Convolutional Neural Networks on Graphs for Deep Knowledge Tracing

Alexandre Attia
MVA
ENS Paris-Saclay
alexandre.attia@ens-paris-saclay.fr

Sharone Dayan
MVA
ENS Paris-Saclay
sharone.dayan@ens-paris-saclay.fr

Abstract

Modern methods in Knowledge Tracing (e.g. predicting students' answers in educational platform) are mainly divided in two categories: on the one hand, deep sequential learning (e.g. deep recurrent neural networks) and on the other hand, highly structured models with parameters that can be interpreted. We propose to investigate the second - deep convolutional graph learning - as a knowledge tracing method and base our approach on Graph Convolutional Networks. We consider the task of predicting unobserved entries on graphs applied to educational data, the goal being the prediction of answers to questions never encountered by a student.

1 Introduction

The project we undertake is based on educational data and how to predict a student behavior regarding a question he has never encountered. We have access to *lelivrescolaire*, a French educational platform, dataset. The goal is to link together questions in order to learn to predict answers from previous performance in the neighborhood, using knowledge tracing. The previous can be formalized as given observations of interactions $\{x_0, ..., x_t\}$ taken by a student on a particular learning task, we would like to predict aspects of their next interaction x_{t+1}, .

In order to link together questions, we make use of graph theory. A weighted graph $\mathcal{G}$ is defined by a tuple ($\mathcal{V}$, $\mathcal{E}$, $\mathcal{W}$), where $\mathcal{V}$ is a set of vertices (nodes), $\mathcal{E}$ a set of couples of $\mathcal{V}$ elements (edges : correlations between nodes), and $\mathcal{W} : \mathcal{E} \to \mathbb{R}$ that assigns weights to edges. The graph $\mathcal{G}$ could be represented as an adjacency matrix where the entries are the edges' weights. We say that two nodes i and j are connected if $w_{i,j} > 0$, hence the number of edges is the number of strictly positive weights. Concretely, for our application, the set of questions can be seen as the nodes of a graph, related to each others. To compute the weights (and hence the edges of the graph), we can use natural language processing (NLP) techniques and/or empirical analysis (correlations). We concentrated on the second and let NLP techniques for future work.

The approach we adopt is based on Graph Convolutional Networks (GCN) which is the adaptation of Convolutional Neural networks (CNN) on graphs. CNN [1] have the ability to extract and learn meaningful patterns from large datasets. They can learn local and complex structures and have shown outstanding performance for different application such as image, video and sound recognition. Precisely, CNNs use shared features across data and thus, we take advantage of the generalization of CNNs to high-dimensional irregular domains represented by graphs.

In the next parts, we will present an overview of the related work on Knowledge Tracing before explaining the methodology of node classification and matrix completion using GCN. Then, we will present the dataset, the implementation and the conducted experiments. For the implementation, we used Tensorflow. Code available at : `github.com/SharoneDayan/EduDeepGraph`.

2 Related Work

Originally, the most popular approach for encoding students learning was based on Bayesian Knowledge Tracing (BKT), which returns a set of binary variables (understood / not understood). However, the model has now evolved and both the assumptions, that once a skill is mastered it is never forgotten as well as the one representing the understanding as a binary variable, do not stand anymore.

For all the previous reasons, student learning can be seen as a temporal model that encodes the learner's latent knowledge state over time. In that way, applying Long Short Term Memory networks (LSTM) recurrent neural network (RNN), where we map an input sequence to an output sequence by computing a sequence of hidden states that retain their values until explicitly cleared by a 'forget' action, can be promising [2].

Also, the task of predicting student answers can fall under the category of a matrix completion problem where we have a very sparse matrix representing the history of answers to questions by students that we would like to complete. This is precisely the problem of user-item recommendation that we transpose to student-question prediction. Recent works have proven that we can model this category of problems with geometric deep learning on graph-structured data using LSTM to encode the answers' temporal dynamics (valid or not valid) [3].

