Supplementary Information for

Graph Representation Learning in Biomedicine and Healthcare

Michelle M. Li[1,3], Kexin Huang[2], and Marinka Zitnik[3,4,5,‡]

[1]Bioinformatics and Integrative Genomics, Harvard Medical School, Boston, MA 02115, USA

[2]Health Data Science, Harvard T.H. Chan School of Public Health, Boston, MA 02115, USA

[3]Department of Biomedical Informatics, Harvard Medical School, Boston, MA 02115, USA

[4]Broad Institute of MIT and Harvard, Cambridge, MA 02142, USA

[5]Harvard Data Science Initiative, Cambridge, MA 02138, USA

‡Corresponding author. Email: marinka@hms.harvard.edu

This PDF file includes:

Contents

List of Figures

1 Further Information on Graph Notation and Definitions

Graph-theoretic elements. Graphs consist of the following key elements:

- *Node* v represents a biomedical entity, ranging from atoms to patients.
- *Edge* $e_{u,v}$ is a relation or link between node entities u and v, such as a bond between atoms, an affinity between molecules, a disease association between phenotypes, and a referral between a patient and a doctor. We denote the edge set in the graph as $\mathcal{E}$ and the complementary non-edge set as $\mathcal{E}^c$. The edge can be directed, oriented such that it points from a source (or head node) to a destination (or tail node). The edge can also be undirected, where the nodes have two-way relations.
- *Graph* $G = (\mathcal{V}, \mathcal{E})$ consists of a collection of nodes $\mathcal{V}$ that are connected by an edge set $\mathcal{E}$, such as a molecular graph or protein interaction network. Adjacency matrix $\mathbf{A}$ is commonly used to represent a graph, where each entry $\mathbf{A}_{u,v}$ is 1 if nodes u, v are connected, and 0 otherwise. $\mathbf{A}_{u,v}$ can also be the edge weight between nodes u, v. We denote the number of the nodes $|\mathcal{V}| = n$ and the number of edges $|\mathcal{E}| = m$.
- *Subgraph* $S = (\mathcal{V}_S, \mathcal{E}_S)$ is a subset of a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}_S \subseteq \mathcal{V}, \mathcal{E}_S \subseteq \mathcal{E}$. Examples include disease modules in a protein interaction network or communities in a patient-doctor referral network.
- *Node Feature* $\mathbf{x}_v \in \mathbb{R}^d$ describes attributes of node v. The node feature matrix is denoted as $\mathbf{X} \in \mathbb{R}^{n \times d}$. Similarly, we can have edge features $\mathbf{x}^e_{u,v} \in \mathbb{R}^c$ for edge $e_{u,v}$ collected together into edge feature matrix $\mathbf{X}^e \in \mathbb{R}^{m \times c}$.
- *Label* is a target value associated with either a node Y_v, an edge Y_e, a subgraph Y_S, or a graph Y_G.

Walks and paths in graphs. A walk of length l from node v_1 to node v_l is a sequence of nodes and edges $v_1 \xrightarrow{e_{1,2}} v_2 \cdots v_{l-1} \xrightarrow{e_{l-1,l}} v_l$. A (simple) path is a type of walks where all nodes in the walk are distinct. For every two nodes u, v in the graph G, we define the distance $d(u, v)$ as the length of the shortest path between them. In a heterogeneous graph, a meta-path is a sequence of node types V_i and their edge types $R_{i,j}$: $V_1 \xrightarrow{R_{1,2}} V_2 \cdots V_{l-1} \xrightarrow{R_{l-1,l}} V_l$.

Local neighborhoods. For a node v, we denote its neighborhood $\mathcal{N}(v)$ as the collection of nodes that are connected to v, and its degree is the size of $\mathcal{N}(v)$. The k hop neighborhood of node v is the set of nodes that are exactly k hops away from node v: $\mathcal{N}^k(v) = \{u|d(u,v) = k\}$.

Graph types. The following types of graphs are commonly considered to model complex biomedical systems.

- *Simple, weighted, and attributed graphs.* A simple graph $G = (\mathcal{V}, \mathcal{E})$ is fully described by nodes $\mathcal{V}$ and edges $\mathcal{E}$. For example, a set of binary PPIs gives rise to an unweighted PPI network. Further, nodes and edges can be accompanied by single-dimensional (e.g., edge weights) or multi-dimensional attribute vectors describing node and edge properties. For example, each node in a cell-cell interaction network can have a gene expression attribute vector encoding the gene's expression profile.

