Member-only story

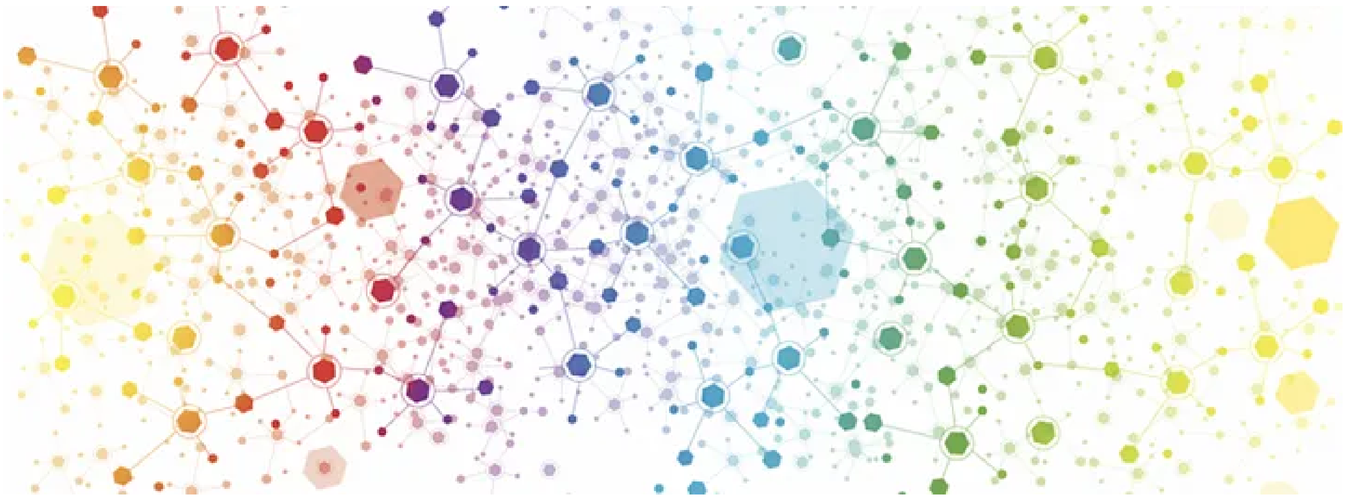
GRAPHEDM-SERIES

Graph Representation Learning — Objective Functions and Encoders (Part 3)

Let’s understand how to optimize GRL models and their ability to incorporate different network features

Giuseppe Futia · Follow
Published in Towards Data Science
8 min read · Nov 3, 2021

Listen Share More


Graph Vis — Original image from NYU Shangai website

This series summarizes a comprehensive taxonomy for machine learning on graphs and reports details on GraphEDM (Chami et al.), a new framework for unifying different learning approaches.

In recent years there has been a massive proliferation of Graph Representation Learning (GRL) techniques. The primary goal of these techniques is to learn functions that map the discrete structure of graph data to a continuous representation in the vector space. The GraphEDM framework introduced in the previous articles is able to unify more than 30 approaches in the GRL context to learn these continuous representations, also known as *embeddings*.

In the unsupervised setting, the learning goal of GRL techniques is to preserve the graph structure. In the supervised setting, the goal is to address a specific downstream task. Different learning approaches imply different optimization methods and various types of encoders to generate node embeddings. Therefore, the first part of this article is dedicated to objective functions implemented in GraphEDM. The second part is instead focused on different types of graph-based encoders. For a brief recap on the GraphEDM framework, I suggest reading the previous articles of this series.

Graph Representation Learning — Network Embeddings (Part 1)

towardsdatascience.com

Graph Representation Learning — The Encoder-Decoder Model (Part 2)

towardsdatascience.com

All the framework details are reported in the article by Chami et al.:

> Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., & Murphy, K. (2020). Machine learning on graphs: A model and comprehensive taxonomy. arXiv preprint arXiv:2005.03675.