3 Weighted edges prediction

3.1 Graph Convolution Network

In order to achieve efficient classification on structured data, we based our approach on a simplified framework of spectral graph convolutions developed by Kipf and Welling [4].

As introduced by Henaff et al. [5], spectral networks are a generalization of convolutional networks with Graph Fourier Transform using the normalized Laplacian. Let N be the number of nodes of a graph $\mathcal{G}$ defined by a $N \times N$ symmetric similarity matrix W, D be the degree matrix, the normalized graph Laplacian is

$$L = I - D^{-1/2} W D^{-1/2}$$

Let U be the matrix of eigenvectors of L which plays the role of the Fourier Transform. The graph convolutions of an input x with a filter g is defined by :

$$x \star g = U^T (Ux \odot Ug) \text{ with } \odot \text{ the point-wise product.}$$

For large graphs, computing the graph Laplacian is computationally expensive so we need to reduce the complexity of the model in the next steps (graph convolutions are also computationally expensive). We are using pooling (average and max pooling) as we would for image recognition thanks to their ability to keep the data relations without the absolute data information. The same layers are defined with graphs using neighborhood and multi-resolution spectral-clustering.

The GCN is a stack of graph convolutions where each layer can then be written as a non-linear function f :

$$H^{(l+1)} = f(H^{(l)}, W)$$

with $H^{(l)}$ the feature matrix of l^{th} layer. Let X be the $N \times D$ feature matrix which contains a description for each node (N denoting the number of samples and D the number of input features). The first network layer is initialized with $H^{(0)} = X$. We can use softmax or ReLU activation [6] and the weights g are learned according to a gradient descent approach using batch of size the full dataset at each epoch. Finally, using a first-order approximation of spectral filters on graphs, we have the following layer-wise propagation rule :

$$H^{(l+1)} = activation(D^{-1/2} W D^{-1/2} H^{(l)} G^{(l)}) \text{ with } G^{(l)} \text{ layer-specific learnable filters.}$$

Repeated multiplication with the adjacency matrix can lead to exploding or vanishing gradients when used in a neural network model. To fix this, we simply add the identity matrix to the similarity matrix $W \leftarrow W + I_N$ as it can be done in a similar way in residual networks [7] and so the degree matrix defined by $D_{ii} \leftarrow \sum_j W_{ij}$ is also preprocessed .
The multi-layer GCN are illustrated in Figure 1.

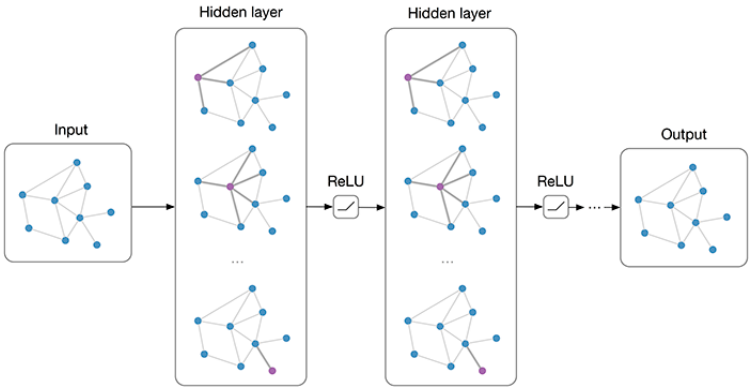

Figure 1: Multi-layer Graph Convolutional Network (GCN) with first-order filters *(source : T. Kipf's website)*

3.2 Graph Convolution for Matrix Completion

The project we undertake can be seen as a link prediction task where we try to recover missing values and hence, it can be processed as a matrix completion problem. In order to tackle this problem, Kipf et al. have introduced Graph Auto-Encoders (GAE) [8]. The underlying model makes use of latent variables (embeddings) to learn interpretable representations and correlations on undirected graphs.

