ECE 5268: Theory of Neural Network

Individual Course Project

By

Joseph Marie Nke

jnke2015@my.fit.edu

https://github.com/Joseph19961997

/ICP_project

Spring 2020

Introduction:

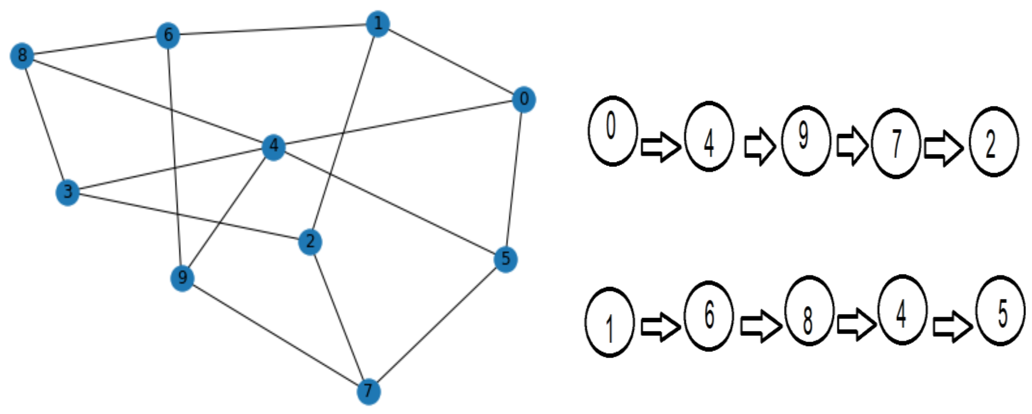
With the increasing number of data transfer, it is important if not detrimental to improve our current computer networks by minimizing the end to end latency which is the time taken for a packet to be transmitted across a network from a source to destination and maximizing the throughput which is the rate of successful delivery over a communication channel. Several machine learning models have been implemented on graph for studies and some of them include links prediction and node classification. In this project, I developed a machine learning technique which is being used in Natural Language Processing that learn the social representation of a graph's vertices by applying short random walks of fixed length. Each node will be mapped to a D dimensional low-level vector space capturing its neighborhood. The main goal of my project is to capture the network's topology information.

Problem Definition

Given a Graph G (V, E), we aim to learn the social representation of a graph by mapping each node to a low dimensional vector space. In the case of a social network, the goal is to classify members of a social network into multiple categories after obtaining a partially labeled social network through random walks. This approach has proven its effectiveness in language modeling which purpose is to estimate the likelihood of a specific sequence of words appearing in a corpus. For instance if we have the following sentence "The quick brown fox jumps over the lazy dog", and given the fact that we observed the words "brown" and "fox", what is the likelihood of each word in the sentence to be in between. With a well-trained model, the word with the highest probability should be "for". This approach is being used in my project but inversely. Instead of predicting the word based on the neighboring words, the neighboring words were predicted based on the given word. For social network, instead of having words, nodes from a graph are being predicted.

Random walks

In order to extract the information from a network (community of a given node), short random walks of length 5 were performed for each node. There are two benefits of applying random walks: The first reason is that they can be parallelized where several random walks can be run on different thread on a GPU reducing the time for computation. Another reason is the facility to accommodate small changes in a graph without global recomputation. In the case new nodes need to be incorporated, random walks can be performed on these specific nodes only. This second feature from random walks is effective for social network knowing the increasing and ongoing number of new connections. The need to incorporate these changing with minimal computations is necessary. Figure 1b is a representation of 2 nodes in the graph G_L after performing random walks of length 5



(a) Graph used during this project (b)

Figure 1: Random walks is being applied on the graph (1a) and some of the results appears on (1b)

Language Modeling

As I previously recalled, language modeling has the goal of estimating the likelihood of a specific sequence of words appearing in a corpus

$W_1^n = (w_0, w_1, \ldots, w_n)$

Where $w_i \in \mathcal{V}$ ($\mathcal{V}$ is the vocabulary) and the goal is to maximize the $\Pr(w_n \mid w_0, w_1, \ldots, w_{n-1})$ over all the training corpus. Analogically, in the case of a computer network graph, we would like to maximize the $\Pr(v_n \mid v_0, v_1, \ldots, v_{n-1})$ where $v_n \in V$

By combining short random walk and the Skip-gram model (neural language model), I was able to generate a representation of social networks that are low dimensional and exist in a continuous vector space.

```
**Algorithm 1**: RandomWalk (G, L)
**Input**:  graph G (V, E)
            length of the walk L
**Output**: Matrix of vertex representation
  1. for each vertex V in the graph G
  2.     perform a random walk of length L
  3. end for
```

The first task of my project involved the representation of the graph as lists of connected nodes (1b) similar to words in a sentence. To achieve that, N random walks of length L were performed in order to have a chain representation of the graph where N is the number of node and L the length of the walk. Only one random walk was performed for each node in order to prove the effectiveness of my implementation with just a subset of the graph**.** Above is the algorithm used to perform the random walks.

