Pattern Recognition 97 (2020) 107000

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Dynamic graph convolutional networks

Franco Manessi[a], Alessandro Rozza[a,*], Mario Manzo[b]

[a] Strategic Analytics, lastminute.com, Chiasso, Switzerland
[b] Information Technology Services, University of Naples “L’Orientale”, Naples, Italy

ARTICLE INFO

Article history:
Received 14 May 2018
Revised 21 May 2019
Accepted 13 August 2019
Available online 13 August 2019

ABSTRACT

In many different classification tasks it is required to manage structured data, which are usually modeled as graphs. Moreover, these graphs can be dynamic, meaning that the vertices/edges of each graph may change over time. The goal is to exploit existing neural network architectures to model datasets that are best represented with graph structures that change over time. To the best of the authors’ knowledge, this task has not been addressed using these kinds of architectures. Two novel approaches are proposed, which combine Long Short-Term Memory networks and Graph Convolutional Networks to learn long short-term dependencies together with graph structure. The advantage provided by the proposed methods is confirmed by the results achieved on four real world datasets: an increase of up to 12 percentage points in Accuracy and F1 scores for vertex-based semi-supervised classification and up to 2 percentage points in Accuracy and F1 scores for graph-based supervised classification.

1. Introduction

In machine learning, data are usually described as points in a vector space ($\boldsymbol{x} \in \mathbb{R}^d$). Nowadays, structured data are ubiquitous. The capability to capture the structural relationships among the data points can be particularly useful in improving the effectiveness of the models trained on them.

To this aim, graphs are widely employed to represent this kind of information in terms of nodes/vertices and edges, including local and spatial information arising from data. For instance, consider a d-dimensional dataset $\mathcal{X} = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n\} \subset \mathbb{R}^d$ representing points in space, a graph can be extracted from $\mathcal{X}$ by considering each point as a node, where edge connectivity and weights can be computed using a metric function. new data representation $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is obtained, where $\mathcal{V}$ is a set, which contains vertices and $\mathcal{E}$ is a set of weighted[1] pairs of vertices (edges).

Applications in a graph domain can be usually divided into two main categories: *vertex-focused* and *graph-focused* [1]. The former includes tasks where one performs classification/regression on the vertices of a graph, whereas in the latter one performs these tasks on the graph itself. For instance, object detection [2] and image annotation [3] are examples of vertex-focused applications, the former consists of finding whether an image contains a given object and its position, the latter consists of extracting a caption that describes an image. Another possible example could be web page classification, where the web is represented by a graph where nodes are the pages and edges are the hyperlinks between them, the aim being to exploit the web connectivity to classify pages in a set of topics. Instead, estimating the probability of a chemical compound to cause certain diseases can be seen a graph-focused application [4]. This is possible due to the fact that a chemical compound can be modeled by a graph where the nodes are the atoms and the edges the chemical bonds.

For the sake of simplicity and without loss of generality, just the classification problem is considered. Under this setting, the *vertex-focused* applications are characterized by a set of labels $\mathcal{Y} = \{1, \ldots, k\}$, a dataset $\mathcal{X} = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n\} \subset \mathbb{R}^d$, the related graph $\mathcal{G}$. Let us assume $\mathcal{L} \subset \mathcal{X}$ is a subset of labeled nodes. The goal is to classify the unlabeled nodes belonging to $\mathcal{X}$ but not to $\mathcal{L}$ by exploiting jointly the node features and the graph structure by means of a semi-supervised learning approach. *Graph-focused* applications are related to the goal of learning a function f that maps different graphs to integer values by taking into account the features of the nodes of each graph: $f(\mathcal{G}, \mathcal{X}) \in \mathcal{Y}$. This task can be solved by supervised classification on the graphs.

A number of research works are devoted to classification, both for *vertex-focused* and *graph-focused* applications [5–10]. Nevertheless, there is a major limitation in existing studies: most of these research works are focused on static graphs. However, many real world graph-structured data are dynamic and nodes/edges in the

* Corresponding author.
E-mail addresses: franco.manessi@lastminute.com (F. Manessi), alessandro.rozza@lastminute.com (A. Rozza), mmanzo@unior.it (M. Manzo).

[1] Note that an unweighted graph can be considered an instance of a weighted graph, where all the weights are equal.

https://doi.org/10.1016/j.patcog.2019.107000
0031-3203/© 2019 Elsevier Ltd. All rights reserved.

graphs may change over time. In such dynamic scenarios, temporal information can also play an important role. For instance, the interactions between individuals inside a building in one day can be modeled as a sequence of graphs, each one describing a time window within the day, where the nodes are the people and the edges are the interactions occurring between them within a time frame. As another example, consider the classification of human activities from motion-capture data, where each frame can be modeled as graph, where the vertices are the skeleton joints of the person in question.

In the last decade, neural networks have shown their great power and flexibility by learning to represent the world as a nested hierarchy of concepts, achieving outstanding results in many different fields of application. It is important to underline that just a few research works have been devoted to encoding graph structures directly using a neural network model [1,11–15]. Among them, to the best of the authors knowledge, no one is able to manage dynamic graphs.