- *Multimodal or heterogeneous graphs.* Multimodal or heterogeneous graphs consist of nodes of different types (node type set $\mathcal{A}$) connected by diverse kinds of edges (edge type set $\mathcal{R}$). For example, in a drug-target-disease interaction network, nodes represent $\mathcal{A} = \{\text{drugs}, \text{proteins}, \text{diseases}\}$ and edges indicate $\mathcal{R} = \{\text{drug-target binding}, \text{disease-associated mutations}, \text{treatments}\}$.

- *Knowledge graphs.* A biomedical knowledge graph is a heterogeneous graph that captures knowledge retrieved from literature and biorepositories. A knowledge graph is given by a set of triplets $(u, r, v) \in \mathcal{V} \times \mathcal{R} \times \mathcal{V}$, where nodes u, v belong to node types $\mathcal{A}$ and are connected by edges with type $r \in \mathcal{R}$.

- *Multi-layer graphs.* Multi-layer graphs capture hierarchical relations by grouping individual networks into different layers. Formally, we have a set of networks $G_1, \cdots, G_n$ and l layers where each layer corresponds to a set of networks. Different layers can represent distinct contexts, such as tissues or diseases. Edges can also be added across layers. For example, each tissue can be represented with a tissue-specific PPI network, and PPIs for every tissue can be organized by tissue taxonomy, where each layer corresponds to a tissue taxonomy. Inter-layer edges can be connected for the same proteins across tissues [1].

- *Temporal graphs.* Biomedical systems evolve over time. A temporal graph consists of a sequence of graphs $G_1, \cdots, G_T$ ordered by time, where at each time step t, we observe a subset of all nodes and their activity. For example, a human brain can be modeled as a temporal graph

of brain regions showing task-related increases in neural activity at time t (e.g., greater activity during an experimental task than during a baseline state) and linked based on functional connectivity at t.

- *Spatial graphs.* Nodes or edges in a spatial graph are spatial elements usually associated with coordinates in one, two, or three dimensions, e.g., a spatial representation of cell-cell interactions in the 3-dimensional (3D) Euclidean tissue environment or a 3D point cloud of the protein's atomic coordinates. Spatial graphs are defined by having nodes or edges with spatial locations. This small modification to aspatial graphs has profound effects on how these graphs are used and interpreted because a spatial graph is a location map of points with the constraints of space rather than an abstract structure.

The graph types described above can be combined to give rise to new objects, such as multi-layer spatial graphs or multimodal temporal graphs.

2 Overview of Graph Machine Learning Tasks

We divide machine learning tasks on biomedical graphs into three broad categories: graph prediction, latent graph learning, and graph generation. Each category is associated with several individual graph machine learning tasks.

Canonical graph prediction. Graph prediction aims to predict a label in the graph. The label can be associated with any unit of the graph. There are four canonical graph learning tasks: (1) *Node classification/regression* aims to find a function $f : \mathcal{V} \rightarrow Y_{\mathcal{V}}$ that predicts the label of a node in the graph; (2) *Link prediction* aims to find a function $f : \mathcal{E} \cup \mathcal{E}^c \rightarrow \{0, 1\}$) to predict whether there exists a link between a given pair of nodes in the graph; (3) *Edge classification/regression* aims to find a function $f : \mathcal{E} \rightarrow Y_{\mathcal{E}}$ that predicts the label of an edge; (4) *Graph classification/regression* aims to find a function $f : \mathcal{G} \rightarrow Y_{\mathcal{G}}$ that maps each graph in a graph set to the correct label.

Other graph prediction tasks. In addition to the four standard prediction tasks on graphs, there are additional tasks that are particularly important for biomedical graphs: (1) *Module detection* aims to detect a subgraph module in the graph that contributes to a variable; (2) *Clustering or community detection* aims to partition the graphs into a set of subgraphs such that each subgraph contains similar nodes; (3) *Subgraph classification/regression* aims to predict a label for the subgraph or module; (4) *Dynamic graph prediction* aims to perform the above prediction tasks in a sequence of dynamic graphs.

Latent graph learning. While graph prediction tasks predict given the graph-structured data,

latent graph learning aims to obtain a function $f : \mathcal{V}, \mathbf{X} \rightarrow \mathcal{E}$ to learn the underlying graph structure (e.g., edges) given only the nodes and their feature attributes. The learned graph can be used to (1) perform graph prediction tasks; (2) obtain the inherent topology of the data; and (3) generate latent low-dimensional representations of the feature attributes.

