Machine Learning on Graph-Structured Data

Claudio D.T. Barros[1], **Daniel N.R. da Silva**[1], **Fabio A.M. Porto**[1]

[1]DEXL Lab – Laboratório Nacional de Computação Científica (LNCC)
Petrópolis, RJ – Brazil

{cdtb,dramos,fporto}@lncc.br

Abstract. *Several real-world complex systems have graph-structured data, including social networks, biological networks, and knowledge graphs. A continuous increase in the quantity and quality of these graphs demands learning models to unlock the potential of this data and execute tasks, including node classification, graph classification, and link prediction. This tutorial presents machine learning on graphs, focusing on how representation learning – from traditional approaches (e.g., matrix factorization and random walks) to deep neural architectures – fosters carrying out those tasks. We also introduce representation learning over dynamic and knowledge graphs. Lastly, we discuss open problems, such as scalability and distributed network embedding systems.*

1. Introduction

Graphs are a ubiquitous data structure and a universal language for describing complex systems, representing a collection of objects and a set of interactions between pairs of these objects. The graph formalism lies both in its generality and in its focus on the relationship between data instances, offering a mathematical foundation to analyze, understand, and learn from real-world networks. Given the ever-increasing scale and complexity of graph datasets, including social networking platforms, massive scientific initiatives to model biological networks, and billions of interconnected web-enabled devices, machine learning plays an important role in advancing the ability to model and predict from graph-structured data [Hamilton 2020].

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ between these nodes. An edge going from node $v_i \in \mathcal{V}$ to node $v_j \in \mathcal{V}$ is depicted as $(v_i, v_j) \in \mathcal{E}$. A graph is **undirected** if the existence of edge $(v_i, v_j) \in \mathcal{E}$ implies the existence of edge $(v_j, v_i) \in \mathcal{E}$, otherwise, the graph is **directed**. It is possible to represent graphs through an **adjacency matrix** $A \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, whose element A_{ij} denotes the existence of an edge from node v_i to node v_j, i.e., $A_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$, and $A_{ij} = 0$ otherwise. Another graph representation is given by an **adjacency list** – an array of lists in which each entry a_i represents the list of nodes connected to v_i through an edge. Moreover, each edge in a graph can be assigned to a unique **weight** whose meaning depends on the problem – from distances between nodes and costs to reach them, to bond strength and link probability.

In real-world systems, nodes and edges are usually **heterogeneous**, i.e., they are assigned to types: edges include a type τ, e.g., $(v_i, \tau, v_j) \in \mathcal{E}$, and nodes are partitioned into k disjoint sets $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup ... \cup \mathcal{V}_k$. Furthermore, nodes, edges and even whole graphs can be associated with **additional feature information**, i.e., numeric, categorical or even other data structures such as strings, images or time series.

2. Machine Learning Fundamentals

Machine learning is broadly defined as computational methods using experience – past information typically in the form of sample data collected and available for analysis – to improve performance and to make accurate predictions. Two major learning paradigms are (i) **supervised learning**, involving the observation of several examples of a random vector x and an associated value y, then learning to predict y from x – hence target y is understood as being provided by an instructor or teacher who shows the machine learning system what to do –, and (ii) **unsupervised learning**, where is attempted to implicitly or explicitly learn from data without target variable y [Goodfellow et al. 2016].

A typical supervised machine learning setting is composed by a **domain set** $\mathcal{X}$ of instances represented by a vector of features, and a **output set** $\mathcal{Y}$ containing possible outputs for each instance, which could be class labels, a regression score, a cluster or a latent vector. The main learning task is to learn a mapping from the vector of features to an specific output. The prediction model has access to **training data**, a finite sequence of labelled domain points in $\mathcal{X} \times \mathcal{Y}$, i.e., $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m))$. The learner is requested to output a **prediction rule** $h : \mathcal{X} \to \mathcal{Y}$, which can be used to predict the output of new instances [Mohri et al. 2018].

It is assumed that the instances are generated by some **probability distribution** $\mathcal{D}$ over $\mathcal{X}$, and that there is some correct **output function** $f : \mathcal{X} \to \mathcal{Y}$ and $y_i = f(\mathbf{x}_i)$ for all i. Each pair in the training data S is generated by first sampling a point $\mathbf{x}_i$ according to $\mathcal{D}$ and then the output being calculated by f. The prediction model is blind to the underlying distribution $\mathcal{D}$ over the world and to the labelling function f. In addition, the examples in the training set are assumed to be **independently and identically distributed (i.i.d.)** according to the distribution $\mathcal{D}$ [Mohri et al. 2018].

