Link Prediction On Large Graph Data

By

Sourin Ghatak
Sachin Kumar
Subham Kothari
Subham Karmakar

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

Session 2016-2017

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY
[Affiliated to West Bengal University of Technology]
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA-700015

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
RCC INSTITUTE OF INFORMATION TECHNOLOGY

श्रमम् बिना न किमपि साध्यम्

TO WHOM IT MAY CONCERN

I hereby recommend that the Project entitled **Link Prediction on Large Graph Data** prepared under my supervision by **Sourin Ghatak** (Reg. No. 14117011072, Class Roll No. CSE/2014/062), **Subham Kothari** (Reg. No. 141170110079, Class Roll No. CSE/2014/065), **Sachin Kumar** (Reg. No. 141170110054, Class Roll No. CSE/2014/086), **Subham Karmakar** (Reg. No. 141170110078, Class Roll No. CSE/2014/082) of B.Tech (7th /8th Semester), may be accepted in partial fulfillment for the degree of **Bachelor of Technology in Computer Science & Engineering** under Maulana Abul Kalam Azad University of Technology (MAKAUT) formely known as WBUT.

....

Project Supervisor
Department of Computer Science and Engineering
RCC Institute of Information Technology

Countersigned:

....
Head
Department of Computer Sc. & Engg,
RCC Institute of Information Technology
Kolkata – 700015.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
RCC INSTITUTE OF INFORMATION TECHNOLOGY

CERTIFICATE OF APPROVAL

The foregoing Project is hereby accepted as a credible study of engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve the project only for the purpose for which it is submitted.

FINAL EXAMINATION FOR EVALUATION OF PROJECT	1. ______________________________
	2. ______________________________
	(Signature of Examiners)

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our teacher Mr Koushik Mullick, who gave us the golden opportunity to do this wonderful project on the topic Link Prediction on Large Graph Data, which also helped us in doing a lot of Research and we came to know about so many new things. We are really thankful to him.

Table of Contents

Contents **Page No.**

Introduction:

Real-world networks evolve over time as new nodes and links are added. Link prediction algorithms use historical data in order to predict the appearance of a new links in the network or to identify links which may exist but are not represented in the data. The application of link prediction is most commonly seen in recommendation engines, such as new connections on social networks or related products on shopping sites. Traditional approaches involve the calculation of a heuristic similarity score for a pair of nodes, such as the number of common neighbours or the shortest path length connecting the nodes, where pairs of nodes with the highest similarity scores are considered the most likely edges.

Link prediction is an important task for analyzing social networks which also has applications in other domains like, information retrieval, bio-informatics and e-commerce. There exist a variety of techniques for link prediction, ranging from feature-based classification and kernel based method to matrix factorization and probabilistic graphical models. These methods differ from each other with respect to model complexity, prediction performance, scalability, and generalization ability. Link prediction aims to uncover the underlying relationship behind networks, which could be utilized to predict missing edges or identify the spurious edges. The key issue of link prediction is to estimate the likelihood of potential links in networks. Most classical static-structure based methods ignore the temporal aspects of networks, limited by the time-varying features. Such approaches perform poorly in evolving networks.

One could apply Link Prediction algorithms to virtually any domain that can be represented as a graph without supervision. The complexity of achieving good performance on the LP task increases with the graph size, as does the problems at faithfully evaluating performance. When a graph grows linearly in vertices, the number of possible links within the graph grows quadratically. This defines a needle in a haystack context where relevant or useful predictions are a tiny fraction of all predictions. Keeping a good precision in this type of problem turns out to be very difficult, as the smallest false positive acceptance rate will amount to huge absolute number of wrongfully predicted edges (i.e. false positives). But in parallel, estimating the quality and applicability of results also becomes particularly difficult.

Review of Literature

There has been a lot of work done in the field of link prediction with pretty interesting results over different datasets. It has been one of the most worked topic since quite a time and has been successfully gathering a lot of attention. The popularization of graph-based data sets (i.e., networks) in the early 21 century motivated research on a new family of machine learning methods. Link prediction problem can be described as a inference of predicting future links/edges of a graph based on present links. So, it is a process extracting knowledge from the present network and deploying them for future exploration [1]. Therefore, link prediction in networks such as the World Wide Web, where the network dynamics evolves overtime, would be prized. These kinds of networks as described by Li et al. [2] are known as

