an inner product on Z -

$$A' = Z * Z^T$$

The training is done using cross entropy loss and updation based on Adam optimization. The above implementation is based on [3].

4.4. Problems anticipated

We anticipated that no embeddings or versions of representation learning would lead to a significant decrease in performance. We expand upon this in further detail in later sections. The graph auto-encoder takes a very long running time for adjacency matrix with nodes in the order of 10^4 on a single CPU. Therefore the later experiments are performed on carefully chosen subgraphs / subset of nodes.

5. Experimentation & Results

After applying the embedding algorithms, we sought to compare the accuracies and performance of various link prediction baseline methods. For this portion of our project, we utilized the Github repository from github.com/lucashu1/link-prediction [3], namely making changes to the link prediction baseline Jupyter notebook by adding additional algorithms such as RA and adapting the code to Python 3 and utilizing NetworkX, a library that is used to create, manipulate, and structure complex networks. We additionally included a visualization of the node graph in Figure 1. We also augment the existing SEAL framework code [11] by adding new embedding options, namely, spectral embedding and a hybrid node2vec and spectral embedding.

5.1. Node neighborhoods (low-order heuristics)

Within the context of link prediction, node neighborhood-based methods have been gaining popularity in use due to their simplicity and effectiveness. We explored the performances and implications between Adamic-Adar, Jaccard's coefficient, preferential attachment, and resource allocation. For this portion, we examined ROC curves and AP (average precision). We first explain the concepts behind the baseline algorithms that were used / excluded from this project.

One popular method that we chose to exclude was Common Neighbors (CN), which is a relatively straightforward algorithm which scores by determining the number of neighbors that a node x and a node y have in common. [5]

$$score(x, y) = |\Gamma(x) \cap \Gamma(y)|$$

Common Neighbors are often combined with either computing the ratio of within and inter-cluster common neighbors of node pairs as was done by Valverde-Rebaza and Lopes [9] or through utilizing community information [8]. However, communities and clusters need to be defined and generated, and there is not enough reliable user data to construct accurate, disparate groups - thus the reason for CN's exclusion.

Jaccard's coefficient can be computed as follows [5]:

$$score(x, y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

The idea behind Jaccard's coefficient is that it measures the probability that both x and y shares a similar feature f which is randomly selected. [5].

Adamic-Adar is a common second order heuristic method for link prediction. The general notion behind this method is that sparse nodes should be weighted strongly because an edge between sparse nodes constitute more significance as opposed to a node with more edges. [5]

Algorithm	Time	ROC	AP
Adamic-Adar	85.29s	0.652	0.443
Jaccard Coeff.	96.56s	0.586	0.435
Preferential Attachment	41.36s	0.861	0.653
Resource Allocation	76.16s	0.604	0.443
Spectral Clustering	8.91s	0.911	0.846

Table 1. Comparisons of link prediction baseline algorithms and spectral clustering.

$$A(x,y) = \sum_{z \in N(x) \cap N(y)} \frac{1}{log|N(z)|}$$

, where $N(z)$ is the set of nodes adjacent to z. [5]

Preferential attachment operates on the premise that the probability that a new edge involves node u is proportional to $|\Gamma(u)|$, which is the current number of neighbors of u. The equation is as follows:

$$|\Gamma(u)| \cdot |\Gamma(v)|$$

Lastly, we also analyzed resource allocation, which is defined as:

$$score(x,y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{|\Gamma(z)|}$$

, where $\Gamma(x)$ denotes the set of neighbors of x . For our results, refer to Table 1. We list out the comparisons of link prediction baseline algorithms on our adjacency matrix. Note that Preferential Attachment has a strong ROC, in addition to having a much higher AP than the rest of the other algorithms. Preferential Attachment also evaluates and performs much quicker.

By looking at the visualized graph in Figure 1, we see that this notion makes sense, as it seems that most of the randomly sampled videos are closely inter-connected with each other. This tight inter-connection between the nodes implies that these nodes often share similar neighbors. As a result, preferential attachment performs the best since it relies on the proportion of the current number of neighbors for the compared node.