3. Machine Learning on Graphs

Traditional learning approaches over graphs follow the standard machine learning paradigm: extract statistics or features based on heuristic functions or domain knowledge, and then use these features as input to machine learning models. Classical **node-level features** include **node degree**, representing the number k_i of edges incident to a node v_i, **node centrality**, measuring the influence or the importance of each node v_i in a graph; and **clustering coefficient**, measuring how tightly clustered a node's neighborhood is. **Edge-level features** are usually expressed by **similarity matrix**, summarizing pairwise node statistics and measuring how two nodes are similar in the network. **Graph-level features** characterize an entire network, in general counting different smaller subgraph structures called **graphlets** or using functions which measure similarity between graphs, i.e., **graph kernels** [Hamilton 2020].

Machine learning tasks on graphs include: (i) **node classification**, whose goal is to predict the label y_i – which could be a type, category, or attribute – associated with each node $v_i \in \mathcal{V}$, given the true labels on a training set of nodes $\mathcal{V}_{\text{train}} \subset \mathcal{V}$; (ii) **relation prediction** (a.k.a. link prediction, graph completion and relational inference), which concerns to inferring the edges between nodes in a graph, and the standard setup is to use the information about a set of nodes $\mathcal{V}$ and an incomplete set of edges between these nodes $\mathcal{E}_{\text{train}} \subset \mathcal{E}$ to infer the missing edges $\mathcal{E} \backslash \mathcal{E}_{\text{train}}$; (iii) **community detection**, the graph analogue of unsupervised clustering, grouping nodes into a finite number of partitions.

Furthermore, it is possible to execute tasks regarding the whole graph (i.e. treating the entire graph as a single instance of a dataset containing several graphs), such as **graph classification**, **graph regression** and **graph clustering**.

4. Graph Representation Learning

Hand-engineered features cannot adapt through a learning process, and designing them can be a time-consuming and expensive process. Therefore, an alternative approach is to learn representations that encode structural and functional information about the network into latent vectors, i.e., embedding vectors. The **graph representation learning** problem is described by an **encoder model** mapping graphs into low-dimensional vectors (i.e., embeddings), and a **decoder model** taking these embeddings to reconstruct structural (e.g., node neighbourhood) or functional information (e.g., classifying nodes or graphs on a supervised task) [Hamilton 2020].

Matrix factorization decomposes some similarity matrix in a graph using linear algebra techniques including singular value decomposition, non-negative factorization and locally linear embedding to obtain node embeddings, assuming that the input data lie in a low-dimensional manifold. **Random walk approaches** generate node sequences from a graph to create contexts for each node, and apply techniques similar to or inspired by natural language processing to learn embeddings, preserving higher-order similarity between nodes by maximizing the probability of occurrence of subsequent nodes in fixed-length random walks. Approaches based on **neural networks** apply neural architectures on graphs, including autoencoders (AEs), convolutional neural networks (CNNs), and variational autoencoders (VAEs) [Cai et al. 2018].

Classical graph embedding methods assume that the encoder model is an embedding lookup, i.e., $\text{ENC}(v_i) = \mathbf{z}_i = \mathbf{Z}_i$, where $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}| \times d}$ (d is the embedding dimension) is a matrix containing embedding vectors for all nodes, and $\mathbf{Z}_i$ denotes the row of $\mathbf{Z}$ corresponding to node v_i. These approaches lack any parameter sharing between nodes in the encoder, meaning that the number of parameters necessarily grows as $O(|\mathcal{V}|)$, fail to leverage node features and are inherently transductive, only generating embeddings for nodes that were present during the training phase. Therefore, shallow encoders can be replaced with more sophisticated encoders that depend more generally on the structure and attributes of the graph [Hamilton 2020].

5. Graph Neural Networks

Deep learning over general graphs requires a new kind of neural architecture, whose operations should be invariant to permuting the order of graph nodes. An important class of neural networks that aims to obey these properties encompasses Graph Neural Networks (GNNs). In short, they employ a form of **differential message passing** along graph edges in which vector messages are exchanged between nodes and are updated using neural networks. During each message-passing iteration in a GNN, an **embedding** $\mathbf{h}_i^{(k)}$ corresponding to each node $v_i \in \mathcal{V}$ is updated according to information aggregated from v_i's graph neighborhood $\mathcal{N}(v_i)$. This message-passing update can be expressed by [Hamilton 2020]:

$$\begin{aligned} \mathbf{h}_i^{(k+1)} &= \text{UPDATE}^{(k)}\left(\mathbf{h}_i^{(k)}, \text{AGGREGATE}^{(k)}\left(\left\{\mathbf{h}_j^{(k)}, \forall v_j \in \mathcal{N}(v_i)\right\}\right)\right) \\ &= \text{UPDATE}^{(k)}\left(\mathbf{h}_i^{(k)}, \mathbf{m}_{\mathcal{N}(v_i)}^{(k)}\right) \end{aligned} \tag{1}$$

where UPDATE and AGGREGATE are arbitrary differentiable functions, i.e., neural networks, and $\mathbf{m}_{\mathcal{N}(v_i)}^{(k)}$ is the message that is aggregated from v_i's graph neighborhood $\mathcal{N}(v_i)$. After running K iterations of the GNN message passing, the output of the final layer is used to define the embeddings for each node, i.e., $\mathbf{z}_i = \mathbf{h}_i^{(K)}$ for all $v_i \in \mathcal{V}$. Note that K may be regarded as the number of GNN layers.