To exploit both graph-structured data and temporal information through the use of a neural network model, two novel approaches are introduced. They combine Long Short Term-Memory network (LSTM, [16]) and Graph Convolutional Network (GCN, [14]), which can be considered the two base elements of the proposed architectures. Both of them are able to deal with *vertex-focused* applications. Respectively, these techniques are able to capture temporal information and to properly manage graph-structured data. The approaches are also extended to deal with *graph-focused* applications.

LSTMs are a special kind of Recurrent Neural Network (RNN, [17]), which are able to improve the learning of long term dependencies. All RNNs take the form of a chain of repeating modules of neural networks. Precisely, RNNs are artificial neural networks where connections among units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic sequential behavior. In standard RNNs, the repeating module is based on a simple structure, such as a single (hyperbolic tangent) unit. LSTMs extend the repeating module by combining four interacting units. Specifically, it is based on: cell state, forget gate, input gate and output gate. The most important part of LSTMs is the cell state, which can be pictured as a conveyor belt. It runs straight down the entire chain, with some linear interactions. The first interaction decides what information is going to be removed from the cell state. This decision is made by a sigmoid layer, called the *forget gate*. The next interaction decides what new information will be stored in the cell state. This can be operationalized in two parts: (i) a sigmoid layer, called the *input gate* decides which values should be updated; (ii) another sigmoid layer creates a vector of new candidate values that could be added to the cell state. The final output is generated by combining the cell state with a sigmoid layer.

A GCN is a neural network model that directly encodes graph structure, which is trained on a supervised target loss for all the nodes with labels. This approach is able to distribute the gradient information from the supervised loss and to learn representations exploiting both labeled and unlabeled nodes, thus achieving state-of-the-art results. The goal of a GCN is to learn a function of signals/features on a graph which takes as input:

- A feature description for each node, summarized in a $N \times D$ feature matrix (where N is the number of nodes and D is the number of input features);
- A representative description of the graph structure in matrix form (typically in the form of an adjacency matrix).

This model produces a node-level output (an $N \times F$ feature matrix, where F is the number of output features per node).

A GCN shares its underlying intuition with Convolutional Neural Networks (CNNs, [18]), specialized kinds of neural networks that are highly successful in practical applications (such as computer vision) employing, within some of their layers, a convolutional operation in place of regular matrix multiplication. A convolutional layer in a CNN exploits the *locality of information* embedded in its input (e.g. pixel adjacency for image inputs) to extract *local* features. It does this by repeatedly looking at small patches of its input and by learning a mixing matrix of the size of the patches that is shared across all the patches, thus reducing the amount of parameters. Similarly, a GCN exploits the locality notion induced by the graph connectivity by looking at small patches of the graph, where the patches are all neighbourhood subgraphs of the original graph.

2. Related work

Many real world datasets are best represented in graph form: e.g. knowledge graphs, social networks, protein-interaction networks and the World Wide Web.

To achieve good classification results on this kind of data, the traditional approaches proposed in the literature mainly follow two different directions: to identify structural properties as features and manage them using traditional learning methods, or to propagate the labels to obtain a direct classification.

Zhu et al. [19] propose a semi-supervised learning algorithm based on a Gaussian random field model (also known as Label Propagation). The learning problem is formulated exploiting Gaussian random fields over a continuous function state space, rather than random fields over the discrete label set. This ensures that the most probable configuration of the field is unique, it is characterized in terms of harmonic functions and it has a closed form solution that can be computed using matrix methods or belief propagation. In contrast, for multi-label discrete random fields, computing the lowest energy configuration is typically NP-hard and approximation algorithms or other heuristics must be used [20]. In [21], the authors have extended the previous approach by introducing Dynamic Label Propagation, a technique to better deal with multi-class/multi-label problems due to the lack in consideration of label correlation.

Xu et al. [8] present a semi-supervised factor graph model that is able to exploit the relationships among nodes. In this approach, each vertex is modeled as a variable node and the various relationships are modeled as factor nodes. Grover and Leskovec, in [22], present an efficient and scalable algorithm for feature learning in networks that optimizes a novel network-aware, neighborhood preserving objective function using Stochastic Gradient Descent. Perozzi et al. [23] propose an approach called DeepWalk. This technique uses truncated random walks to efficiently learn representations for vertices in graphs. These latent representations, which encode graph relations in a vector space, can be easily exploited by statistical models thus producing state-of-the-art results.

Another technique used to extract vector representations starting from graphs is the Bag of Graphs method introduced in [24], where the authors exploit the combination of graphs with a bag-of-words model to create a discriminant and efficient representation based on local structures of an object, leading to fast and accurate results in classification tasks.

Unfortunately, the described techniques cannot deal with graphs that dynamically change over time. There are a few methodologies that have been designed to classify nodes in dynamic networks [25,26]. Li et al. [25] propose an approach that is able to learn the latent feature representation and to capture dynamic patterns. Yao et al. [26] present a Support Vector Machine-based approach that combines the support vectors of the previous temporal instant with the current training data to exploit temporal relationships. Pei et al. [27] define an approach known as the Dynamic Factor Graph Model, which they apply to node classification in dynamic social networks. This approach organizes the dynamic graph data in a sequence of graphs. Three types of factors, called node, correlation and dynamic factors, are designed to capture node features, node correlations and temporal correlations, respectively. Node and correlation factors are designed to capture the global and local properties of the graph structures, while the dynamic factor exploits the temporal dependencies.