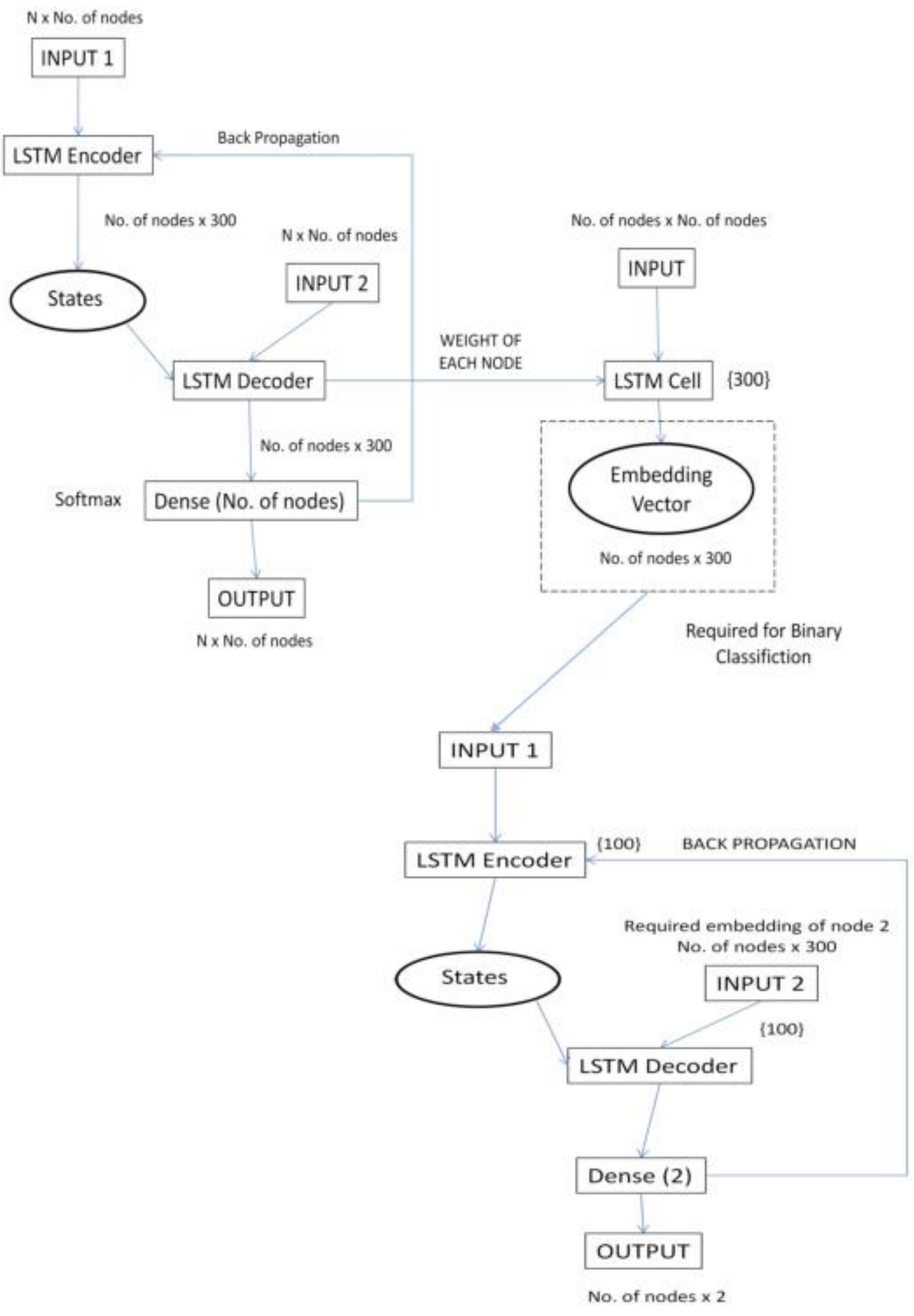
Dynamic Networks. Link prediction has a wide variety of applications from Recommendation Systems, Social Networks to biological data analysis in Molecular Interactions (MI), Protein-Protein Interactions (PPI) and Drug- Target Interactions (DTI). Mainly there are two approaches to solve the link prediction problem; (i) collecting the features from graph topology information [3] and (ii) extracting the individual feature of nodes.

It was proposed to use neural networks to map sequences to sequences [4] [5] [6] .This framework has been used for neural machine translation and achieves improvements on the English-French and English-German translation tasks from the WMT'14 dataset [7]. It has also been used for other tasks such as parsing [8] and image captioning [9]. Since it is well known that vanilla RNNs suffer from vanishing gradients, most researchers use variants of the Long Short Term Memory (LSTM) recurrent neural network [10]. Kalchbrenner and Blunsom [4] were the first to map the input sentence into a vector and then back to a sentence, although they map sentences to vectors using convolutional neural networks, which lose the ordering of the words. Similarly to this work, Cho et al. [5] used LSTM-like RNN architecture to map sentences into vectors and back, although their primary focus was on integrating their neural network into an SMT system. Bahdanau et al. [2] also attempted direct translations with a neural network that used an attention mechanism to overcome the poor performance on long sentences experienced by Cho et al. [11] and achieved encouraging results. Likewise, Pouget-Abadie et al. [12] attempted to address the memory problem of Cho et al. [11] by translating pieces of the source sentence in way that produces smooth translations, which is similar to a phrase-based approach. End-to-end training is also the focus of Hermann et al. [13], whose model represents the inputs and outputs by feed forward networks, and map them to similar points in space. However, their approach cannot generate translations directly: to get a translation, they need to do a look up for closest vector in the pre-computed database of sentences, or to rescore a sentence.