Graph generation. The objective of graph generation is to generate a never-before-seen graph G with some properties of interest. Given a set of training graphs $\mathcal{G}$ with certain shared characteristics, the task is to learn a function $f : \mathcal{G} \rightarrow \mathcal{D}_{\mathcal{G}}$ to obtain a distribution $\mathcal{D}_{\mathcal{G}}$ that characterizes the training graphs. Then, the learned distribution can be used to generate a new graph G', which has the same characteristics as or optimized properties compared to the training graphs.

3 Further Details on Representation Learning Approaches

We here review four important graph representation learning approaches beyond those surveyed in the main text. In particular, we describe:

- *Graph theoretic techniques (Section 3.1):* Networks model relations among real world subjects. Such relations form patterns of structures in the network. These patterns can be quantified by expert-defined statistics to characterize the role of nodes, links, or subgraphs. These statistics can be used to represent elements in the graph.

- *Network propagation methods (Section 3.2):* Nodes in a graph influence each other along the paths. Diffusion measures these spreads of influences. By aggregating diffusion from neighboring nodes to the target node, a diffusion profile captures the local connectivity patterns of the target node.

- *Topological data analysis (Section 3.3):* A dataset has an underlying structure. Topological data analysis (TDA) analyzes the topology of the data to generate an underlying graph structure. Two main frameworks exist. One is called persistent homology [2], which obtains a vector that quantifies various topological shapes at different spatial resolutions. The other is called Mapper [3,4], which first clusters topologically similar data into a node, where similarity is defined by a filter function, and then connects the clusters as the backbone of the data topology. The output is a graph that characterizes the topology of data.

- *Manifold learning (Section 3.4):* Real-world data is usually high-dimensional. To better interpret them, a mapping to find the low-dimensional characterization of the data is ideal. This mapping is called manifold learning, or non-linear dimensionality reduction.

3.1 Graph theoretic techniques

Networks model relations among real world subjects. Such relations form patterns of structures in the network. Among the network science community, many have studied these patterns of graph structures, and proposed graph statistics to measure their characteristics. We summarize such statistics into the following three categories: node-, link-, and subgraph-level statistics. See Figure 1a for an illustration of network statistics.

Node-level techniques. The goal of node-level statistics is to measure the role of a node u in a graph. For instance, betweenness [5] calculates the number of shortest paths that pass through the node. A node with high betweenness is called a bottleneck because it controls the information flow of the network. Various centrality statistics are proposed to measure the various roles of a node regarding its structure and function. For example, k-core [6] measures the position of the node in the graph.

Link-level techniques. Statistics have been demonstrated as a powerful tool for link prediction. They are used to represent the likelihood that a link exists between two nodes. Classic statistics include common neighbor index, Adamic–Adar index, resource allocation index, etc [7]. Most of them rely on the homophily principle. However, [8] recently showed that the protein-protein interaction (PPI) network does not follow homophily because interacting proteins are not necessarily similar, and similar proteins do not necessarily interact. They propose L3, which calculates the number of paths of length 3, and it shows strong performance in learning PPI networks. For a complete list of link-level graph statistics methods, we refer readers to [9].

Subgraph-level techniques. Many subnetwork patterns recur in the network. Such patterns are called motifs, and they are shown to be the basic blocks of complex networks. For instance, a feed-forward loop is an important three-node motif in gene regulatory networks [10]. They have functional roles, such as increasing the response to signals [11]. Individualized motifs for each network can also be computed through frequent subgraph mining algorithms [12].

3.2 Network diffusion

Nodes in a graph influence each other along the paths. Diffusion measures these spreads of influences. The typical resulting outcomes of interest include a scalar ψ between every node v to the source node u that measures the influence, or a diffusion profile ψ_u for source node u, which captures the local connectivity patterns (Figure 1b). Many have studied the effect of diffusion in physics, economics, epidemiology and various formulations have been proposed [13–15]. On a very

related line of work, label propagation leverages connected links to propagate labels [16].

Diffusion state distance. One effective method for biomedical networks is the diffusion state distance [17], which first calculates the number of times a random walk starting at source node will visit a destination node given a fixed number of steps, and iterates this process for every destination node in the graph.

Unsupervised extension. Recent efforts, such as GraphWave [18], adopt an unsupervised learning method to learn an embedding for each node by leveraging heat wavelet diffusion patterns. The resulting embeddings allow nodes residing in different parts of a graph to have similar structural roles within their local network topology.