It is important to underline that very little attention has been devoted to the generalization of neural network models to graph-structured datasets. In the last few years, a number of research works have revisited the problem of generalizing neural networks to work on structured graphs, some of them achieving promising results in domains that have been previously dominated by other techniques. Gori et al. [28] and Scarselli et al. [1] formalize a novel neural network model, the Graph Neural Network (GNN). This model is based on extending a neural network method with the purpose of processing data in form of graph structures. The GNN model can process different types of graphs (e.g., acyclic, cyclic, directed and undirected) and it maps a graph and its nodes into a D-dimensional Euclidean space to learn the final classification/regression model. Li et al. [15] extend the GNN model, by relaxing the contractivity requirement of the propagation step through the use of a Gated Recurrent Unit [29] and by predicting sequence of outputs from a single input graph. Bruna et al. [11] describe two generalizations of CNNs: one based on a hierarchical clustering of the domain and another based on the spectrum of the graph (computed using the Laplacian matrix). Duvenaud et al. [13] present another variant of CNNs that works on graph structures. This model allows an end-to-end learning on graphs of arbitrary size and shape. Defferrard et al. [12] introduce a formulation of CNNs in the context of spectral graph theory. The model provides efficient numerical schemes to design fast localized convolutional filters on graphs, achieving the same computational complexity of classical CNNs working on any graph structure. Kipf and Welling [14] propose an approach for semi-supervised learning on graph-structured data (the GCN) based on CNNs. In their work, they exploit a localized first-order approximation of the spectral graph convolutions framework [30]. Their model linearly scales in the number of graph edges and learns hidden layer representations that encode local and structural graph features. Seo et al. [31] address the problem of generating sequences on *static* graphs by extending [12] through the usage of a RNN. Monti et al. [32] tackle the problem of matrix completion (one of the most common formulations of recomender systems) by means of recurrent multi-graph neural networks.

Note that none of the aforementioned neural network architectures are able to properly deal with dynamic graph-structured data.

3. Methods

In this section, two novel network architectures are introduced to deal with *vertex/graph-focused* applications. Both of them rely on the following intuitions:

- GCNs can effectively deal with graph-structured information, but they lack the ability to handle data structures that change over time. This limitation is (at least) twofold: (i) inability to manage dynamic vertex features, (ii) inability to manage dynamic edge connections.
- LSTMs excel in finding long and short range sequence dependencies, but they lack the ability to explicitly exploit graph-structured information.

The new network architectures proposed in this paper will work on ordered sequences of graphs and ordered sequences of vertex features. For sequences of length one, this reduces to the *vertex/graph-focused* applications described in Section 1.

The paper contributions are based on the idea of combining an extension of the Graph Convolution (GC, the fundamental layer of the GCNs) and a modified version of an LSTM, which learn the downstream recurrent units by exploiting both graph-structured data and vertex features.

Two GC-like layers are here proposed. They take as input a graph sequence and the corresponding ordered sequence of vertex features. The output is an ordered sequence of a new vertex representation. These layers are:

- The *Waterfall Dynamic-GC* layer, which performs at each step of the sequence a graph convolution on the vertex input sequence. An important feature of this layer is that the trainable parameters of each graph convolution are shared among the various steps of the sequence;
- The *Concatenated Dynamic-GC* layer, which performs at each step of the sequence a graph convolution on the vertex input features and concatenates it to the input. Again, the trainable parameters are shared among the steps in the sequence.

Each of the two layers can jointly be used with a modified version of an LSTM to perform a semi-supervised classification of a sequence of vertices or a supervised classification of a sequence of graphs. The difference between the two tasks just consists in how the final step of processing of the data is performed (for further details, see Eqs. (8) and (10).

In the following section the mathematical definitions of the two modified GC layers are given, together with the modified version of the LSTM, as well as some other handy definitions that will be useful when the final network architectures will be described.

3.1. Definitions

Let $[\boldsymbol{Y}]_{i,j}$ be the ith row, jth column element of the matrix $\boldsymbol{Y}$ and $\boldsymbol{Y}'$ its transpose. $\boldsymbol{I}_d$ is the identity matrix of $\mathbb{R}^d$; softmax and ReLU are the *soft-maximum* and the *rectified linear unit* activation functions [33]. Note that all the activation functions act element-wise when applied to a matrix $\boldsymbol{M} \in \mathbb{R}^{n\times m}$, e.g. $[\mathrm{ReLU}\,\boldsymbol{M}]_{i,j} = \mathrm{ReLU}[\boldsymbol{M}]_{i,j}$.

