Recurrent Multi-task Graph Convolutional Networks for COVID-19 Knowledge Graph Link Prediction

Remington Kim[1] and Yue Ning[2]

[1] Bergen County Academies, Hackensack, NJ 07601, USA
`remingtonskim@gmail.com`
[2] Stevens Institute of Technology, Hoboken, NJ 07030, USA
`yue.ning@stevens.edu`

Abstract. Knowledge graphs (KGs) are a way to model data involving intricate relations between a number of entities. Understanding the information contained in KGs and predicting what hidden relations may be present can provide valuable domain-specific knowledge. Thus, we use data provided by the 5th Annual Oak Ridge National Laboratory Smoky Mountains Computational Sciences Data Challenge 2 as well as auxiliary textual data processed with natural language processing techniques to form and analyze a COVID-19 KG of biomedical concepts and research papers. Moreover, we propose a recurrent graph convolutional network model that predicts both the existence of novel links between concepts in this COVID-19 KG and the time at which the link will form. We demonstrate our model's promising performance against several baseline models. The utilization of our work can give insights that are useful in COVID-19-related fields such as drug development and public health. All code for our paper is publicly available at `https://github.com/RemingtonKim/SMCDC2021`.

Keywords: Recurrent Graph Convolutional Networks · Multi-task Learning · COVID-19 Knowledge Graph · Link Prediction

1 Introduction

In recent years, the amount of data in a variety of domains has skyrocketed. Amidst this vast collection of information, there exist entities and relationships that can be modelled as a knowledge graph (KG) in which the entities are the nodes and the relationships are the edges. Although KGs can be extremely illuminating, the quantity of data contained in many KGs makes them prohibitively vast for manual review. Thus, a computational model that predicts novel connections in these KGs can provide valuable insight without time-consuming labor and reveal relationships unapparent to humans.

A relevant example of a KG is a COVID-19 KG of biomedical concepts and research papers formed using the scientific literature. Predictions of novel links in a COVID-19 KG may be helpful in areas such as drug development, public health, etc. To this end, we make the following contributions in this paper:

1. We construct a COVID-19 KG using data provided by the Oak Ridge National Laboratory Smoky Mountains Computational Sciences Data Challenge and auxiliary data processed with natural language processing techniques. We then analyze this network and its properties.
2. We propose a multi-task recurrent graph convolutional network (MTL-Recurrent GCN) that predicts the existence of novel links in a COVID-19 KG as well as the time at which a link will form.
3. We compare the performance of our model to several baselines and evaluate each model's performance using accuracy, AUC, precision, recall, and F1 score.

2 Related Work

COVID-19 Knowledge Graphs. In response to the COVID-19 crisis, several KGs centered around COVID-19 have been formed. A cause-and-effect COVID-19 KG was constructed from the scientific literature that contains 10 node types, including proteins and genes, and 10,232 edge types, including increases and association [4]. For ease of access, this KG was made available on a public web application. Wise *et al.* [14] built a COVID-19 KG of research papers and trained a model that utilizes KG embeddings for retrieving similar papers. Giarelis *et al.* [5] developed a method for predicting future research collaboration links in a COVID-19 KG that incorporates structured and unstructured text data using a graph-of-docs representation.

Temporal Link Prediction. Temporal link prediction is the task of predicting the formation of links in the future state of a dynamic network [3]. Peddada and Kostas [11] performed temporal link prediction on a Pinterest network using temporal feature extraction of proximity measures and machine learning models. Li *et al.* [8] developed a self-attention and graph convolutional network-based temporal link prediction model that outputs the probability of a link between every pair of nodes at the next timestep given a sequence of graphs. They apply their model to a private message, an interaction, and two email networks. Their model focuses on the sole task of link prediction, unlike our work which predicts the time of link formation as well.