3.3 Topological data analysis

For a large and high-dimensional dataset D, it is hard to directly gauge their characteristics or obtain a summary of the data. However, all data have an underlying shape or topology T, which can be considered as a network. Topological data analysis (TDA) analyzes the topology of the data to generate an underlying graph structure. There are two major diagrams in TDA: persistent homology (Figure 1c) and mapper (Figure 1d).

Persistent homology. Persistent homology [2] obtains a vector that quantifies various topological shapes at different spatial resolutions. As the resolution scale expands, the noise and artifacts would disappear while the important structure persists. Recent works have used neural networks on top of persistent diagrams to learn augmented topological features [19,20]. [21] propose a differentiable persistent homology layer in the network to make any GNN topology-aware.

Mapper. While persistent homology provides a vector of topology, mapper generates a topology graph [3,4]. This graph is obtained by first clustering topologically similar data into a node, where similarity is defined by a filter function, and then connecting the clusters as the backbone of the data topology. The resulting shape can be used to visualize the data and understand data subtypes [22] and trajectories of development [23]. Mapper is highly dependent on the filter function, and it is usually constructed with domain expertise. Recent works have integrated neural networks to automatically learn the filter function from the data [24].

3.4 Manifold learning

Real-world data is usually high-dimensional. To better interpret them, a mapping to find the low-dimensional characterization of the data is ideal. This mapping is called manifold learning, or non-linear dimensionality reduction (Figure 1e). Note that the underlying manifold can be considered

as a weighted network such that higher weights are assigned to edges between data points that are closer in the manifold.

In a typical setting, we only have a set of data points, or nodes $u_1, \cdots, u_n$, and their associated attributes $\mathbf{x}_1, \cdots, \mathbf{x}_n$ without any connections among them. To learn the underlying manifold using graphs, the first step is to construct an edge set $\mathcal{E}$ that connects nodes given some distance measure. With this connectivity graph, one approach is to apply a graph statistics operator to directly compute the low-dimensional embedding, such as laplacian eigenmap [25] and isomap [26]. Another approach is to optimize various kinds of cost functions to generate low-dimensional embeddings that preserve distance measures on the graph, such as t-SNE [27]. However, such methods are all multi-stage processes in which the outcome depends on the defined distance measures. Recently, a line of research has emerged that can generate embeddings in an end-to-end learnable manner, where the manifold is learned by the signals from the downstream prediction task [28,29].

4 Further Details on Mathematical Formulations

Formulation 1 ***Graph theoretic techniques***

Graph theoretic techniques are functions that map network components to real-values representing aspects of graph structure, such as node proximity [30–32] and node centrality [33]. We use betweenness as an example.

Example. *For node u in graph G, the betweenness is calculated as: $B_u = \sum_{s \neq u \neq t} \frac{\sigma_{s,t}(u)}{\sigma_{s,t}}$, where $\sigma_{s,t}$ is the number of shortest paths between nodes s and t, and $\sigma_{s,t}(u)$ is the number of shortest paths that pass through node u. Basically, the larger the betweenness, the larger the influence of this node u on the network.*

Formulation 2 ***Network diffusion***

Network diffusion computes the network influence signatures based on propagation on the networks. We use Diffusion State Distance (DSD) [17] as an example.

Example. *For a node u, we calculate its diffusion distance $D(u, v_i)$ from every node $v_i \in \mathcal{V}$ as the expected number of times that p_u, a random walk of length k starting from u, will visit v_i. Formally, $D(u, v_i) = \mathbb{E}[sign(v_i, p_u)]$, where $sign(v_i, p_u)$ is 1 if node $v_i \in p_u$ and 0 otherwise. So, we obtain a vector $D(u) = (D(u, v_1), \cdots, D(u, v_n))$. Then, the DSD between nodes u and*

v is defined as $DSD(u,v) = \|D(u) - D(v)\|_1$, the L1-norm of the diffusion vector difference. Intuitively, DSD measures the differences in the node influence from every other node to see if they have similar local connectivity.

