Journal of Machine Learning Research 21 (2020) 1-39 Submitted 6/19; Revised 3/20; Published 6/20

Probabilistic Learning on Graphs via Contextual Architectures

Davide Bacciu BACCIU@DI.UNIPI.IT
Federico Errica[*] FEDERICO.ERRICA@PHD.UNIPI.IT
Alessio Micheli MICHELI@DI.UNIPI.IT
Department of Computer Science
University of Pisa
56127, PI, Italy

Editor: Pushmeet Kohli

Abstract

We propose a novel methodology for representation learning on graph-structured data, in which a stack of Bayesian Networks learns different distributions of a vertex's neighbourhood. Through an incremental construction policy and layer-wise training, we can build deeper architectures with respect to typical graph convolutional neural networks, with benefits in terms of context spreading between vertices. First, the model learns from graphs via maximum likelihood estimation without using target labels. Then, a supervised readout is applied to the learned graph embeddings to deal with graph classification and vertex classification tasks, showing competitive results against neural models for graphs. The computational complexity is linear in the number of edges, facilitating learning on large scale data sets. By studying how depth affects the performances of our model, we discover that a broader context generally improves performances. In turn, this leads to a critical analysis of some benchmarks used in literature.

Keywords: Structured domains, deep graph networks, graph neural networks, deep learning, maximum likelihood, graph classification, node classification.

1. Introduction

The development of machine learning methodologies has always been driven by the interest in modelling the human brain's inner workings, from the Perceptron (Rosenblatt, 1957) to Spiking neurons (Izhikevich, 2003) and feedforward neural networks, to name a few. These models can only process flat inputs, i.e., vectors, so they are not the best candidates to deal with more involved data representations. Indeed, we often reason about the complex dynamics of the real world in relational terms by considering entities that interact with each other. The complexity of these interactions gives rise to different *structures*, such as sequences, trees, and graphs. A sequence may be well suited for temporal data, but it is not adequate to represent a molecule or a social network where entities have multiple relationships. Hence, choosing a particular structure to solve a problem affects the kind of information we want to preserve in the data, also referred to as *relational inductive bias* (Battaglia et al., 2018). Structured domains allow representing more articulated informa-

[*]. Corresponding author

tion than flat data. However, they also need models that explicitly take into account the chosen relational inductive bias and learn to extract structural patterns. For these reasons, the adaptive processing of structures is of fundamental importance in machine learning, data mining, and, more generally, artificial intelligence. Many impactive applications can take advantage of graph processing methods, such as chemistry (Swamidass et al., 2005), bioinformatics (Sharan and Ideker, 2006), and social science (Yanardag and Vishwanathan, 2015). Consequently, further investigation is necessary from both theoretical and practical perspectives.

Depending on the assumptions made about the underlying interactions, researchers developed a plethora of models: Recurrent Neural Networks (Elman, 1990) and Hidden Markov Models (Rabiner and Juang, 1986) for sequences; Recursive Neural Networks (Goller and Kuchler, 1996; Sperduti and Starita, 1997) and Hidden Tree Markov Models (Frasconi et al., 1998; Diligenti et al., 2003; Bacciu et al., 2012) for trees and directed acyclic graphs (Micheli et al., 2004); lastly, Deep Neural Graph Networks (DNGNs) as defined in Bacciu et al. (2020) for arbitrary structures, i.e., graphs, which are the focus of this paper. Notice that the term DNGN is very general as it comprises, in our context, both iterative and convolution-based neural architectures (Section 2).

Yet, dealing with structured data is often associated with higher computational costs and theoretical issues. As a matter of fact, we cannot use an (unconstrained) Recursive Neural Network on the cyclic dependencies present in a graph because that can induce a non-converging process to compute the representation of its entities. In the recent past, *kernels* have been a very successful approach to deal with graph-structured information (Ralaivola et al., 2005; Da San Martino et al., 2012), allowing to map pairs of graphs to similarity scores between the structures. Interestingly, there also exist hybrid methods between kernels, relational, and logical learning (Frasconi et al., 2014). However, kernels are typically *non-adaptive*, as they rely on the computation of human-defined similarity measures that are often expensive to compute, restricting their applicability to medium-sized data sets. This is why, in recent years, researchers have designed architectures for graph learning that focus on each entity's neighbourhood. There are at least two reasons for such rapid change: first, the *adaptive* exploitation of graphs, whose richness of representation is far superior to simpler structures, could replace kernels' feature engineering. Secondly, the increasing availability of this kind of data will soon require methods that scale to large data sets. Nonetheless, technical obstacles often limit the interaction between far-away entities, such as the vanishing of the gradient and over-smoothing (Li et al., 2018). As a result, convolution-based DNGNs are typically not very deep networks with less than five layers (Kipf and Welling, 2017; Xu et al., 2019), making it challenging to capture long dependencies.

In this paper, we propose to address these limitations with a novel technique for graph-structured data called Contextual Graph Markov Model (CGMM), which is flexible in its construction, efficient, and scalable. It comprises a stack of Bayesian Networks trained in an incremental layer-wise fashion, whose role is to extract different patterns for each entity in the structure. As the model can learn from data without any supervised target label, it is suitable for embedding purposes like *struc2vec* (Ribeiro et al., 2017) and transfer-learning. To evaluate its effectiveness, we target graph and node classification, and we obtain competitive performances when compared to expensive kernels and neural models, thus empirically showing the richness of the information captured by the model. Notably, computational complexity is linear in the number of edges, and we can achieve fine-grained scalability by distributing each vertex's job across different computational resources.

We extend our preliminary work (Bacciu et al., 2018a) in the following ways. First, we devise a more general formulation that accounts for both discrete and continuous vertex representations, thus increasing the expressiveness of the model. Secondly, we present a new set of experiments that compare our work with kernels and state of the art DNGNs. In particular, we thoroughly analyze the effect of depth on performances, showing that we can build deeper architectures than the typical neural network counterparts. Moreover, we simplify the construction of the architecture with a considerable saving of training time, and we study the model's behaviour by visualizing how messages flow in a real graph. Lastly, we investigate the usefulness of some commonly used classification benchmarks, showing that, for some of them, shallow networks seem to be sufficient.

The rest of the article is organized as follows: Section 2 reviews the literature, focusing on models that are relevant from an architectural point of view; Section 3 presents our method; Section 4 gives details about the experimental setting; Section 5 discusses our key findings; finally, Section 6 contains our last remarks.

2. Background

The methods for processing graphs can be divided into four main families, namely kernels, spectral methods, DNGNs, and probabilistic models. Despite the importance of the former two, we prioritize *vertex-centric* ones because they are more efficient and based on the same underlying principles as our work, though radically different from an architectural point of view. The rest of the section is intended to be an analysis of different architectures rather than a comprehensive review of graph processing methodologies. From now on, we will refer to entities as *vertices* and the relations between them as *edges*. Finally, the set of vertices to which one entity is related is called a *neighbourhood*.

We can trace the first ideas on neural networks for graphs back to Sperduti and Starita (1997), who proposed the Generalized Recursive Neuron (GRN) for supervised learning on hierarchical structures (trees and Directed Positional Acyclic Graphs). Notably, the authors introduced the idea of neighbourhood aggregation to gather information from a *variable* number of vertices. More recently, Micheli (2009) and Scarselli et al. (2009) managed to deal with *cyclic* structures using two different neural architectures: the Neural Network for Graphs (NN4G) and the Graph Neural Network (GNN). These are based on layering and iterative message-passing schemes, respectively; nowadays, the former goes by the name of "spatial convolution". NN4G relaxes the causality assumption of the GRN to use a feedforward network, and it exploits Cascade Correlation (Fahlman and Lebiere, 1990) to automatize the construction of the multi-layer architecture; we take inspiration from its incremental and layer-wise approach to design our model. The Graph Neural Network, instead, models a graph as a dynamical system, where vertices are guaranteed to reach a stable state by imposing contractive constraints on the recursive dynamics of the units. However, the procedure may be slow to converge, which results in higher computational requirements for training and inference.

Interestingly, these two formalisms can be unified under an abstract framework called Message Passing Neural Network (MPNN) (Gilmer et al., 2017); in fact, layers implement an information spreading mechanism which is very similar to message-passing, as we will see in the following sections. Nonetheless, the *content* of the messages is not the same in different models, being heavily dependent on the architecture.

Recently, the Graph Convolutional Network (GCN) (Kipf and Welling, 2017) has become a popular model that fits into the MPNN framework. GCN's formulation approximates the spectral decomposition of the Laplacian matrix L with a Chebyshev polynomial of degree one, i.e., $L+I$ where I is the identity matrix. Moreover, the convolution layer is a weighted sum of a vertex's neighbourhood, where the weight is given by the combination of learned parameters and Laplacian entries. The final architecture is a stack of such layers interleaved by non-linear activations. Li et al. (2018) recently proved that GCNs suffer from the *over-smoothing* effect, meaning that the vertices' representations tend to become similar after a few layers. As a result, the convolution itself prevents the model from learning long dependencies.

PATCHY-SAN (PSCN) (Niepert et al., 2016) processes graphs thanks to a very different strategy. In particular, the method requires determining a normalized neighbourhood ordering for each vertex that is consistent across the entire data set, as well as to impose an upper bound on the neighbourhood size. The ordering is obtained via a *non-adaptive* procedure, which may not work well in all situations, and fixing the size of the neighbourhood makes it necessary to introduce padding or to exclude some of the neighbours. This is why, generally speaking, an *ordering-free* procedure should be preferred. GraphSAGE (Hamilton et al., 2017), on the other hand, exploits a spatial convolution mechanism very similar to that of NN4G. However, the authors test different neighbourhood aggregation schemes and extend the self-supervised link-reconstruction loss of Kipf and Welling (2016). Similarly to PSCN, GraphSAGE relies on a neighbourhood *sampling* scheme, which needs additional hyper-parameters, to keep computational complexity constant. Instead, the Graph Isomorphism Network (GIN) (Xu et al., 2019) builds upon the limitations of GraphSAGE by extending it with arbitrary aggregation functions on multi-sets. The model is proven to be as theoretically powerful as the 1-dim Weisfeiler-Lehman test of graph isomorphism. Very recently, Wagstaff et al. (2019) gave an upper bound to the number of hidden units needed to realize permutation-invariant functions over sets and multi-sets.

Some models also apply a pooling scheme after convolutional layers to reduce the size of a graph. Among those, we find the Edge-Conditioned Convolution (ECC) (Simonovsky and Komodakis, 2017). ECC is different from most DNGNs in that it learns different parameters for each edge. Therefore, edge parameters act as weights for the neighbours' aggregation, which we can interpret as a simple form of attention (Bahdanau et al., 2015). The pooling scheme of ECC is based on a *predefined* algorithm, which outputs a pooling map that maintains differentiability. On the other hand, DiffPool (Ying et al., 2018) proposes an adaptive pooling mechanism that collapses vertices based on a supervised criterion. DiffPool is more of a framework rather than a single model: it combines a differentiable encoder for graphs, such as a GCN, with this pooling strategy so that each part of the architecture is trainable. DiffPool may have restricted application in contexts where supervised labels are not available, as there is no easy way to define a self-supervised loss and learn the pooling maps. ARMA (Bianchi et al., 2019) is another DNGN whose pooling strategy deserves mention. For large data sets, ARMA uses the Fiedler vector of a graph to drop approximately half of the vertices in the graph at each pooling step. The convolutional layer, based on spectral theory, exploits the ARMA filter, which is known to be more flexible than the polynomial one. Similarly to ECC, the pooling scheme of ARMA is not adaptive as it depends on the Laplacian of each graph.

The attention mechanism in the context of graphs was introduced by the Graph Attention Network (GAT) (Velickovic et al., 2018). Indeed, GAT uses multi-head attention on pairs of vertices to determine the importance of an edge, and weight sharing reduces the number of parameters. Similarly to other DNGNs, GAT's architecture was fixed to three layers, and it seems to be slower than simpler methods (Fey and Lenssen, 2019). The authors report good experimental results on vertex classification tasks, which were later revised by Shchur et al. (2018).

The Deep Graph Convolutional Neural Network (DGCNN) (Zhang et al., 2018) differs from other works in two ways. First, it *concatenates* the output of several graph convolutional layers; secondly, vertices are sorted and aligned by a specific algorithm called SortPool that produces *truncated* graph representations. A 1-dimensional Convolutional Neural Network, followed by fully-connected layers, is then used to solve graph classification tasks. Its extension, the Parametric Graph Convolution DGCNN (PGC-DGCNN) (Tran et al., 2018), further generalizes the model in order to consider neighbourhoods of radius larger than one. The sorting scheme of these models, inspired by graph colouring algorithms, relies on the ability of the convolutional layers to extract consistent colouring across graphs, but this ability may be hampered by the presence of noise in the data. Our proposal does not need a sorting scheme because it aggregates the representations of all vertices in the graph.