Random Walks

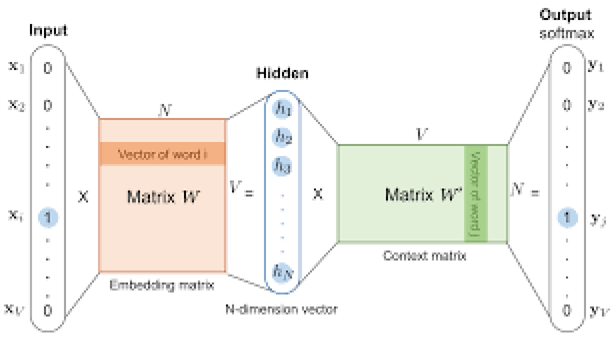
Figure (1a) graphic representation of the graph that was part of my experiments. It is a strongly connected graph with 10 vertices. The logic applied to this graph is alike Word-to-vector used in Natural Language Processing where nodes from a graph are being passed to a machine learning algorithm instead of words. In fact, nodes in a Graph somewhat have similar representation to words in a corpus in the sense that a word has a meaning in a sentence depending on its context. What I mean is that, a word in a sentence can be characterize by its neighbors which means that word that share the same neighbors tends to have similar meaning. Likewise, nodes that share the same neighbors share some similarity, but it is important to define what is meant by similarity. In the context of this study, I define two nodes to be similar if they share the same neighbors which means if they are connected. Below is a one representation of the graph in figure 1a after applying a random walk of length 5 on each node and let's denote it G_L

G_L =[['0', '4', '9', '7', '2'], ['1', '6', '9', '7', '2'], ['2', '1', '6', '8', '3'], ['3', '4', '5', '0', '1'], ['4', '8', '3', '2', '7'], ['5', '7', '9', '6', '1'], ['6', '9', '4', '3', '2'], ['7', '2', '3', '8', '6'], ['8', '3', '4', '5', '7'], ['9', '6', '8', '4', '5']].

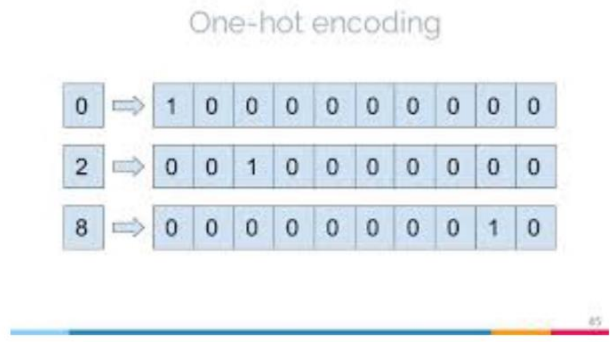
Performing random walks of length equal or bigger than two have some great advantages in the sense that each node does not just get a representation of its directs neighbors, but also have a picture of its surrounding (neighbors of its neighbors) .

<u>Node to Vector</u>

After obtaining G_L which is just a subset of graph G from figure (1a), we need to represent nodes as input to the machine learning model and the approach used in my project is a vector representation. Node2Vec is a model used to capture the relation between nodes in a graph or a social network and it is derived from Word2Vec in Natural Language Processing. Two main model are used when applying Word2Vec which are the Skip-gram and the Continuous Bag of Words (CBOW) models. In the skip-gram model, the goal is to predict contexts words based on the center word. The model will output a probability for each word in the corpus to be the context word. Similarly, given a node u_i€ V, the machine learning model will output the probability of each node in the graph on figure 1a to be a connection. The second approach (CBOW) used in words turn the problem to the head where the center node is predicted by summing vectors of the surrounding nodes. The Skip-gram model was used in this project and the figure below is its representation.



The model takes as input a training data and propagate it to the hidden layers in order to get the output. In order to assign unique label to each node, a one hot encoding was used which is just a vector representation of each node. One hot encoding is a process is a process by which categorical variables are converted into a form that could be provided to machine learning algorithms for prediction. The picture below is an example of a one hot encoding



One-hot encoding

0 ⇨	1	0	0	0	0	0	0	0	0	0
2 ⇨	0	0	1	0	0	0	0	0	0	0
8 ⇨	0	0	0	0	0	0	0	0	1	0

These One-Hot Embeddings have '1' only at the index mentioned on the left side

The training data is obtained by concatenating the target node (center node) and the context nodes (neighbors). Let’s denote C as a list from G_L.