Formulation 3 ***Geometric representations along preassigned guiding functions called filters***

A mapper generates the data topology T from the data D [4,34]. A standard mapper procedure consists of the following steps:

1. **Reference Map.** *Given a set of data D, we first define a continuous filter function $f : D \to Z$ that assigns every data point in D to a value in Z.*

2. **Construction of a Covering.** *A finite covering $\mathbb{U} = \{U_\alpha\}_{\alpha \in A}$ is constructed on Z. Each cover consists of a set of points D_α, and is the pre-image of the cover $D_\alpha = f^{(-1)}(U_\alpha)$.*

3. **Clustering.** *For each subset D_α, we apply a clustering algorithm C that generates N_α clusters.*

4. **Topology Graph.** *Each cluster forms a node. If two clusters share data points in D, then an edge is formed. The resulting graph is the topology graph T of data D. Formally, the topology graph is defined as the nerve of the cover by the path-connected components.*

Formulation 4 ***Shallow network embedding***

Shallow network embedding generates a mapping that preserves the similarity in the network [35]. Formally, typical shallow network embeddings are learned in the following three steps:

1. ***Mapping to an embedding space.*** *Given a pair of nodes u, v in network G, we obtain a function f to map these nodes to an embedding space to generate $\mathbf{h}_u$ and $\mathbf{h}_v$.*

2. ***Defining network similarity.*** *We next define the network similarity as $f_n(u, v)$, and the embedding similarity as $f_z(\mathbf{h}_u, \mathbf{h}_v)$.*

3. ***Computing loss.*** *Then, we define the loss $\mathcal{L}(f_n(u,v), f_z(\mathbf{h}_u, \mathbf{h}_v))$, which measures whether the embedding preserves the distance in the original networks. Finally, we apply an optimization procedure to minimize the loss $\mathcal{L}(f_n(u,v), f_z(\mathbf{h}_u, \mathbf{h}_v))$.*

Formulation 5 ***Persistent homology***

Given a dataset D, persistent homology generates a persistence diagram (often referred to as a barcode) that captures the significant topological features in D, such as connected components, holes, and cavities [36,37]. It consists of the following steps:

1. **Construction of the Rips Complex.** *Consider each data point in the original dataset D as a vertex. For each pair of vertices u, v, create an edge if the distance between them is at most ϵ, i.e., $\mathcal{E} = \{(u,v)|d(u,v) \leq \epsilon\}$, given the distance metric d. Consider a monotonically increasing sequence of $\epsilon_0, \cdots, \epsilon_n$, we then generate a filtration of Rips complexes $G_0, \cdots, G_n$.*

2. **Homology.** *Homology characterizes topological structures (e.g., 0-th order homology measures connected components, 1-st order homology measures holes, 2-nd order homology measures voids). A class in a k-th order homology is an instantiation of the homology. In each Rips complex, we can track the various homology classes, which capture various topological structures. For a rigorous definition of homology, we refer reader to [38].*

3. **Persistence Diagrams.** *At each ϵ, we denote the emergence of a new homology class a as its birth using the current ϵ. Similarly, we denote the disappearance of a previous homology class a as its death. Thus, for each homology class a, we can represent them as $(\epsilon_{Birth}, \epsilon_{Death})$. After ϵ reaches the end of the sequence, we have a set of 2-dimensional points (x, y), each corresponding to the birth and death of a homology class. The persistence diagram is a plot of these points. The farther away from the diagonal, the longer the lifespan of the homology class, implying that the structure is an important shape of the data topology.*

Formulation 6 ***Manifold learning***

The goal of manifold learning is to learn a low-dimensional embedding that captures the data manifold from high-dimensional data [39]. In the following, we use isomap [26] as an example.

Example. *First, given a set of data points with high-dimensional feature vectors* $\mathbf{x}_1, \cdots, \mathbf{x}_n$*, we apply a* k*-nearest neighbor algorithm and connect the nearest neighbors to form a neighborhood graph* G*. Next, we calculate the distance matrix* $\mathbf{D}$*, where each entry* $\mathbf{d}_{i,j}$ *is the length of the shortest path between nodes* i *and* j *in the neighborhood graph. Then, we apply an optimization algorithm to obtain a set of low-dimensional vectors* $\mathbf{h}_1, \cdots, \mathbf{h}_n$ *that minimizes* $\sum_{i<j} \left(\|\mathbf{h}_i - \mathbf{h}_j\| - d_{i,j}\right)^2$.

Formulation 7 ***Graph neural networks***

GNNs learn compact representations or embeddings that capture network structure and node features [40–42]. A GNN generates outputs through a series of propagation layers [43], where propagation at layer l *consists of the following three steps:*

1. ***Neural message passing.*** *The GNN computes a message* $\mathbf{m}_{u,v}^{(l)} = \textsc{Msg}(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)})$ *for every linked nodes* u, v *based on their embeddings from the previous layer* $\mathbf{h}_u^{(l-1)}$ *and* $\mathbf{h}_v^{(l-1)}$.