The matrix $\boldsymbol{P} \in \mathbb{R}^{d\times d}$ is a *projector* on $\mathbb{R}^d$ if it is a symmetric, positive semi-definite matrix and where $\boldsymbol{P}^2 = \boldsymbol{P}$. In particular, it is a *diagonal projector* if it is a diagonal matrix (with possibly some zero entries on the main diagonal). In other words, a diagonal projector on $\mathbb{R}^d$ is the diagonal matrix with some 1s on the main diagonal that when it is right-multiplied by a d-dimensional column vector $\boldsymbol{v}$ it zeroes out all the entries of $\boldsymbol{v}$ corresponding to the zeros on the main diagonal of $\boldsymbol{P}$:

$$\overbrace{\begin{pmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}}^{\boldsymbol{P}}\overbrace{\begin{pmatrix}a\\b\\c\\d\end{pmatrix}}^{\boldsymbol{v}}=\overbrace{\begin{pmatrix}a\\0\\c\\d\end{pmatrix}}^{\boldsymbol{Pv}}.$$

Let $(\mathcal{G}_i)_{i\in\mathbb{Z}_T}$ with $\mathbb{Z}_T := \{1, 2, \ldots, T\}$ be a finite sequence of undirected graphs $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, with $\mathcal{V}_i = \mathcal{V}\ \forall i \in \mathbb{Z}_T$, i.e. all the graphs in the sequence share the same vertices. Considering the graph $\mathcal{G}_i$, for each vertex $v^k \in \mathcal{V}$ let $\boldsymbol{x}_i^k \in \mathbb{R}^d$ be the corresponding feature vector. Each step i in the sequence $\mathbb{Z}_T$ can completely be defined by its graph $\mathcal{G}_i$ (modeled by the adjacency matrix[2] $\boldsymbol{A}_i$)

[2] Given a finite undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, its adjacency matrix is the square matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ where $[A]_{i,j} = [A]_{j,i} = w_{ij}$ if and only if there is an edge between

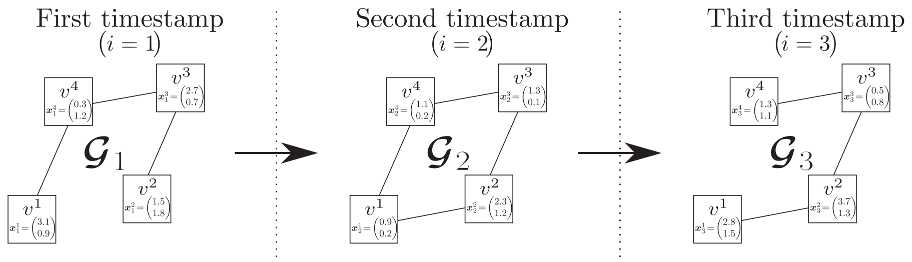

First timestamp
$(i = 1)$
Second timestamp
$(i = 2)$
Third timestamp
$(i = 3)$
v^4 $x_1^4 = \begin{pmatrix}0.3\\1.2\end{pmatrix}$
v^3 $x_1^3 = \begin{pmatrix}2.7\\0.7\end{pmatrix}$
$\mathcal{G}_1$
v^2 $x_1^2 = \begin{pmatrix}1.5\\1.8\end{pmatrix}$
v^1 $x_1^1 = \begin{pmatrix}3.1\\0.9\end{pmatrix}$
v^4 $x_2^4 = \begin{pmatrix}1.1\\0.2\end{pmatrix}$
v^3 $x_2^3 = \begin{pmatrix}1.3\\0.1\end{pmatrix}$
$\mathcal{G}_2$
v^2 $x_2^2 = \begin{pmatrix}2.3\\1.2\end{pmatrix}$
v^1 $x_2^1 = \begin{pmatrix}0.9\\0.2\end{pmatrix}$
v^4 $x_3^4 = \begin{pmatrix}1.3\\1.1\end{pmatrix}$
v^3 $x_3^3 = \begin{pmatrix}0.5\\0.8\end{pmatrix}$
$\mathcal{G}_3$
v^2 $x_3^2 = \begin{pmatrix}3.7\\1.3\end{pmatrix}$
v^1 $x_3^1 = \begin{pmatrix}2.8\\1.5\end{pmatrix}$

Fig. 1. Representation of a sequence of unweighted graphs $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ made of 4 nodes (v^1, v^2, v^3, v^4), where each vertex-feature vector $\boldsymbol{x}_i^k \in \mathbb{R}^2$, with $i \in \{1, 2, 3\}$ and $k \in \{1, 2, 3, 4\}$. The graphs share the same vertex elements, while the edge connectivity and the vertex-feature values change for different graphs of the sequence. For instance, the feature vector for vertex v^1 changes from $\boldsymbol{v}_2^1 = (0.9\ 0.2)'$ to $\boldsymbol{v}_3^1 = (2.8\ 1.5)'$ between steps $i = 2$ and $i = 3$. Similarly, the edge between vertices v^1 and v^4 disappears between the same steps in the graph sequence.

and by the vertex-features matrix $\boldsymbol{X}_i \in \mathbb{R}^{|\mathcal{V}| \times d}$ (the matrix whose row vectors are the $\boldsymbol{x}_i^k$). It is worth mentioning that despite sharing the same vertex elements, the graphs within the sequence do not share the same vertex-feature values. See Fig. 1 for a pictorial representation of a sequence of graphs.