From the Reservoir Computing field, the GraphESN (Gallicchio and Micheli, 2010) is an Echo State Network for graphs. GraphESN generates a graph encoding when vertices reach a stable state in a recursive system subject to contractive constraints. While this may seem similar to the Graph Neural Network of Scarselli et al. (2009), GraphESN does not require training except for the last layer, thus addressing the efficiency issue of the former approach.

A different strategy is taken by Deep Graph Infomax (DGI) (Velickovic et al., 2019), which incorporates DNGNs into a framework that maximizes the mutual information between vertex and graph representations. DGI relies on a corruption function to generate a corrupted version of the input graph, and it learns representations that allow discriminating between the original and corrupted vertices without having access to supervised target labels. In this sense, the DNGN that produces such representations is encouraged to highlight differences between graphs, while our approach focuses on modelling the distribution of a vertex neighbourhood. When solving classification tasks, DGI trains a standard supervised model on the learned representations, which is the same approach we will adopt in this work.

The family of probabilistic models for graph data is much smaller than the previous ones, possibly because there still is no straightforward way to formalize neighbourhood aggregation in probability terms. The foundational work of Macskassy and Provost (2007) summarized the first relational probabilistic classifiers for network data. All the proposed classifiers are motivated by the *homophily* assumption, which states that linked entities tend to belong to the same class. More recently, the Variational Graph Auto-Encoder (VGAE)

(Kipf and Welling, 2016) was proposed as an extension of the Variational Auto-Encoder (Kingma and Welling, 2014) to address link-prediction on graphs. VGAE can implement both its encoder and decoder as DNGNs, and the model is trained to reconstruct the graph structure through vertex embeddings. This model has shown good performances on link prediction tasks, but it is not directly applicable to other problems such as vertex or graph classification.

Another model that takes advantage of DNGNs to approximate probability distributions is the Graph Markov Neural Network (GMNN) Qu et al. (2019), which is defined for semi-supervised vertex classification but can also be applied to link prediction tasks. Its formulation borrows from statistical relational learning theory, and it approximates intractable distributions through DNGNs. Specifically, training is performed using the Variational Expectation-Maximization algorithm (Neal and Hinton, 1998), with an E-step that captures the posterior distribution of each vertex target label conditioned on its neighbours and an M-step that maximizes the probability of the labelled vertices (given the posterior distribution computed at the E-step). Independent DNGNs approximate both variational steps, and the overall architecture is trained in an end-to-end fashion. Similarly to VGAE, GMNN cannot be readily applied to graph classification tasks.

On the other hand, Graph2Gauss (Bojchevski and Gnnemann, 2018) is a probabilistic model that learns to map a vertex's attributes to the sufficient statistics of a multivariate Gaussian distribution i.e., mean and covariance matrix. Consequently, the graph structure is not used at inference time while being taken into account during learning. The catch is the following: vertices that are structurally closer to each other in the graph should have similar representations, i.e., statistics of a Gaussian distribution. Graph2Gauss shows impressive performances on link prediction and an efficient inference strategy, but it cannot deal with attributed edges, which usually carry important information e.g., the bond type in molecules.

With respect to the above architectural choices, we propose the first deep probabilistic model for graphs that can handle arbitrary vertex features (with an extension for discrete edge features) and model the learning process in a fully probabilistic way. To this end, we generalize Hidden Tree Markov models (Bacciu et al., 2012) to graphs, using an incremental layering policy similar to NN4G's that is necessary to correctly define the probabilistic layers of our model. As a by-product of this incremental construction, the model does not suffer from the vanishing of the gradient: this means that each vertex can capture longer dependencies by just increasing the depth of the architecture as much as we need, without having to introduce other techniques such as Batch Normalization (Ioffe and Szegedy, 2015) and Layer Normalization (Ba et al., 2016). Thanks to layering analyses (Sections 5.3-5.5), we will show how context spreads across the graph and what are the benefits of depth in terms of classification performance. Instead of relying on a fixed neighbourhood or sampling schemes, we consider all vertices without discarding information. The probabilistic formulation efficiently leverages examples with respect to their *structure* and *vertex features*, without the need for supervised target labels. Then, the model is combined with standard classification methods to solve graph and vertex classification tasks. As we will see, our model recalls a probabilistic version of deep neural graph networks, also called *Deep Bayesian Graph Networks* (DBGNs) in Bacciu et al. (2020).

3. Contextual Graph Markov Model

With CGMM, we aim to bridge the gap between probabilistic learners for trees and Deep Neural Graph Networks. Our convolutional layer is indeed a Bayesian Network that captures the neighbourhood's distribution of each vertex. As in convolutional DNGNs, stacking layers is necessary to spread information between vertices, but CGMM differs from them in the construction of the architecture. The model is trained incrementally (layer by layer) on a set of graphs to construct vertex or graph embeddings without relying on any supervised label related to the classification task. Then, classical methods such as logistic regression or a Multi-Layer Perceptron (MLP) use these embeddings to solve vertex/graph supervised tasks.

We adopt a top-down approach to describe CGMM. To begin with, we give a high-level overview of the whole architecture so that the reader can understand the rationale behind the design of the core parts. Next, we discuss the learning process and a straightforward extension via a structure-aware approximation technique.

3.1. Notation

We consider the problem of learning from a population of graphs $\mathcal{G} = \{\mathbf{g}_1, \dots, \mathbf{g}_N\}$, where each sample $\mathbf{g}_i$ is a graph with varying topology. A graph $\mathbf{g} = (\mathcal{V}_g, \mathcal{E}_g, \mathcal{X}_g, \mathcal{A}_g)$ is formally defined by a set of vertices $\mathcal{V}_g$ and by a set of edges $\mathcal{E}_g$ between pairs of vertices. Each vertex u has an associated feature vector $x_u \in \mathcal{X}_g$, which can be continuous or discrete. A directed edge (u,v) between vertices u and v may hold a feature vector $a_{uv} \in \mathcal{A}_g$. Instead, an undirected edge can be modeled by two directed edges having the same features. The neighbourhood of a vertex $u \in \mathcal{V}_g$ is defined as $\mathcal{N}(u) = \{v \in \mathcal{V}_g | (v,u) \in \mathcal{E}_g\}$, that is the set of vertices associated to incoming edges. Without loss of generality, we consider an *open* neighbourhood of u i.e., one that does not include u itself unless an explicit self-loop is present. Depending on the vertex or graph classification task at hand, we will rely on *supervised target labels* y_v $\forall v \in \mathcal{V}_g$ and y_g, respectively. Finally, we speak of *context* of a vertex u to denote the set of vertices that, directly or indirectly, influence vertex u.

3.2. Architecture

Figure 1 presents the overall structure of the architecture. The input is a graph represented by black interconnected vertices, which is initially fed to the first layer of our model. For now, it is sufficient to mention that each layer maps every vertex into a vector. This is called an isomorphic transduction (Frasconi et al., 1998), and details are given in Section 3.3. Any subsequent layer can use these intermediate representations together with the input graph; in general, the output of each layer ℓ will be conditioned on an arbitrary subset $\mathbb{L}(\ell)$ of previous layers' outputs.

Once the process completes, there will be n different vectors for each vertex, where n is the total number of layers of our architecture. Subsequent concatenation yields a richer representation for all vertices in the graph; if one wants to tackle vertex classification tasks, a classifier must be trained on these representations. However, in case we are interested in graph classification, the outputs have to be first aggregated into a single graph embedding; Section 3.2.2 describes this process in detail.

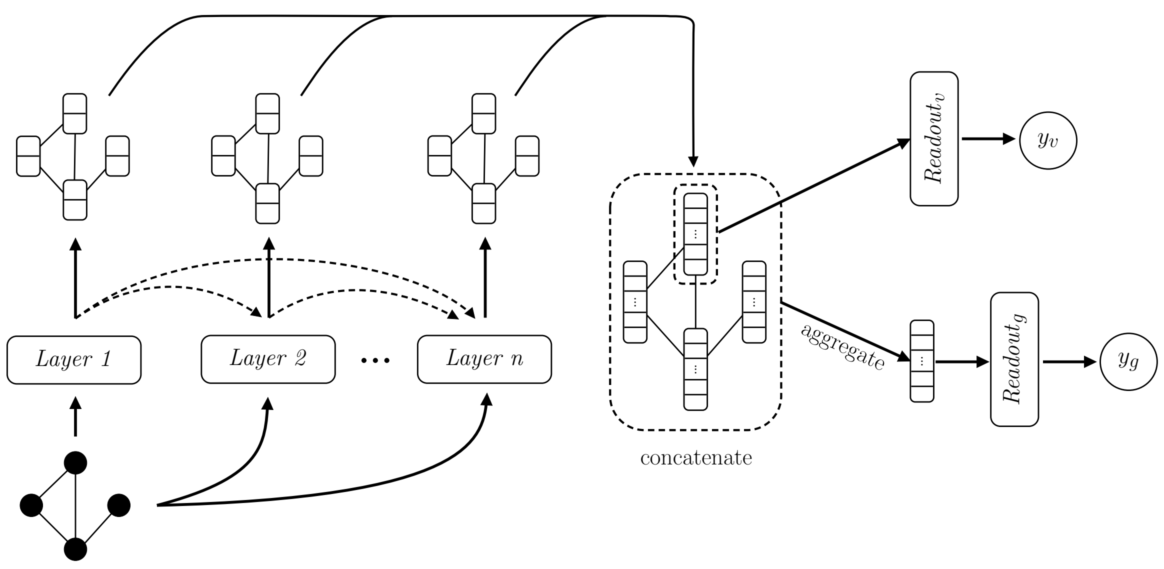


Figure 1: The figure shows a high-level overview of the architecture. The goal is to output a vertex target label y_v (shown for one vertex only in the figure) or a graph target label y_g depending on the supervised task at hand. We incrementally build the network, one layer after another, by fitting all layers on data without using supervised target labels. Each layer produces a vector representation for each vertex of the input graph. Dashed arrows emphasize that what is passed on to subsequent layers is contextual information. The final representation of a vertex v is obtained by concatenating intermediate representations of that vertex across different layers. Such representations are used to solve both vertex classification and graph classification by applying a supervised method to vertex or graph representations.

Our probabilistic architecture puts forward two significant differences with respect to recent DNGNs. First of all, the construction of the model is *incremental* as training is performed one layer at a time, and parameters are frozen before moving to the next layer. Secondly, CGMM models the distribution of each vertex in a *fully probabilistic* fashion, whereas DNGNs usually rely on supervised labels or self-supervised loss functions (see Section 2). Incremental training was introduced by Fahlman and Lebiere (1990) in a supervised learning context, with the nice advantage that there are no exploding/vanishing gradient issues. In our case, the incremental construction is also necessary to define the layers of the model by relaxing the causality assumptions, i.e., avoiding mutual dependencies between latent variables in the same layer, as discussed in the following Section 3.3.1. Moreover, it allows us to construct deeper and efficient networks capable of spreading information without being limited by gradient-related issues (Li et al., 2018).

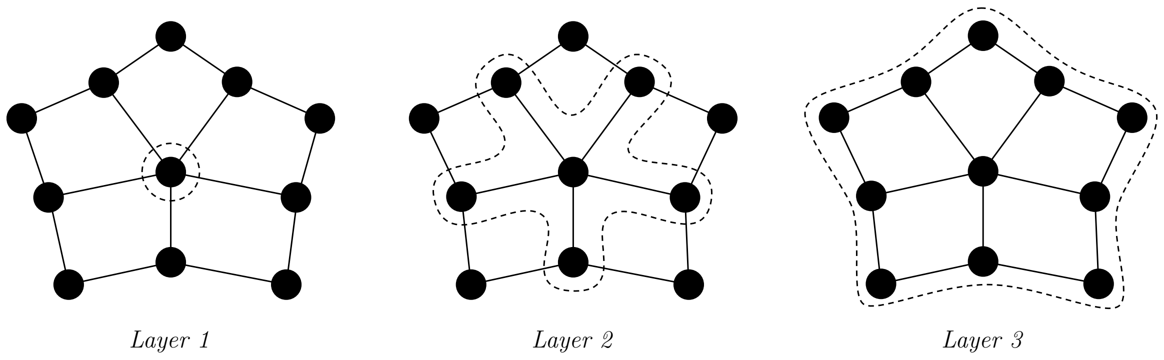


Figure 2: We show how the context window of the central vertex in the graph widens. At the beginning (left graph), the vertex has access to its information only. Then, from layer two, the vertex is allowed to see its neighbourhood. By the same token, at layer three, the neighbouring vertices have already encoded their own neighbourhood knowledge, so the context window broadens.