Objective of the Project

Given a large graph data containing known links/edges between the nodes/vertices, our objective is to learn the function by which the nodes have established a relationship with other nodes. The learned function can then be implemented to establish probable links among the unknown pair of nodes.

System Design



Methodology for implementation (Formulation/Algorithm)

Here we convert the paired nodes into paired and fixed-length vectors and pass it through a binary classifier to learn the interaction between them.
The input node is converted into a vector of dimensions having maximum existing nodes in the graph using one-hot encoding. The following LSTM encoder converts the vectors into a fixed given length vectors. The hidden states and the cell states of the encoder is calculated. The calculated states are used as context for the decoder LSTM to reproduce the output node which is given as input for the decoder. The output sequence from the decoder is fed into a dense layer activated by a softmax function having classes equal to maximum number of nodes existing in the graph. After training the model on the dataset, we now have updated hidden and cell states of both the encoder and the decoder as well.
To generate the required embedding for each node we are using the updated weight matrix of the nodes from the decoder as it is closest to the output. This weight matrix is preset in a separate LSTM cell having the inputs as square matrix of the form n*n, where n = maximum number of nodes existing in the graph, which means all the nodes are passed as one-hot encoding into the LSTM cell. Now, based on the preset weight matrix, we made the prediction with LSTM having a fixed dimension. The output sequence given by the LSTM cell is treated as our required embedding which is further fed as the input for the corresponding nodes into the classifier which determines whether a link exists between a pair of nodes or not.