The mathematics of the GC layer [14] and the LSTM [16] are here briefly recalled, since they are the basic building blocks of the contribution of this paper. Given a graph with adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ and vertex-feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, the GC layer with M output nodes and $\boldsymbol{B} \in \mathbb{R}^{d \times M}$ weight matrix is defined as the function:

$$\mathrm{GC}_{M,\boldsymbol{A}}^{\boldsymbol{B}} : \mathbb{R}^{|\mathcal{V}| \times d} \to \mathbb{R}^{|\mathcal{V}| \times M}, \qquad \mathrm{GC}_{M,\boldsymbol{A}}^{\boldsymbol{B}}(\boldsymbol{X}) := \mathrm{ReLU}(\hat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{B}), \tag{1}$$

where $\hat{\boldsymbol{A}}$ is the re-normalized adjacency matrix, i.e. $\hat{\boldsymbol{A}} := \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ with $\tilde{\boldsymbol{A}} := \boldsymbol{A} + \boldsymbol{I}_{|\mathcal{V}|}$ and $[\tilde{\boldsymbol{D}}]_{kk} := \sum_l [\tilde{\boldsymbol{A}}]_{kl}$. Note that the GC layer can be seen as localized first-order approximation of spectral graph convolution [34], with the additional *renormalization trick* in order to improve numerical stability [14].

Given the sequence $(\boldsymbol{x}_i)_{i \in \mathbb{Z}_T}$ with $\boldsymbol{x}_i d$-dimensional row vectors for each $i \in \mathbb{Z}_T$, a *returning sequence-LSTM* with N output nodes, is the function $\mathrm{LSTM}_N : (\boldsymbol{x}_i)_{i \in \mathbb{Z}_T} \mapsto (\boldsymbol{h}_i)_{i \in \mathbb{Z}_T}$, with $\boldsymbol{h}_i \in \mathbb{R}^N$ and

$$\begin{aligned} &\boldsymbol{h}_i = \boldsymbol{o}_i \odot \tanh(\boldsymbol{c}_i), && \boldsymbol{f}_i = \sigma(\boldsymbol{x}_i \boldsymbol{W}_f + \boldsymbol{h}_{i-1}\boldsymbol{U}_f + \boldsymbol{b}_f), \\ &\boldsymbol{c}_i = \boldsymbol{j}_i \odot \tilde{\boldsymbol{c}}_i + \boldsymbol{f}_i \odot \boldsymbol{c}_{i-1}, && \boldsymbol{j}_i = \sigma(\boldsymbol{x}_i \boldsymbol{W}_j + \boldsymbol{h}_{i-1}\boldsymbol{U}_j + \boldsymbol{b}_j), \\ &\boldsymbol{o}_i = \sigma(\boldsymbol{x}_i \boldsymbol{W}_o + \boldsymbol{h}_{i-1}\boldsymbol{U}_o + \boldsymbol{b}_o), && \tilde{\boldsymbol{c}}_i = \sigma(\boldsymbol{x}_i \boldsymbol{W}_c + \boldsymbol{h}_{i-1}\boldsymbol{U}_c + \boldsymbol{b}_c), \end{aligned} \tag{2}$$

where $\odot$ is the Hadamard product, $\sigma(x) := 1/(1 + e^{-x})$, $\boldsymbol{W}_l \in \mathbb{R}^{d \times N}$, $\boldsymbol{U}_l \in \mathbb{R}^{N \times N}$ are weight matrices and $\boldsymbol{b}_l$ are bias vectors, with $l \in \{o, f, j, c\}$.

Definition 1 (wd-GC layer)**.** Let $(\boldsymbol{A}_i)_{i \in \mathbb{Z}_T}$, $(\boldsymbol{X}_i)_{i \in \mathbb{Z}_T}$ be, respectively, the sequence of adjacency matrices and the sequence of vertex-feature matrices for the considered graph sequence $(\mathcal{G}_i)_{i \in \mathbb{Z}_T}$, with $\boldsymbol{A}_i \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ and $\boldsymbol{X}_i \in \mathbb{R}^{|\mathcal{V}| \times d}$ $\forall i \in \mathbb{Z}_T$. The *Waterfall Dynamic-GC* layer with M output nodes is the function wd-GC$_M$ with weight matrix $\boldsymbol{B} \in \mathbb{R}^{d \times M}$ defined as follows:

$$\text{wd-GC}_M : ((\boldsymbol{X}_i)_{i \in \mathbb{Z}_T}, (\boldsymbol{A}_i)_{i \in \mathbb{Z}_T}) \mapsto (\mathrm{GC}_{M,\boldsymbol{A}_i}^{\boldsymbol{B}}(\boldsymbol{X}_i))_{i \in \mathbb{Z}_T}. \tag{3}$$

The wd-GC layer can be seen as multiple copies of a standard GC layer, all of them sharing the same training weights. Then, the resulting training parameters are $d \times M$, independently of the length of the sequence.

In order to introduce the Concatenated Dynamic-GC layer, the definition of the *Graph of a Function* is now given: considering a function f from A to B, $[\mathrm{GF}\, f] : A \to A \times B$, $x \mapsto [\mathrm{GF}\, f](x) := (x, f(x))$. Namely, the GF operator transforms f into a function returning the concatenation between x and $f(x)$.