3 Methodology

3.1 Problem Formation

We assume that $G_t = (V_t, E_t, X_t)$ is an undirected graph at time t, where V_t is the set of nodes, E_t is the set of edges, $N_t = |V_t|$ is the number of nodes, and $X_t \in \mathrm{R}^{N_t \times F}$ is the matrix containing the F features for each node. Every edge in E_t has an associated r, which is its relation type. Let $S_{t,j} = \{G_{t-j-(\alpha-1)l}, ..., G_{t-j-l}, G_{t-j}\}$ represent a sequence of graphs where j is the leadtime, l is the time length between two consecutive graphs in $S_{t,j}$, and α is the number of graphs in $S_{t,j}$.

The task of the model is to predict the existence of a link between target nodes u and v that forms at time t as well as j given $(S_{t,j}, u, v)$. In other words, the model predicts whether or not a link will form and *when* a link will form. As there are two objectives, this can be modeled as a multi-task learning problem.

3.2 Model

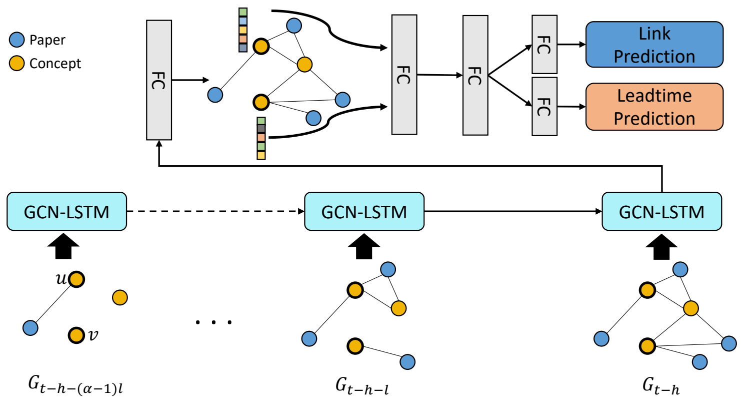


Fig. 1: Architecture of MTL-Recurrent GCN. GCN-LSTM represents a Chebyshev Graph Convolutional Long Short Term Memory cell and FC represents a fully connected layer. u and v are the target nodes.

The architecture of our model, MTL-Recurrent GCN, is displayed in Fig 1. We utilize a Chebyshev Graph Convolutional Long Short Term Memory cell (GCN-LSTM) [13] for the recurrent unit of our model. A GCN-LSTM employs a Chebyshev Graph Convolutional operator to learn graph features and a Long Short Term Memory cell to learn temporal dependencies within the sequence of input graphs [13].

Given X, the node feature matrix of graph G, Chebyshev Graph Convolutional operator [2] computes

$$\begin{aligned} X' &= \sum_{k=0}^{K-1} T^{(k)} \cdot W^{(k)} \\ T^{(0)} &= X \\ T^{(1)} &= \hat{L} \cdot X \\ T^{(k \geq 2)} &= 2 \cdot \hat{L} \cdot T^{(k-1)} - T^{(k-2)}. \end{aligned} \tag{1}$$

Here, $W^{(k)}$ is a learnable weight matrix, and $\hat{L} = \frac{2(I - D^{\frac{-1}{2}} A D^{\frac{-1}{2}})}{\lambda_{\max}} - I$ is the normalized Laplacian where I is the identity matrix, D is the diagonal degree matrix of G, A is the adjacency matrix of G, and $\lambda_{\max}$ is the largest eigenvalue of $I - D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$.