3.2.1. Context Spreading

One can see how the architecture of Figure 1 resembles Convolutional Neural Networks (CNNs) (LeCun et al., 1995). Layers are indeed responsible for the aggregation of *local* information by generalizing the concept of convolution to non-Euclidean spaces (Bruna et al., 2014). Intuitively, by repeated application of such convolution, we increase the context window that each vertex can capture, i.e., the local receptive field. We clarify this aspect in Figure 2. Consider the central vertex of the depicted graph; at the beginning, context is limited to the vertex itself. From layer two, the vertex will always process information coming from its neighbourhood, and at the same time every other neighbour will do the same. Therefore, the context that a vertex captures at layer three is given by the union of the neighbouring contexts computed at layer two; for a formal description and proof of this context behaviour, please refer to Micheli (2009). In a similar vein to hierarchical models for flat data, our goal is to extract distinct patterns from different layers of the network. When the structure is of practical importance the architecture will be deeper, but there may be cases in which a shallow model is sufficient; Section 5 provides an interesting example.

3.2.2. Graph Fingerprint Construction

We can build a graph fingerprint in 2 steps, as shown in Figure 3. Vertex representations of different layers are concatenated into $|\mathcal{V}_g|$ vectors of longer size. We choose concatenation as the most conservative choice because others do not guarantee the same expressivity (e.g., a weighted sum). This choice is also necessary to keep the process completely separated from any supervised target label. At this point, the concatenated vectors have to be merged together to obtain the graph fingerprint, so we use either *sum* or *mean*. Notice that the mean abstracts from the size of a graph and focuses on vertex distributions, whereas the sum tells us which features are most prevalent in the graph. It is reasonable to assume that the best choice is task-dependent. To simplify our analysis and generate task-agnostic

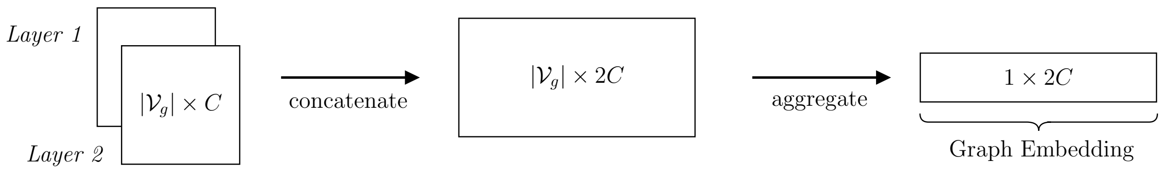


Figure 3: Example of a graph embedding construction in an architecture of two layers. Each layer outputs a representation of size C for each vertex $u \in \mathcal{V}_g$. After a concatenation step, vertex representations (of size $2C$) are aggregated to form the required graph embedding. The permutation invariant function over multisets, which we use to perform the aggregation, influences the final result.

graph representations, we do not incorporate adaptive aggregation functions as in Zaheer et al. (2017) and Xu et al. (2019).

So far, we have described how we construct a deep architecture to create vertex and graph embeddings. We now turn our attention to the core module of CGMM that enables fast probabilistic reasoning on vertices.

3.3. Probabilistic Layer

The goal of each CGMM's probabilistic layer is to learn a mapping from a vertex to a fixed-size representation, conditioned on the graph structure. To this end, we develop a Bayesian Network that models the generation of a vertex's features while dealing with a variable number of neighbours.

3.3.1. Model Definition

The network of a generic layer ℓ is shown in Figure 4. We follow the convention that a shaded vertex corresponds to an observable random variable, whereas a white vertex is associated with an unobserved random variable. A latent factor Q_u with C attainable values is responsible for the generation of vertex u's features x_u via an emission distribution $P(x_u|Q_u)$. Here, we assume that all vertices are *i.i.d* when the value of Q_u is observable, which works well because structural dependencies are encoded in Q_u through layering (as discussed in Section 3.2.1). To take into account u's neighbourhood $\mathcal{N}(u)$, we condition the posterior distribution of the variable Q_u on the set of vertex *states* $\mathbf{q}$ inferred at previous layers. The *state* of a vertex u is defined as the posterior distribution of its latent factor Q_u, which has the form of a vector of size C. Specifically, we identify with q_v the observable state of vertex v that has been inferred at layer $\ell - 1$, with j-th component $q_v(j)$. We then formally define the probability of a graph at layer ℓ as

$$P(\mathbf{g}) = \prod_{u \in \mathcal{V}_g} \sum_{i=1}^{C} P(x_u|Q_u = i) P(Q_u = i|\mathbf{q}^{\ell-1}_{\mathcal{N}(u)}). \tag{1}$$

By means of marginalization, we have introduced the latent variable Q_u; furthermore, $\mathbf{q}^{\ell-1}_{\mathcal{N}(u)}$ denotes the set of observable states associated to u's neighbourhood. Importantly, not knowing the size of $\mathcal{N}(u)$ makes the definition of the posterior distribution of a vertex challenging

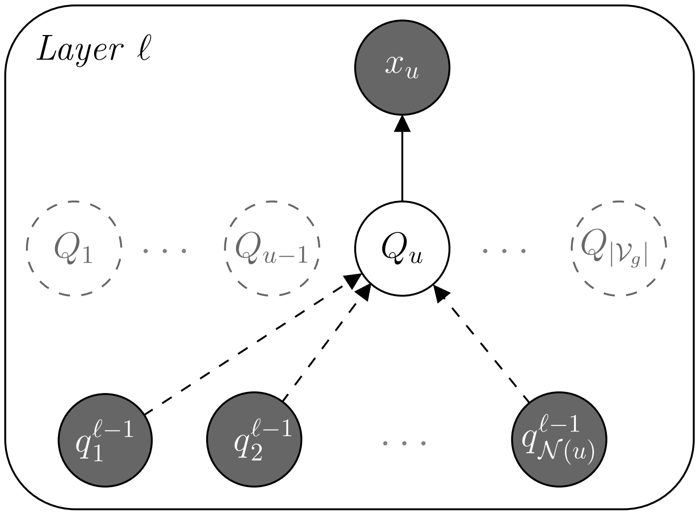


Figure 4: The Bayesian Network that implements layer ℓ of our model. Shaded and white vertices denote observable and unobserved random variables, respectively. The generation of a vertex features x_u is conditioned on a latent factor Q_u, which in turn depends on the contextual information coming from previous layers (dashed arrows). Depending on the form of x_u, we can use a multivariate Gaussian for continuous attributes or a categorical distribution for discrete ones. Finally, dashed vertices correspond to the latent factors of other vertices that depend on different sets of observable neighbours.

to formalize, and the joint distribution of all neighbours quickly becomes intractable (due to the exponential growth in the number of parameters needed). Notice that the cardinality of $\mathbf{q}_{\mathcal{N}(u)}^{\ell-1}$ may vary, and if we do not assume any vertex ordering we can weigh contributions of the states equally:

$$P(Q_u = i|\mathbf{q}_{\mathcal{N}(u)}^{\ell-1}) \approx \frac{1}{|\mathcal{N}(u)|} \sum_{j}^{C} P(Q = i|q = j) \sum_{v \in \mathcal{N}(u)} q_v^{\ell-1}(j), \tag{2}$$

i.e., we take the average of the elements in $\mathbf{q}_{\mathcal{N}(u)}^{\ell-1}$. We want to stress that this may not be the only possible choice, but it returns a correct probability value. With Equation 2, we extend the formalization used in Bacciu et al. (2018a) to handle observable states defined in terms of their posterior distribution. To recover the discrete formulation, it suffices to represent a state $q_v \in \{1, \dots, C\}$ as a one-hot vector where all the probability mass is placed on the i-th entry of maximum value. Nonetheless, the extension leads to a different form of the learning equations. Overall, the parameters of our model are the emission distribution $P(x|Q)$ and the transition distribution $P(Q|q)$.

Before we move to the training and inference procedures, we have to make important remarks. The probabilistic model we have just described is an extension of the Hidden Tree

Markov Model (HTMM) of Bacciu et al. (2012) to the processing of cyclic dependencies. Because a Bayesian Network is defined for acyclic dependencies only, we cannot connect random variables according to the graph structure, as done in HTMMs for trees. Indeed, if two latent variables Q_u and Q_v had mutual dependencies due to an explicit cycle or undirected edges, we would incur in a recursive and impractical definition of the latent states. Therefore, we must rely on a strategy that relaxes the causality assumptions between variables of the same layer. As discussed in Figure 4, we achieve this by considering a frozen representation of the neighbouring states of the latent Q_u, thus approximating the effect of mutual dependencies with what has been computed at previous layers and avoiding the need for recursive definitions of the state space. Not only is the incremental construction functional to the spreading of context between the vertices (Figure 1), but it is *necessary* to implement the Bayesian Network and to enhance efficiency. In this context, an "end-to-end" training of many layers is impractical because it would require not to freeze the previous latent variables. In turn, this amounts to consider exponentially large combinations of hidden states that depend on the structure of the graph, thus hampering the creation of deep architectures.

Secondly, a stationary assumption on all parameters is used to cope with graphs of varying sizes, i.e., we learn the same distributions for all vertices. This means that stationarity allows the model to generalize to unseen instances rather than assuming a maximum graph size in the true data distribution. To cope with varying neighbourhood dimensions, we aggregate neighbours as done in other works (Micheli, 2009; Hamilton et al., 2017). This strategy does not require any kind of neighbourhood sampling scheme, nor it imposes limitations on the neighbourhood size.

3.3.2. Training

We define learning as a maximum likelihood estimation problem. Given the graphical model in Figure 4, the likelihood of a data set $\mathcal{G}$ of i.i.d graphs at level ℓ is

$$\mathcal{L}(\theta|\mathcal{G}) = \prod_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i=1}^{C} P(x_u|Q_u=i)P(Q_u=i|\mathbf{q}^{\ell-1}_{\mathcal{N}(u)}), \tag{3}$$

which we now know how to compute from Section 3.3.1. Equation 3 is similar to a mixture model with a *variable* number of additional "inputs"; when $\mathbf{q}$ is not available, as is the case for the first layer, we recover the usual mixture model (Bishop, 2006). Likelihood can be thus maximized with the Expectation Maximization (Moon, 1996) algorithm; to this aim we define the indicator variable z_{uij}, which is one when vertex u is in state i and its neighbours are in state j.

During the E-Step, we need to compute the posterior of the indicator variables, which can be shown to be equivalent to

$$\begin{aligned} E[z_{uij}|\mathcal{G},\mathbf{q}] &= P(Q_u=i,q=j|\mathcal{G},\mathbf{q}) \\ &= \frac{1}{Z}P(x_u|Q_u=i)P(Q_u=i|q=j)\frac{\sum_{v\in\mathcal{N}(u)} q^{\ell-1}_v(j)}{|\mathcal{N}(u)|}, \end{aligned} \tag{4}$$

where Z is a simple normalization term. Such posterior probabilities are then used to update the model parameters at the M-Step:

$$P(Q=i|q=j) = \frac{1}{Z}\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} E[z_{uij}|\mathcal{G},\mathbf{q}].$$

Instead, the update rule for a categorical emission with K attainable values is

$$P(y=k|Q=i) = \frac{1}{Z}\sum_{\substack{\mathbf{g}\in\mathcal{G}\\ u\in\mathcal{V}_g}} \delta(x_u,k)E[z_{ui}|\mathcal{G},\mathbf{q}],$$

where $\delta(\cdot,\cdot)$ is the Kronecker delta, and $E[z_{ui}|\mathcal{G},\mathbf{q}]$ is obtained by straightforward marginalization of the posterior term computed at the E-step. When dealing with a continuous label, one could use a univariate Gaussian distribution, and the corresponding M-step equations are

$$\mu_i = \frac{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} x_u E[z_{ui}|\mathcal{G},\mathbf{q}]}{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} E[z_{ui}|\mathcal{G},\mathbf{q}]},$$

$$\sigma_i = \sqrt{\frac{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} E[z_{ui}|\mathcal{G},\mathbf{q}](x_u-\mu_i^{old})^2}{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} E[z_{ui}|\mathcal{G},\mathbf{q}]}}.$$

3.3.3. Inference

The inference phase takes the most likely *index* associated to the posterior of Q_u as representative for vertex u. In other words, it assigns u to one of the C potential clusters. Formally, this can be expressed as

$$\max_i P(Q_u=i|g,\mathbf{q}^{\ell-1}_{\mathcal{N}(u)}) = \max_i \frac{P(x_u|Q=i)P(Q=i|\mathbf{q}^{\ell-1}_{\mathcal{N}(u)})}{\cancel{P(x_u|\mathbf{q}^{\ell-1}_{\mathcal{N}(u)})}}. \tag{5}$$

The equivalence is obtained by straightforward application of the Bayes Theorem; moreover, the denominator does not contribute to the maximization because it is independent of i, so it can be dropped. To compute the numerator, we apply again Equation 2.

3.3.4. Different Vertex Representations

As noted previously, the inference phase returns the most likely *index* of Q_u. Hence, we convert this discrete value into a one-hot vector of C entries, as discussed above. This way, the graph fingerprint is constructed as a bag-of-states encoding that counts how many vertices are in a particular cluster at each layer.