Definition 2 (cd-GC layer)**.** Let $(\boldsymbol{A}_i)_{i \in \mathbb{Z}_T}$, $(\boldsymbol{X}_i)_{i \in \mathbb{Z}_T}$ be, respectively, the sequence of adjacency matrices and the sequence of vertex-feature matrices for the considered graph sequence $(\mathcal{G}_i)_{i \in \mathbb{Z}_T}$, with $\boldsymbol{A}_i \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ and $\boldsymbol{X}_i \in \mathbb{R}^{|\mathcal{V}| \times d}$ $\forall i \in \mathbb{Z}_T$. A *Concatenated Dynamic-GC* layer with M output nodes is the function cd-GC$_M$ defined as follow:

$$\text{cd-GC}_M : ((\boldsymbol{X}_i)_{i \in \mathbb{Z}_T}, (\boldsymbol{A}_i)_{i \in \mathbb{Z}_T}) \mapsto ([\mathrm{GF}\,\mathrm{GC}_{M,\boldsymbol{A}_i}^{\boldsymbol{B}}](\boldsymbol{X}_i))_{i \in \mathbb{Z}_T} \tag{4}$$

where $[\mathrm{GF}\,\mathrm{GC}_{M,\boldsymbol{A}_i}^{\boldsymbol{B}}](\boldsymbol{X}_i) \in \mathbb{R}^{|\mathcal{V}| \times (M+d)}$.

Intuitively, cd-GC is a layer made of T copies of GC layers, each copy acting on one instant of the sequence. Each output of the T copies is then concatenated with its input, thus resulting in a sequence of graph-convoluted features together with the vertex-features matrix. Such *skip-connections* of the vertex-input features counters the possible *blurring* of the graph features due to the GC units alone. Indeed, by looking at Eq. (1) it can be seen that GC recombines the vertex-features matrix by means of the re-normalized adjacency $\hat{\boldsymbol{A}}$, thus *averaging* each vertex feature with its neighbourhood. This behavior might be counter-productive in a graph with high connectivity, to the extreme case that in a *complete graph* (i.e. a graph with $[\tilde{\boldsymbol{A}}]_{i,j} = 1\ \forall i, j$) all the recombined vertex features are equal for all the vertices, since all the elements of $\hat{\boldsymbol{A}}$ are the same. The weights $\boldsymbol{B} \in \mathbb{R}^{d \times M}$ are shared among the T copies. The number of learnable parameters of this layer is $d \times (d + M)$, independently of the number of steps in the sequence $(\mathcal{G}_i)_{i \in \mathbb{Z}_T}$.

Note that both the input and the output of wd-GC and cd-GC are sequences of matrices (loosely speaking, third order tensors).

Three additional layers will be defined now. These layers will help in reducing the clutter with the notation when, in Sections 3.2 and 3.3, the network architectures used to solve the semi-supervised classification of sequences of vertices and the supervised classification of sequences of graphs will be introduced. They are: (i) the recurrent layer used to process in a parallel fashion the convoluted vertex features, (ii) the two final layers (one per task) used to map the previous layers outputs into k-class probability vectors.

Definition 3 (v-LSTM layer)**.** Consider $(\boldsymbol{Z}_i)_{i \in \mathbb{Z}_T}$ with $\boldsymbol{Z}_i \in \mathbb{R}^{L \times M}$, the *Vertex LSTM* layer with N output nodes is given by the function v-LSTM$_N$:

$$\text{v-LSTM}_N : (\boldsymbol{Z}_i)_{i \in \mathbb{Z}_T} \mapsto \begin{pmatrix} \mathrm{LSTM}_N((\boldsymbol{V}_1' \boldsymbol{Z}_i)_{i \in \mathbb{Z}_T}) \\ \vdots \\ \mathrm{LSTM}_N((\boldsymbol{V}_L' \boldsymbol{Z}_i)_{i \in \mathbb{Z}_T}) \end{pmatrix} \in \mathbb{R}^{L \times N \times T}, \tag{5}$$

the ith and jth vertices and the edge has weight w_{ij}. In the case of an unweighted graph, $w_{ij} = 1$.

where $\boldsymbol{V}_p$ is the isometric embedding[3] of $\mathbb{R}$ into $\mathbb{R}^L$ defined as $[\boldsymbol{V}_p]_{i,j} = \delta_{ip}$ and δ is the Kronecker delta function. The training weights are shared among the L copies of the LSTMs.

Definition 4 (vs−FC layer)**.** Consider $(\boldsymbol{Z}_i)_{i\in\mathbb{Z}_T}$ with $\boldsymbol{Z}_i \in \mathbb{R}^{L\times N}$, the *Vertex Sequential Fully Connected* layer with k output nodes is given by the function vs-FC$_k$, parameterized by the weight matrix $\boldsymbol{W} \in \mathbb{R}^{N\times k}$ and the bias matrix $\mathbb{R}^{L\times k} \ni \boldsymbol{B} := (\boldsymbol{b}', \ldots, \boldsymbol{b}')'$:

$$\text{vs-FC}_k : (\boldsymbol{Z}_i)_{i\in\mathbb{Z}_T} \mapsto (\operatorname{softmax}(\boldsymbol{Z}_i\boldsymbol{W} + \boldsymbol{B}))_{i\in\mathbb{Z}_T} \tag{6}$$

with $\operatorname{softmax}(\boldsymbol{Z}_i\boldsymbol{W} + \boldsymbol{B}) \in \mathbb{R}^{L\times k}$.

