Contextual Graph Markov Model: A Deep and Generative Approach to Graph Processing

Davide Bacciu [1] **Federico Errica** [1] **Alessio Micheli** [1]

Abstract

We introduce the Contextual Graph Markov Model, an approach combining ideas from generative models and neural networks for the processing of graph data. It founds on a constructive methodology to build a deep architecture comprising layers of probabilistic models that learn to encode the structured information in an incremental fashion. Context is diffused in an efficient and scalable way across the graph vertexes and edges. The resulting graph encoding is used in combination with discriminative models to address structure classification benchmarks.

1. Introduction

Learning in structured domains (SDs) deals with data of varying size and topology and amounts to identifying, synthesizing and embedding structural relationships into the model. A naive approach, widely diffused in the past, would preprocess the structure to obtain a fixed vectorial representation of hand-designed features, and feed it to standard learning models. In doing so, many of the relationships carrying useful information are definitively lost. Recurrent and recursive models have been widely used to learn such encodings by imposing a topological order on directed acyclic structures (sequences, trees and DAGs) known as the *causality* assumption which, however, restricts the classes of data that can be handled. In particular, the recursive definition of the state space cannot be applied to cyclic inputs without expensive diffusion mechanisms and constraints on the state transition function of the model. The paper introduces a novel generative model for learning unsupervised encodings of cyclic structures, that borrows from previous SD processing works, namely the generative Bottom-Up Hidden Tree Markov Model (Bacciu et al., 2012a) and the constructive approach of Neural Network for Graphs (Micheli, 2009). We believe this to be the first scalable realization of a generative model for the processing of variable size graphs. Its probabilistic nature allows to model labeling uncertainty as well as to perform inference on graph vertexes and arcs. The unsupervised generative encoders can capture rich patterns in the data that do not depend on the problem at hand, allowing wide reuse of the encoding across different tasks and to eventually leverage unlabeled examples to boost accuracy in a semi-supervised setting. In addition, the model is trained in an incremental fashion, which allows automatic construction of the architecture in supervised tasks. Information diffusion is achieved thanks to layering in a deep learning fashion. Computation within each layer is fully local, highly scalable and does not require iterative procedures or convergence conditions. Depth has a profound impact on context shaping since it allows each vertex to gather a wider picture of its surroundings, with a number of layers that is functional to the generative contextual encoding. We will focus on classification tasks that require a transduction $\mathcal{T}$ from the domain of graphs to a discrete or continuous space. Such mapping has the advantage to be adaptive, in contrast to kernel functions which are fixed, and it is built by composition of an encoding function $\mathcal{T}_{enc}$ and an output function $\mathcal{T}_{out}$. The former takes as input a graph $\mathbf{g}$ and produces an hidden representation $z(\mathbf{g})$, while the latter takes such encoding as input and outputs the prediction. We will consider an output function implemented through a Support Vector Machine (SVM), taking in input the hidden representation $z(\mathbf{g})$ learned by the latent variables of the generative model. The representational capability of our model, referred to as Contextual Graph Markov Model (CGMM), is tested on popular benchmarks for tree classification and on biochemical datasets where compounds are naturally represented as undirected graphs.

2. Background

Let us first define the class of graphs we deal with in the rest of the paper together with the associated notation. We consider the problem of learning from a population of graphs structured data $\mathcal{G} = \{\mathbf{g}_1, \ldots, \mathbf{g}_N\}$ where each sample $\mathbf{g}_n$

*Equal contribution [1]Department of Computer Science, University of Pisa. Correspondence to: Davide Bacciu <bacciu@di.unipi.it>, Federico Errica <f.errica@protonmail.com>, Alessio Micheli <micheli@di.unipi.it>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

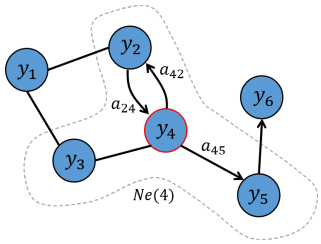

Figure 1. Example graph highlighting the notation used throughout the paper: y_u denotes the label of a vertex u, a_{uv} is the label of the edge connecting u to v, while $Ne(u)$ is the set of neighbors of a vertex u. The figure highlights the neighborhood of vertex 4 as a dashed area. Undirected edges denote the presence of two arcs oriented in different directions between the vertices and sharing the same label $a_{uv} = a_{vu}$. In general, a graph can include both directed and undirected edges.

is a labeled graph with varying topology, number of nodes and edges with respect to the other samples. No restrictions are posed on the graph structure, in particular in terms of acyclicity. A graph $\mathbf{g} = (\mathcal{V}_g, \mathcal{E}_g, \mathcal{Y}_g, \mathcal{A}_g)$ is defined by a set of vertexes $\mathcal{V}_g$ and by a set of edges (also referred to as arcs) $\mathcal{E}_g$ between two vertexes. A directed arc (u,v) can be associated to a label a_{uv} which, for the purpose of this paper, we assume to be taken from a finite alphabet $\{1, \dots, A\}$. Similarly, a vertex u is associated to a label y_u from a finite vocabulary $\{1, \dots, M\}$. When clear from the context, we omit the subscript g to avoid cluttering. Figure 1 shows an exemplar graph to clarify the notation: we use undirected edges, to ease the drawing, to denote the presence of two oriented arcs having the same label. In the remainder of the paper, we will use the concept of neighborhood of a vertex u, defined as $Ne(u) = \{v \in \mathcal{V}_g | (v,u) \in \mathcal{E}_g\}$. In the following, we will consider an *open* neighborhood of u i.e. one that does not include u itself. Undirected connections are modeled by two oriented arcs. Figure 1 shows an example of the neighborhood for a target node $u = 4$, depicted as a dashed area.