However, consider the case of the posterior distributions of two vertices, with $C=3$, being $(0.4,0,0.6)$ and $(0,0.4,0.6)$; picking the most likely cluster discards important information about the probability of being in the others. This is why we may want to use the continuous vector of posterior probabilities. The trade-off is between a less noisy but approximate representation and a possibly noisy but exact one; we treat this choice as a hyper-parameter. In both cases, we call such C-sized representation a *unigram*.

In addition, we can augment the vertex representation with a *bigram*, i.e., a C^2-sized vector which reflects how neighbours of vertices are distributed. Formally, the bigram $\Phi(u)$ (Bacciu et al., 2018b) of a vertex u is defined as

$$\Phi_{i_j}(u) = \sum_{v \in \mathcal{N}(u)} q_u(i) q_v(j), \quad i, j \in 1, \dots, C.$$

In the remainder of this paper, whenever a bigram is used, we concatenate it with its corresponding unigram, and the resulting vector is called *unibigram*. Unigrams are clearly less expensive to compute; in turn, unibigrams constitute a richer vertex representation.

3.3.5. Extending CGMM to Incorporate Contributions from Multiple Previous Layers

So far, we have assumed that only the states computed at the previous layer contribute to the maximization of Equation 1. However, it is possible to modify the CGMM layer to consider contributions from an arbitrary subset $\mathbb{L}(\ell)$ of previous layers. To this aim, we introduce a random variable L_u, also called Switching Parent (SP) variable (Saul and Jordan, 1999; Bacciu et al., 2013). Formally, an SP's role is to decompose a complex distribution into a convex combination of simpler ones:

$$P(i|i_1, i_2, ..., i_k) \approx \sum_{\mu=1}^{k} P(SP = \mu) P^{\mu}(i|i_\mu).$$

In this work, the role of L_u is to weigh states according to the layer they belong to. Simply put, this variable approximates the original posterior distribution by "grouping" the previous states $\mathbf{q}$ by their layer. Similarly to other neural networks for graphs (Micheli, 2009; Kipf and Welling, 2017), the use of L_u can be seen as a form of weighted skip connections over previous layers.

The finite cardinality of the set $\mathbb{L}(\ell)$ makes it possible to apply the SP approximation to our problem. We therefore use the random variable L_u to rewrite Equation 1 as

$$P(\mathbf{g}) \approx \prod_{u \in \mathcal{V}_g} \sum_{i=1}^{C} P(x_u | Q_u = i) \sum_{\ell' \in \mathbb{L}(\ell)} P(L_u = \ell') P^{\ell'}(Q_u = i | \mathbf{q}^{\ell'}_{\mathcal{N}(u)}). \tag{6}$$

Here, $\mathbf{q}^{\ell'}_{\mathcal{N}(u)}$ is the subset of u's neighbouring states that are associated to layer ℓ'. Notice that this definition requires additional transition distributions $P^{\ell'}(Q_u = i|\mathbf{q}^{\ell'}_{\mathcal{N}(u)})\ \forall \ell' \in \mathbb{L}(\ell)$. From a learning point of view, the introduction of a SP variable only requires the addition of a new indicator variable in the posterior estimate, i.e., $E[z_{ui\ell'j}|\mathcal{G}, \mathbf{q}] = P(Q_u = i, L_u = \ell', q = j|\mathcal{G}, \mathbf{q})$. The interested reader can find the complete derivations in Appendix A, where we also show how to deal with discrete edge features using the same technique.

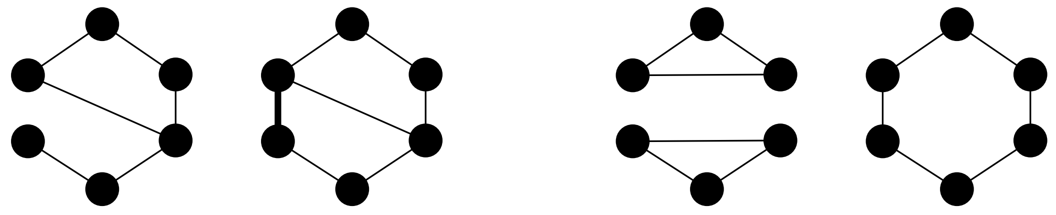


Figure 5: From left to right: two non-regular graphs that can be discriminated by adding degree information to each vertex; two *regular* graphs that cannot be discriminated by the 1-dimensional Weisfeiler-Lehman test of graphs isomorphism.

Importantly, the SP approximation fits inside our framework because it does not require reasoning about all layers at the same time. Different decompositions, e.g., conditioning the state of a vertex on the "history" of previous posteriors (like RNNs or Hidden Markov Models), assume that states are not frozen and can change altogether through a *shared* transition function. While more flexible than an SP approximation, it is not possible to apply any of these to CGMM, as it would reintroduce mutual dependencies between unobserved variables (that we have broken thanks to the incremental construction). This technique could be helpful in some tasks, and it provides a principled way to consider skip connections in a probabilistic framework. In Section 5, we study how considering all previous layers affects performances.

3.3.6. Limitations

As we have seen, the probabilistic nature of the model allows it to learn the distribution of each vertex's neighbourhood. However, care must be taken when discriminating between structures with different connectivity and the same local distributions. For example, the two leftmost graphs in Figure 5 have neither vertex nor edge features, and they share the same structure but for one edge. From a probabilistic point of view, the distribution of the neighbourhood's states is always the same, regardless of a vertex's degree, which makes learning difficult. We can mitigate this issue by introducing the notion of vertex degree via a special state $\perp$ called *bottom*. The idea is to connect each vertex to several "ghost" vertices with a *special edge type* and apply an SP variable for discrete edges that weighs the importance of "bottom" states (see Appendix A). Thus, by denoting with $deg(g)$ the maximum degree of a graph g, we can connect vertex u to $deg(g) - deg(u)$ ghost vertices in bottom state. Moreover, we can also encode vertex u's degree into x_u and use a Gaussian emission distribution.

Conversely, there are classes of graphs that cannot be distinguished so easily, such as *k-regular graphs*, that is those such that $deg(u) = k \ \forall u \in g$. In particular, this is the family of structures that can be discriminated by the k-dim Weisfeiler-Lehman (WL) test of Graph Isomorphism (Douglas, 2011). Recently, Xu et al. (2019) formally proved that almost all convolutional DNGNs are at most as powerful as the 1-dim WL test. We provide an example of two regular graphs on the right-hand side of Figure 5. To be able to discriminate these

Algorithm 1: Probabilistic Incremental Training

```
Input: Data set $\mathcal{G}$, maximum number of layers $\ell_{max}$ and epochs $\text{epoch}_{max}$.
Output: A data set of node graph representations
1  for $\ell \leftarrow 1$ to $\ell_{max}$ do
2      Initialize layer $\ell$ according to $\mathbb{L}(\ell)$ and $C$;
3      Load $\mathbf{q}_{\mathcal{G}}^{\ell'} \;\; \forall \ell' \in \mathbb{L}(\ell)$;
4      for $epoch \leftarrow 1$ to $epoch_{max}$ do
5          $\Delta_{likelihood}$, $posteriors \leftarrow$ E-Step($\mathcal{G}, \mathbf{q}_{\mathcal{G}}^{\mathbb{L}(\ell)}$) (Equation 4);
6          M-Step($posteriors$) (update equations);
7          if $\Delta_{likelihood} < threshold$ then
8              break;
9          end
10     end
11     $\mathbf{q}_{\mathcal{G}}^{\ell} \leftarrow$ Inference($\mathcal{G}$) (Equation 5);
12     Store $\mathbf{q}_{\mathcal{G}}^{\ell}$;
13 end
14 $R_{\mathcal{V}_{\mathcal{G}}} \leftarrow$ concatenate($\mathbf{q}_{\mathcal{G}}^{\ell} \;\; \forall \ell \in \{1, \ldots, \ell_{max}\}$);
15 $R_{\mathcal{G}} \leftarrow$ aggregate($R_{\mathcal{V}_{\mathcal{G}}}$);
16 return $R_{\mathcal{V}_{\mathcal{G}}}, R_{\mathcal{G}}$;
```

graphs, Morris et al. (2019) proposed a DNGN that can be theoretically as powerful as the k-dim WL test, with a computational cost that, however, grows exponentially with k.

3.4. Complexity and Scalability

Thanks to CGMM's architectural flexibility, training cost of each layer ℓ can range from constant ($|\mathbb{L}(\ell)| = 1$) to depth-specific ($|\mathbb{L}(\ell)| = \ell - 1$). Time and space complexity of a training epoch can be bounded by the cost of learning the transition and emission distributions, which is $\mathcal{O}(|\mathcal{V}_{\mathcal{G}}|(|\mathbb{L}(\ell)||C^2| + KC))$, where $\mathcal{V}_{\mathcal{G}}$ stands for the set of vertices in the whole data set. The computation of $\mathbf{q}$, on the other hand, has time complexity $\mathcal{O}(|\mathcal{E}_{\mathcal{G}}|)$ because it only needs access to the structure. The overall time computation is therefore bounded by the sum of these two asymptotic terms that can be written as $\mathcal{O}(|\mathcal{V}_{\mathcal{G}}| + |\mathcal{E}_{\mathcal{G}}|)$.

Similarly to DNGNs, the *i.i.d* assumption on vertices given the latent factors makes it easy to implement mini-batch training, with which we can arbitrarily reduce the memory fingerprint; this is especially important in hardware-constrained scenarios. Also, data parallelism could be trivially achieved by distributing the epoch's mini-batches on different computing vertices, such as CPUs or a cluster of machines. For these reasons, CGMM is a suitable candidate to handle large-scale graph learning. Up to now, our implementation[1] does not exploit intra-epoch parallelization: each training epoch was just fast enough for our purposes.

1. The implementation is available here: `https://github.com/diningphil/CGMM`.

Data set	Graphs	Task	Vertices	Edges	Vertex Feat.	Classes
D&D	1178	Graph Class.	284.32	715.66	89	2
NCI1	4110	Graph Class.	29.87	32.30	37	2
PROTEINS	1113	Graph Class.	39.06	72.82	3	2
IMDB-B	1000	Graph Class.	19.77	96.53	1	2
IMDB-M	1500	Graph Class.	13.0	65.94	1	3
COLLAB	5000	Graph Class.	74.49	2457.78	1	3
PPI	24	Vertex Class.	2372.67	34113.17	50	121

Table 1: Data sets' statistics. Note that PPI is a multilabel classification problem.

To summarize this Section, we provide a pseudo-code of the incremental training procedure in Algorithm 1, showing how the ideas described so far are put together to construct vertex (line 14) and graph (line 15) representations.

4. Experiments

The Section illustrates the experimental setting. First of all, we present the data sets used to assess CGMM's efficacy. Then, in order to foster reproducibility, the set of hyper-parameters and the risk assessment procedure are thoroughly described.

4.1. Data Sets

We extend the comparison in Bacciu et al. (2018a) with six common graph classification benchmarks (Kersting et al., 2020) and one vertex classification task. As regards graph classification, three data sets come from the chemical domain, while the others are social network data sets. D&D (Dobson and Doig, 2003) is a data set of proteins in which we have to distinguish between enzymes and non-enzymes; this data set is computationally challenging for kernels because of the high connectivity and size of each graph. Similarly, in PROTEINS (Borgwardt et al., 2005), vertices are connected if they are neighbours in the amino-acid sequence or 3D space. The last chemical data set, NCI1 (Wale et al., 2008), is the biggest in terms of the number of graphs. It contains two balanced subsets of chemical compounds that need to be classified according to their ability to suppress the growth of a panel of human tumour cell lines. Instead, IMDB-BINARY and IMDB-MULTI (Yanardag and Vishwanathan, 2015) are movie-collaboration data sets where vertices represent actors/actresses, and there is an edge every time two of them appeared in the same movie; the task is to classify the associated movie genre. Finally, COLLAB (Leskovec et al., 2005) is a scientific-collaboration benchmark derived from 3 other data sets, and the task is to classify the specific research field each community belongs to. To test CGMM performance on vertex classification tasks, we use the protein-protein interaction data set (PPI) introduced in Hamilton et al. (2017). In this task, we are given a set of distinct large graphs, and our goal is to classify their vertices.

We provide a summary of the data sets statistics in Table 1. Notice that all collaboration data sets have no information except their structure. The absence of vertex features is a degenerate case that prevents a probabilistic model from learning any vertex distribution.

As done in Niepert et al. (2016), when a vertex has no features, we add its degree as a continuous value as well as the bottom states discussed in Section 3.3.6.

4.2. Hyper-parameters

We now list the hyper-parameters needed by CGMM and by the supervised classifier working on graph and vertex representations. We achieve constant per-layer complexity by setting $|\mathbb{L}| = 1$, but we also evaluate the impact of all previous layers at the cost of additional computation. The number of EM epochs is fixed to 10: the reason is that likelihood always stabilizes around that value in our experiments.