3.2.1 Graph Auto-Encoder

The classical version of an Auto-Encoder is a neural network that is trained to attempt to reproduce its input as its output. The network is composed of two parts: the first encodes the input $h = f(x)$ and the second uses a decoder to reconstruct the encoded input $y = g(h)$. Auto-encoders are restrained in certain ways to avoid building a neural network which is the identity. The way they are constructed forces the prioritization of the most important features and aspects of the input data and thus, captures the most salient features of the input. Auto-encoders are handy for feature learning and dimensionality reduction. They constrain the dimension of the hidden layer h which is often smaller than x and learn an under-complete representation of the input [9]. Actually, a simple auto-encoder often ends up learning a low-dimensional representation similar to PCAs.

The Graph Auto-Encoder version differs slightly buth the goal stays exactly the same, namely learn correlations in the input data in order to construct another representation in the output data. For the matrix completion problem, we can add additional data (side information) that helps the auto-encoder learn the most valuable representation, in order to predict a reconstructed (completed) representation of the input data. GAEs are end-to-end trainable neural network models based on GCN.

3.2.2 Bipartite Graph

Interaction data such as question answering can be represented as a bipartite user-item graph with labeled edges denoting binary success ratings [10]. We can see this problem as a link prediction task where the prediction would be whether the user succeeded in correctly answering a question he has never encountered (unobserved) as described in Figure 2.

Concretely, taking N_u the number of users and N_v the number of questions, the encoding part consists in building two matrices U, V of user and item embeddings :

$$[U, V] = f(X, M_1, M_2)$$

where X is the feature matrix (side information) and M_1 (respectively M_2) is the adjacency matrix associated with Success (respectively Failure). In the Graph Convolutional Matrix Completion (GC-MC) [11], we make use of local graph convolutions and we assign a specific transformation to each type parameter (Success/Failure) such that we obtain two types of specific messages. This encoding helps the network detect correlations and intrications in the bipartite graph.
As for the decoding part - once the network has captured the most salient features - we compute

$$\widetilde{M} = g(U, V)$$

that acts both on the User and Item embeddings. Finally, the outcome is a reconstructed (and completed) bipartite graph with predicted links on unobserved interactions.

The added-value of this approach is that it treats each parameter (Success/Failure) as a separate class that has a specific encoding and hence, it guides the hidden representation.

For link prediction in students' answers validity, the encoder contains a graph convolution layer that constructs user and questions embeddings on the bipartite user-question interaction graph. Combined with a decoder, new answers are predicted in the form of new labeled edges. The GAE generalizes to include side information for questions (similarity matrix with correlations between questions).

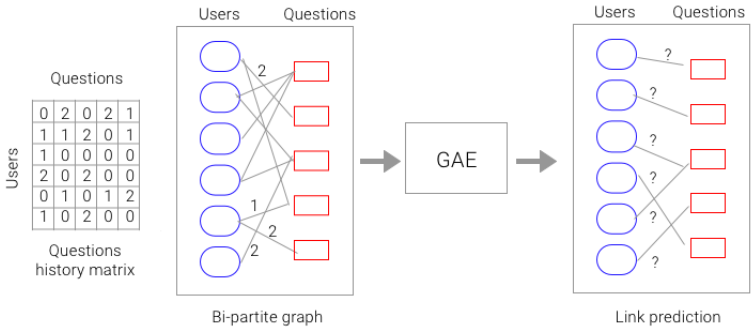

Figure 2: *Left :* Questions history matrix with 0 for unobserved entries, 1 for failure and 2 for success. *Right :* Architecture for link prediction using a bipartite graph between users and questions

4 Experiments

The aim of this project is to predict a student's answer to an educational question using knowledge tracing. We aim at using GCN and GAE to predict students' answers to unobserved questions, making use of their history of answers to other questions.