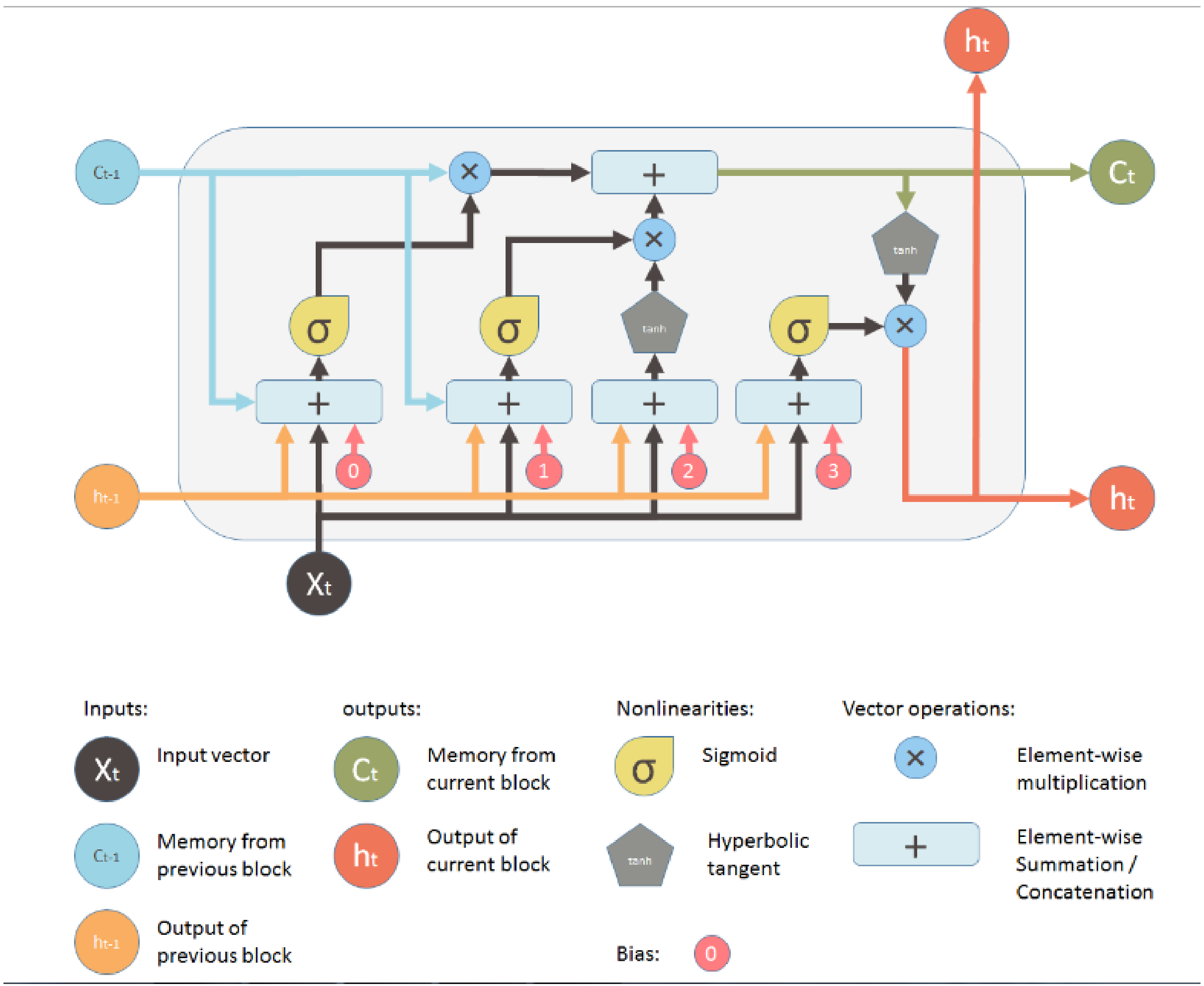
Implementation Details

About LSTM:

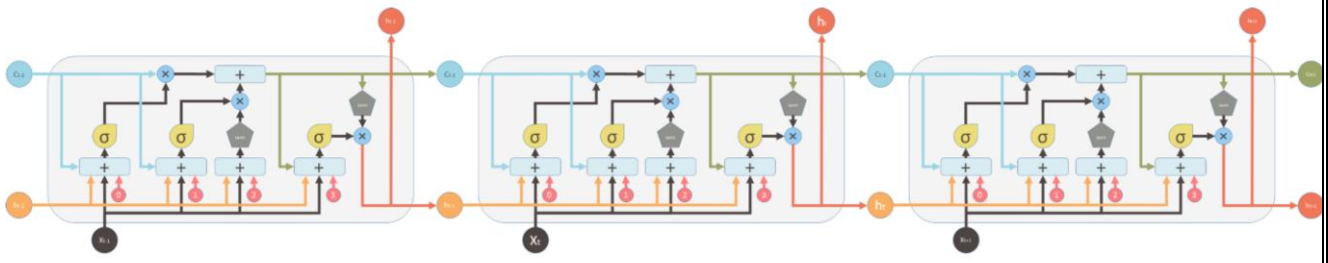
Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

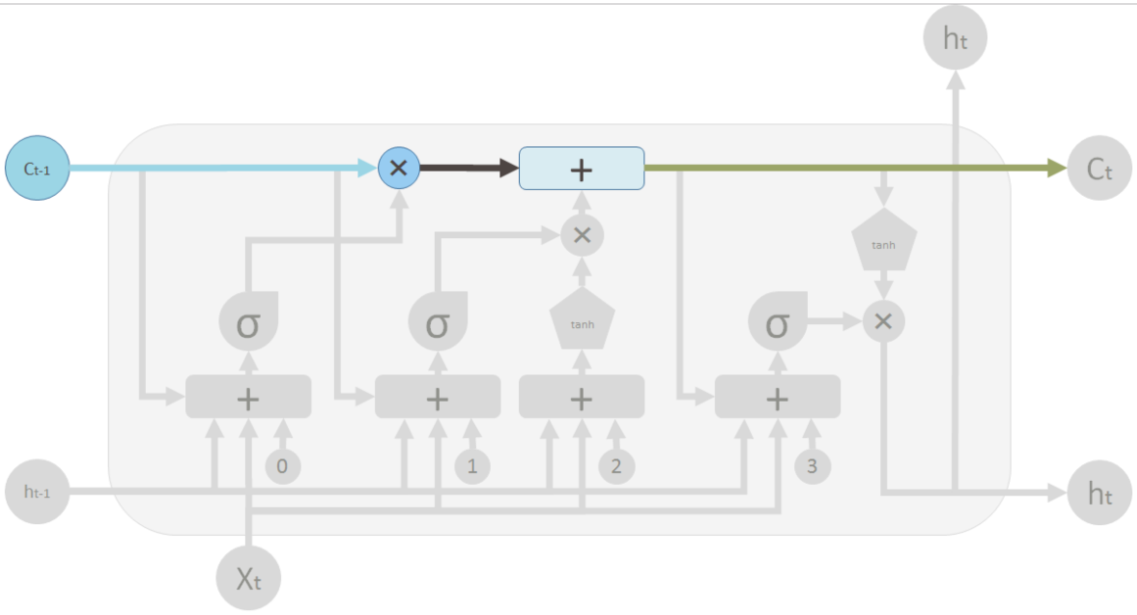


At a first sight, this looks intimidating. Let's ignore the internals, but only look at the inputs and outputs of the unit. The network takes three inputs. X_t is the input of the current time step. H_t-1 is the output from the previous LSTM unit and C_t-1 is the "memory" of the previous unit, which I think is the most important input. As for outputs, h_t is the output of the current network. C_t is the memory of the current unit.
Therefore, this single unit makes decision by considering the current input, previous output and previous memory. And it generates a new output and alters its memory.