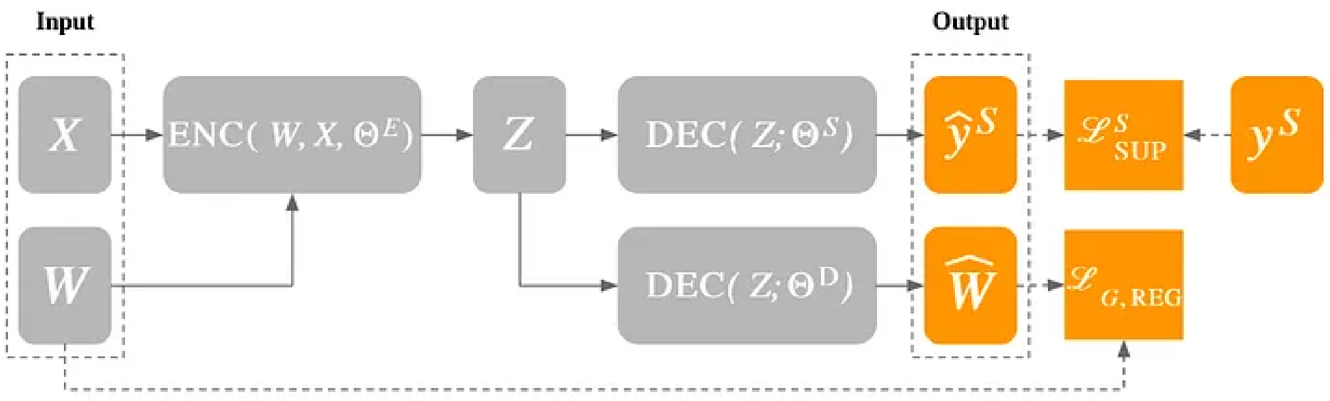
Objective Functions

In machine learning, the *objective function*, or *criterion*, is the general function that we want to minimize (or maximize) during the optimization of the model's parameters. If we're going to minimize the output, the objective function is denoted as the *loss* function for a single training example, while it is known as the *cost* function when it is considered over the entire training set.

Hence, the loss function is defined on a data point: it compares the predicted label from the model to the target label and computes the penalty. The cost function might be the sum or the average of the single penalties plus a complexity penalty, defined as *regularization*. The regularization allows the model to stabilize the learning process and enforce its generalization capacity. In the regularization, the model parameters are constrained (or regularized), discouraging the learning of noise signals and avoiding overfitting.

As we will see shortly, in the specific case of GRL, the graph structure can also be employed during the learning process as a regularization term.

GraphEDM Loss Functions



Loss functions in the GraphEDM Framework (posted with permission)

In the case of GRL techniques included in the GraphEDM framework, we want to optimize three different parameter classes (see the previous article of the series for further details):

- Θ^E represents the parameters of the encoder network;
- Θ^D represents the parameters of the decoder network;
- Θ^S represents the parameters of the classification network. In this specific case, *S* is a collection of supervised signals including the features and the labels of nodes, edges, and graphs: *S* ∈ *{N, E, G}*.

All these parameters are optimized by combining a supervised loss term, a graph-regularization loss term, and a weight regularization loss term.

Supervised Loss

The *supervised loss* term computes the penalties between the predicted labels and the target labels.

$$L^{G}_{SUP}(y^{S}, \hat{y}^{S}; \Theta) = l(y^{S}, \hat{y}^{S}; \Theta)$$

Graph Representation Learning — Supervised loss term (posted from the Chami et al. manuscript)

The graph signals *S* involved in this loss term are strictly related to the downstream task that involves the model training phase. If the task to perform is a node classification, then *S = (N)*. For the cases in which the task is focused on the link prediction, *S = (E)*. Otherwise, in the case of task performed at graph level, *S = (G)*. In this mathematical formalization, *ŷ^S* represents the prediction space, while *y^S* represents the label space.