Definition 5 (gs−FC layer)**.** Consider $(\boldsymbol{Z}_i)_{i\in\mathbb{Z}_T}$ with $\boldsymbol{Z}_i \in \mathbb{R}^{L\times N}$, the *Graph Sequential Fully Connected* layer with k output nodes is given by the function gs-FC$_k$, parameterized by the weight matrices $\boldsymbol{W}_1 \in \mathbb{R}^{N\times k}$, $\boldsymbol{W}_2 \in \mathbb{R}^{1\times L}$ and the bias matrices $\mathbb{R}^{L\times k} \ni \boldsymbol{B}_1 := (\boldsymbol{b}', \ldots, \boldsymbol{b}')'$ and $\boldsymbol{B}_2 \in \mathbb{R}^{1\times k}$:

$$\text{gs-FC}_K : (\boldsymbol{Z}_i)_{i\in\mathbb{Z}_T} \mapsto (\operatorname{softmax}(\boldsymbol{W}_2 \operatorname{ReLU}(\boldsymbol{Z}_i\boldsymbol{W}_1 + \boldsymbol{B}_1) + \boldsymbol{B}_2))_{i\in\mathbb{Z}_T} \tag{7}$$

with $\operatorname{softmax}(\boldsymbol{W}_2 \operatorname{ReLU}(\boldsymbol{Z}_i\boldsymbol{W}_1 + \boldsymbol{B}_1) + \boldsymbol{B}_2) \in \mathbb{R}^{1\times k}$.

Informally: (i) the v−LSTM layer acts as L copies of LSTM, each one evaluating the sequence of one row of the input tensor $(\boldsymbol{Z}_i)_{i\in\mathbb{Z}_T}$; (ii) the vs−FC layer acts as $L \times T$ copies of a Fully Connected layer (FC, [33]), all of the copies sharing the parameters and each one of them returning a k-class probability vectors thanks to the softmax activation; (iii) the gs−FC layer acts as T copies of two FC layers with softmax-ReLU activation, all the copies sharing the parameters. This layer outputs one k-class probability vector for each step in the input sequence, thanks to the softmax activation. Note that both the input and the output of vs−FC and v−LSTM are sequences of matrices, while for gs−FC the input is a sequence of matrices and the output is a sequence of vectors.

3.2. Semi-supervised classification of sequence of vertices

Definition 6 (Semi-Supervised Classification of Sequence of Vertices)**.** Let $(\mathcal{G}_i)_{i\in\mathbb{Z}_T}$ be a sequence of T graphs each one made of $|\mathcal{V}|$ vertices and $(\boldsymbol{X}_i)_{i\in\mathbb{Z}_T}$ the related sequence of vertex-features matrices.

Let $(\boldsymbol{P}_i^{\text{Lab}})_{i\in\mathbb{Z}_T}$ be a sequence of diagonal projectors on the vector space $\mathbb{R}^{|\mathcal{V}|}$. Define the sequence $(\boldsymbol{P}_i^{\text{Unlab}})_{i\in\mathbb{Z}_T}$ by means of $\boldsymbol{P}_i^{\text{Unlab}} := \boldsymbol{I}_{|\mathcal{V}|} - \boldsymbol{P}_i^{\text{Lab}}$, $\forall i \in \mathbb{Z}_T$; i.e. $\boldsymbol{P}_i^{\text{Lab}}$ and $\boldsymbol{P}_i^{\text{Unlab}}$ identify the labeled and unlabeled vertices of $\mathcal{G}_i$, respectively. Moreover, let $(\boldsymbol{Y}_i)_{i\in\mathbb{Z}_T}$ be a sequence of T matrices with $|\mathcal{V}|$ rows and k columns, satisfying the property $\boldsymbol{P}_i^{\text{Lab}}\boldsymbol{Y}_i = \boldsymbol{Y}_i$, where the jth row of the ith matrix represents the one-hot encoding of the k-class label of the jth vertex of the ith graph in the sequence, with the jth vertex being a labeled one. Then, a *semi-supervised classification of a sequence of vertices* consists in learning a function f such that $\boldsymbol{P}_j^{\text{Lab}} f((\mathcal{G}_i)_{i\in\mathbb{Z}_T}, (\boldsymbol{X}_i)_{i\in\mathbb{Z}_T})_j = \boldsymbol{Y}_j$ and $\boldsymbol{P}_j^{\text{Unlab}} f((\mathcal{G}_i)_{i\in\mathbb{Z}_T}, (\boldsymbol{X}_i)_{i\in\mathbb{Z}_T})_j$ is the right labeling for the unlabeled vertices for each $j \in \mathbb{Z}_T$.