Processing structured data requires to cater for samples that change size and connectivity, extracting from this variability patterns useful for predictive or explorative analyses. Several learning models have been dealing successfully with sequences and trees. When moving to more general and possibly cyclic topology, as in this work, things get considerably more cumbersome. Literature in this respect reports a number of works dealing with the problem of loops through simplifications and relaxations of the original problem.

Within the neural paradigm, two foundational approaches have been proposed, almost at the same time, to design neural networks for graphs. They are able to learn both the hidden representation (state encoding) and the output function for classification or regression tasks. By exploiting weight sharing techniques they also implement a stationarity assumption in order to efficiently treat variable-size structures. The Graph Neural Network (Scarselli et al., 2009) (GNN) is a supervised model using a direct extension of recursive models. In an iterative fashion, at each step, information is exchanged between neurons associated to adjacent vertexes. Contractive state transition functions (obtained by weight constraints) are implemented by the neurons to ensure that the diffusion process can reach an equilibrium at each epoch. Hence GNN can process directed, undirected and cyclic graphs, at the cost of introducing a constraint on the weights values and a recursive dynamics for cycles that have drawbacks in terms of efficiency. The Neural Network for Graphs (Micheli, 2009) (NN4G), on the other hand, takes advantage of a feed-forward neural network construction whose layered architecture is determined during training e.g by a Cascade Correlation approach (Fahlman & Lebiere, 1990). Cyclic structures are easily handled without recursive state definitions because computation is local to each vertex by exploiting frozen states, while the incremental construction allows to capture larger and larger contexts in a symmetric way (as formally proved in (Micheli, 2009)). It follows that NN4G can be applied to any kind of graphs (as for GNN) despite the constraints on weights values being relaxed. NN4G also achieves a more efficient approach both by avoiding the need of a dynamical process for cycles and by exploiting an incremental approach to the task through progressive layers. In (Duvenaud et al., 2015) it has been proposed a hierarchical approach akin to NN4G and inspired by circular fingerprints in chemical structures. Differently from NN4G (and CGMM), the graph encoding is learned by end-to-end backpropagation on all layers, instead of incrementally. More recently, alternatives have been proposed extending Convolutional Neural Networks (CNN) to graphs: PATCHY-SAN (Niepert et al., 2016) has been the first to extend convolution to adjacent nodes in a graph by centering it on vertexes. When dealing with a graph dataset of varying topology, such an approach requires to determine a vertex ordering for each graph which is consistent across the entire dataset, as well as to impose an upper bound on the neighborhood size, limiting flexibility and context propagation. Two related approaches (Defferrard et al., 2016; Kipf & Welling, 2017) extended CNN to graphs by exploiting a spectral analysis of the graph's Laplacian, which however limits applicability to a single graph of fixed size with undirected edges. Very recently (Hamilton et al., 2017), it has appeared a preliminary work on extending graph convolutions to structures of varying size, using an information spreading mechanism very similar to that of (Micheli, 2009).

Kernels have been the most popular methodology for dealing with graph data in the last ten years. They are typically based on a relaxation of the original problem, counting matchings between simpler (computationally affordable)

substructures/features in the original graphs, e.g. those generated by random walks from each vertex (Borgwardt & Kriegel, 2005; Kashima et al., 2003). The solution proposed by (Da San Martino et al., 2012) is to transform the graph into a multi-set of Directed Acyclic Graphs (DAGs) and to define similarity as a multi-set similarity score on the DAGs. The Fast-Subtree (Shervashidze & Borgwardt, 2009) kernel (FS) compares graphs by means of the Weisfeiler-Lehman (WL) test for graphs' isomorphism. This is based on an iterative algorithm which outputs a relabeled graph, where each vertex is the result of a multi-set compression. The FS kernel uses the relabeling from different WL iterations as structural features, fundamentally computing graph similarity in terms of the number of common strings in such relabeling. The use of these kernels on large datasets is limited by their computational costs. Further, they lack adaptivity, since the similarity function is defined by hand-engineered features rather than being learned from the data, which might limit the extent of their applicability.

The application of probabilistic models to graph data is largely constrained by the impact of cycles on the probabilistic relationship between random variables used to encode structural information.Cycles make the inferential and training procedures too complex for practical use.In this respect, much of the contributions have been dealing with acyclic structures such as trees, typically through recursive extensions of the Hidden Markov Model (Frasconi et al., 1998; Diligenti et al., 2003; Bacciu et al., 2012a). These probabilistic models learn a distribution $P(\mathbf{x})$ over a tree $\mathbf{x}$ assuming nodes are generated by hidden states, with a transition probability that depends on the node's children, siblings or parents. These (typically unsupervised) generative models can be used to solve supervised tasks, e.g. by defining generative kernels on top of the learned probabilistic process as in (Bacciu et al., 2018).