Moreover, we tried to use both continuous and discrete vertex representations, as described in Section 3.3.4. On the contrary, the aggregation function (sum or mean) was usually fixed after an initial experimental screening, which helped to reduce the grid dimension. Overall, the different configurations of CGMM's architecture depended on *two* hyper-parameters only: the number of hidden states C and the number of layers. Furthermore, we can exploit the incremental nature of our model to reduce the dimension of the grid search space: if the set of depth hyper-parameters has maximum value N, we train a single network of N layers and "cut" it to obtain different architectures of depth $M < N$.

Contrarily to our previous work (Bacciu et al., 2018a), we did not use the "pooling" strategy of Fahlman and Lebiere (1990): in short, it consists of training a pool of candidate Bayesian Networks at each layer and selecting the best one according to some metric, e.g., validation performances. This choice is motivated by empirical studies that will be presented in Section 5.4. The training time improvement factor is equal to the number of candidates that were used in Bacciu et al. (2018a), i.e., *10×*.

Another point to consider is whether the learned representations are sufficient to linearly separate classes; to this end, we introduced a logistic regressor in the grid search. However, we add a flexible alternative in the form of a one-hidden-layer MLP with ReLU activations. Both of them are trained with Adam (Kingma and Ba, 2015) optimizer and Cross-Entropy loss (Mean Squared Error for PPI); to contrast overfitting, we introduced L2 regularization. Also, we used the early stopping technique called Generalization Loss (Prechelt, 1998) with $\alpha = 5$, which is considered to be a good compromise between training time and performances. In Table 5, we report the number of epochs after which early stopping starts; at the beginning of training, validation loss smoothly oscillates and accuracy does not steadily increase, so it would be unwise to apply this technique immediately. For space reasons, we list the hyper-parameters table in Appendix B.

4.3. Risk Assessment

To assess CGMM's classification performance, we used a *Double Cross-Validation* with 10 external folds and 5 internal folds. The internal folds are needed for grid-search model selection, which returns a *biased* estimate (optimization is done on validation folds) of the best hyper-parameters for each external split. Then, these configurations are retrained on the corresponding external training fold (using a validation set to do early stopping) and evaluated on the external *test* fold. We repeat each final retraining three times and average results in order to mitigate possible bad random initializations. The output of this procedure is an estimate of CGMM's performance. Importantly, the test fold was *never*

Kernel	Cost	Reference
GK	$\mathcal{O}(\lvert\mathcal{G}\rvert^2 n d^{k-1})$	Shervashidze et al. (2009)
RW	$\mathcal{O}(\lvert\mathcal{G}\rvert^2 n^3)$	Vishwanathan et al. (2010)
PK	$\mathcal{O}(m(h-1) + h\lvert\mathcal{G}\rvert^2 n)$	Neumann et al. (2012)
WL	$\mathcal{O}(\lvert\mathcal{G}\rvert hm + \lvert\mathcal{G}\rvert^2 hn)$	Shervashidze et al. (2011)
CGMM	$\mathcal{O}(\lvert\mathcal{G}\rvert n + \lvert\mathcal{G}\rvert m)$	

Table 2: Computational costs of graph kernels compared to CGMM. We assume that all graphs have size $n = |\mathcal{V}_g|$, $m = |\mathcal{E}_g|$ edges and maximum degree d. Moreover, k is the size of the graphlets that GK counts, and h is the number of iterations needed by different procedures to compute the final similarity scores.

seen throughout the entire process; unfortunately, we have observed this is not always the case in DNGNs' literature, where *model selection's best results*, i.e., those of a k-fold cross-validation, are reported. This trend is also criticized in Klicpera et al. (2019). Consequently, we compare with results from Tran et al. (2018), where the authors performed an external 10-fold cross-validation with a simple hold-out internal model selection procedure. Their process is repeated ten times with different splits, and results are averaged. This approach, which is similar to bootstrapping, is less subject to unlucky splits with consequent variance reduction; on the other hand, it might be more biased than a k-fold cross validation (Kohavi, 1995), which is why we use the latter. Despite these differences, both evaluations are robust and comparable estimates of a model's performance. A train/validation/test split was already provided for PPI, so we used a standard hold-out technique for this task.

5. Results and Discussion

In the following, we present CGMM's results on graph and vertex classification, along with empirical studies on the beneficial effects of depth. Indeed, depth gives hints on CGMM's ability to extract useful information, as well as how much structural features are essential for the task at hand. We support our reasoning via stability results, which led us to a relevant insight into some of the data sets used. Then, we perform hyper-parameters' analyses to study their impact on the final performance. Finally, we visualize how the model moves information among vertices and what is the average contribution of an input state to the new output representation.

5.1. Graph Classification

We evaluate the performance of our method against different kernels and deep learning techniques for graph classification. Table 2 provides a comparison between the kernels considered in terms of computational costs. As we can see, kernels are not adequate when it comes to large scale training and inference because of their (at least) quadratic time complexity in the number of graphs. Moreover, the considered kernel for graphs are not applicable to continuous vertex features, which limits their scope. Apart from these and some of the DNGNs introduced in Section 2, we will consider the Deep Graphlet Kernel

	D&D	**NCI1**	**PROTEINS**	**IMDB-B**	**IMDB-M**	**COLLAB**
GK	74.38 ± 0.7	62.49 ± 0.3	71.39 ± 0.3	-	-	-
RW	> 3 days	> 3 days	59.57 ± 0.2	-	-	-
PK	78.25 ± 0.5	82.54 ± 0.5	73.68 ± 0.7	-	-	-
WL	78.34 ± 0.6	$\mathbf{84.46} \pm 0.5$	74.68 ± 0.5	-	-	-
ARMA	74.86	-	75.12	-	-	-
PSCN	76.27 ± 2.6	76.34 ± 1.7	75.00 ± 2.5	71.00 ± 2.3	45.23 ± 2.8	72.60 ± 2.15
DCNN	58.09 ± 0.5	56.61 ± 1.0	61.29 ± 1.6	49.06 ± 1.4	33.49 ± 1.4	52.11 ± 0.7
ECC	72.54	76.82	-	-	-	-
DGK	-	62.48 ± 0.3	71.68 ± 0.5	66.96 ± 0.6	44.55 ± 0.5	73.09 ± 0.3
DGCNN	$\mathbf{79.37} \pm 0.9$	74.44 ± 0.5	75.54 ± 0.94	70.03 ± 0.86	47.83 ± 0.9	73.76 ± 0.5
PGC-DGCNN	78.93 ± 0.9	76.13 ± 0.7	$\mathbf{76.45} \pm 1.02$	71.62 ± 1.2	47.25 ± 1.4	75.00 ± 0.58
CGMM-nb	77.35 ± 1.6	77.02 ± 1.8	75.11 ± 2.8	71.07 ± 3.5	47.36 ± 3.4	73.3 ± 2.9
CGMM-full	77.20 ± 3.1	76.94 ± 1.6	75.45 ± 4.4	$\mathbf{72.30} \pm 3.5$	49.42 ± 3.6	$\mathbf{76.06} \pm 2.4$
CGMM	77.15 ± 3.5	$\mathbf{77.80} \pm 1.9$	75.56 ± 3.0	72.1 ± 2.3	$\mathbf{49.73} \pm 1.6$	75.50 ± 2.74

Table 3: CGMM's results of a 10-Fold Double Cross Validation for graph classification. Best results are reported in bold. We report CGMM's accuracy on NCI1 in bold because it performs better than the other neural models. CGMM-nb indicates that the model is not using bigram features, whereas CGMM-full represents the extended CGMM using all previous layers rather than just the previous one.

(DGK) framework (Yanardag and Vishwanathan, 2015) and the Diffusion Convolutional Neural Network (DCNN) (Atwood and Towsley, 2016), which is a precursor of GCN.

Results for graph classification are shown in Table 3. CGMM performs well in all data sets (scoring top-3 on five of them), even though the probabilistic architecture was not optimized to solve a classification task. In particular, we achieve state of the art results on all three collaborative data sets, and we improve the best result on NCI1 compared to all supervised DNGNs. This suggests that learning the distribution of a vertex's neighbourhood at different abstraction's levels produces a good representation. As a matter of fact, in 9 out of 10 external folds on NCI1, the model selection procedure chose a configuration with 20 layers; in contrast, the DNGNs of Table 3 exploit a maximum of 4 layers only, possibly interleaved by batch normalization (Ioffe and Szegedy, 2015) to accelerate training. Furthermore, results also highlight that CGMM performs well even when the only source of information is structural.

Note that kernels can process and compare graphs more explicitly than vertex-centric models: one of the reasons why the WL kernel has higher accuracy on NCI1 may be due to the kind of structural patterns it uses to compute the similarity score. However, the number of scores to compute may quickly become impractical, as we can see from Table 2. Please notice the difference between context-expansion via neighbourhood aggregation and the standard kernel approach: the former provides higher-level representations of a graph, while the latter looks for similar substructures between each pair of vertices or graphs.

Some chemical benchmarks deserve a more in-depth analysis. Because the accuracy on D&D and PROTEINS seems comparable for both kernels and vertex-centric models, we decided to study the effect of context on these data sets. Section 5.4 will show that we can trivially achieve state of the art performances on PROTEINS and D&D. This raises the question of whether or not these benchmarks should be considered for future evaluations of deep learning methods for graphs in that the structure does not seem to be that relevant for the task.

5.2. Vertex Classification

We now turn our attention to vertex classification. Because CGMM and most of the other DNGNs in literature produce vertex embeddings, it is reasonable to assess the model performance on PPI, a common and robust benchmark for vertex classification. To this aim, we compare against GraphSAGE (Hamilton et al., 2017) and DGI (Velickovic et al., 2019), as well as a structure-agnostic baseline that applies logistic regression to the vertex features. Indeed, GraphSAGE, DGI, and CGMM all share a first pre-training step in which vertex embeddings are learned on data with no supervised target labels, before being given to a supervised classifier for vertex prediction. Results are shown in Table 4: we can see that CGMM has good performances, improving against all GraphSAGE variants except for DGI. Considering that DGI uses GraphSAGE as part of its framework, it seems that the learning procedure is what generates the gap between the two methods. While GraphSAGE relies on a link prediction loss and an entropy penalization term, DGI learns to discriminate vertices according to a contrastive noise procedure that maximizes the mutual information between the input and the target value.

	Data Used	**Micro F1**
Baseline	X	42.2
GraphSAGE-GCN	X, A	46.5
GraphSAGE-mean	X, A	48.6
GraphSAGE-LSTM	X, A	48.2
GraphSAGE-pool	X, A	50.2
DGI	X, A	**63.8**
CGMM-full	X, A	58.4
CGMM	X, A	60.2

Table 4: CGMM's results of inductive vertex classification on PPI. We report the Micro Average F1 score. Here, X refers to the input features and A to the adjacency matrix information.

5.3. Hyper-parameters Analysis

We enrich our empirical analysis with three further studies. The first concerns the impact of the unibigram technique (introduced in Section 3.3.4) on performances, whereas the second regards the potential advantages of using all previous layers when training a new layer. Finally, the last one investigates whether a wider model with fewer layers can perform as well as a deep model with fewer hidden states. With the exception of Section 5.3.2, in our analyses we will refer to the standard version of CGMM with $\mathbb{L}(\ell) = \{\ell - 1\}$.

5.3.1. Unibigram Ablation

In Table 3, we re-evaluated the model on all data sets by constraining CGMM to only use unigrams. Results indicate a slight performance drop on chemical data sets and a larger decrease in social data sets. This result suggests that, especially when we only have access to structural information (i.e., the degree distribution), computing a graph representation that takes the structure into account can be helpful. Nevertheless, CGMM performances remain good with respect to the state of the art.

5.3.2. Impact of Previous Layers

Similarly to Section 5.3.1, we have repeated all experiments by conditioning each layer of the architecture on the entire subset of previous layers, i.e., $\mathbb{L}(\ell) = \{1, \ldots, \ell-1\}$. This way, each layer is free to weigh the previous layers to maximize the likelihood of a graph. Results (Table 3 and 4, CGMM-full) indicate that the model performs almost always on par (with a difference within the standard deviation intervals) compared to the significantly more efficient version that does not use the SP variable L_u. Nonetheless, despite the negligible performance advantage obtained on these tasks, we recommend treating the use of L_u as a hyper-parameter of the model when dealing with other node or graph classification tasks.

5.3.3. Sensitivity Analysis

When designing any deep network (let it be neural or probabilistic), it is useful to analyze the relation between the dimension of each layer's hidden representation and the number of layers in terms of performance. In the case of CGMM (and more generally DNGNs), the depth of the architecture is functional to context spreading, so we would expect that having a larger hidden representation for each layer is not enough to compensate for the flow of information between vertices. To show this, we provide an example in Figure 6, where we show how validation accuracy of a logistic regressor on NCI1 varies while changing the number of hidden states C. For instance, we observe that the graph representation associated with point A of size 60 is not sufficient to achieve the performance of the representation associated with point B, which has size 45. This means that smaller representations associated with deeper networks can encode more information than those associated with shallower ones.