The way its internal memory C_t changes are pretty similar to piping water through a pipe. Assuming the memory is water, it flows into a pipe. You want to change this memory flow along the way and this change is controlled by two valves.

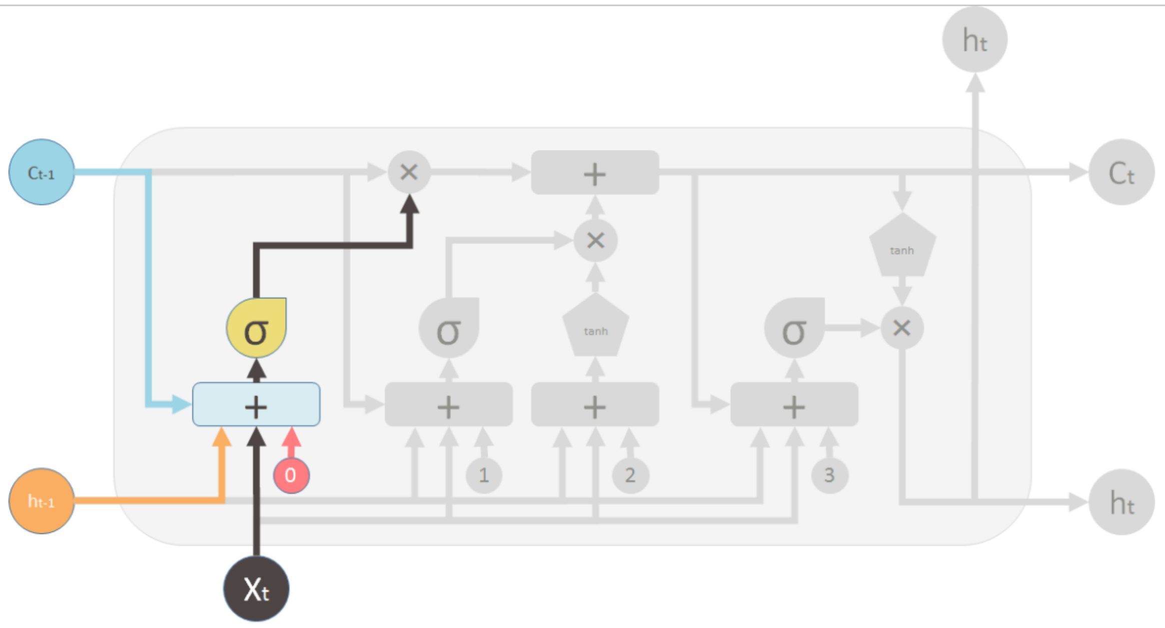
The second valve is the new memory valve. New memory will come in through a T shaped joint like above and merge with the old memory. Exactly how much new memory should come in is controlled by the second valve.