2. ***Neighborhood aggregation.*** *The messages between node* u *and its neighbors* $\mathcal{N}_u$ *are aggregated as* $\hat{\mathbf{m}}_u^{(l)} = \textsc{Agg}(\mathbf{m}_{uv}^{(l)} | v \in \mathcal{N}_u)$.

3. ***Update.*** *The GNN applies a non-linear function to update node embeddings as* $\mathbf{h}_u^{(l)} = \textsc{Upd}(\hat{\mathbf{m}}_u^{(l)}, \mathbf{h}_u^{(l-1)})$ *using the aggregated message and the embedding from the previous layer.*

Formulation 8 ***Generative modeling***

Generative models optimize and learn data distributions in order to generate graphs with desirable properties. In the following, we use VGAE [44] as an example.

Example. *VGAE is a graph extension of variational autoencoders [45]. Given a graph* G *with*

adjacency matrix $\mathbf{A}$ *and node features* $\mathbf{X}$*, we use two GNNs to encode the graph and generate the latent mean vector* $\mu_{\mathbf{Z}}^{u}$ *and the log-variance* $\log(\sigma_{\mathbf{Z}}^{u})$ *parameters for each node* u*. Formally,* $\mu_{\mathbf{Z}} = \mathrm{GNN}_{\mu}(\mathbf{A}, \mathbf{X}) \in \mathbb{R}^{|\mathcal{V}| \times d}$*, and* $\log(\sigma_{\mathbf{Z}}) = \mathrm{GNN}_{\sigma}(\mathbf{A}, \mathbf{X}) \in \mathbb{R}^{|\mathcal{V}| \times d}$*. We can then obtain the latent distribution* $q(\mathbf{Z}|G) \sim \mathcal{N}(\mu_{\mathbf{Z}}, (\log(\sigma_{\mathbf{Z}})))$*, where we can draw a latent embedding sample* $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}| \times d}$ *from the latent distribution. Then, given the latent embedding, we feed into a probabilistic decoder* $p_{\theta}(\hat{\mathbf{A}}|\mathbf{Z})$ *to generate a network* $\hat{G}$ *with adjacency matrix* $\hat{\mathbf{A}}$*. In VGAE, the decoder is a dot product, i.e.,* $p_{\theta}(\hat{\mathbf{A}}_{uv} = 1|\mathbf{Z}) = \mathrm{sigmoid}(\mathbf{z}_{u}^{T}\mathbf{z}_{v})$*. This way, we obtain a newly generated graph* $\hat{G}$*. Finally, we optimize the weight using the variational reconstruction loss* $\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|G)}[p(G|\mathbf{Z})] - \mathrm{KL}(q(\mathbf{Z}|G)\|p(\mathbf{Z}))$.