The model proposed in this paper introduces the first scalable generative model for graphs of varying size, by building on the context propagation scheme of NN4G. It uses hidden variables to encode structural information using diffusion from neighboring nodes. In order not to incur in problems with cycles, context propagation is achieved thanks to the frozen representation of the neighbors' hidden states at preceding generative layers.

3. Contextual Graph Markov Model

In the following, we introduce the formalization of the Contextual Graph Markov Model (CGMM)[1], providing a brief summary of the main procedures required to run the model, i.e. parameter learning, inference of the vertex encoding and incremental layering policies.

[1] https://github.com/diningphil/CGMM

3.1. The model

CGMM is a probabilistic model for graphs that learns to encode structural information, adopting a modular approach that exploits both a stack of base modules and a layer-wise pooling strategy to increase discriminative efficacy along the lines of (Fahlman & Lebiere, 1990). As anticipated, the CGMM architecture borrows from NN4G, whereas the realization of each layer is strongly influenced by the recursive probabilistic models for trees, such as the Bottom-up Hidden Tree Markov Model (BHTMM) (Bacciu et al., 2012a).

The base CGMM component models a graph using discrete hidden state variables which encode information regarding vertexes and their context. Each vertex u is thus associated with a multinomial random variable Q_u with values on the finite alphabet $\{1, \dots, C\}$, where C is the hidden state size. Like in hidden Markov models, the hidden state assignment for a node u determines the emission of the observed label y_u (again assumed discrete) through the emission distribution $P(y_u|Q_u)$. Differently from a standard hidden Markov model, however, the current hidden state assignment, i.e. at layer l of the architecture, is not determined by a transition from another set of hidden states at the same layer l. Instead, a vertex state Q_u at layer l is determined by the frozen states' assignments of the neighboring vertexes $Ne(u)$ at previous layers $l' < l$. This is a key difference as such assignments can be considered fully observable at level l, thus avoiding mutual causal dependencies between the layers and preventing to get stuck in indefinite inference loops due to cycles. Figure 2 shows a representation of the approach as a graphical model, focusing on a target layer l. Hidden states at current level are represented by capital Q_u terms, which are empty nodes of the graphical model, since they are unobserved. Vertex labels are observed nodes y_u and the probabilistic model unfolds on the structure of the graph like a hidden Markov model unfolds on the structure of a sequence. Figure 2 also depicts how the hidden state for node $u = 4$ is determined through the distribution $P(Q_u|\mathbf{q}^{l-1}_{Ne(u)})$ based on the state of its neighbors (and possibly himself) at the previous level $l - 1$. Such states are determined by another probabilistic model, trained previously and then frozen. The resulting state assignments are marked with lowercase q_u terms and the corresponding nodes are black to denote the fact that they have to be considered observed.

The scheme depicted in Figure 2 is iterated across a number of layers, each independently trained with respect to the previous ones but using the knowledge extracted by them under the form of hidden state labels. Figure 3 shows the unrolling of the CGMM on a four-layers structure. The process starts at level one where hidden states are assigned without taking any context into consideration, except for the vertex label. At next iteration, vertexes have access to information concerning their direct neighbors. At level

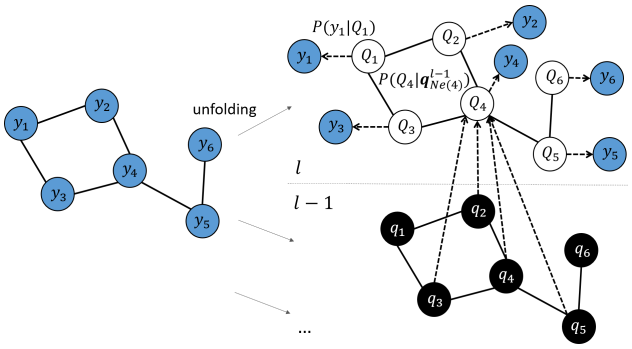

Figure 2. Representation as a graphical model of the unfolding of the l-th layer of the CGMM on the structure of the graph on the left. As usual, empty nodes correspond to unobserved variables, while full nodes denote observed vertex labels (shaded) and state assignments at previous level (black nodes). Thick undirected edges denote arcs while dashed arrows represent probabilistic causation relationships.

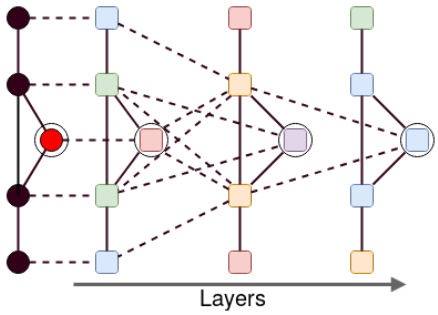

Figure 3. Unfolding of the hidden state variables for a cyclic structure (on the left).In this diagram, solid lines represents edges while dashed lines represent the flow of contextual information. At layer 1 vertexes are unaware of any information concerning surrounding structures. If we focus on the red vertex we see how increasing the depth leads to states (rounded squares) that encode a larger portion of the graph. Perhaps more importantly, the spreading of information is symmetric thanks to relaxation of the causality assumption. This cannot be achieved by a recurrent or recursive model without introducing a cyclic state definition.