C= ['0', '4', '9', '7', '2']

Each node from C will be at some point a target node or context node that will then be passed to the machine learning accordingly. If node ‘9’ is the target node and the window is of size to in both directions, therefore the context nodes (neighbors) are node ‘0,4,7 and 2’. They will then be encoded as one hot vector and will be passed to a machine learning model for training purposes. For instance, let’s denote T_c to be the training data set for C.

T_c=[[[0, 1, 0, 0, 0, 0, 0, 0, 0, 0], [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]]]].

Training the Model

The training function takes one input which is just the training data constitute of the target nodes and the context nodes. The training data constitute the input for the Skip-model which is then propagated to the hidden layer in order to get the output. This propagation was performed by the **forward_prop** function which takes as input the training node and output the predicted context node (neighbor)as well the hidden layer vectors and the output vector(before being passed to softmax function). Both the hidden layer and the output are vectors of size 10. Since there are more than two labels, ten in my case, the output was passed to a Softmax function. The error was then computed by performing the difference between the one hot representation of the target node and the one hot representation of the context node. That error was then propagated backward in order to obtain the gradient. In fact, the **back_prop** function takes as input the error, the hidden layer matrix and the outputs in order to update the weights. The algorithm used to perform both forward and back propagation is described below.

```
1:  procedure TRAIN
2:      $X \leftarrow$ Training Data Set of size mxn
3:      $y \leftarrow$ Labels for records in X
4:      $w \leftarrow$ The weights for respective layers
5:      $l \leftarrow$ The number of layers in the neural network, 1...L
6:      $D_{ij}^{(l)} \leftarrow$ The error for all l,i,j
7:      $t_{ij}^{(l)} \leftarrow 0$. For all l,i,j
8:      $For\ \ i = 1$ to $m$
9:        $a^l \leftarrow feedforward(x^{(i)}, w)$
10:       $d^l \leftarrow a(L) - y(i)$
11:       $t_{ij}^{(l)} \leftarrow t_{ij}^{(l)} + a_j^{(l)} \cdot t_i^{l+1}$
12:     if $j \neq 0$ then
13:       $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)} + \lambda w_{ij}^{(l)}$
14:     else
15:       $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)}$
16:       where $\frac{\partial}{\partial w_{ij}^{(l)}} J(w) = D_{ij}^{(l)}$
```

The loss function was then computed (according to the function below) in order to check the accuracy of the gradient descent.

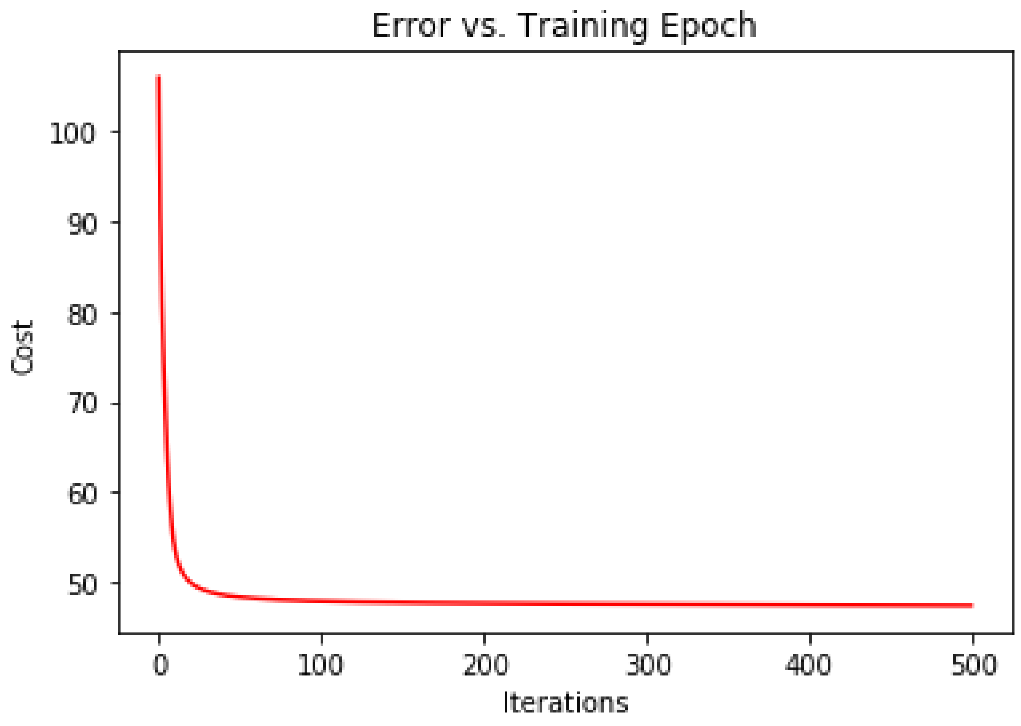
$$\mathcal{L} = \underbrace{\boxed{\sum_{u \in V}}}_{\text{sum over all nodes } u} \underbrace{\boxed{\sum_{v \in N_R(u)}}}_{\text{sum over nodes } v \text{ seen on random walks starting from } u} -\log \left(\underbrace{\boxed{\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}}}_{\text{predicted probability of } u \text{ and } v \text{ co-occuring on random walk}} \right)$$