Graph Regularization Loss

The *graph regularization loss* term has the goal regularize the parameters of the GRL model by exploiting the network structure.

$$L_{G,REG}(W, \hat{W}; \Theta) = d(s(W), \hat{W}; \Theta)$$

Graph Representation Learning — Graph regularization loss term (posted from the Chami et al. manuscript)

The penalty computation is based on the distance between the reconstructed similarity matrix $\hat{W}$ and the target similarity matrix *s(W)*. The dimension of both the matrices $\hat{W}$ and *W* is equal to|V|×|V|, where |V|is the number of nodes. *d* executes the distance computation between $\hat{W}$ and *s(W)*.

To better understand the general concept of similarity matrices in GRL, consider the following clarification. The most simple usage of graph regularization involves the reconstruction of the adjacency matrix, whose dimension is |V|×|V|. As described with an example in my introductory article on GNNs, the adjacency matrix describes the (weighted) edges between nodes.

We can state that the adjacency matrix shapes the first-order proximity between two nodes. To clarify this aspect, consider two distinct nodes v_i and v_j: the local similarity value of these nodes, according to the first-order proximity, is defined by the edge weight w_{ij}. Hence, the first-order proximity is able to capture the existence and the effectiveness of an edge between two distinct nodes.

However, we can expand this first-order proximity concept, trying to encapsulate further information from the graph structure. For instance, the second-order proximity between two nodes does not consider the congruence between two nodes based on their weighted edge, but it is based on the homogeneity of their neighborhood nodes. In other words, two nodes achieve a closer representation according to the second-order proximity if they lean to have many neighbors in common.

The notion of node proximity is crucial for learning graph embeddings because the training goal of many GRL models is to maintain one or multiple orders of proximity.

Weight Regularization Loss

The *weight regularization loss* term is a traditional approach applied to generally regularize model parameters during the training step.

$$L_{REG}(\Theta) = \sum_{\theta \epsilon \Theta} ||\theta||_2^2$$

Graph Representation Learning — Weight regularization loss term (posted from the Chami et al. manuscript)

The main idea behind the concept of weight regularization is to inject a small amount of bias in the model, in order to obtain a significant drop in variance. As a consequence, with a slightly worse fit, we can provide better predictions with real data. The most common regularization is *L2*, where the squared value of θ parameters makes the predictions less sensitive to the changes in the training data.

Loss term combination

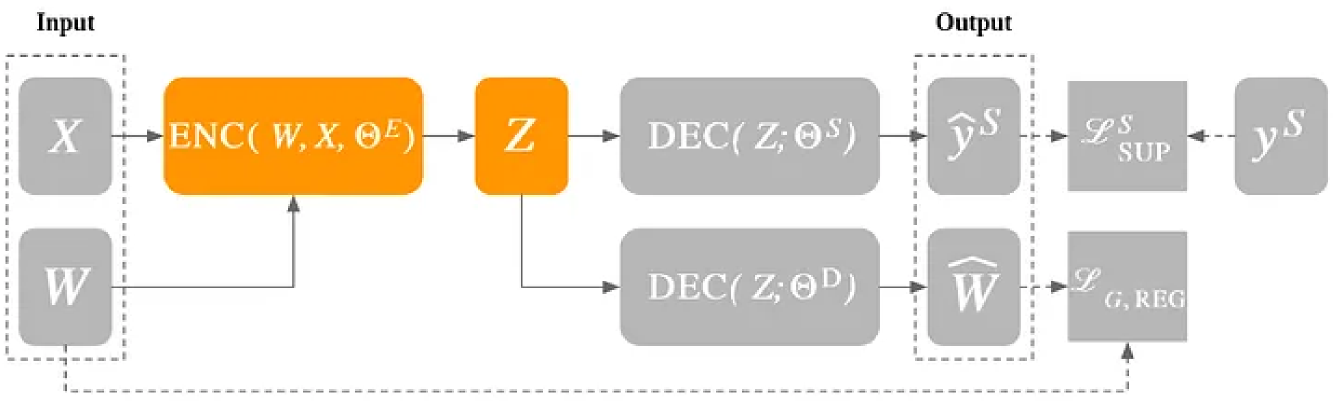
As explained at the beginning of the section, the loss defined for the models encapsulated in the GraphEDM framework includes all the previous components. For this reason, the loss is formulated as follows.

$$L = \alpha L_{SUP}^{G} + \beta L_{G,REG} + \gamma L_{REG}$$

Graph Representation Learning — Loss term combination (posted from the Chami et al. manuscript)

α, β, and γ are hyper-parameters that can be learned during the training process. As you can guess, in the supervised setting for graph embedding learning $\alpha \neq 0$, while in the unsupervised setting $\alpha = 0$ to exclude the contribution of the supervised loss term.