5.4. On the Effects of Depth

This Section is meant to answer to two further research questions. First, we want to quantify the effect of depth on the architecture when coupled with a classifier. The second is about understanding how much the model's performance is affected by a random initialization of the layers. To address both questions, we took a random train-validation-test split of NCI1 to conduct a new experiment. With its 4110 graphs, NCI1 was chosen to minimize the effect of the data split on results, and consequently on the random initialization of the classifier. In contrast, the other chemical data sets are much more split-dependent. We trained a 20-layer network for each configuration defined in Table 5. We repeated each process five times, and the results were averaged. Figure 7 reports accuracy against the number of layers for one of such configurations, with logistic regression (Figure 7a) and MLP (Figure

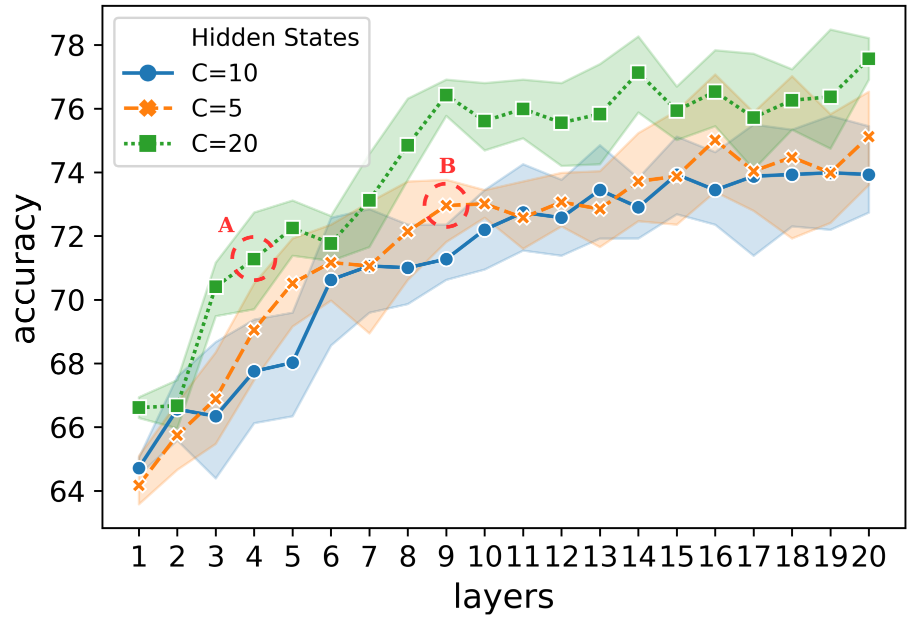
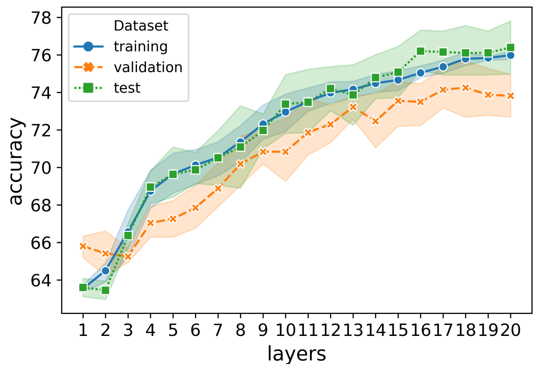


Figure 6: This picture shows that having a larger graph representation with fewer layers is not always enough to reach the same performance of a deeper network with a smaller graph representation. Points A and B are associated with representations of dimensions 60 and 45, respectively. However, we see that the latter is associated with better validation performances of a logistic regressor on NCI1, which means that the flowing of contextual information is indeed more beneficial than increasing the number of hidden states C. Results are averaged over five independent runs.

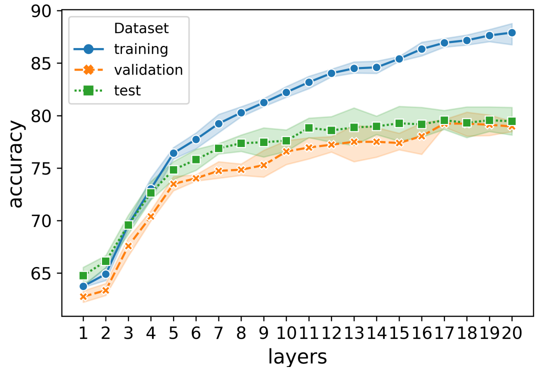
7b) classifiers. We see that, in both cases, depth has a beneficial effect on test accuracies, with slightly worse results on test accuracy from logistic regression due to its strong bias. Notice how test curves in Figure 7b tend to an asymptote after ten layers; this information may be used as stopping criterion when constructing the architecture for supervised tasks, as proposed in Marquez et al. (2018) for CNNs on images.

One interesting thing to notice is that we do not necessarily incur in the curse of dimensionality as the size of the fingerprint grows with the layers. This is because the fingerprint construction at layer ℓ is guided by layer $\ell - 1$, and this dependency may well cause the fingerprint to lie in a sub-space of the original one. This may explain why we do not quickly overfit after the first few layers.

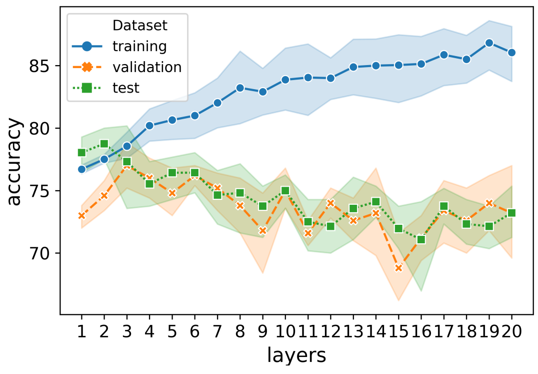
In addition to the above considerations, we note the following. Since training accuracy on NCI1 does not significantly vary between different runs (due to different weights' initialization), that is an indication that the pooling strategy of Fahlman and Lebiere (1990), mentioned in Section 4.2, brings negligible advantage in our case. This also holds for PRO-



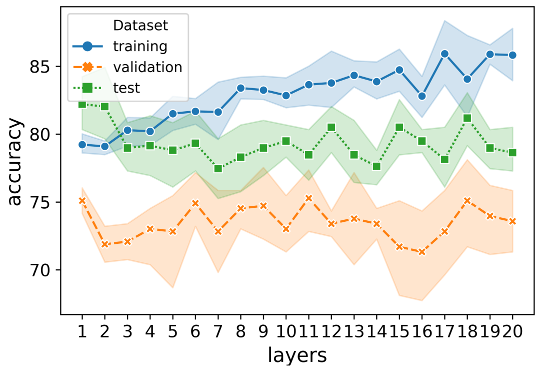
(a) NCI1, logistic regression



(b) NCI1, MLP with 128 units



(c) PROTEINS, MLP with 8 units



(d) D&D, MLP with 8 units

Figure 7: Stability experiment on NCI1, PROTEINS and D&D that shows how accuracy varies from 1 to 20 layers. Results are averaged over five independent runs. Colored bands denote standard deviation. While the effect of depth is beneficial for NCI1, stability experiment on PROTEINS and D&D report a different behavior, which is not to be attributed to random splits. Indeed, we discover that competitive results can be achieved by trivial methods on PROTEINS and D&D.

TEINS and D&D when the architecture is very shallow (up to three layers), though results need to be interpreted from a different perspective, as we are about to show.

5.4.1. A Critical Investigation of Chemical Benchmarks

The stability experiments we did on PROTEINS and D&D led to surprisingly different results. We can see from Figure 7c and 7d that accuracy on validation and test tends to decrease almost immediately, while standard deviation increases as we add layers. One may think this behaviour is caused by the way the data set was randomly split. However, repeating the experiments did not change the final result. It appears that a minimal number of layers (even 1) is *enough* to get the best results. Thus, we took inspiration from Ralaivola et al. (2005) and from the technique of *molecular fingerprint* to investigate the matter and validate our hypothesis. In order to assess the practical relevance of the structured information for each chemical task, we constructed graph fingerprints by only counting atoms of the same type. Then, a 10-fold Nested Cross-Validation with Hold-out model selection was used to assess the performance of different MLP classifiers: the average test accuracy was 76.0 ± 3.4 for PROTEINS and 77.33 ± 3.7 for D&D, entirely in line with the state of the art. Instead, this trivial approach was not enough to reach a satisfying accuracy on NCI1 (69.80 ± 3.0), meaning that being able to capture structural patterns is very important for discrimination purposes on this task.

This investigation shows how important it is to establish structure-agnostic baselines, like the one we used, to understand how much the structure plays a role in the final result. Our study on depth effects, which is rather lacking in DNGNs' literature, revealed that more care should be taken when considering benchmark results. Our beliefs further strengthen similar statements made in Shchur et al. (2018) for semi-supervised vertex classification tasks.

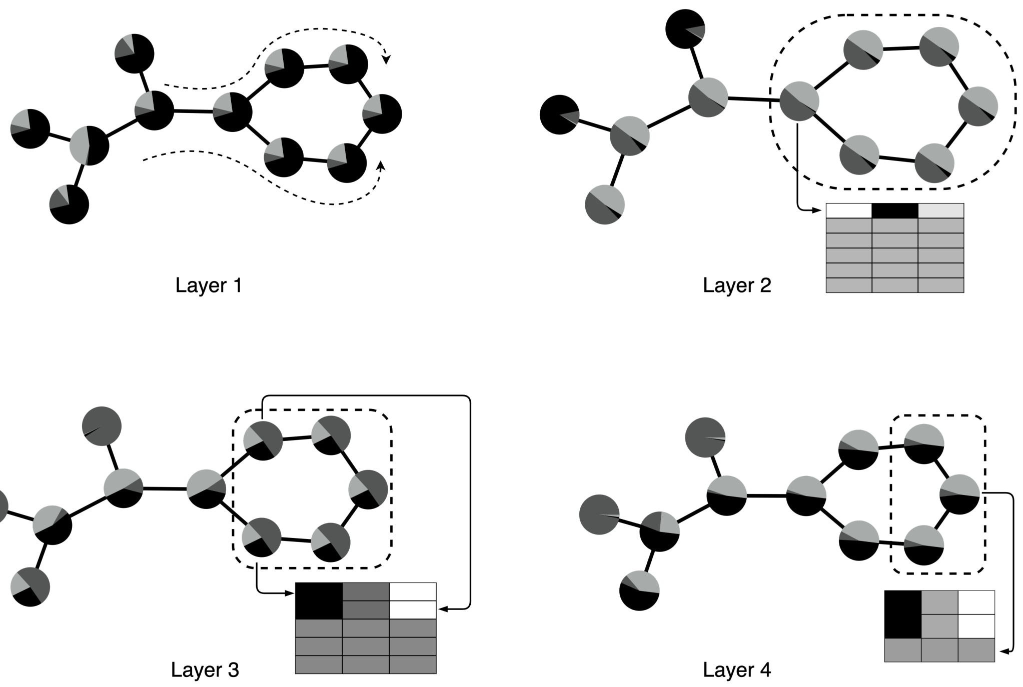


Figure 8: Context flow on a real NCI1 graph. We focus on the highly regular rightmost subgraph, which is influenced by the left part of the structure. Heatmaps show that, although relatively small, posteriors change as we expected.

5.5. Visualization: a Case Study

This part provides a visual exploration and interpretation of CGMM's internal dynamics. This kind of analysis is meant to demonstrate how the model extracts different patterns at each layer of the architecture in a way that is consistent with what stated in Section 3.2.1. Therefore, Figure 8 shows how information spreads in a real NCI1 molecule. We represent each vertex's posterior as a circle chart with C different colours; in this experiment, we use a 4-layer CGMM with $C = 3$. Please keep in mind that colours assignment between different layers is irrelevant. The rightmost six atoms of the molecule have the same atomic symbol so that they will be assigned an identical state at layer 1. What is more, these six atoms alone form a 2-regular subgraph, which means that their state can only change if context flows from left to right, as shown by the dashed arrows on the top-left side of the figure. As discussed in Section 3.3.6, if it were not for the five leftmost vertices context could not flow, because all "messages" would be identical (both qualitatively and quantitatively). This is indeed what happens: at each layer, we highlight the vertices we are interested in via dashed regions, and the associated heatmap (one state per row) proves that posteriors change as we expected. Thanks to the simplicity of Equation 2, it is easy to understand how CGMM behaves in this particular situation.

Similarly, we can look at how the model exploits the learned parameters (and the "bottom" states introduced in Section 3.3.6) through $E[z_{uij}]$: Figure 9 illustrates what CGMM infers on average. First of all, each layer has a different average "activation" pattern that adds new useful information to the final fingerprint. Moreover, we can inspect how the $\perp$ contribution, i.e., degree information, is encoded into the new states. In particular, some components of the previous states, called "$C2$" because they now include the $\perp$ symbol, have little influence on the average posterior of the random variables, as is the case for component 2 at layer 2. Instead, bottom states are always taken into consideration, possibly because they encode information about the degree of a vertex.