4.1 Lelivrescolaire's dataset

We have been furnished *lelivrescolaire* dataset which contains the history of answers for multiple students from different sessions and the questions features related to the answers.
This database has $m \sim 25$k different students and $n \sim 15$k unique questions. For each exercise, we have the information of whether the student has answered correctly or skipped and complementary information about the exercise (time information, topic, chapter, difficulty, type of question, etc).
The first step was to build a graph of questions : each node represents a questions and the edges correspond to correlations between questions. We first computed the mean for five parameters : if the question is skipped, its validity, type, difficulty, topic (when available) and the time spent by the student before answering. Then, we calculated the pairwise L_1-distance (Manhattan) between questions for the categorical features and a pairwise Canberra distance for the continuous last feature (student's spent time). The Canberra distance is a weighted version of L_1 distance :

$$d_C(p,q) = \sum_i \frac{|p_i - q_i|}{|p_i| + |q_i|}$$

We give more significant weights to the most answered questions and average them to create a similarity measure. Taking $V = \{\textit{spentTime, skip, valid, type, difficulty, topic}\}$, we have :

$$W'_{i,j} = \sum_{k \in V \setminus \{\text{spentTime}\}} | \, k_j - k_i \, | + d_C(\text{spentTime}_i, \text{spentTime}_j)$$

$$C_{i,j} = \begin{cases} 0.8, & \text{if } | \, \text{nbUser}_i - \text{nbUser}_j \, | > q_{0.75} - q_{0.25}. \\ 0.9, & \text{if } | \, \text{nbUser}_i - \text{nbUser}_j \, | > q_{0.75} - q_{0.50} \\ 1.0 & \text{otherwise} \end{cases}$$

$$W_{i,j} = 1 - \frac{1}{6} W'_{i,j} \odot C_{i,j}$$

Eventually, we had a $n \times n$ adjacency matrix representing our graph of questions. We then needed to build knowledge graphs that are meaningful for our application.

4.2 Graph Convolution Network

Before going deeper on predicting answers for multiple students at a time, we found it useful to test on synthesized datasets. Indeed, the major issue and challenge stands in handling missing values in the feature matrix, that correspond to the case where a student has never encountered a question. Usually, in classical deep learning approaches, we have an input matrix and an output matrix/vector, both of same shape; and when training, we have to be cautious to only select the part of the dataset that is complete.

4.2.1 Random dataset for GCN

In order to understand how GCNs work, we have used Keras-based implementation for semi-supervised classification by Kipf [12]. Keras is a minimalist Python library for deep learning on top of Tensorflow backend.

We have synthetized a random dataset : a 2500×2500 adjacency matrix, a 2500×600 feature matrix and 2500×2 binary label matrix. The aim is to use the feature matrix to predict a label for each node using the correlations between nodes of the graph. With random data and binary labels, we were conscious that we could not have an accuracy better than 50%. The goal of this part was mainly to get familiar with the implementation and how we could take advantage of it.

As explained previously, we need to preprocess our dataset. We add the identity matrix to the adjacency matrix to avoid exploding or vanishing gradients and we normalize the feature matrix. First, we made the assumption that there was no missing values and thus we achieved accuracy of around 0.5 which is the expected value for a randomized dataset.
The main issue with an actual dataset is indeed the lack of information. We often encounter missing information in the feature matrix and we cannot keep *Not-a-Number* (NaN) values, otherwise we would get an infinite loss. We have binary data so it means two categories 0 and 1, and NaN values. We use a simple translation to encode NaN as 0s ($1 \leftarrow 2$, $0 \leftarrow 1$, $NaN \leftarrow 0$). With neural networks, if there are missing input values, it is recommended to fill them with 0s (once the input has been preprocessed). Indeed, when we develop the chain rule, we easily understand that an input value filled with 0s would cancel the loss gradient so the the weights regarding that input value would remain stable. When adding NaN values for missing data, we got around the same accuracy but it is difficult to assume that the network had learned the way it should.