3 they start receiving information from the neighbors of their neighbors. The iteration of this process allows an effective context propagation from each node of the graph (provided that a sufficient number of layers is used). The propagation of context in this architecture is symmetric with no need for causal dependencies between hidden states. As a consequence, CGMM can efficiently process graphs of any dimension without assumptions on topological properties, in contrast with models that need limiting the size of the vertex neighborhood or add padding to work with a fixed-size representation.

Each layer of the CGMM is implemented using different instantiations of the same probabilistic model, for which we provide the likelihood function in the following. Let us start by defining $L^{-1}(l)$ as the set of layers l' that precede current layer l. We let $\hat{\mathbf{q}}^{\mathbf{L^{-1}(l)}}_{\mathbf{Ne(u)}}$ denote the set of neighboring hidden states for vertex u, provided by previous layers $l' \in L^{-1}(l)$. We can then define the likelihood of a model at level l as

$$\mathcal{L}(\theta|\mathcal{G}) = \prod_{\mathbf{g}\in\mathcal{G}} \prod_{u=1}^{\mathcal{V}_g} \sum_{i=1}^{C} P(y_u|Q_u = i) P(Q_u = i|\hat{\mathbf{q}}^{L^{-1}(l)}_{Ne(u)}). \tag{1}$$

Please recall that, when clear from the context, we will abstract from the indexing term g to ease notation. Equation (1) assumes that vertexes are independent from each other, which makes training of the model efficient and scalable. Structural information is indirectly conveyed through conditioning on the state assignment from previous layers. The complexity of the rightmost term can become infeasible due to the (potentially) combinatorial number of states in the conditioning part. This can be made tractable by resorting to a Switching Parent (SP) approximation such as the one in (Bacciu et al., 2012a), so as to control complexity of children-to-parent transitions. Here, we introduce a SP variable L_u whose probability $P(L_u = l')$ denotes the weight of level l' in determining the state transition, and another variable S_u whose distribution $P^{l'}(S_u = a)$ controls the weight that an arc with label a should be given when the layer considered is l'. The resulting likelihood is

$$\mathcal{L} = \prod_{\mathbf{g}\in\mathcal{G}} \prod_{u=1}^{\mathcal{V}_g} \sum_{i=1}^{C} P(y_u|Q_u = i) \sum_{l'\in L^{-1}(l)} P(L_u = l') \sum_{a=1}^{A} P^{l'}(S_u = a) P^{l',a}(Q_u = i|\hat{\mathbf{q}}^{\mathbf{l',a}}_{Ne(u)}) \tag{2}$$

where we have introduced $\hat{\mathbf{q}}^{l',a}_{Ne(u)}$ to denote the states' assignments at level l' of all neighbors of u having label a on their arc. Such a parameterization allows to differentiate vertexes based on their hidden state, the label on the connecting edge as well as the depth where the contextual information is generated. The hidden state distribution can be further decomposed as

$$P^{l',a}(Q_u = i|\hat{\mathbf{q}}^{\mathbf{l',a}}_{Ne(u)}) = \frac{\sum_{v\in Ne^{l',a}(u)} P^{l',a}(Q_u = i|q_v)}{|Ne^{l',a}(u)|}$$

assuming that all vertexes in $Ne^{l',a}(u)$ which share the arc label a contribute equally (for each level l'). Clearly, alternative decompositions of this distribution can be devised, but for the sake of this paper we stick to the most simple and intuitive.

Summarizing, the model parameters for a level l are the emission distribution $P(y|Q)$, the arc-and-level-dependent state distribution $P^{l',a}(Q_u|q_v)$, plus the two weighting

terms $P(L = l)$ and $P^l(S = a)$ (all multinomials). All are shared between vertexes allowing the model to easily scale to graphs of any size. The first CGMM layer has no previous hidden states as inputs, requiring to fit only prior and emission distributions. Model complexity is dominated by $\mathcal{O}(\mathcal{V}_g * |L^{-1}(l)| * A * C^2)$ and can be regulated by tweaking the number of preceding layers L^{-1} and hidden states C. In its current implementation CGMM can process $20 - 22$ vertices per 1ms.

3.2. Training

Learning of a CGMM is achieved by training independently the layers composing the model. This can be performed by maximization of the likelihood in (2) using an Expectation-Maximization approach. Given the space limitations, we consider vertex labels to be discrete and unidimensional, but an extension to multidimensional and continuous labels is straightforward.

Formally, we introduce into the likelihood the indicator variables z_{uilaj}, that are one when vertex u is in state i while its neighbors with arc label a are in state j at level l, and zero otherwise. By these means we can rearrange the terms in Equation (2) to comprise only multiplications, which are later transformed into summations through the application of a logarithm. When applying the E-Step on such a function, we are seeking the posterior of the indicator variables, which can be shown to be equivalent to

$$\begin{aligned} E[z_{uilaj}|\mathbf{g}, \hat{\mathbf{q}}] &= P(Q_u = i, L_u = l, S_u = a, q = j|\mathbf{g}, \hat{\mathbf{q}}) \\ &= \frac{1}{Z} P(y_u|Q_u = i) P^{l,a}(Q_u = i|q = j) \\ &\quad \times P(L_u = l) P^l(S_u = a) \end{aligned} \tag{3}$$