6. Conclusions and Future Works

We have introduced a new probabilistic perspective for scalable deep learning on graph-structured data. CGMM processes graphs by exploiting an incremental construction of the architecture to spread contextual information between vertices. Moreover, the model does not suffer from the vanishing like most typical DNGNs. This new *probabilistic* formulation allows us to efficiently and effectively cope with graph and node classification tasks, with good performances compared to state of the art models. Also, we have thoroughly analyzed the impact of depth on performance improvements. Therefore, we hope that our work may lay the foundation of Deep Bayesian Graph Networks (DBGNs), a new probabilistic family of graph processing methods.

Several research directions can further enhance this architecture for representation learning on structures. One is the automatic choice of the hyper-parameter C via Bayesian nonparametric methods, while another is the use of more general aggregation functions (Castellana and Bacciu, 2019). Other than these, it would be critical to automatically determine when to stop in the construction of the architecture with a criterion that depends on information-theoretic properties of the inferred states, e.g., the entropy, or to assess transfer learning capabilities in social or chemical domains.

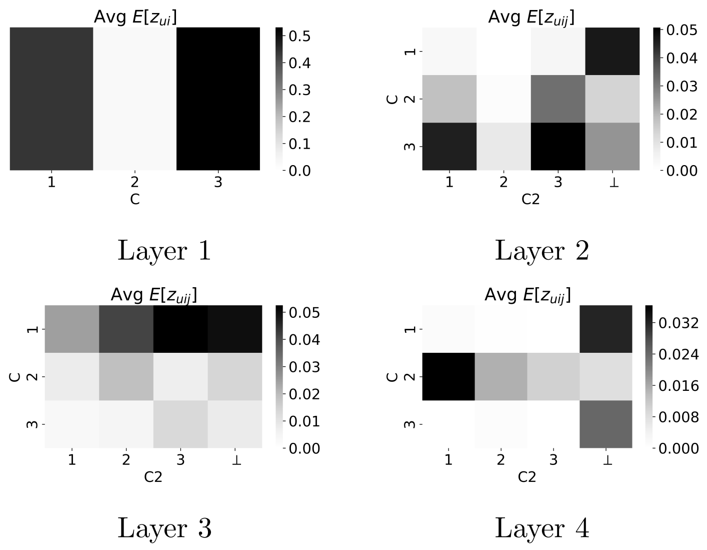


Figure 9: Unnormalized posteriors computed on NCI1 data set and averaged over vertices. By inspection, we can understand how much of a value $q(j)$ is mapped, on average, to the different components of the new state. The first layer displays averaged states.

In conclusion, the study of probabilistic models for graphs that automatize the choice of their hyper-parameters opens appealing challenges for the future development of the field.

Acknowledgments

This work has been partially supported by the Italian Ministry of Education, University, and Research (MIUR) under project SIR 2014 LIST-IT (grant n. RBSI14STDE).

Appendix A. Learning Equations

We start again from Equation 6 to derive the learning equations for a generic layer ℓ; remember that layer 0 corresponds to a simple mixture model, whose learning equations are well known. Here, however, we also deal with discrete edge types (taken from a finite alphabet $\{1, \dots, A\}$) by introducing an SP variable S_u that weights the contribution of each edge. The only additional modifications are the introduction of a different transition distribution for each edge type a, an additional indicator variable, and the definition of $\mathbf{q}^{\ell',a}_{\mathcal{N}(u)}$ as the set of neighbouring posteriors of u that are connected to u via edge a and were computed at layer ℓ'. The resulting model is a more general version than that of Bacciu et al. (2018a), in which continuous rather than discrete vertex posteriors are considered. Finally, note that time and space complexities are now bounded by $\mathcal{O}(|V_{\mathcal{G}}|(|\mathbb{L}|A|C^2|+KC))$. The extended graphical model is shown in Figure 10.

With the addition of the SP variable S_u, the likelihood of the model becomes

$$
\begin{aligned}
\mathcal{L} &= \prod_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} P(x_u|Q_u=i) \sum_{\ell'\in\mathbb{L}(\ell)} P(L_u=\ell') P^{\ell'}(Q_u=i|\mathbf{q}^{\ell'}_{\mathcal{N}(u)}) \\
&= \prod_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} P(x_u|Q_u=i) \sum_{\ell'\in\mathbb{L}(\ell)} P(L_u=\ell') \sum_{a=1}^{A} P^{\ell'}(S_u=a) P^{\ell',a}(Q_u=i|\mathbf{q}^{\ell',a}_{\mathcal{N}(u)}) \\
&= \prod_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} P(x_u|Q_u=i) \sum_{\ell'\in\mathbb{L}(\ell)} P(L_u=\ell') \sum_{a=1}^{A} \Big(P^{\ell'}(S_u=a) \times \\
&\qquad\qquad \times \sum_{j}^{C} \frac{P^{\ell',a}(Q=i|q=j) \sum_{v\in\mathcal{N}^a(u)} q^{\ell'}_v(j)}{|\mathcal{N}^a(u)|} \Big),
\end{aligned}
$$

where $\mathcal{N}^a(u)$ is the set of u's neighbours connected via an edge of type a. We dropped the ℓ' subscript from $|\mathcal{N}^a(u)|$ because it is independent from the specific layer. In pratice, S_u splits $\mathbf{q}^{\ell'}_{\mathcal{N}_u}$ into multiple groups of vertices depending on the type of edge they are connected to u. Notice that we have introduced all the random variables via marginalization. By means of the indicator variables introduced in Section 3.3, we can write the *complete* log-likelihood

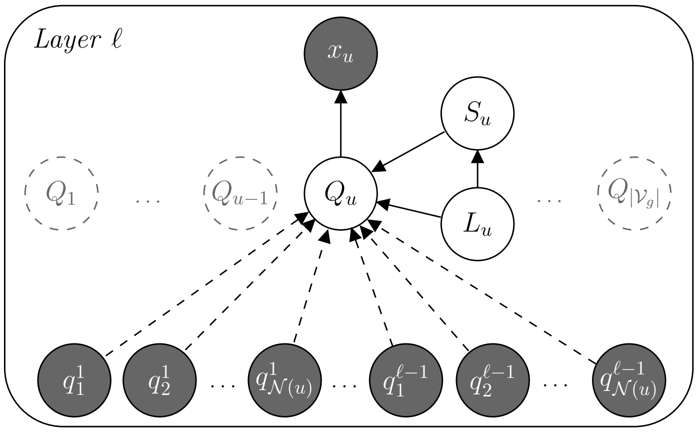


Figure 10: The extended Bayesian Network implementing layer ℓ of our model. The SP variables L_u and S_u are used to weigh the contribution of previous layers and different discrete edge features.

as

$$\begin{aligned}
\log \mathcal{L}_c = \log \prod_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \prod_{i}^{C} \Bigg\{ P(x_u|Q_u=i) \prod_{\ell'\in\mathbb{L}(\ell)} \Big\{ P(L_u=\ell') \prod_{a=1}^{A} \Big\{ P^{\ell'}(S_u=a) \\
\prod_{j}^{C} \Big\{ \frac{P^{\ell',a}(Q=i|q=j)\sum_{v\in\mathcal{N}^a(u)} q_v^{\ell'}(j)}{|\mathcal{N}^a(u)|} \Big\}^{z_{ui\ell' aj}} \Big\}^{z_{ui\ell' a}} \Big\}^{z_{ui\ell'}} \Bigg\}^{z_{ui}} \\
= \sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} z_{ui} \log P(x_u|Q=i) \\
+ \sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} \sum_{\ell'\in\mathbb{L}(\ell)} z_{ui\ell} \log P(L=\ell) \\
+ \sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} \sum_{\ell'\in\mathbb{L}(\ell)} \sum_{a=1}^{A} z_{ui\ell a} \log P^{\ell'}(S=a) \\
+ \sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} \sum_{\ell'\in\mathbb{L}(\ell)} \sum_{a=1}^{A} \sum_{j}^{C} z_{ui\ell' aj} \log \frac{P^{\ell',a}(Q=i|q=j)\sum_{v\in\mathcal{N}^a(u)} q_v^{\ell'}(j)}{|\mathcal{N}^a(u)|}. \qquad (7)
\end{aligned}$$

The E-step of the EM algorithm requires to compute the expectation of the log likelihood conditioned on the data. With slight abuse of notation and when clear from the context, we use $\mathbf{q}$ in place of $\mathbf{q}_{\mathcal{N}(u)}$:

$$E_{Z|\mathbf{q},\mathcal{G}}[\log \mathcal{L}_c(\theta|\mathcal{G},\mathbf{q},Z)] = \sum_{\substack{g\in\mathcal{G}\\u\in\mathcal{V}_g}} \sum_{i}^{C} E[z_{ui}|\mathcal{G},\mathbf{q}] \log P(x_u|Q=i)$$

$$+ \sum_{\substack{g\in\mathcal{G}\\u\in\mathcal{V}_g}} \sum_{i}^{C} \sum_{\ell'\in\mathbb{L}(\ell)} E[z_{ui\ell'}|\mathcal{G},\mathbf{q}] \log P(L=\ell)$$

$$+ \sum_{\substack{g\in\mathcal{G}\\u\in\mathcal{V}_g}} \sum_{i}^{C} \sum_{\ell'\in\mathbb{L}(\ell)} \sum_{a=1}^{A} E[z_{ui\ell' a}|\mathcal{G},\mathbf{q}] \log P^{\ell'}(S=a)$$

$$+ \sum_{\substack{g\in\mathcal{G}\\u\in\mathcal{V}_g}} \sum_{i}^{C} \sum_{\ell'\in\mathbb{L}(\ell)} \sum_{a=1}^{A} \sum_{j}^{C} E[z_{ui\ell' aj}|\mathcal{G},\mathbf{q}] \log \frac{P^{\ell',a}(Q=i|q=j)\sum_{v\in\mathcal{N}^a(u)} q_v^{\ell'}(j)}{|\mathcal{N}^a(u)|}.$$

Moreover, the expectation of an indicator variable is just the probability of its associated event:

$$\begin{aligned}
E[z_{ui}|\mathcal{G},\mathbf{q}] &= P(Q_u=i|\mathcal{G},\mathbf{q})\\
E[z_{ui\ell}|\mathcal{G},\mathbf{q}] &= P(Q_u=i,L_u=\ell|\mathcal{G},\mathbf{q})\\
E[z_{ui\ell a}|\mathcal{G},\mathbf{q}] &= P(Q_u=i,L_u=\ell,S_u=a|\mathcal{G},\mathbf{q})\\
E[z_{ui\ell aj}|\mathcal{G},\mathbf{q}] &= P(Q_u=i,L_u=\ell,S_u=a,K_u=j|\mathcal{G},\mathbf{q}).
\end{aligned}$$

K_u is a categorical random variable with C possible states, such that $P^{\ell,a}(K_u=j) = \sum_{v\in\mathcal{N}^a(u)} q_v^{\ell}(j)/|\mathcal{N}^a(u)|$. Consequently, we can apply the Bayes Theorem on $E[z_{ui\ell aj}|\mathcal{G},\mathbf{q}]$, yielding

$$\begin{aligned}
&E[z_{ui\ell aj}|\mathcal{G},\mathbf{q}] = P(Q_u=i,L_u=\ell,S_u=a,K_u=j|\mathcal{G},\mathbf{q})\\
&= \frac{P(x_u|Q_u=i)P(Q_u=i,L_u=\ell,S_u=a,K_u=j|\mathbf{q})}{P(x_u|\mathbf{q})}\\
&= \frac{P(x_u|Q_u=i)P(Q_u=i|L_u=\ell,S_u=a,K_u=j,\mathbf{q})P(L_u=\ell)P^{\ell}(S_u=a)P^{\ell,a}(K_u=j)}{Z}\\
&= \frac{P(x_u|Q_u=i)P^{\ell',a}(Q_u=i|q=j)P(L_u=\ell)P^{\ell}(S_u=a)P^{\ell,a}(K_u=j)}{Z}\\
&= \frac{P(x_u|Q_u=i)P(L_u=\ell)P^{\ell}(S_u=a)P^{\ell,a}(Q=i|q=j)P^{\ell,a}(K_u=j)}{Z},
\end{aligned} \tag{8}$$

where Z is the normalization term, obtained by $P(x_u|\mathbf{q})$ via marginalization over all the latent variables (including K_u). Similarly, $E[z_{ui}]$, $E[z_{ui\ell}]$ and $E[z_{ui\ell a}]$ can be obtained by straightforward marginalization of $E[z_{ui\ell aj}]$. We now develop the M-step equations exploiting the expectations computed in the E-step. If we compute the gradient with respect of $P(L=\ell)$ and we add a Lagrange multiplier to enforce probability requirements we get

$$\begin{aligned}
&\frac{\partial E_{Z|\mathcal{G},\mathbf{q}}[\log \mathcal{L}_c(\theta|\mathcal{G},\mathbf{q},Z)] - \gamma(\sum_{\ell'\in\mathbb{L}(\ell)} P(L=\ell')-1)}{\partial P(L=\ell)} \\
&= \sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} \frac{E[z_{ui\ell}|\mathcal{G},\mathbf{q}]}{P(L=\ell)} - \gamma = 0 \\
&= \sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} \frac{E[z_{ui\ell}|\mathcal{G},\mathbf{q}]}{\gamma} = P(L=\ell). \qquad (9)
\end{aligned}$$