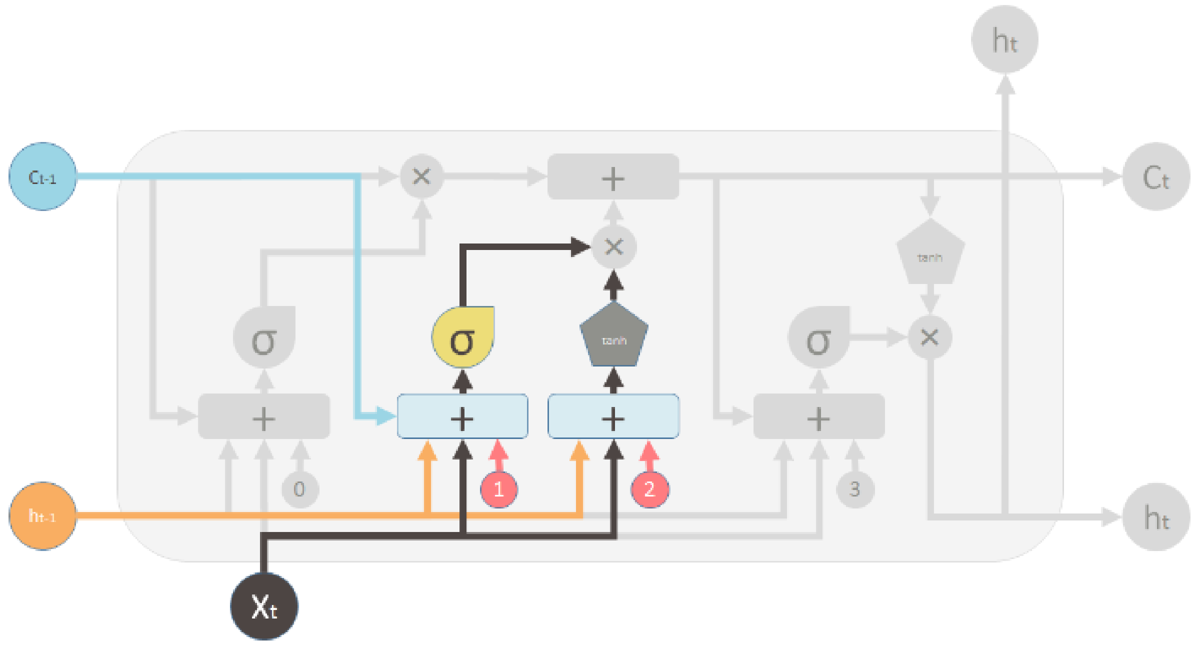
On the LSTM diagram, the top "pipe" is the memory pipe. The input is the old memory (a vector). The first cross ✖ it passes through is the forget valve. It is actually an element-wise multiplication operation. So if you multiply the old memory C_t-1 with a vector that is close to 0, that means you want to forget most of the old memory. You let the old memory goes through, if your forget valve equals 1.

Then the second operation the memory flow will go through is this + operator. This operator means piece-wise summation. It resembles the T shape joint pipe. New memory and the old memory will merge by this operation. How much new memory should be added to the old memory is controlled by another valve, the ✖ below the + sign.

After these two operations, you have the old memory C_t-1 changed to the new memory C_t.

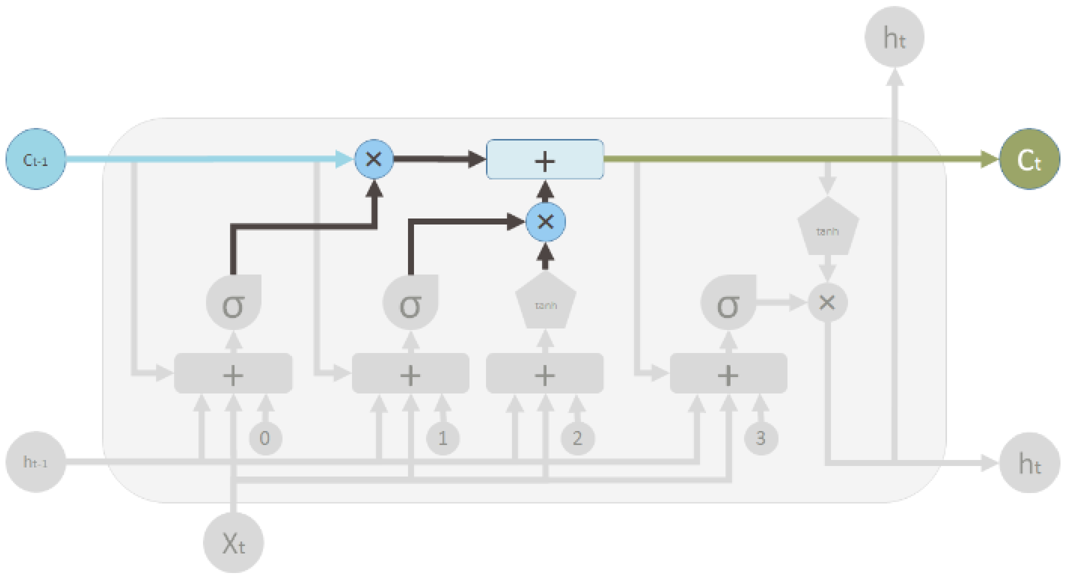


Now let’s look at the valves. The first one is called the forget valve. It is controlled by a simple one layer neural network. The inputs of the neural network is h_t-1, the output of the previous LSTM block, X_t, the input for the current LSTM block, C_t-1, the memory of the previous block and finally a bias vector b_0. This neural network has a sigmoid function as activation, and its output vector is the forget valve, which will applied to the old memory C_t-1 by element-wise multiplication.