5.2. Spectral clustering

In addition to measuring the performance of the link prediction algorithms, we also researched spectral clustering. We used scikit-learn in order to generate spectral embeddings for our adjacency matrix, and then evaluated ROC and AP. The results can be found at Table 1, though will be discussed briefly in towards the latter half of this section.

We first talk briefly about the process of spectral clustering, [6].

- The partitioning algorithm initially builds a Laplacian matrix L for the graph.
- Find eigenvalues λ and eigenvectors x of L.
- Map vertices to the corresponding components of λ_2,
- Group these components by sorting them within a reduced 1-dimensional vector, with cluster separations done via a splitting point.

In all, this spectral clustering performs the best in situations where points in clusters are infinitely far from each other. [6]

Overall, spectral clustering seemed to have significantly better results (0.911 ROC, 0.846 AP) than the other link prediction baseline algorithms. This was most likely due to spectral embedding also taking into account distance between the nodes, not just the connections. As can be seen in Figure 1, there are clear groups of separated clusters for the nodes, that are disconnected from many other nodes.

5.3. SEAL (Subgraphs, Embeddings and Attributes for Link prediction)

The following definition will be useful in the section.

Given a graph G = (V, E), a local enclosing sub-graph for any two vertices (or nodes) x, y ∈ V is as follows

$$enclosing\_sub\_graph(x,y,h) = \Gamma(x,h) \cup \Gamma(y,h)$$

where $\Gamma(v,h)$ denotes the h-hop neighborhood of vertex v. Formally,

$$\Gamma(v,h) = \{u | D(u,v) \leq h\}$$

where $D(x,y)$ = length of shortest path between nodes x and y.

High-order heuristics often outperform lower order heuristics, but they come with the cost of being computationally expensive, as they require the entire network. This has motivated work to locate a compromise between the two. It has been shown that local enclosing sub-graphs can effectively approximate many high-order heuristics. To be more precise, most high-order heuristics can be classified as γ-decaying heuristics and, under certain conditions, these can be approximated using h-order enclosing subgraph, with exponentially decreasing approximation error as h grows.

By training a Graph Neural Network (GNN) to learn a h-order heuristic from smaller local enclosing sub-graphs, the SEAL framework[11] is able to exploit the computational efficiency of low-order heuristics while preserving useful features and graph structure knowledge gained from high-order heuristics.

SEAL specifically uses a Deep Convolutional Graph Neural Net [10]. In addition to the adjacency matrix, A, of the local sub-graphs, the DCCNN also takes as input, a feature matrix, X, which contains node labels and optionally includes node embeddings and attributes. Node labeling is the only necessary component of this feature matrix, as it encodes structural information of the sub-graph, by differentiating nodes by their functions in the graph. There are target nodes, which form the links we are trying predict the existence of. Remaining nodes serve different roles, depending on their relative positioning to the target nodes. SEAL uses Double-Radius Node Labeling, which meets the following criteria:

1. Target nodes x and y have a constant and unique label.
2. Nodes u and v have the same label if and only if they are equidistant to the targets, i.e. D(x, u) = D(x, v) and D(u, y) = D(v, y).

The DGCNN architecture feeds these inputs through a pipeline with three sections: Graph Convolutional Layers, Sorted Pooling Layers, and a regular Convolutional Neural Network. Omitting normalization factors for simplification, a single Graph Convolutional Layer can be described as the function $f(A, X) = NonLinearActivation(AXW)$. Intuitively, this layer learns network substructure information by optimizing W, which helps distributes node information through a local neighborhood. These can be composed to learn more intricate sub-graph features. The motivation for the next phase, Sorted Pooling, comes from the inherent ordering in other common inputs to CNN, such as images or text. The assumption is that the features extracted from the Graph Convolutions should be ordered for better performance. For graphs, the ordering is based on the node structure, which has already been extracted from the matrix X during the graph convolution. Thus, the Sorted Pooling Layers can operate solely on the output from the previous layer. Finally, the CNN with 1D convolutional layers learns additional patterns and outputs probabilities for link existence, using a composition of dense and softmax layers.