GraphEDM Encoders



The GraphEDM Encoder (posted with permission)

The main goal of the encoder network is to map each node in the graph into a low-dimensional vector. Diverse GRL methods significantly differ based on the encoder to generate the node embeddings. The GraphEDM framework summarizes four main classes: shallow embedding methods, graph-regularization methods, graph auto-encoding methods, and neighborhood aggregation methods.

Shallow embedding methods

In the case of shallow embedding methods, the encoder function is a simple projection operation.

$$Z = \mathrm{ENC}(\Theta^{E}) = \Theta^{E} \epsilon \mathbb{R}^{|V| \mathrm{x} d}$$

Shallow Encoder Network (posted from the Chami et al. manuscript)

Therefore, the parameters of the models are directly used as node embeddings (in the math formulation, $|V|\ x\ d$ corresponds to the number of the model parameters).

Considering their nature, these methods are particularly useful in a transductive setting, which assumes that all graph nodes are observed during the training process. Unlike the inductive setting, the requirement of the transductive setting is a settled graph, which does not evolve over time. To concretely understand the projection step, you can see my article related to the introduction on GNNs (section "The Input Layer").

Graph regularization methods

In the case of graph-regularization methods, the encoder exploits only the node features and labels.

$$Z = \mathrm{ENC}(X; \Theta^E)$$

Graph Regularization Encoder Network (posted from the Chami et al. manuscript)

Even if it is not directly used in the propagation into the encoded network, the graph structure is used as a regularization constrain ($\beta \neq 0$ to preserve the graph regularization loss term). As a main consequence, the node features are learned using the training step, but at the same time, they are regularized using the graph structure.

Graph auto-encoding methods

In the case of graph auto-encoding methods, the encoder component leverages only the graph structure.

$$Z = \mathrm{ENC}(W; \Theta^E)$$

Graph Auto-encoding Network (posted from the Chami et al. manuscript)

Compare to shallow embedding methods, graph autoencoders are able to learn non-linear features, employing deep learning architectures. In this specific case, the graph structure is not exploited as a graph regularization term, but it is directly injected (using an auto-encoder) in the encoder component as an adjacency matrix W.

Neighborhood aggregation methods

In the case of neighborhood aggregation methods, both the node features (X) and the graph structure (W) are exploited in the propagation step.

$$Z = \mathrm{ENC}(X, W; \Theta^E)$$

Neighborhood Aggregation Encoder Network (posted from the Chami et al. manuscript)

These GRL approaches exploit the message passing framework (see one of my previous articles for a clear intuition) to update the node features with the features of its neighbors. In this case, the neighbor features are *passed* to the target node as *messages* through the edges. As a main consequence, the embeddings are able to encode the local structures of the graph. It is worth mentioning that Graph Convolutional Networks (GCNs) is an effective example of a neighborhood aggregation method.

For all the articles on the GraphEDM framework, you can use the following link: https://towardsdatascience.com/tagged/graphedm-series.

For further readings on Graph Representation Learning, you can follow my series at the following link: https://towardsdatascience.com/tagged/grl-series.

If you like my articles, you can support me using this link https://medium.com/@giuseppefutia/membership and becoming a Medium member.