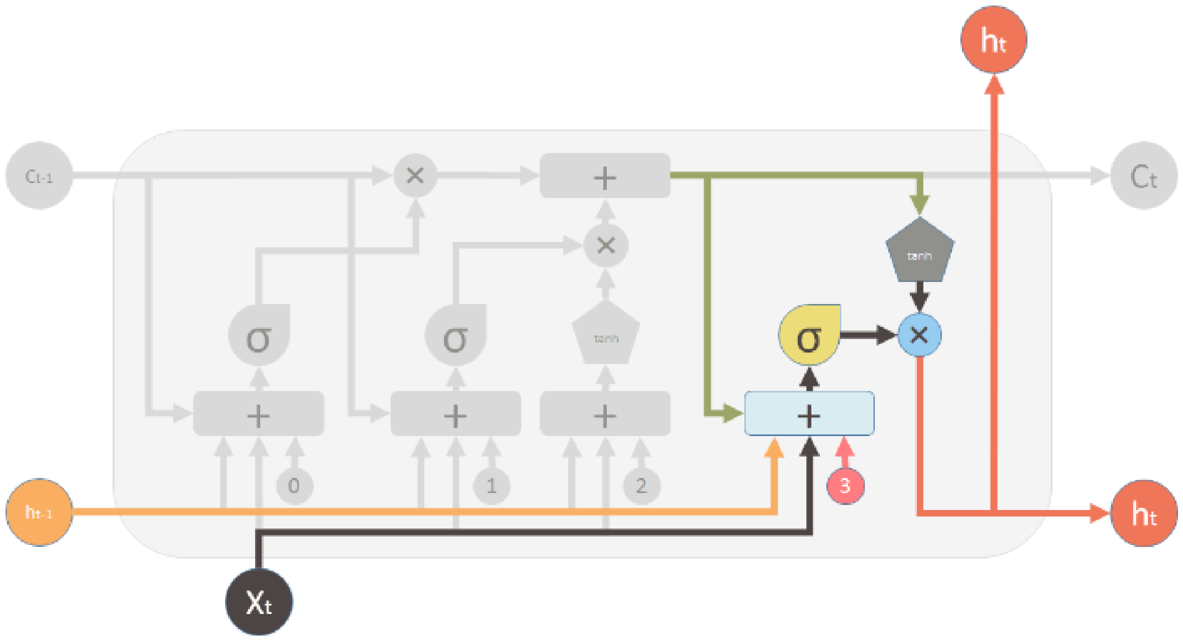


Now the second valve is called the new memory valve. Again, it is a one layer simple neural network that takes the same inputs as the forget valve. This valve controls how much the new memory should influence the old memory.

The new memory itself however is generated by another neural network. It is also a one layer network, but uses tanh as the activation function. The output of this network will element-wise multiple the new memory valves, and adds to the old memory to form the new memory.



These two ✖ signs are the forget valve and the new memory valve.



And finally, we need to generate the output for this LSTM unit. This step has an output valve that is controlled by the new memory, the previous output h_t-1, the input X_t and a bias vector. This valve controls how much new memory should output to the next LSTM unit.

Model

We have designed an encoder-decoder model for the purpose of generating an embedding vector of the input and use the same as a context for the output. After training the model on the complete dataset, we extract the features of the nodes i.e., the embedding vector of the nodes. The extracted features of the nodes are used to represent the nodes themselves in another similar type of model used as a binary classifier.

Formulae and Data flow

The following are the formulae on which LSTMs works.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{1}$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{2}$$

$$o_t = \sigma_g(W_0 x_t + U_0 h_{t-1} + b_0) \tag{3}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{4}$$

$$h_t = o_t \circ \sigma_h(c_t) \tag{5}$$

where

x_t : input vector to the LSTM unit

f_t : forget gate's activation vector

i_t : input gate's activation vector

o_t : output gate's activation vector

h_t : output vector of the LSTM unit

c_t : cell state vector

W, U and b : weight matrices and bias vector parameters which need to be learned during training.

So this is how an LSTM calculates and updates its weight matrix, hidden states and cell states.

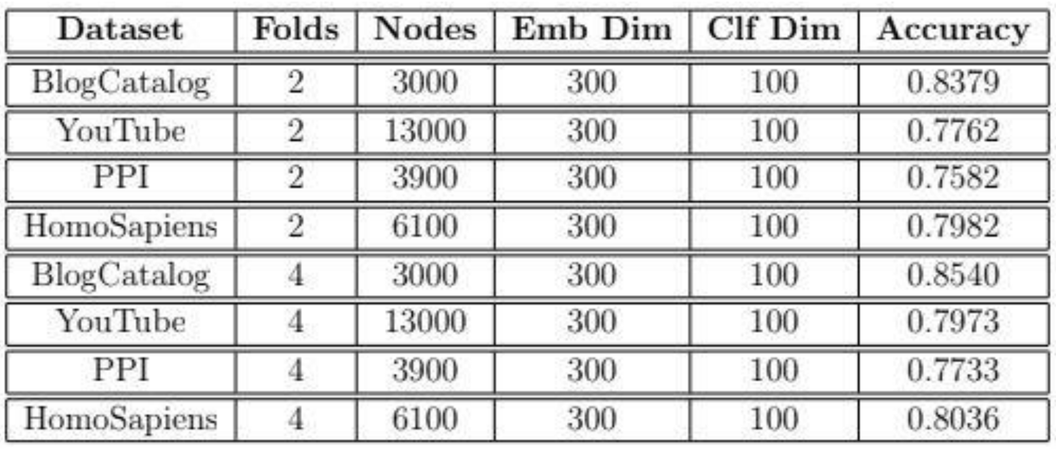
The decoder LSTM is followed by a softmax function which is as follows,