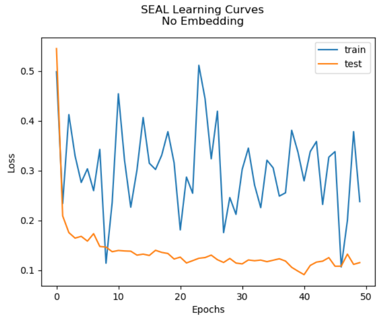
By default SEAL can toggle the use of node2vec with 10 walks, each of length 80. Since, spectral embedding is used in spectral clustering from an earlier section, we also incorporated spectral embedding as an extra embedding option. Our intuitive explanation for this addition is that node2vec utilizes random walks, unlike the more sophisticated Laplacian Eigenmaps technique for spectral embedding. As previously noted, the node2vec uses 10 random walks of length 80, meaning it captures information for at most 800 nodes. For perspective, our train and test split had 4423 and 10269 nodes respectively. This is in contrast with the Laplacian Eigenmaps, which has compressed information for all nodes. Thus, we believe spectral embedding will provide a more complete representation of the sub-graphs and perform better than node2vec.

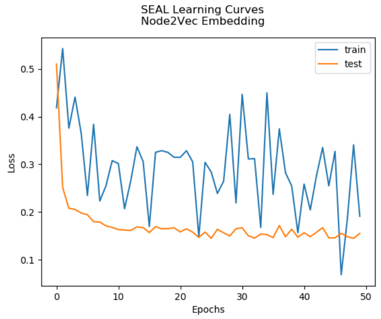
As for the empirical results, refer to Table 2, which delineates the ROC and AP scores of running various configurations of SEAL on our data set. We expected the embedding settings to perform better. As mentioned before, another assumption was the dominance of spectral embedding over node2vec.

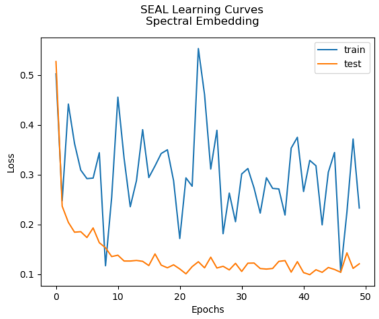
For the most part, AP scores reflected our hypotheses. One oddity was that the no embedding setting beat node2vec in AP score and beat all embeddings in ROC score. We believe this can be attributed to the depth of our data set, and not to over-fitting.

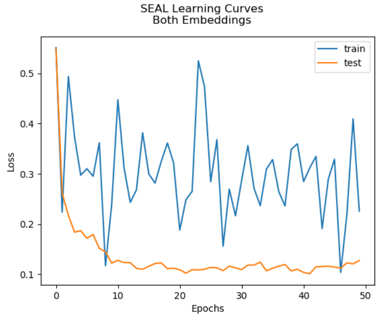
First, we notice that our data set 080327.zip has a maximum depth of 3, i.e. it is a 3-hop network. The GNN without embeddings may be complex enough to deal with this relatively small size. To verify this hypothesis, we ran the same experiments with a deeper 5-hop data set 0222.zip. With this, the ROC and AP scores match our understanding and expectations. See Table 3 for the scores.

Second, to verify that the GNN is not over-fitting, we confirm the testing loss curves are not above the training loss curves. Since this is the case for both 3-hop and 5-hop data sets, we only include the plots for the 3-hop data set.









5.4. VGAE - Variational Graph Autoencoders

As the graph is too large and experiments show that categories are strongly inter-related, we decide to choose top-K nodes with the highest degree and their neighbours so that we can still encode the important information in the graph. For our feature matrix, we only use the following attributes - ’age’, ’length’, ’views’, ’rate’, ’rating’, ’comments’. For our experiments below we report the ROC-AUC score and the Average-Precision score which can be defined as the area under the area under the receiver operating characteristic curve and the area under the precision-recall curve respectively.
We first run an experiment taking the top 1000 nodes with highest degree and varying the number of related videos considered. The results are as follows -

Number of related videos	ROC-AUC Score	AP Score
20	0.816	0.793
15	0.802	0.777
10	0.761	0.740

We then also decided to take the top 1000 viewed and rated videos and see if these subsets perform any better. We observed that -

	ROC-AUC Score	AP Score
Top 1000 viewed videos	0.787	0.775
Top 1000 rated videos	0.827	0.809

We can therefore conclude that we want to use the highly rated videos and all the twenty related videos to make better predictions. We can train on more nodes by using a GPU.