The GCN model architecture is straight-forward - as described in section 3 - with two graph convolution blocks, each block including a dropout layer (with 50% of dropout rate, to prevent overfitting [13]) and a graph convolution layer. The first block contains a ReLU activation with a L_2- weights regularizer although the second one contains a softmax activation. The GCN is compiled with a categorical cross entropy loss and an Adam optimizer [14]. It runs pretty quickly but obviously does not learn anything because of the randomness of the data. However, we were then able to try and adapt this implementation on the actual dataset.

4.2.2 Application to the educational database

The dataset furnished by Lelivrescolaire is very sparse, with around fifty questions answered per student (for a total number of $\sim$ 15k). Once all preliminar pre-processing was achieved and in order to reduce the size of our data, we decided to focus on only one topic (*'Histoire 3e'*), which led us to having $n = 878$ questions and $m = 290$ students.

We tried two types of encoding regarding the feature matrix: the first one with the unobserved data encoded as NaN (Not a Value) and applying a custom loss (cross-entropy error) on top where we mask out all the unobserved values; and the second one by pre-processing the input data such that the unobserved values are encoded as 0s.

For the first encoding, we implemented a custom loss function on TensorFlow such that, when unobserved values are encountered, we mask them and propagate the loss only for not NaN entries :

- The feature matrix of shape (n, m) contains 1 (valid answer), 0 (non-valid answer) and NaN (unobserved).
- The adjacency matrix of shape (n, n) includes the similarities between the exercises
- The output that is a vector of shape $(n \times m, 1)$ represents the history matrix (n, m) flattened (which contains 0,1)

It seems that the model does not learn from the input: our custom loss does not decrease along epochs.

Then, after discussing with Kipf, we attempted a second encoding using 2 for valid, 1 for non-valid and 0 for NaN, as following :

- The feature matrix of shape (n, $2m$) which contains only 1 and 0 for valid/not valid answers. This matrix encodes the history of answers by the students. Indeed, for each exercise, there are two columns : 1 (not-valid) and 2 (valid) to get a matrix filled with dummy variables :
  - If the student has correctly answered the exercise, the column encoding non-validity ($exercice\_id\_1$) is filled with 0 and the column encoding validity ($exercice\_id\_2$) is filled with 1
  - If the student has not correctly answered the exercise, the column encoding non-validity ($exercice\_id\_1$) is filled with 1 and the column encoding validity ($exercice\_id\_2$) is filled with 0
  - If the student has not done the exercise, the column encoding non-validity ($exercice\_id\_1$) is filled with 0 and the column encoding validity ($exercice\_id\_2$) is filled with 0

  This encoding is illustrated in Figure 3.
- We keep the same (n, n) adjacency matrix
- We predict the same $(n \times m, 1)$ output

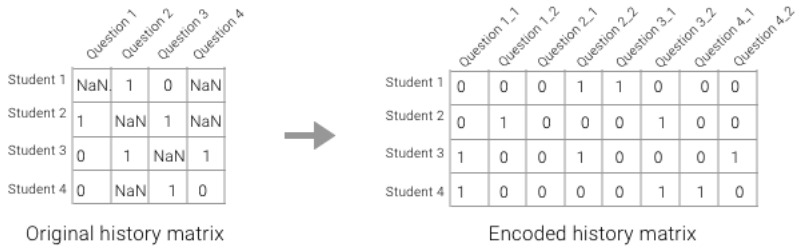

Figure 3: Feature matrix encoding example

The main advantage with this encoding is that we do not need a custom loss anymore. Indeed, as said previously, the weights remains stable in the neural network whenever we have 0s. So, we are using a regular binary cross-entropy loss. Sadly, even with this encoding, the loss does not decrease.

4.3 Graph Convolutional Matrix Completion

After unexpected results with GCN, we aim at adapting the code furnished by Kipf on GAE to our problem [15] by achieving link prediction with matrix completion tasks. We would like to predict a student's answer to unobserved question as described in Figure 2. We base our approach on the one proposed by van den Berg [11] which totally relies on GAE, which themselves depend on GCN. In the cited paper, the authors' goal is to predict music ratings using a bipartite graph representing user/music interactions and as side information, the similarity between the songs. By considering the questions as the musics and the ratings as a binary variable (Success/Failure), we get exactly the problem we would like to solve.