$$p(x_{t,j} \mid x_{t-1}, \ldots\ldots, x_1) = \frac{exp(w_j h_{\langle t \rangle})}{\sum_{j'=1}^{k} exp(w_{j'} h_{\langle t \rangle})} \tag{6}$$

Now we can move to the flow of data inside our algorithm. At first we are sampling the graph data on the basis of DFS (Depth First Search) which extracts the pairs on the basis of nearest neighbours topology and getting the links as sources and destinations. Now for the embedding generation model, the training is done as a neural machine translation from the source node to destination node and vice-versa, to make sure that if one node leads to a particular node, then the reverse is also true. After this training, the embedding for the nodes are generated by a simple LSTM layer set by the weights of the decoder LSTM after the model is trained. The one-hot encoded input of all the nodes are then fed into the LSTM mentioned just now and the output is to be treated as the embedding for the nodes. The embedding for the nodes now represent the nodes themselves for the classifier which is again based encoder-decoder algorithm in which the two nodes of a pair and treated as the two inputs of the model and the output is either 0 or 1. As said earlier the nodes are now represented by their embedding that is generated by the previous translation model. Now these embedding of the nodes are fed into the classifier model as inputs to both encoder and decoder. The model learns the embedding of the nodes instead of the nodes' number and then learns the functions for the embedding of two nodes so that they are linked or not linked.

Results

Our results are based on the embedding vectors generated at 300 dimensions for the model which generates the vectors and dimension of 100 for the model which classifies the pair as linked or not linked. We have tested our algorithm on various datasets namely BlogCatalog, PPI, YouTube and HomoSapiens graph data. The results are provided in the table below,



Dataset	Folds	Nodes	Emb Dim	Clf Dim	Accuracy
BlogCatalog	2	3000	300	100	0.8379
YouTube	2	13000	300	100	0.7762
PPI	2	3900	300	100	0.7582
HomoSapiens	2	6100	300	100	0.7982
BlogCatalog	4	3000	300	100	0.8540
YouTube	4	13000	300	100	0.7973
PPI	4	3900	300	100	0.7733
HomoSapiens	4	6100	300	100	0.8036

Conclusion

Hence, after trying many different algorithms, we finally have found that an encoder-decoder based algorithm works best for the datasets that we have worked on. The results are considerably more accurate and faster than other well known approaches like word2vec and node2vec.

References

1. D. Liben-Nowell and J. Kleinberg, “The link-prediction problem for social networks,” journal of the Association for Information Science and Technology, vol. 58, no. 7, pp. 1019–1031, 2007.
2. X. Li, N. Du, H. Li, K. Li, J. Gao, and A. Zhang, “A deep learning approach to link prediction in dynamic networks,” in Proceedings of the 2014 SIAM International Conference on Data Mining. SIAM, 2014, pp. 289–297.
3. M. Fire, L. Tenenboim, O. Lesser, R. Puzis, L. Rokach, and Y. Elovici, “Link prediction in social networks using computationally efficient topological features,” in Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational

Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on. IEEE, 2011, pp. 73–80.

4. Kalchbrenner, N. and Blunsom, P. Recurrent continuous translation models. In EMNLP, 2013.
5. Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.
6. Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In NIPS, 2014.
7. Luong, T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. Addressing the rare word problem in neural machine translation. arXiv preprint arXiv:1410.8206, 2014.
8. Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. Grammar as a foreign language. arXiv preprint arXiv:1412.7449, 2014a.
9. Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. Show and tell: A neural image caption generator. arXiv preprint arXiv:1411.4555, 2014b.
10. Hochreiter, S. and Schmidhuber, J. Long short-term memory. Neural Computation, 1997.
11. K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Arxiv preprint arXiv:1406.1078, 2014
12. J. Pouget-Abadie, D. Bahdanau, B. van Merrienboer, K. Cho, and Y. Bengio. Overcoming the curse of sentence length for neural machine translation using automatic segmentation. arXiv preprint arXiv:1409.1257, 2014.
13. K. M. Hermann and P. Blunsom. Multilingual distributed representations without word alignment. In ICLR, 2014.