Deriving with respect to γ yields

$$\begin{aligned}
&\frac{\partial E_{Z|\mathcal{G},\mathbf{q}}[\log \mathcal{L}_c(\theta|\mathcal{G},\mathbf{q},Z)] - \gamma(\sum_{\ell'\in\mathbb{L}(\ell)} P(L=\ell')-1)}{\partial\gamma} \\
&= \cancel{-}\big((\sum_{\ell'\in\mathbb{L}(\ell)} P(L=\ell'))-1\big) = 0.
\end{aligned}$$

Substituting Equation 9 in the above formula yields

$$\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} E[z_{ui\ell}|\mathcal{G},\mathbf{q}] = \gamma,$$

which can be plugged into Equation 9 to obtain the update rule for $P(L=\ell)$:

$$P(L=\ell) = \frac{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i=1}^{C} E[z_{ui\ell}|\mathcal{G},\mathbf{q}]}{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i=1}^{C} \sum_{\ell'\in\mathbb{L}(\ell)} E[z_{ui\ell'}|\mathcal{G},\mathbf{q}]}. \qquad (10)$$

The derivations for the categorical emission and edge selector distribution are not shown because they are almost identical to the one above. The corresponding Lagrangian constraints are expressed as $\sum_{i'}^{C} \lambda_{i'}(\sum_{k'}^{K} P(x=k'|Q=i')-1)$ and $\sum_{\ell'\in\mathbb{L}(\ell)} \alpha_{\ell'}(\sum_{a'=1}^{A} P(S^{\ell'}=a')-1)$. Hence, we obtain the following learning rules:

$$P(x=k|Q=i) = \frac{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \delta(x_u,k) E[z_{ui}|\mathcal{G},\mathbf{q}]}{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{k'}^{K} \delta(x_u,k') E[z_{ui}|\mathcal{G},\mathbf{q}]} \qquad (11)$$

$$P^{\ell}(S=a) = \frac{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} E[z_{ui\ell a}|\mathcal{G},\mathbf{q}]}{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i}^{C} \sum_{a'=1}^{A} E[z_{ui\ell a'}|\mathcal{G},\mathbf{q}]}. \qquad (12)$$

We now show the update rules for the transition distribution, which is slightly more troublesome.

$$\frac{\partial E_{Z|\mathcal{G},\mathbf{q}}[\log \mathcal{L}_c(\theta|\mathcal{G},\mathbf{q},Z)] - \sum\limits_{\ell',a',j'} \mu_{\ell',a',j'}(\sum\limits_{i'=1}^{C} P^{\ell',a'}(Q=i'|q=j')-1)}{\partial P^{\ell,a}(Q=i|q=j)}$$

$$= \frac{\partial \sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum\limits_{i',\ell',a'} \sum\limits_{j'} E[z_{ui'\ell'a'j'}|\mathcal{G},\mathbf{q}] \log \frac{P^{\ell',a'}(Q=i'|q=j')\sum_{v\in\mathcal{N}^{a'}(u)} q_v^{\ell'}(j')}{|\mathcal{N}^{a'}(u)|}}{\partial P^{\ell,a}(Q=i|q=j)} -$$

$$\frac{\partial \sum_{\ell',a',j'} \mu_{\ell',a',j'}(\sum_{i'=1}^{C} P^{\ell',a}(Q=i'|q=j')-1)}{\partial P^{\ell',a}(Q=i|q=j)}$$

$$= \sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \frac{E[z_{ui\ell aj}|\mathcal{G},\mathbf{q}]}{P^{\ell,a}(Q=i|q=j)} - \mu_{\ell,a,j} = 0$$

$$= \sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \frac{E[z_{ui\ell aj}|\mathcal{G},\mathbf{q}]}{\mu_{\ell,a,j}} = P^{\ell,a}(Q=i|q=j).$$

Deriving with respect to the specific Lagrange multiplier $\mu_{l,a,j}$ and using the above formula yields

$$\sum_{i'=1}^{C} P^{\ell,a}(Q=i'|q=j) = 1$$

$$\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i'=1}^{C} E[z_{ui'\ell aj}|\mathcal{G},\mathbf{q}] = \mu_{\ell,a,j}.$$

This leaves us with the transition distribution update formula:

$$P^{\ell,a}(Q=i|q=j) = \frac{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} E[z_{ui\ell aj}|\mathcal{G},\mathbf{q}]}{\sum_{\substack{g\in\mathcal{G}\\ u\in\mathcal{V}_g}} \sum_{i'=1}^{C} E[z_{ui'\ell aj}|\mathcal{G},\mathbf{q}]}. \tag{13}$$

To conclude, we note that the learning equations for the emission distributions are identical to those of a mixture model, with the only difference that we substitute the expectation on the prior with $E[z_{ui}]$, which depends on the model's parameters. This is why we do not repeat the derivation for other emission distribution e.g., a univariate or multivariate Gaussian. Moreover, to introduce the bottom state $\perp$, it suffices to use an additional edge feature and an additional value for the variable Q.

Appendix B. Hyper-parameters

	D&D	**NCI1**	**PROTEINS**	**IMDB-***	**COLLAB**	**PPI**
C	{5,10,20}	{5,10,20}	{5,10,20}	{5,10,20}	{5,10,20}	{5,10,20}
$\mathbb{L}$	$\{\ell-1\}$	$\{\ell-1\}$	$\{\ell-1\}$	$\{\ell-1\}$	$\{\ell-1\}$	$\{\ell-1\}$
# layers	{5,10,15,20}	{10,20}	{5,10,15,20}	{5,10,15,20}	{5,10,15,20}	{5,10,20}
EM epochs	10	10	10	10	10	10
Discrete/Posterior	both	both	both	both	both	both
Unigram/Unibigram	both	both	both	both	both	unigram
Sum/Mean	sum	both	sum	both	mean	-
Classifier	{mlp}	{mlp}	{mlp}	{logistic, mlp}	{logistic, mlp}	{mlp}
# Hidden Units	{8,16,32,128}	{32,128}	{8,16,32,128}	{8,32,128}	{32,64,128}	{128,256,512}
Learning Rate	{1e-3,1e-4}	{1e-3}	{1e-3,1e-4}	{1e-3,1e-4}	{1e-3,1e-4}	{1e-2,1e-3}
L2 Weight Decay	{1e-2,5e-2,5e-3}	{1e-3,5e-4}	{1e-2,5e-2,5e-3}	{1e-2,1e-3}	{1e-3,5e-3,5e-4}	{0.,1e-5}
Classifier epochs	5000	2000	5000	5000	5000	5000
Early-Stopping	1000	100	1000	1000	1000	500
Batch-Size	100	200	100	100	100	20

Table 5: Hyper-parameters tried. IMDB-* refers to both binary and multi-class tasks. Early-Stopping refers to the epoch after which we apply early stopping on validation loss. The number of hidden units per layer are ignored when using logistic regression.

References

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS)*, pages 1993–2001, 2016.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. In *Neural Information Processing Systems (NIPS) Deep Learning Symposium*, 2016.

Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. Compositional generative mapping for tree-structured data - part I: Bottom-up probabilistic modeling of trees. *IEEE Transactions on Neural Networks and Learning Systems*, 23(12):1987–2002, 2012. Publisher: IEEE.

Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. An inputoutput hidden Markov model for tree transductions. *Neurocomputing*, 112:34–46, 2013. Publisher: Elsevier.

Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual Graph Markov Model: A deep and generative approach to graph processing. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 294–303. PMLR, 2018a.

Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. Generative kernels for tree-structured data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(10): 4932–4946, 2018b. Publisher: IEEE.

Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 2020.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, and others. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *arXiv preprint arXiv:1901.01343*, 2019.

Christopher M Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

Aleksandar Bojchevski and Stephan Gnnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schnauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005. Publisher: Oxford University Press.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

Daniele Castellana and Davide Bacciu. Bayesian tensor factorisation for bottom-up hidden tree Markov models. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2019.

Giovanni Da San Martino, Nicolo Navarin, and Alessandro Sperduti. A tree-based kernel for graphs. In *Proceedings of the 12th International Conference on Data Mining (ICDM)*, pages 975–986. SIAM, 2012.

Michelangelo Diligenti, Paolo Frasconi, and Marco Gori. Hidden tree Markov models for document image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(4):519–523, 2003. Publisher: IEEE.

Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003. Publisher: Elsevier.

Brendan L Douglas. The Weisfeiler-Lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*, 2011.

Jeffrey L Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. Publisher: Wiley Online Library.

Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. In *Proceedings of the 3rd Conference on Neural Information Processing Systems (NIPS)*, pages 524–532, 1990.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. *Workshop on Representation Learning on Graphs and Manifolds, International Conference on Learning Representations (ICLR)*, 2019.

Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9(5):768–786, 1998. Publisher: IEEE.

Paolo Frasconi, Fabrizio Costa, Luc De Raedt, and Kurt De Grave. klog: A language for logical and relational learning with kernels. *Artificial Intelligence*, 217:117–143, 2014. Publisher: Elsevier.

Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1263–1272, 2017.

Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of the International Conference on Neural Networks (ICNN)*, volume 1, pages 347–352. IEEE, 1996.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, pages 1024–1034, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, 2003.

Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels. 2020. URL `http://www.graphlearning.io/`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. In *Workshop on Bayesian Deep Learning, Neural Information Processing System (NIPS)*, 2016.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Gnnemann. Predict then propagate: graph neural networks meet personalized PageRank. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 14, pages 1137–1145, 1995.

Yann LeCun, Yoshua Bengio, and others. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):1995, 1995.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the 11th International Conference on Knowledge Discovery in Data Mining (SIGKDD)*, pages 177–187. ACM, 2005.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Sofus A Macskassy and Foster Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8(May):935–983, 2007.

Enrique S Marquez, Jonathon S Hare, and Mahesan Niranjan. Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5475–5485, 2018. Publisher: IEEE.

Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009. Publisher: IEEE.

Alessio Micheli, Diego Sona, and Alessandro Sperduti. Contextual processing of structured data by recursive cascade correlation. *IEEE Transactions on Neural Networks*, 15(6): 1396–1410, 2004. Publisher: IEEE.

Todd K Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, 1996. Publisher: IEEE.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 4602–4609, 2019.

Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

Marion Neumann, Novi Patricia, Roman Garnett, and Kristian Kersting. Efficient graph kernels by randomization. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pages 378–393. Springer, 2012.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 2014–2023, 2016.

Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

Meng Qu, Yoshua Bengio, and Jian Tang. GMNN: Graph Markov Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 5241–5250, 2019.

Lawrence R Rabiner and Biing-Hwang Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986. Publisher: IEEE.

Liva Ralaivola, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005. Publisher: Elsevier.

Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 385–394. ACM, 2017.

Frank Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

Lawrence K Saul and Michael I Jordan. Mixed memory Markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Machine Learning*, 37(1):75–87, 1999. Publisher: Springer.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20 (1):61–80, 2009. Publisher: IEEE.

Roded Sharan and Trey Ideker. Modeling cellular machinery through biological network comparison. *Nature Biotechnology*, 24(4):427, 2006. Publisher: Nature Publishing Group.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Gnnemann. Pitfalls of graph neural network evaluation. *Workshop on Relational Representation Learning, Neural Information Processing Systems (NeurIPS)*, 2018.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 488–495, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3693–3702, 2017.

Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997. Publisher: IEEE.

S Joshua Swamidass, Jonathan Chen, Jocelyne Bruand, Peter Phung, Liva Ralaivola, and Pierre Baldi. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, 21(suppl_1):i359–i368, 2005. Publisher: Oxford University Press.

Dinh V Tran, Nicol Navarin, and Alessandro Sperduti. On filter size in graph convolutional networks. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1534–1541. IEEE, 2018.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

Petar Velickovic, William Fedus, William L. Hamilton, Pietro Li, Yoshua Bengio, and R. Devon Hjelm. Deep Graph Infomax. In *Proceedings of the 7th International Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 6-9, 2019*, 2019.

S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. On the limitations of representing functions on sets. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 6487–6494, 2019.

Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3): 347–375, 2008. Publisher: Springer.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining (SIGKDD*, pages 1365–1374. ACM, 2015.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, pages 3391–3401, 2017.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.