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

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

$$P(L = l) = \frac{1}{Z} \sum_{\substack{\mathbf{g}\in\mathcal{G} \\ u\in\mathcal{V}_g \\ i\in\{1,\dots,C\}}} E[z_{uil}|\mathbf{g}, \hat{\mathbf{q}}]$$

$$P^l(S = a) = \frac{1}{Z} \sum_{\substack{\mathbf{g}\in\mathcal{G} \\ u\in\mathcal{V}_g \\ i\in\{1,\dots,C\}}} E[z_{uila}|\mathbf{g}, \hat{\mathbf{q}}]$$

$$P^{l,a}(Q = i|q = j) = \frac{1}{Z} \sum_{\substack{\mathbf{g}\in\mathcal{G} \\ u\in\mathcal{V}_g}} \Big(E[z_{uilaj}|\mathbf{g}, \hat{\mathbf{q}}] \Big)$$

where δ is the Kronecker delta. The different posteriors reported above can be obtained by straightforward marginalization of $E[z_{uilaj}|\mathbf{g}, \hat{\mathbf{q}}]$.

3.3. Inference

Inference serves to determine the most likely hidden state assignment given the observed data (at each level of the CGMM model). Specifically, for the first layer, we determine state assignment as

$$\max_i P(Q_u = i|\mathbf{g}) = \max_i P(y_u|Q_u = i) P(Q_u = i) \tag{4}$$

while for $l > 1$ this is computed as

$$\begin{aligned} \max_i P(Q_u = i|\mathbf{g}, \hat{\mathbf{q}}) = \max_i \Bigg\{ & P(y_u|Q_u = i) \times \\ & \sum_{l'\in L^{-1}(l)} P(L = l') \sum_{a=1}^{A} P^{l'}(S = a) \times \\ & \sum_{v\in Ne^{l',a}(u)} \frac{P^{l',a}(Q_u = i|q_v)}{|Ne^{l',a}(u)|} \Bigg\} \end{aligned} \tag{5}$$

The latent state assignment is fundamental to propagate the context to the higher layers of the CGMM. At the same time, it can be used to obtain a fixed-size vectorial encoding of the graph in terms of its hidden states occurrences, as discussed in the next section.

3.4. CGMM supervision, layering and pooling

The process described in previous sections serves to learn a probabilistic unsupervised encoder of graph structured information. In order to apply the CGMM approach in a supervised setting we need to define a mean to exploit the knowledge captured in its hidden state space. Following what has been proposed in (Bacciu et al., 2012b), we can define an encoding of the graph as a vector of state frequency counts for each layer, which are then concatenated into a fixed-size vector summarizing contributions from all layers. Figure 4 shows an high level view of such a process. The encoding obtained from the unsupervised CGMM can then be used as input to a standard classifier/regressor for performing supervised tasks. In this paper, we will use a SVM to test the CGMM encoding in graph classification tasks. Another question that needs answering to provide a fully functional and general implementation of the CGMM is: how do we determine the correct number of layers? Simple answer is, we rely on model selection, allowing it to decide how many layers are needed to achieve the best generalization performance. The procedure implemented in our CGMM is straightforward: after the addition of a new layer, this is first trained using the EM equations described above. Then a supervised model is trained and tested using the current graph encoding as input. If the newly generated

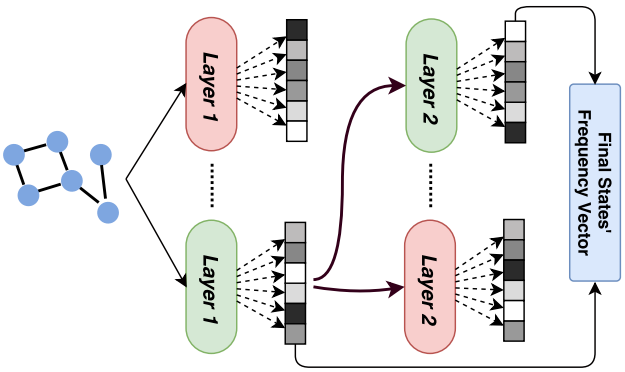

Figure 4. An overview of the training process for an architecture of final depth 2. At each layer a pool of CGMMs is trained and one of them is selected (shown in green) according to a supervised criterion. The second layer exploits the most likely hidden states inferred. States' frequency vectors computed at each layer can be concatenated to become the structure's fingerprint.

layer has a positive effect on the performance, then we try adding another layer, otherwise we stop.

A last feature that we have bundled in our implementation of the model is the use of pooling, motivated by the considerations in (Fahlman & Lebiere, 1990): as shown in Figure 4, at each layer we do not train a single probabilistic model. Rather we train different randomly initialized models, testing their respective encodings as described before. Then, for each level, we pick the best performing model of the pool and use it as input for the next layer of the architecture.

4. Experiments

The section provides an empirical assessment of CGMM's ability in extracting meaningful structural patterns from the data. Since our model is deeply rooted in hidden Markov models for trees, we first test it on tree structured data classification, confronting its performance with that of state-of-the-art probabilistic models and kernels for trees. Then we extend the analysis to more general structures testing CGMM on graph classification tasks.

4.1. Tree Classification Tasks

Previous works try to tackle tree classification by recursively unfolding structures in a top-down or bottom-up (Diligenti et al., 2003; Bacciu et al., 2012a; 2013; 2014) fashion, i.e. from the root to the leaves or from the frontier to the root node. The *INEX2005* and *INEX2006* (Denoyer & Gallinari, 2007) are intensively studied benchmarks in this context. They concern the classification of XML formatted documents from two IEEE *structure only corpora*. These datasets are characterized by a large number of unbalanced classes and discrete node labels. Such trees are generally shallow and with large out-degree. Table 1 summarizes the characteristics of the two datasets.

