# Temporal Graph Embedding

Justin Mücke
justin.muecke@uni-ulm.com
University Ulm
Ulm, Germany

## ABSTRACT

## KEYWORDS

Temporal Graph, Embedding

**ACM Reference Format:**
Justin Mücke. 2021. Temporal Graph Embedding. In *University Ulm: Seminar Data Science.* ACM, New York, NY, USA, 2 pages.

## 1 INTRODUCTION

To start this paper out, let's first create a level playing field for all readers by laying out what exactly a Graph is, how it can change over time and what is meant when we talk about embedding.

### 1.1 Graphs

A graph is a mathematical construct which is used in a variety of tasks. It is often used to model relationships between entities, thus making it possible to operate on such structures to, for example, analyze them.

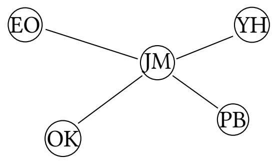

**Figure 1: Part of the follower-relationship-graph of personal Instagram account**

Mathematically it is consisting of two sets $G = (V, E)$, where V consists of all the vertices of the Graph, and E of all the edges, represented through a tuple of two vertices. Our graph above would thus look like following:
$G_i = (\{E, O, Y, P, J\}, \{(J, E), (J, O), (J, P), (J, Y)\})$.

### 1.2 Temporal Graphs

As we want to embed not just graphs, but temporal graph, lets bring in the temporal aspect. Let's imagine a camera that makes a picture of the graph everytime a certain amount of time passed, so we get our Graphs $G_1$ after the first picture and so on. We then take all of our pictures and put them together into one construct, thus creating our temporal graph $G = \{G_1, G_2, \ldots, G_t\}$ consisting of many other graphs.[1]

### 1.3 Embedding

Our goal is, as previously mentioned, to analyze those temporal graphs, but graphs as mathematical structure are rather hard to compute with. Thus, we want to transform it into a more accessible structure. For that we now try to transfer the information, that is modeled by the graph, into a vector space. Those Vectors have to capture the graph topology, the vertex-to-vertex relationship as well as other relevant information. Overall we can divide graph embedding techniques into two categories; vertex embedding and graph embedding. When using vertex embedding, we create for each vertex its own vector representation. This is used when we want to perform predicition on a node level.

When using graph embedding, on the other hand, we create one vector representation of the whole graph. Most useful is this method when we want to analyze our data on a graph level e.g. comparing two graph structures with each other.
To better understand how these methods are working we are now going to look at a few examples

*1.3.1 Word2Vec.* First lets take a look at the word2vec algorithm. This is used as the basis of many other embedding algorithms. The goal hereby is, that words that are similar to each other get a similar vector representation, and that the text can be represented solely through our vectors. To achieve this, we start by training a neural network over a sample text. This neural network consists of one input-, one hidden- and one output layer. We then train this network to detect how likely it is to have two words in one neighborhood. Here, a neighborhood contains the word in question as well as all words that are either in a fixed distance before or after it. A classical distance for this window is 5. As input, we use a one-hot code of all the words in our test file, e.g. the first word has a vector of $(1, 0, \ldots, 0)$, and as output we use a softmax vector to describe the probability of the words being next to the starting word. With this achievement we can now satisfy our problem conditions, because we now have all the information from the text assembled in multiple vectors.

*DeepWalk.* This method is a continuation of the word2vec approach. Overall, the method consists of three steps. First the sampling, which is done using random walks on our graph starting on a selected node. Hereby, a random walk describes a random succession of nodes of defined length. It is sufficient to perform 32 to 64 random Walks per node. Those random walks are now of the same form as sentences used in the word2vec method, meaning we can

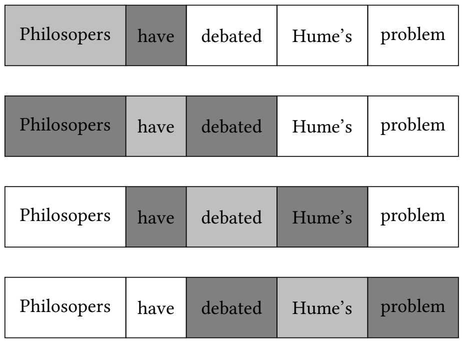

**Figure 2: The light gray words are our Input and the dark-gray words our neighbors for a window size of 1**

train a similar neural network for predicting the neighborhood of our nodes.

*Node2Vec.* Here we modify our DeepWalk method. For that purpose we introduce 2 variables $P$ and $Q$, where $Q$ defines how likely it is to go to an unvisited node, while $P$ describes how probable it is to revisit a node.

*Structural Deep Network Embedding.* In contrast to the methods used before, SDNE does not use random walks. It aims to preserve local pairwise similarity which characterizes the local structure, and as well as the global network structure. To achieve this, we use two autoencoder neural network. These get an adjacency vector as input and want to construct node adjacency as output. We then compute the distance between the two outputs and add it to the loss function of the network. The total loss function is then computed through summation of the distance loss plus the losses of the two encoders. At the end we remain with a collection of adjacency vectors which describe the graph structure.

*Graph2Vec.* Now we don't want to represent the nodes as vectors, but the whole graph. For that we have once again three steps. In the first step we create sub-graphs for each node and encode then once again in a one-hot code. We then use these sub-graphs to train the network used in word2vec to maximize the probability that a predicted sup-graph exists in the input graph. The embedding is then the result of the network. [2]

## 2 METHODS

### 2.1 tbGraphEmbed

Starting paper

### 2.2 sub2vec

Is used as comparison in starting paper -> Look into

### 2.3 Comparison

How do Methods differ -> nodelevel / Graphlevel?

### 2.4 Application

Why do we use Embedding

*2.4.1 Similarity.* Differences Between graphs (exp - googletrends)

*2.4.2 Anomaly.* Where does it differ

## 3 CONCLUSION

## REFERENCES

[1] George B. Mertzios and Hendrik Molter and Rolf Niedermeier and Viktor Zamaraev and Philipp Zschoche, *Computing Maximum Matchings in Temporal Graphs*, 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020), 2020

[2] Primož Godec *Graph Embeddings — The Summary* https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007 Dec 31, 2018

[3] Moran Beladev, Lior Rokach, Gilad Katz, Ido Guy, Kira Radinsky, *tdGraphEmbed: Temporal Dynamic Graph-Level Embedding*, CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020