A GCN-LSTM [13] builds upon this Chebyshev Graph Convolutional operator with an Long Short Term Memory cell. Given X_t, the node feature matrix of graph G_t, it computes

$$\begin{aligned} i_t &= \sigma(\zeta(W_{xi}, X_t) + \zeta(W_{hi}, h_{t-1}) + w_{ci} \odot c_{t-1} + b_i) \\ f_t &= \sigma(\zeta(W_{xf}, X_t) + \zeta(W_{hf}, h_{t-1}) + w_{cf} \odot c_{t-1} + b_f) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(\zeta(W_{xc}, X_t) + \zeta(W_{hc}, h_{t-1}) + b_c) \\ o &= \sigma(\zeta(W_{xo}, X_t) + \zeta(W_{ho}, h_{t-1}) + w_{co} \odot c_t + b_o) \\ h_t &= o \odot \tanh(c_t). \end{aligned} \tag{2}$$

Here, $\zeta(W, X)$ denotes the Chebyshev Graph Convolutional operator where W is the tensor of learnable weight matrices utilized in (Eq. 1) and X is a node feature matrix. Weights w and biases b are also learnable parameters. h and c are the hidden and cell state matrices, respectively. $\odot$ denotes the Hadamard product, and σ and tanh denote the sigmoid and hyperbolic tangent activation functions, respectively.

In the case of MTL-Recurrent GCN, the input is $(S_{t,j}, u, v)$. First, the graphs in $S_{t,j}$ are sequentially inputted into the GCN-LSTM to produce a final output of h_{t-j}. Then, MTL-Recurrent GCN computes

$$\begin{aligned} h_{t-j} &= \mathrm{ReLU}(h_{t-j}) \\ h_1 &= \mathrm{ReLU}(W_1 \cdot h_{t-j} + b_1) \\ h_2 &= \mathrm{ReLU}(W_2 \cdot (h_1[u] \oplus h_1[v]) + b_2) \\ h_3 &= \mathrm{ReLU}(W_3 \cdot \mathrm{Dropout}(h_2) + b_3) \\ \hat{y}^b &= \sigma(W_4 \cdot \mathrm{Dropout}(h_3) + b_4) \\ \hat{y}^m &= \mathrm{Softmax}(W_5 \cdot \mathrm{Dropout}(h_3) + b_5). \end{aligned} \tag{3}$$

Here, $\hat{y}^b$ and $\hat{y}^m$ are the outputs of the link prediction and leadtime prediction tasks, respectively. $\oplus$ denotes the concatenation operator, and weights W and biases b are learnable parameters. Note that leadtime prediction is a multiclass classification task where each class is a possible leadtime window or the negative class. The negative class denotes that the link will not form in the future. The link prediction task is a binary classification task. We use hard parameter sharing multi-task learning, meaning the fully connected (FC) layers for both tasks are shared before the output layer, as it prevents overfitting [12].

This multi-task learning architecture has two advantages: performance improvements [1] and computational efficiency, as only one model is needed even though there are multiple prediction tasks.

Finally, we define a custom loss function (Eq. 4) for training MTL-Recurrent GCN that combines binary cross entropy and categorical cross entropy losses via a weighted average:

$$L = \beta[y^b \cdot \log(\hat{y}^b) + (1 - y^b) \cdot \log(1 - \hat{y}^b)] + (1 - \beta)[\sum_{i=0}^{p} y_i^m \cdot \log(\hat{y}_i^m)], \tag{4}$$

where β is the weight hyperparameter and p is the number of possible leadtime values.

In summary, MTL-Recurrent GCN learns graph features and temporal dependencies from a sequence of a graphs and performs the classification tasks with FC layers.

4 Experiments

4.1 Datasets

We conduct our experiments on a COVID-19 KG with three auxiliary datasets. The main KG dataset is provided by the Oak Ridge National Laboratory as a part of their 5th Annual Smoky Mountains Computational Sciences Data Challenge 2. This dataset contains a network of biomedical concepts nodes and research paper nodes formed using the PubMed, Semantic MEDLINE, and CORD-19 databases. The three relations present in the network are paper-concept edges (paper references concept relation; e_{pc}'s), paper-paper edges (citation relation; e_{pp}'s), and concept-concept edges (paper links two concepts relation; e_{cc}'s). The existence of e_{cc}'s and when they will form are to be predicted.