Results

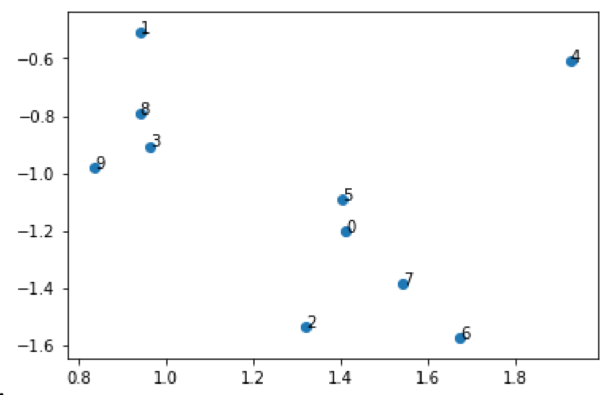
After the data was train, the model was tested by predicting which nodes have similar embeddings. This was achieved by applying the cosine similarity between two nodes. In fact, I identify two nodes to be identical if they have a similarity of 1 and different if they have a similarity=0.

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

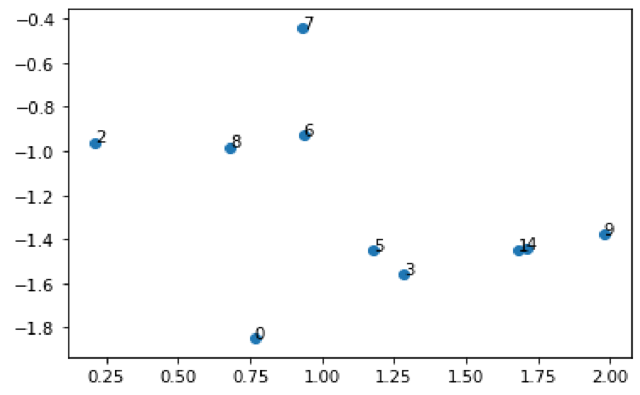
My model was able to predict at least one of the directs neighbors of a node in most of the case as it can be seen in example below. My gradient descent was able to minimize my loss however, I was not able to make it fall below 40. The figure below shows the lost function as gradient descent was applied to minimize the cost.



I was later on trying to see if my model was representing nodes that are similar close to each other but it was quiet hard to do so because all my vectors are of dimensions 10. To have an idea of what the look like in space, I took the maximum and the minimum of each vectors to be the X and Y coordinate. This approach doesn't picture the real representation of the nodes in space and the best way to visualize them is on a 10-dimensional space. The result when trying to map each vector to a 2D space can be pictured below.



.



Depending on the random walk, each node has a different representation in space. In both figures, node 2, 7 and 6 appear to be closer to each other. But based on this graph, my results are not really accurate because both graphs should share a lot of similarities which is not the case. This can be because I was not able to correctly map each vectors of dimensionality 10 to a 2D space. In fact, I only used the maximum value of a vector and the minimum to be the X and Y coordinates which is not very representative.

References:

D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. Journal of the American society for information science and technology, 58(7):1019–1031, 2007.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. CoRR, abs/1301.3781, 2013.

G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. Audio, Speech, and Language Processing, IEEE Transactions on, 20(1):30–42, 2012.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research, 9:1871–1874, 2008.

Chen, Ivan. “Word2vec From Scratch with NumPy.” *Medium*, Towards Data Science, 18 Feb. 2019, towardsdatascience.com/word2vec-from-scratch-with-numpy-8786ddd49e72.