Table 1. Details of the *INEX2005* and *INEX2006* datasets. The total number of nodes is, respectively, 124,359 and 218,537.

DATA SET	SIZE	CLASSES	DEGREE	LABELS
INEX2005	9361	11	32	366
INEX2006	12107	18	66	65

Train and test splits are defined by the benchmark and comprise about 50% of the data each. Model selection decisions have been taken using an hold-out validation set of 20% of the training data. In these tasks, we have considered a CGMM without pooling and with a number of layers ranging from 1 to 4, which are reasonable choices considering that the INEX structures are, on average, quite shallow. Rather than aiming at identifying the best number of layers, in this experiment we will concentrate on assessing the effect of layering on information diffusion. CGMM hidden state size C has been chosen in $\{20, 40\}$, while for context propagation we have considered only the influence from the immediate predecessor of the current layer. The undirected tree edges have been transformed into two directed edges, one incoming and one outgoing, to account for context propagation both in an upwards and downwards direction (as occurs in the Hidden Tree Markov Model thanks to the upwards-downwards inference). Edges have been labelled with the relative position in the children subtree of the parent node.

Tree classification has been performed by placing a SVM on top of the CGMM hidden state encodings as described in Section 3.4. To compute similarity between these representations we have considered both a standard RBF kernel as well as a Jaccard kernel, such as the one introduced in (Bacciu et al., 2018) in its unigram or bigram variant. SVM hyperparameters C_{svm} and γ_{svm} have been selected from a limited set in $\{5, 50, 100\}$, as their choice had little impact on the model performance. The performance of CGMM is reported in Table 2, where it is compared to that of state of the art tree classification approaches. These include both the related input-driven generative model for trees (IOBHTMM) (Bacciu et al., 2013), the syntactic PT kernel (Moschitti, 2006), the generative tree kernels IOBHTMM-J, AM-GTM and Fisher (Bacciu et al., 2012b) as well as the bi-directional BDIO-J kernel (Bacciu et al., 2014). For the INEX 2006 data we also provide results for TreeESN (Gallicchio & Micheli, 2013), a recursive neural network model based on the reservoir computing paradigm.

CGMM shows a competitive performance on both datasets, matching the results of the two best models in literature.It is worth to note that CGMM relaxes the causality assump-

Table 2. Test accuracies for *INEX2005* and *INEX2006*.

MODEL	INEX2005	INEX2006
IOBHTMM	90.17 (3.13)	27.61 (1.34)
TreeESN	-	42.62
PT	**97.04**	41.13
Fisher	96.82 (0.1)	38.47 (0.8)
BHTMM-J	96.12 (0.1)	45.06 (0.2)
BDIO-J	95.24 (0.17)	**45.19**(0.12)
CGMM-RBF	96.60 (0.05)	45.1 (0.2)
CGMM-Jaccard	96.73 (0.06)	43.18 (0.3)

tions that in the BHTMM-based models are fundamental to capture the relationships between substructures composing the tree. Notwithstanding such a simplification, the context propagation mechanism allows CGMM to capture and summarize sufficient structural patterns to be competitive with approaches specialized in treating trees, which often result in high computational complexities such as for the Partial Tree (PT) (Moschitti, 2006).

CGMM allows to gain an insight on the organization of the state space, by inspecting how the hidden states' activations behave at different layers and for different tree classes. Figure 5 presents an example of such hidden state fingerprint for the INEX 2005 classes, obtained by averaging hidden state activations of the composing trees. One can clearly appreciate the effect of context diffusion between first and second layer, where a certain number of states seem to become tuned to class-specific structural patterns. Although the use of more layer has a positive effect on accuracy, the differences between layers fingerprints fade for $l > 2$. This is not surprising, considering the shallow nature of INEX treesfor which a couple of layers is sufficient to fully propagate the context.

4.2. Graph Classification Tasks

The second part of the experimental analysis shows how CGMM effectively processes a more general class of structures than trees. To this end, we consider a set of standard graph classification benchmarks from the biochemical domain, that are MUTAG (Debnath et al., 1991), CPDB (Helma et al., 2004) and AIDS (Smola & Vishwanathan, 2003). All benchmarks are binary classification tasks, where graph vertexes are labeled with discrete values representing atomic symbols. Edges are undirected and their label encodes the type of atomic bond: Table 3 summarizes the statistics of the datasets. Again, we have tested the CGMM model with varying number of levels (from 1 to 8) and considering state information only from the direct predecessor layer. Both RBF and Jaccard kernels have been tested, as well as two different fingerprint construction strategies: one

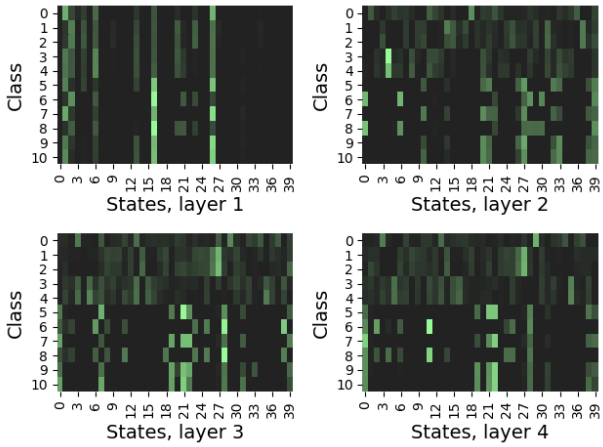

Figure 5. Average fingerprints for each target class of *INEX2005*. The model has been trained with C=40. Lighter areas of the plot denote more "popular" states.