Graphedm Series Grl Series Graph Neural Networks Graph Learning Data Science

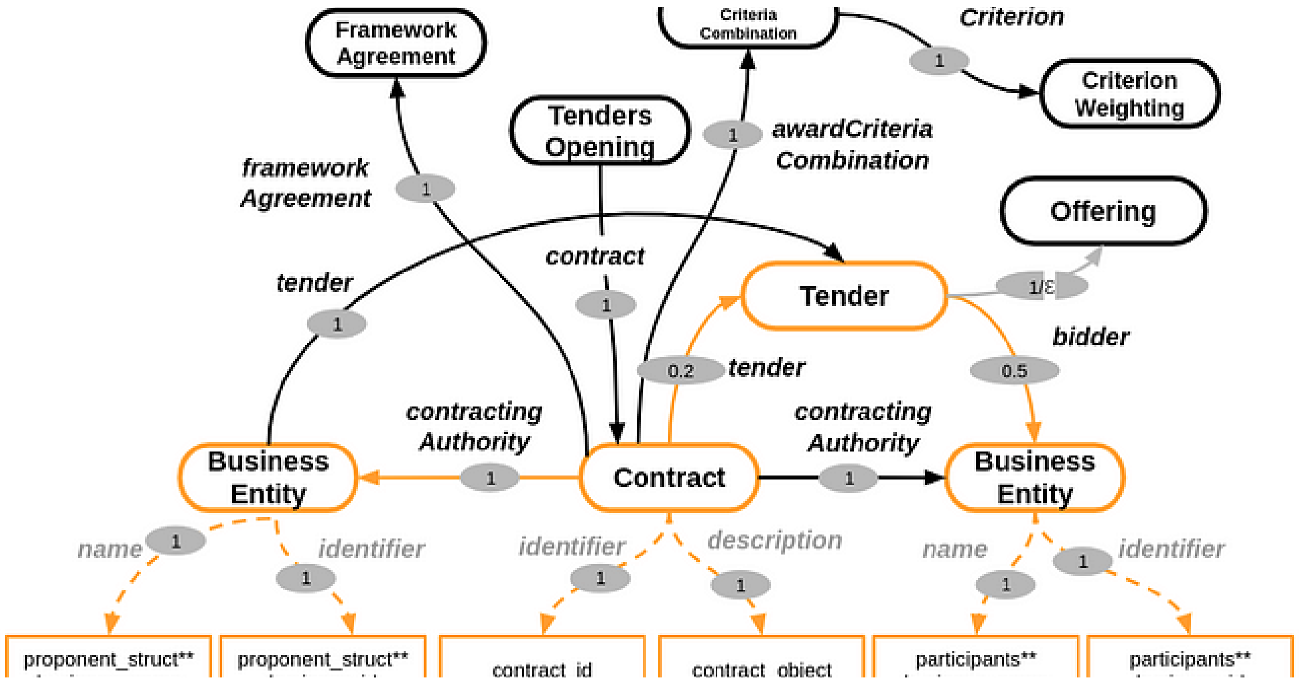
Follow

Written by Giuseppe Futia

768 Followers · Writer for Towards Data Science

Senior Data Scientist at GraphAware | Ph.D. at Politecnico di Torino. Passionate about Knowledge Graphs, Semantic Modeling, and Graph Neural Networks.