The three auxiliary datasets we utilize are the Unified Medical Language System (UMLS) API - for retrieving the names of the biomedical concepts in the network - and the CORD-19 metadata dataset and the PubMed NCBI E-utilities API - for retrieving the abstracts of the scientific papers in the network.

4.2 Preprocessing

We first drop all papers that are missing a valid publication date or a valid abstract. We then drop all concepts that have zero associated papers as a result. We remove e_{pc}'s and e_{pp}'s that stem from an invalidated paper node as well as e_{pc}'s and e_{cc}'s that stem from an invalidated concept node. We also remove e_{cc}'s that have no valid papers linking the two relevant concepts.

As only the publications dates of the papers are given, we assign dates to the edges in the following manner: the date of a e_{pc} is the publication date of the relevant paper, the date of a e_{pp} is the publication date of the citing paper, and the date of a e_{cc} is the publication date of the paper that links the two relevant concepts (if more than one paper linking the concepts exists, the earliest publication date is used).

We then create node feature vectors using a pretrained Google News Word2Vec model with the papers' abstracts and concepts' names as input. For each document, we generate a 300-dimensional Word2Vec [10] embedding for every one of its words and take the term frequency-inverse document frequency (TF-IDF) weighted average of them to get the document embedding. We then perform dimensionality reduction using principal component analysis (PCA) to get a 32-dimensional feature vector for each node.

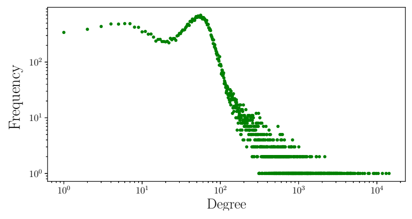
Finally, due to the network's prohibitively large size, we employ the Forest Fire sampler with a burning probability of 40% and an 85% reduction in the number of nodes because of its ability to retain the original properties of the graph [7].

4.3 Analysis

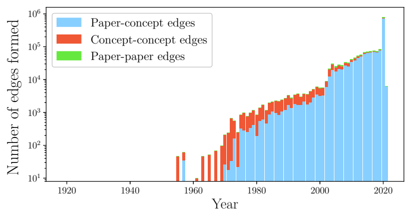
The final network contains 42, 062 nodes and 1, 744, 025 edges. There are 11, 741 concept nodes and 30, 321 paper nodes. e_{pc}'s are the most prevalent with a total of 1, 502, 372 of them in the network, followed by e_{cc}'s with 146, 704 and e_{pp}'s with 94, 967.

The disparity between the number of e_{pc}'s and the number of e_{cc}'s is explained by the fact that a paper referencing multiple concepts does not necessarily mean that all the referenced concepts are linked to each other by e_{cc}'s. For example, a paper references both "RNA" and "degrees Celsius"; however, there does not exist an e_{cc} between these two concepts as there is no substantial association between them. Additionally, there are papers that only reference a single concept, which makes extracting an e_{cc} from them impossible.

The average node degree of the network is 41.46. Concept nodes have a much higher average degree (152.95 ± 496.50) than paper nodes (55.81 ± 29.03); however, their degrees vary more. Additionally, a concept is referenced by an average of 127.96 ± 456.17 papers while a paper references an average of approximately 49.55 ± 22.54 concepts. Fig 2a displays the degree distribution of the network, and Fig 2b plots the formation dates of all the links in the network. The majority of new links were formed in 2020, which is expected due to the influx of COVID-19 research.



(a) Degree Distribution



(b) Distribution of Link Formation Dates

Fig. 2: Distributions of Network

4.4 Baseline Models

We compare the performance of MTL-Recurrent GCN to several link prediction baseline models.

Heuristic Algorithms. We utilize the following link prediction heuristic algorithms: Common Neighbors, Jaccard Coefficient, Preferential Attachment, Adamic-Adar, and Resource Allocation [9,15]. For a node pair between which a link is formed at time t with leadtime j, we extract the heuristics for the node pair from G_{t-j}. For a node pair between which a link is not present with leadtime j, we extract the heuristics for the node pair from G_{T-j}, where T is the last available timestep. We then train a logistic regression model on each heuristic.