Table 3. Statistics of the graphs in the biochemical datasets: the "Pos" column reports the percentage of members of the positive class, while nodes and edge values are averaged.

DATA SET	SIZE	POS(%)	ATOMS	EDGES
MUTAG	188	66.48	45.1	47.1
CPDB	684	49.85	14.1	14.6
AIDS	1503	28.7	58.9	61.4

using the encoding produced by the last layer; the other using the concatenation of the encodings produced by all layers. The hidden states size has been chosen in $\{20, 40\}$. A pooling strategy has been used with pool size set to 10.

Experimental assessment is performed according to two model selection schemes, following the approaches used in literature. One assesses performance using a single cross-validation (CV) on the standard 10-fold splits provided for the dataset (i.e. performance is reported for the best model identified on the 10-fold validation error). The second approach, less used in literature but more robust, uses a nested CV, where a 5-fold CV is applied to each training fold of the outer 10-fold: in this case, model selection decisions are taken based on the internal 5-fold CV performance. Tables 4 reports the classification accuracy for the two assessment schemes: note that the set of models with which we confront varies in function of the method used in the respective papers. To provide a comparison, we quote results from related approaches: FS kernel (Shervashidze & Borgwardt, 2009), WL-DDK kernels (Da San Martino et al., 2014), ODD-ST kernel (Da San Martino et al., 2012), Marginalised Graph Kernel (MGK) (Kashima et al., 2003), Correlated Pattern Mining (CPM) (Bringmann et al., 2006) gBoost (Saigo et al., 2009), Convolutional Neural Network for Graphs

Table 4. Performances (classification accuracy, in %) for model selection and risk assessment.

	CPDB	AIDS	MUTAG
10-FOLD CV			
CPM	76.0	83.2	80.8
MGK	76.5	76.2	-
gBoost	78.8	80.2	85.2
Fast Subtree	76.3	79.1	89.3
$ODD-ST_h$	80.4	83.5	87.8
PATCHY-SAN	-	-	**92.63** (4.21)
CGMM	**81.04** (4.00)	**84.16** (2.31)	91.18 (6.02)
NESTED CV			
IOBHTMM-J	69.03 (3.35)	79.17 (3.46)	
AM-GTM	75.44 (3.74)	81.33 (3.89)	
Fisher	68.87 (3.41)	76.65 (3.45)	
ST	75.29 (1.64)	82.00 (2.00)	
SST	76.59 (2.16)	80.17 (1.53)	
Fast Subtree	73.22 (0.78)	75.61 (1.00)	
$ODD-ST_h$	76.44 (0.62)	81.51 (0.74)	
WL_{NS-DDK}	77.03 (1.18)	82.8 (0.66)	
WL_{DDK}	76.52 (1.16)	82.93 (0.71)	
CGMM	**78.06** (6.47)	**83.15** (2.17)	

(PATCHY-SAN) (Niepert et al., 2016). We also provide results obtained by applying tree kernels to all the spanning trees generated from each graph vertex, using generative tree kernels from (Bacciu et al., 2012b) as well as syntactic tree kernels, such as ST (Smola & Vishwanathan, 2003) and SST (Collins & Duffy, 2002). The results for the MUTAG dataset are provided only for the single 10-fold CV as no model in literature appears to use it with a nested CV assessment (CGMM scores an accuracy of 85.3% when assessed with a nested CV on MUTAG).

Results highlight that CGMM obtains the highest accuracies on both CPDB and AIDS using assessment schemes compared to a number of state-of-the-art graph kernels of various nature (syntactical, convolutional, generative). The standard deviation of CGMM is relatively high on the CPDB dataset for what looks like an unfortunate split of the folds (only 2 folds out of 10 have accuracies considerably higher and lower than the average, hence the high variance). Results on MUTAG are encouraging as well, basically matching the state of the art of the PATCHY-SAN neural model.

The effect of stacking on the quality of the representation learned by the CGMM can be assessed by looking at the accuracy of a linear SVM trained on the CGMM fingerprints for a varying number of layers.Figure 6 depicts this information for the 3 graph classification benchmarks, considering a CGMM model without pooling to discount its effect on model performance. One can clearly note that depth allows the propagation of informative structural context with a beneficial effect on accuracy on all the datasets, with an eventual performance plateau. On MUTAG, the addition of a new layer leads to a minor degradation of performance at some point, which can be explained by the greedy layer-wise and unsupervised nature of the approach. This can be easily contrasted using early stopping as in Section 3.4.