More from Giuseppe Futia and Towards Data Science



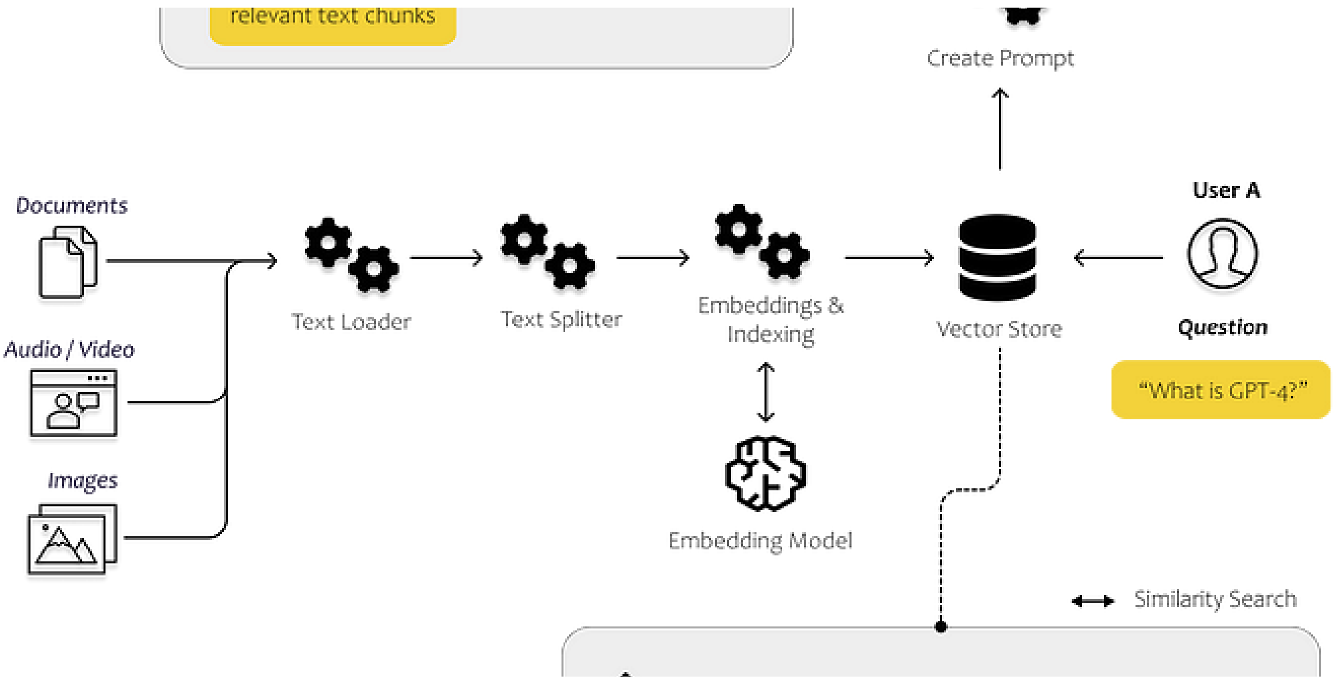
Giuseppe Futia in Towards Data Science

Semantic Models for Constructing Knowledge Graphs

Capturing the Semantics of Data Sources with Graph-Based Structures

· 6 min read · Dec 16, 2020

69



Dominik Polzer in Towards Data Science

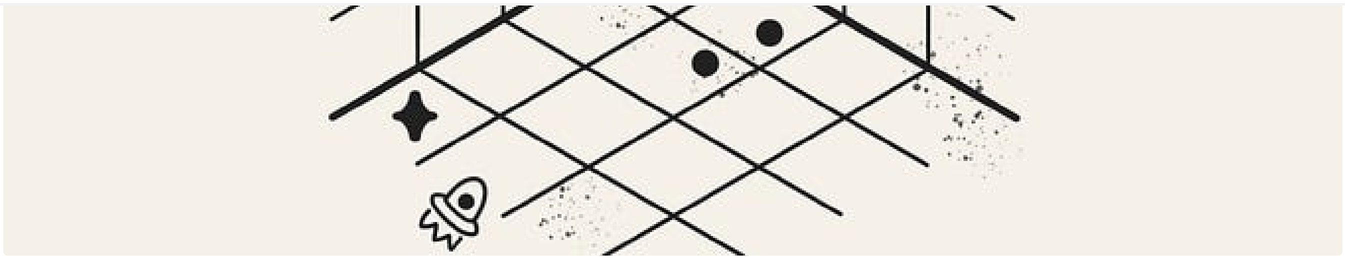
All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

✦ · 26 min read · Jun 22

3.4K 31

Open in app ↗



Leonie Monigatti in Towards Data Science

Explaining Vector Databases in 3 Levels of Difficulty

From noob to expert: Demystifying vector databases across different backgrounds

✦ · 8 min read · Jul 4

1.8K 18

Giuseppe Futia in Towards Data Science

Knowledge Graphs at a glance

Incorporate human knowledge into intelligent systems, exploiting a semantic graph perspective

· 7 min read · Sep 27, 2020

182 1

See all from Giuseppe Futia

See all from Towards Data Science