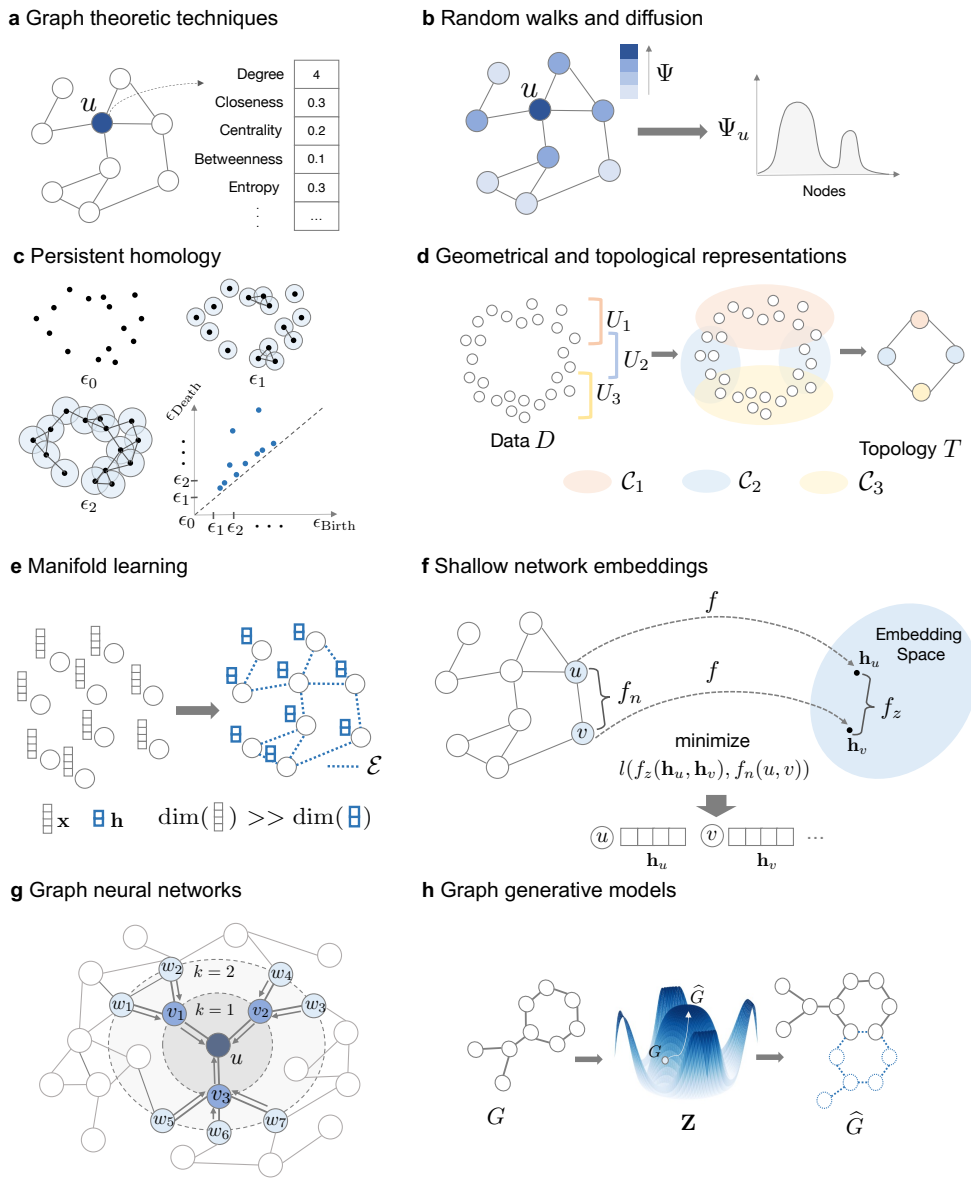

Figure 1: Predominant graph learning paradigms. Graph machine learning is a large field with a diverse set of methods. Here, we summarize and categorize seven major graph machine learning methods paradigms. **(a)** Graph theoretic techniques compute a deterministic value that describes patterns in the graph; **(b)** diffusion process captures the importance and influence of nodes through network diffusion; **(c-d)** topological data analysis provides summarized views of the shape of the data; **(e)** manifold learning aims to obtain the underlying graph structure of data and a low-dimensional embedding; **(f)** shallow network embeddings generate node representations though direct encoding of node similarities in the input graph; **(g)** graph neural networks learn graph embeddings through supervised signals by neural networks; **(h)** generative models generate novel graphs that have desirable properties. Note that panels (c) and (d) are two major diagrams of TDA.

Supplementary References

1. Zitnik, M. & Leskovec, J. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* (2017).

2. Edelsbrunner, H. & Harer, J. Persistent homology-a survey. *Contemporary Mathematics* (2008).

3. Nicolau, M., Levine, A. J. & Carlsson, G. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *PNAS* (2011).

4. Singh, G., Mémoli, F. & Carlsson, G. E. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *SPBG* (2007).

5. Freeman, L. C. A set of measures of centrality based on betweenness. *Sociometry* (1977).

6. Kitsak, M. *et al.* Identification of influential spreaders in complex networks. *Nature Physics* (2010).

7. Gao, F., Musial, K., Cooper, C. & Tsoka, S. Link prediction methods and their accuracy for different social networks and network metrics. *Scientific Programming* (2015).

8. Kovács, I. A. *et al.* Network-based prediction of protein interactions. *Nature Communications* (2019).

9. Liu, C. *et al.* Computational network biology: Data, models, and applications. *Physics Reports* (2020).

10. Milo, R. *et al.* Network motifs: simple building blocks of complex networks. *Science* (2002).

11. Martin, O. C., Krzywicki, A. & Zagorski, M. Drivers of structural features in gene regulatory networks: From biophysical constraints to biological function. *Physics of Life Reviews* (2016).

12. Jiang, C., Coenen, F. & Zito, M. A survey of frequent subgraph mining algorithms. *Knowledge Engineering Review* (2013).