GAE learn new representations of the input adjacency matrix to predict new weighted edges between nodes (completing the similarity matrix) by using the feature matrix as side information. The model is adapted to the prediction of edges between either users or questions (in the adjacency matrix) using the feature matrix to help the auto-encoder detect the most valuable features and thus, the intrications. It is implemented to make use of two graph convolution layers. The first one corresponds to the sparse hidden layer with 32 units and a ReLU activation although the second one is a basic graph convolution with 16 units and a ReLU activation too, both having a L_2 weight normalization. This neural network uses a custom Adam optimizer for graphs.

We aim at predicting weighted edges between users and questions, e.g. the validity of the answer to a question by a student. We model this interaction by a bipartite graph where the nodes on the left represent the users and the ones on the right the questions. The weights on the edges are binary encoding valid (2) or not valid (1) and we would like to predict new links as shown in Figure 4.

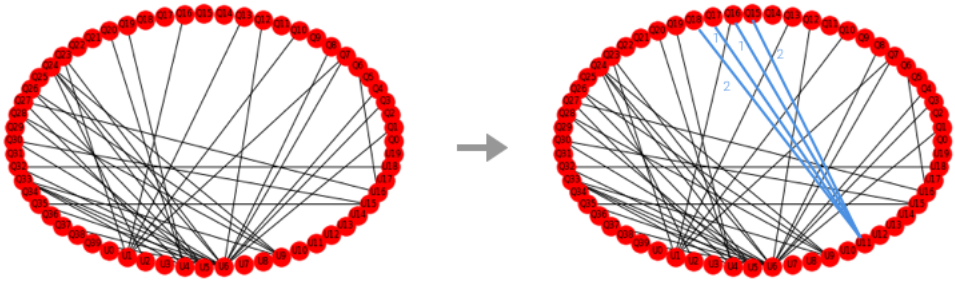

Figure 4: Questions (Q) and Users (U) are plotted on the same graph. *Left :* Initial Bi-Partite Graphs with questions and users. *Right :* Bi-Partite Graph with new predicted edges for 'User 11' and for four unencountered questions (from 'Question 15' to 'Question 18')

Because of the lack of time and the failures we faced with GCN, we did not have enough time to implement GAEs to our problem but we present how we would have achieved it.

Indeed, we would use a (n, m) feature matrix as the one we used for the random dataset with GCN. The encoding is still 2 for a valid answer, 1 for a wrong answer and 0 for no answer. In order to reduce the number of edges and to avoid having a fully connected graph (which indeed does not make sense because all questions should not connected i.e. knowledge regarding a subject might be independent from the knowledge with respect to another subject), we threshold the questions adjacency matrix. As side information, we would of course use the questions adjacency matrix denoting the similarities.

In order to measure the performance of the model, we would need to split the dataset into three sets: training (70%) , validation (15%) and test (15%). We would also need to pre-process the data as exposed previously (adding the identity matrix and normalization). Measure of performance of our model would have been done using two metrics : the average precision score (ratio of true positive predictions and all positive predictions) and the area under the ROC curve (AUC, equal to the probability that the model rank a random positive example higher than a random negative one).

In a future work, we would like to have more time to implement this model which seems particularly interesting and pertinent with respect to our problem. This approach seems to perform well for the similar tasks introduced by van den Berg, Kipf and Welling [11], so we are hopeful that it can be transposed to our knowledge tracing problem.

4.4 Extensions

As explained previously, we have manually built an adjacency matrix for the questions. In order to add information, we could have also built it for the users taking into account the results and their behavior regarding the different questions. Then, we would use this similarity matrix as an added side information. Indeed, in classical recommender systems, we would use collaborative filtering in order to link together the users based on their past performance and their rate of success. Then, we would have tried to complete their matrix of ratings and finally we would have recommended items that are "neighbors" (the notion of proximity depends on the distance we apply). Collaborative Filtering

succeeds in recommending items either in a content-based way (linked questions) or in a user-based approach (linked users) and can even combine the two [16]. Hence, using both collaborative filtering to link together users and similarity matrix to connect questions as side information could have led to better results.