N2V. node2vec (N2V) generates node embeddings by feeding random walks sequences on the graph into a word2vec model [6]. We generate 128 dimensional node embeddings on the training graph G_c, where c is the cutoff date between training and validation/testing dates, using node2vec and take the Hadamard product of node pairs' embeddings to train a 2-layer neural network model.

GCN. We utilize two different graph convolutional neural network (GCN) models as baselines: GCN and MTL-GCN. Both are standard GCN models with one Chebyshev spectral graph convolutional layer. GCN performs a single task while MTL-GCN performs multi-task learning using MTL-Recurrent GCN's custom loss function. For these models, the only graph inputted for predicting a link formed at time t with leadtime j is G_{t-j}.

Recurrent GCN. Recurrent graph convolutional network (Recurrent GCN) has the same architecture as MTL-Recurrent GCN; however, it forgoes the multi-task learning and performs a single task only.

4.5 Experimental Setup

In our experiments, we set Chebyshev filter sizes $k = 3$, and manually set $\beta = 0.6$ (Eq. 4). For our input graph sequences $S_{t,j}$, the leadtime $j \in \{1, 12, 24, 36\}$ months, the length of time between graphs $l = 12$ months, and the number of graphs $\alpha = 3$.

All e_{cc}'s formed between January 2019 and July 2020 are training, formed in August 2020 are validation, and formed between September 2020 and May 2021 are testing sampled. Negative node pairs (i.e., concept nodes pairs without a link by May 2021) are randomly sampled in equivalent numbers. In total, there are $45,922$ samples and a train : validation : test split of 82.3 : 7.4 : 10.3%. Every sample is randomly assigned a leadtime j.

Our GCN and Recurrent GCN models are trained for 10 epochs with AUC early stopping using the Adam optimizer, a learning rate of 0.001, and a dropout value of 0.5 on a NVIDIA TITAN V GPU server. All models are implemented using PyTorch.

4.6 Results

Table 1 summarizes the link and leadtime prediction performance of MTL-Recurrent GCN along with the baseline models. Note that two separate models are trained for all non-multi-task learning baselines: one for each task.

MTL-Recurrent GCN achieves that best link prediction performance out of all the models in terms accuracy, AUC, recall, and F1 score; however, MTL-GCN exhibits the best precision. This suggests that forcing the model to learn *when* a link will form via a multi-task learning architecture also improves the model's ability to predict *if* a link will form.

Additionally, although the link prediction performance of Recurrent GCN is slightly worse than that of MTL-Recurrent GCN, both give 1-2% AUC performance gains compared to GCN and MTL-GCN, which suggests the effectiveness of the recurrent architecture in learning temporal dependencies for link prediction. In general, all four GCN models give appreciable link prediction AUC and

Table 1: Prediction Results. Best results for each metric are bolded.

Model	**Link Prediction**					**Leadtime Prediction**	
	Accuracy	**AUC**	**Precision**	**Recall**	**F1**	**AUC**	**Accuracy**
Common Neighbors	0.6682	0.7380	0.8173	0.4333	0.5664	0.5983	0.6192
Jaccard Coefficient	0.6880	0.7415	0.7984	0.5030	0.6171	0.6176	0.6257
Preferential Attachment	0.5276	0.5469	0.5627	0.2477	0.3440	0.4864	0.6274
Adamic-Adar	0.6608	0.7332	0.7983	0.4303	0.5592	0.5984	0.6194
Resource Allocation	0.5489	0.7071	0.6448	0.2175	0.3253	0.5909	0.6249
node2vec	0.7326	0.8043	0.8249	0.5905	0.6883	N\A	N\A
GCN	0.8137	0.8994	0.8545	0.7562	0.8023	0.7640	0.5423
MTL-GCN	0.7802	0.9051	**0.9019**	0.6287	0.7409	0.7661	0.5516
Recurrent GCN	0.8141	0.9149	0.8890	0.7179	0.7944	**0.8839**	**0.6653**
MTL-Recurrent GCN	**0.8335**	**0.9193**	0.8508	**0.8088**	**0.8293**	0.7770	0.5603

F1 score performance gains of 11-13% and 7-19%, respectively, compared to the best non-GCN baseline: node2vec.