The basic intuition behind the GNN message-passing framework is that, at each iteration, every node aggregates information from its local neighborhood, and as these iterations progress each node embedding contains more and more information from further reaches of the graph. This approach captures both **structural information** and **features of each node**. After k iterations of GNN message passing, the embeddings for each node also encode information about features in their k-hop neighborhood. Many GNN variants with different neighborhood aggregation and graph-level pooling schemes have been proposed, these GNNs have empirically achieved state-of-the-art performance in many tasks such as node classification, link prediction, and graph classification [Hamilton 2020].

6. Machine Learning on Dynamic Networks and Knowledge Graphs

It is possible to extend machine learning to handle dynamic networks, where nodes and edges are being added or removed from the system, features are changing over time, and diffusion processes are taking place. Embedding methods for time-varying graphs improve tasks such as link prediction and node classification, while enabling novel applications, including event prediction, anomaly detection and diffusion prediction. Embedding algorithms range from **matrix factorization** over graph snapshots, **node sequence sampling** leveraging temporal order, **neural networks** and **deep learning approaches**, **tensor factorization approaches**, and **temporal point process based** methods, which handles similarity matrix changes as stochastic processes [Barros et al. 2021].

Moreover, machine learning models can be applied on **knowledge graphs** – graph structured knowledge bases that store factual information in the form of relationships between entities – to infer new facts about the world and to predict the existence or the probability of correctness of facts (i.e., **knowledge graph completion**), to identify which objects in relational data refer to the same underlying entities (i.e., **entity resolution**) and to cluster entities according to their features and links (i.e., **link-based clustering**) [Nickel et al. 2015].

7. Open Problems and Future Perspectives

This tutorial seeks to bring a comprehensive view of machine learning on graphs, discussing shallow and deep approaches, along with applications in static, dynamic and knowledge graphs. Some important challenges in the field include the limitations and scalability of graph neural networks. In practice, GNNs suffer from so called over-smoothing, since node representations become indistinguishable when the number of GNN layers increases. Moreover, the number of nodes in each node's receptive field grows exponentially, causing an over-squashing: information from the exponentially-growing receptive field is compressed into fixed-length node vectors, and the graph fails to propagate messages flowing from distant nodes, thus learning only short-range signals from the training data [Alon and Yahav 2020]. In addition, the exponential growth of neighborhood size corresponds to an exponential input-output overhead, and a common

strategy for scaling GNNs is to sample the graph structure during training, e.g. sample a fixed number of nodes from the k-hop neighborhood of a given node to generate its prediction. Since these approaches still rely on an expensive multi-hop message passing procedure, there is a trade-off between runtime and accuracy. Therefore, an efficient approximation of information diffusion in GNNs resulting in significant speed gains while maintaining state-of-the-art prediction performance is demanding [Bojchevski et al. 2020].

Moreover, a scalable and integrated database system with fully-functional training and inference for machine learning on graphs presenting distributed network embedding systems is also demanding. Existing systems require the in-memory storage of graph data either in a single machine that could not handle real industrial-scale data, or in a customized graph store that could lead to huge amount of communications between graph stores and workers. Moreover, these systems do not exploit classical infrastructures, such as MapReduce or parameter server, for fault tolerant purpose. As a consequence, they focus on training rather than the optimization of inference over graphs, which makes them an unintegrated system. Several efforts to design ingenious system architectures for various graph machine learning techniques include PyTorch Geometric, Deep Graph Library, AliGraph, and Ant Graph Machine Learning System (AGL) [Zhang et al. 2020].

References

Alon, U. and Yahav, E. (2020). On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*.

Barros, C. D., Mendonça, M. R., Vieira, A. B., and Ziviani, A. (2021). A survey on embedding dynamic graphs. *arXiv preprint arXiv:2101.01229*.

Bojchevski, A., Klicpera, J., Perozzi, B., Kapoor, A., Blais, M., Rózemberczki, B., Lukasik, M., and Günnemann, S. (2020). Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2464–2473.

Cai, H., Zheng, V. W., and Chang, K. C.-C. (2018). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Hamilton, W. L. (2020). Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of Machine Learning*. MIT Press.

Nickel, M., Murphy, K., Tresp, V., and Gabrilovich, E. (2015). A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33.

Zhang, D., Huang, X., Liu, Z., Zhou, J., Hu, Z., Song, X., Ge, Z., Wang, L., Zhang, Z., and Qi, Y. (2020). Agl: A scalable system for industrial-purpose graph machine learning. *Proc. VLDB Endow.*, 13(12):3125–3137.