Moreover, we could have improved our question adjacency matrix. Indeed, we have correlated questions based on only a small portion of the available features (difficulty, question type, topic, etc.) and questions' answers (time spent, success, skipped, etc.) across the students. However, one could think that adding information about the structure and content of the question would give more insights and connect more accurately the items (questions). We could have used the content of the questions and applied some NLP techniques such as a Graph-of-words in order to target the most relevant words and encode context [17]. The previous would find new correlations and compute the weights on the graph represented by the adjacency matrix. Ideally, we would have built a similarity matrix such that $A = B_{NLP} + \alpha C_{features}$ with A the adjacency matrix representing the similarities between the questions based on NLP techniques and questions features, and α empirically chosen (using cross-validation).

During our project, we had to restrain our dataset to few questions and few students because the full database was too computationally expensive. We made the naive assumption that between two topics, the questions are not related but if we had added NLP techniques to encode the similarity, this hypothesis does not stand anymore. In a future work, we would like to use a more powerful server to work with the entire knowledge graph and history matrix.

5 Conclusion

To conclude, let us recall that the ultimate goal of the project was to predict unobserved entries as a knowledge tracing task. We first achieved the construction of a knowledge graph linking together questions empirically using students' answers history and some features of the questions. Then, we tackled the problem using two different approaches both based on graph convolutions. The first endeavors to make predictions on a feature matrix thanks to the similarity matrix of questions. As for the second, it completes the matrix of features using as side information the adjacency matrix to help the network encode the most salient feature.

Using Graph Convolutional Networks for this task was ambitious and with uncertain results. We struggled to adapt Kipf's implementations to our tasks because our project is indeed a cross-task problem. Eventually, we have learned how to use convolutional networks on graphs, when to use them and when they might work. We also discovered other neural networks methods for graphs such as graph auto-encoders for the task of graph matrix completion.
We wished we had more time for this interesting project, in order to succeed in our experiments of predicting students' answers with GCN and GC-MC and finally, compare the two previous.

Acknowledgments

We would like to deeply thank Julien Seznec for the helpful and enriching discussions during our e-mail and numerous video conversations and also for providing us an interesting and meaningful dataset.
We would also like to thank Thomas Kipf for taking the time to answer some of our questions and for having enlightened some tricky points by e-mail.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[2] C. Piech, J. Spencer, J. Huang, S. Ganguli, M. Sahami, L. Guibas, and J. Sohl-Dickstein. Deep Knowledge Tracing. *ArXiv e-prints*, June 2015.

[3] F. Monti, M. M. Bronstein, and X. Bresson. Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. *ArXiv e-prints*, April 2017.

[4] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv e-prints*, September 2016.

[5] M. Henaff, J. Bruna, and Y. LeCun. Deep Convolutional Networks on Graph-Structured Data. *ArXiv e-prints*, June 2015.

[6] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling Relational Data with Graph Convolutional Networks. *ArXiv e-prints*, March 2017.

[11] R. van den Berg, T. N. Kipf, and M. Welling. Graph Convolutional Matrix Completion. *ArXiv e-prints*, June 2017.

[12] Deep learning on graphs with keras. `https://github.com/tkipf/keras-gcn`. Accessed: 2017-12-15.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[15] Implementation of graph auto-encoders in tensorflow. `https://github.com/tkipf/gae`. Accessed: 2018-01-2.

[16] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 501–508, New York, NY, USA, 2006. ACM.

[17] François Rousseau and Michalis Vazirgiannis. Graph-of-word and tw-idf: New approach to ad hoc ir. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 59–68, New York, NY, USA, 2013. ACM.