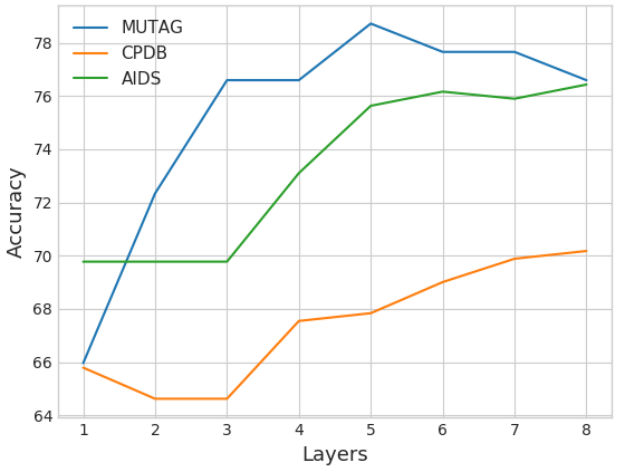

Figure 6. CGMM test accuracy as a function of the number of layers. For this experiment, we have used a CGMM without pooling and a vanilla SVM classifier trained on the concatenation of the hidden states fingerprints up to the current layer l.

5. Conclusions

We have presented a novel framework that tackles learning in the structured domain by combining generative and discriminative approaches. It works with graphs of any size and shape exploiting a rich set of concepts such as full stationarity, incremental depth of hierarchical layers and a pooling strategy. Importantly, CGMM does not squash the graph into a simpler representation prior to learning. The hierarchical state construction is functional to the relaxation of causal dependencies, allowing a symmetric context spreading between states representing vertexes and to elegantly cope with cycles. Experimental analysis shows that the model is competitive with both state-of-the-art recursive models for trees and kernels for graphs. The authors hope that the exploitation of the proposed framework, which can be extended in many directions, can contribute to the extensive use of both generative and discriminative approaches to the adaptive processing of structured data.

Acknowledgements

D. Bacciu would like to acknowledge support from the Italian Ministry of Education, University, and Research (MIUR) under project SIR 2014 LIST-IT (grant n. RBSI14STDE).

References

Bacciu, D., Micheli, A., and Sperduti, A. 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, 2012a.

Bacciu, D., Micheli, A., and Sperduti, A. A generative multiset kernel for structured data. *Artificial Neural Networks and Machine Learning–ICANN 2012*, pp. 57–64, 2012b.

Bacciu, D., Micheli, A., and Sperduti, A. An input–output hidden Markov model for tree transductions. *Neurocomputing*, 112:34–46, 2013.

Bacciu, D., Micheli, A., and Sperduti, A. Modeling bidirectional tree contexts by generative transductions. In *International Conference on Neural Information Processing*, pp. 543–550. Springer, 2014.

Bacciu, D., Micheli, A., and Sperduti, A. Generative kernels for tree-structured data. *IEEE transactions on neural networks and learning systems*, pp. 1–15, 2018.

Borgwardt, K. M. and Kriegel, H.-P. Shortest-path kernels on graphs. In *Data Mining, Fifth IEEE International Conference on*, pp. 8–pp. IEEE, 2005.

Bringmann, B., Zimmermann, A., De Raedt, L., and Nijssen, S. Dont be afraid of simpler patterns. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 55–66. Springer, 2006.

Collins, M. and Duffy, N. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 263–270. Association for Computational Linguistics, 2002.

Da San Martino, G., Navarin, N., and Sperduti, A. A tree-based kernel for graphs. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pp. 975–986. SIAM, 2012.

Da San Martino, G., Navarin, N., and Sperduti, A. Graph kernels exploiting weisfeiler-lehman graph isomorphism test extensions. In *International Conference on Neural Information Processing*, pp. 93–100. Springer, 2014.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34 (2):786–797, 1991.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 3844–3852, USA, 2016. Curran Associates Inc.

Denoyer, L. and Gallinari, P. Report on the xml mining track at inex 2005 and inex 2006: categorization and clustering of xml documents. In *ACM SIGIR Forum*, volume 41, pp. 79–90. ACM, 2007.

Diligenti, M., Frasconi, P., and Gori, M. Hidden tree markov models for document image classification. *IEEE Transactions on pattern analysis and machine intelligence*, 25 (4):519–523, 2003.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Fahlman, S. E. and Lebiere, C. The cascade-correlation learning architecture. In Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 2*, pp. 524–532. Morgan-Kaufmann, 1990.

Frasconi, P., Gori, M., and Sperduti, A. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 9(5):768–786, 1998.

Gallicchio, C. and Micheli, A. Tree echo state networks. *Neurocomputing*, 101:319–337, 2013.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017. URL http://arxiv.org/abs/1706.02216.

Helma, C., Cramer, T., Kramer, S., and De Raedt, L. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *Journal of chemical information and computer sciences*, 44 (4):1402–1411, 2004.

Kashima, H., Tsuda, K., and Inokuchi, A. Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 321–328, 2003.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *Proceedings of ICLR 2017*, 2017. URL http://arxiv.org/abs/1609.02907.

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

Moschitti, A. Efficient convolution kernels for dependency and constituent syntactic trees. In *ECML*, volume 4212, pp. 318–329. Springer, 2006.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pp. 2014–2023, 2016.

Saigo, H., Nowozin, S., Kadowaki, T., Kudo, T., and Tsuda, K. gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75(1):69–89, 2009.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Shervashidze, N. and Borgwardt, K. M. Fast subtree kernels on graphs. In *Advances in neural information processing systems*, pp. 1660–1668, 2009.

Smola, A. J. and Vishwanathan, S. Fast kernels for string and tree matching. In *Advances in neural information processing systems*, pp. 585–592, 2003.