13. Cowan, R. & Jonard, N. Network structure and the diffusion of knowledge. *Journal of Economic Dynamics and Control* (2004).

14. Raj, A., Kuceyeski, A. & Weiner, M. A network diffusion model of disease progression in dementia. *Neuron* (2012).

15. Akbarpour, M. & Jackson, M. O. Diffusion in networks and the virtue of burstiness. *PNAS* (2018).

16. Wang, F. & Zhang, C. Label propagation through linear neighborhoods. In *ICML* (2006).

17. Cao, M. *et al.* Going the distance for protein function prediction: a new distance metric for protein interaction networks. *PLOS One* (2013).

18. Donnat, C., Zitnik, M., Hallac, D. & Leskovec, J. Learning structural node embeddings via diffusion wavelets. In *KDD* (2018).

19. Hofer, C. D., Kwitt, R., Niethammer, M. & Uhl, A. Deep learning with topological signatures. In *NeurIPS* (2017).

20. Carrière, M. *et al.* Perslay: A neural network layer for persistence diagrams and new graph topological signatures. In *AISTATS* (2020).

21. Hofer, C. D., Graf, F., Rieck, B., Niethammer, M. & Kwitt, R. Graph filtration learning. In *ICML* (2020).

22. De Cecco, L. *et al.* Head and neck cancer subtypes with biological and clinical relevance: Meta-analysis of gene-expression data. *Oncotarget* (2015).

23. Madhobi, K. F. *et al.* A visual analytics framework for analysis of patient trajectories. In *ACM-BCB* (2019).

24. Bodnar, C., Cangea, C. & Liò, P. Deep graph mapper: Seeing graphs through the neural lens. *arXiv:2002.03864* (2020).

25. Belkin, M. & Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NeurIPS* (2001).

26. Tenenbaum, J. B., De Silva, V. & Langford, J. C. A global geometric framework for nonlinear dimensionality reduction. *Science* (2000).

27. Maaten, L. v. d. & Hinton, G. Visualizing data using t-sne. *JMLR* (2008).

28. Halcrow, J., Mosoi, A., Ruth, S. & Perozzi, B. Grale: Designing networks for graph learning. In *KDD* (2020).

29. Wang, Y. *et al.* Dynamic graph CNN for learning on point clouds. *ACM Transactions On Graphics* (2019).

30. Menche, J. *et al.* Uncovering disease-disease relationships through the incomplete interactome. *Science* (2015).

31. Baryshnikova, A. Systematic functional annotation and visualization of biological networks. *Cell Systems* (2016).

32. Agrawal, M., Zitnik, M., Leskovec, J. *et al.* Large-scale analysis of disease pathways in the human interactome. In *PSB* (2018).

33. Goh, K.-I. *et al.* The human disease network. *PNAS* (2007).

34. Wang, R., Li, S., Cheng, L., Wong, M. H. & Leung, K. S. Predicting associations among drugs, targets and diseases by tensor decomposition for drug repositioning. *BMC Bioinformatics* (2019).

35. Hosseini, A., Chen, T., Wu, W., Sun, Y. & Sarrafzadeh, M. Heteromed: Heterogeneous information network for medical diagnosis. In *CIKM* (2018).

36. Hofer, C. D., Kwitt, R. & Niethammer, M. Learning representations of persistence barcodes. *JMLR* (2019).

37. Bubenik, P. Statistical topological data analysis using persistence landscapes. *JMLR* (2015).

38. Edelsbrunner, H. & Harer, J. *Computational topology: an introduction* (2010).

39. Burkhardt, D. B. *et al.* Quantifying the effect of experimental perturbations in single-cell rna-sequencing data using graph signal processing. *bioRxiv* (2019).

40. Han, P. *et al.* GCN-MF: disease-gene association identification by graph convolutional networks and matrix factorization. In *KDD* (2019).

41. Ingraham, J., Garg, V. K., Barzilay, R. & Jaakkola, T. S. Generative models for graph-based protein design. In *NeurIPS* (2019).

42. Xie, Y., Peng, J. & Zhou, Y. Integrating protein-protein interaction information into drug response prediction by graph neural encoding (2019).

43. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. Neural message passing for quantum chemistry. In *ICML* (2017).

44. Kipf, T. N. & Welling, M. Variational graph auto-encoders. *NeurIPS Workshop* (2016).

45. Kingma, D. P. & Welling, M. Auto-encoding variational bayes. In *ICLR* (2014).