Lastly, Recurrent GCN exhibits the best leadtime prediction AUC and accuracy out of all the models. We suspect that this is due to the model's recurrent architecture and its loss function being comprised solely of categorical cross entropy loss.

5 Conclusions

In this paper, we present MTL-Recurrent GCN, a recurrent graph convolutional neural network model for temporal link prediction that utilizes a multi-task learning architecture. We also construct and analyze a COVID-19 KG of biomedical concepts and research papers using data provided by the Oak Ridge National Laboratory Smoky Mountains Computational Sciences Data Challenge and auxiliary textual data. Finally, we demonstrate MTL-Recurrent GCN's ability to outperform several baseline models at predicting novel links between biomedical concepts within this KG.

Currently, the leadtime prediction is limited to a set number of classes. Therefore, future works involves altering the multi-task learning models to allow for leadtime regression. Additional future work involves grid searching β for our custom loss function.

References

1. Caruana, R.: Multitask learning: A knowledge-based source of inductive bias. In: Proceedings of the Tenth International Conference on Machine Learning. pp. 41–48. Morgan Kaufmann (1993)
2. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering (2017)

3. Divakaran, A., Mohan, A.: Temporal link prediction: A survey. New Generation Computing **38**(1), 213–258 (Aug 2019)
4. Domingo-Fernández, D., Baksi, S., Schultz, B., Gadiya, Y., Karki, R., Raschka, T., Ebeling, C., Hofmann-Apitius, M., Kodamullil, A.T.: COVID-19 knowledge graph: a computable, multi-modal, cause-and-effect knowledge model of COVID-19 pathophysiology. Bioinformatics **37**(9), 1332–1334 (Dec 2020)
5. Giarelis, N., Kanakaris, N., Karacapilidis, N.: On the utilization of structural and textual information of a scientific knowledge graph to discover future research collaborations: A link prediction perspective. In: Discovery Science, pp. 437–450. Springer International Publishing (2020)
6. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks (2016)
7. Leskovec, J., Faloutsos, C.: Sampling from large graphs. p. 631–636. KDD '06, Association for Computing Machinery, New York, NY, USA (2006)
8. Li, J., Peng, J., Liu, S., Weng, L., Li, C.: Tsam: Temporal link prediction in directed networks based on self-attention mechanism (2020)
9. Liben-Nowell, D., Kleinberg, J.: The link prediction problem for social networks. In: Proceedings of the Twelfth International Conference on Information and Knowledge Management. p. 556–559. CIKM '03, Association for Computing Machinery, New York, NY, USA (2003)
10. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013)
11. Peddada, A.V., Kostas, L.: Users and pins and boards , oh my ! temporal link prediction over the pinterest network (2016)
12. Ruder, S.: An overview of multi-task learning in deep neural networks (2017)
13. Seo, Y., Defferrard, M., Vandergheynst, P., Bresson, X.: Structured sequence modeling with graph convolutional recurrent networks (2016)
14. Wise, C., Ioannidis, V.N., Calvo, M.R., Song, X., Price, G., Kulkarni, N., Brand, R., Bhatia, P., Karypis, G.: Covid-19 knowledge graph: Accelerating information retrieval and discovery for scientific literature (2020)
15. Zhou, T., Lü, L., Zhang, Y.C.: Predicting missing links via local information. The European Physical Journal B **71**(4), 623–630 (Oct 2009)