6. Conclusion

In conclusion, we were able to show that given a dataset that could be represented and shown as a network through the use of the related IDs field, representation learning algorithms / embeddings in combination with a GNN yielded surprisingly strong results. Based on the results that we gathered prior to running the GNN, we conclude that the feature learning process was crucial for high performance and accuracy. After running the GNNs, we observe that the features that are used heavily impact the model’s performance. Furthermore, our experimental results suggest that feeding pre-processed input, in the form of explicitly computed features, to GNNs gives better performance, as seen in SEAL. Models incorporating spectral algorithms also seem to do well with our data set.

The work here demonstrates that given an accurate representation of a network, graph neural networks could be a practical application - especially so for non-linear dimensional data sets.

Future work could also involve the use of other well-known, high heuristic order algorithms such as PageRank (used by Google for representation learning on pages on the WorldWide Web) and the Katz index.

7. Work Division

Our work delegation of work among team members can be seen at Table 4.

References

[1] Jure Leskovec Aditya Grover. node2vec: Scalable feature learning for networks. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2018. 2

[2] X. Cheng, C. Dale, and J. Liu. Statistics and social network of youtube videos. pages 229–238, 2008. 2

[3] Lucas Hu; Thomas Kipf; Gökçen Eraslan. Representation learning for link prediction within social networks. 2018. 3

[4] Andrew V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. 2

[5] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. *In Proceedings of the Twelfth International Conference on Information and Knowledge Management, CIKM ’03*, pages 556–559. 1, 3, 4

[6] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. pages 849–856, 2001. 4

[7] Jay Adams Paul Covington and Emre Sargin. Deep neural networks for youtube recommendations. *Proceedings of the 10th ACM conference on recommender systems*, 2016. 1

[8] Sucheta Soundarajan and John Hopcroft. Using community information to improve the precision of link prediction methods. *In Proceedings of the 21st international conference companion on World Wide Web (WWW ‘12 Companion).* 3

SEAL	Time	ROC after 50 epochs	AP after 50 epochs	Best ROC	Best AP
No Embedding	312.98s	0.936	0.916	0.936	0.916
Node2Vec	384.77s	0.923	0.906	0.924	0.906
Spectral	360.52s	0.932	0.916	0.935	0.917
Both	403.85s	0.934	0.921	0.934	0.921

Table 2. Comparisons of different embeddings with SEAL trained on 50 epochs of 3-hop data set.

SEAL	Time	ROC after 50 epochs	AP after 50 epochs	Best ROC	Best AP
No Embedding	363.96s	0.936	0.919	0.936	0.919
Node2Vec	461.40s	0.932	0.919	0.932	0.920
Spectral	428.12s	0.939	0.926	0.939	0.926
Both	458.80s	0.939	0.927	0.939	0.927

Table 3. Comparisons of different embeddings with SEAL trained on 50 epochs of 5-hop data set.

[9] Jorge Carlos Valverde-Rebaza and Alneu de Andrade Lopes. Link prediction in complex networks based on cluster information. *In Proceedings of the 21st Brazilian conference on Advances in Artificial Intelligence= (SBIA'12). https://doi.org/10.1007/978-3-642-34459-6*$_1$0. 3

[10] Cui Zhicheng Neumann Marion Zhang, Muhan and Yixin Chen. An end-to-end deep learning architecture for graph classification. *AAAI*, 2018. 5

[11] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *arXiv preprint arXiv:1802.09691*, 2018. 3, 4

Student Name	Contributed Aspects	Details
David Gong	Writing, implementation, article research	Implemented / analyzed link prediction algorithms + spectral embedding
Reagan Kan	Writing, implementation, article research	Parse/split data set. Add spectral embedding to SEAL + analyzed results.
Sreyans Sipani	Writing, implementation and research	Researched+implemented+experimented-variational graph auto-encoder
Jeremy Kwok	Writing, implementation and research	Implemented / analyzed node2vec / DeepWalk

Table 4. Contributions of team members.