To address the above task, the networks defined by the following functions are proposed:

$$\text{v\_wd-GC\_LSTM}_{M,N,k} : \quad \text{vs-FC}_k \circ \text{v-LSTM}_N \circ \text{wd-GC}_M, \tag{8a}$$

$$\text{v\_cd-GC\_LSTM}_{M,N,k} : \quad \text{vs-FC}_k \circ \text{v-LSTM}_N \circ \text{cd-GC}_M, \tag{8b}$$

[3] An isometric embedding f of a vector space V to a vector space W, with $\dim(V) < \dim(W)$, is a linear map between V and W such that $\|f(v)\| = \|v\|$, $\forall v \in V$.

where $\circ$ denote the function composition. Both the architectures take $((\boldsymbol{X}_i)_{i\in\mathbb{Z}_T}, (\boldsymbol{A}_i)_{i\in\mathbb{Z}_T})$ as input and produce a sequence of matrices whose row vectors are the probabilities of each vertex of the graph: $(\boldsymbol{Z}_i)_{i\in\mathbb{Z}_T}$ with $\boldsymbol{Z}_i \in \mathbb{R}^{|\mathcal{V}|\times k}$. For the sake of clarity, in the rest of the paper, the networks defined by Eqs. (8a) and (8b) will be called *Waterfall Dynamic-GCN*(WD−GCN) and *Concatenated Dynamic-GCN* (CD−GCN), respectively.

As it can be seen in Fig. 2a, the wd−GC layer acts as T copies of a regular GC layer, each one working on one instant of the sequence. The output of this first layer is then processed by the v−LSTM layer, which acts as $|\mathcal{V}|$ copies of the returning sequence-LSTM layer, each one working on one vertex of the sequence of graphs. The last layer, which produces the k-class probability vector for each vertex and for each instant of the sequence, can be seen as $|\mathcal{V}| \times T$ of a FC layer. Fig. 2b shows the CD−GCN architecture; the interpretation is analogous of that of WD−GCN, where now the v−LSTM unit takes as input both the graph convolutional output and the vertex input features thanks to the cd−GC layer.

Since the output layers return probability vectors, the weights of the architectures can be learned using gradient descent methods, employing as a loss function the *cross entropy*, evaluated only on the labeled vertices:

$$-\sum_{t\in\mathbb{Z}_T}\sum_{c\in\mathbb{Z}_k}\sum_{v\in\mathbb{Z}_{|\mathcal{V}|}} [\boldsymbol{Y}_t]_{v,c} \log[\boldsymbol{P}_t^{\text{Lab}}\boldsymbol{Z}_t]_{v,c}, \tag{9}$$

with the convention that $0 \times \log 0 = 0$.

3.3. Supervised classification of sequence of graphs

Definition 7 (Supervised Classification of Sequence of Graphs)**.** Let $(\mathcal{G}_i)_{i\in\mathbb{Z}_T}$ be a sequence of T graphs each one made of $|\mathcal{V}|$ vertices and $(\boldsymbol{X}_i)_{i\in\mathbb{Z}_T}$ the related sequence of vertex-features matrices. Moreover, let $(\boldsymbol{y}_i)_{i\in\mathbb{Z}_T}$ be a sequence of T one-hot encoded k-class labels, i.e. $\boldsymbol{y}_i \in \{0, 1\}^k$. Then, *graph-sequence classification task* consists in learning a predictive function f such that $f((\mathcal{G}_i)_{i\in\mathbb{Z}_T}, (\boldsymbol{X}_i)_{i\in\mathbb{Z}_T}) = (\boldsymbol{y}_i)_{i\in\mathbb{Z}_T}$.

The proposed architectures are defined by the following functions:

$$\text{g\_wd-GC\_LSTM}_{M,N,k} : \quad \text{gs-FC}_k \circ \text{v-LSTM}_N \circ \text{wd-GC}_M, \tag{10a}$$

$$\text{g\_cd-GC\_LSTM}_{M,N,k} : \quad \text{gs-FC}_k \circ \text{v-LSTM}_N \circ \text{cd-GC}_M, \tag{10b}$$

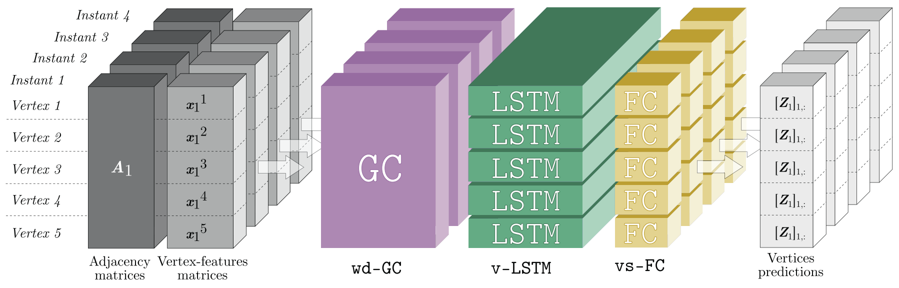
The two architectures take as input $((\boldsymbol{X}_i)_{i\in\mathbb{Z}_T}, (\boldsymbol{A}_i)_{i\in\mathbb{Z}_T})$. The output of wd−GC and cd−GC is processed by a v−LSTM, resulting in a $|\mathcal{V}| \times N$ matrix for each step in the sequence. It is a gs−FC duty to transform this vertex-based prediction into a graph based prediction, i.e. to output a sequence of k-class probability vectors $(\boldsymbol{z}_i)_{i\in\mathbb{Z}_T}$. Again, WD−GCN and CD−GCN will be used to refer to the networks defined by Eqs. (10a) and (10b), respectively.

Fig. 3a and b show the two architectures WD−GCN and CD−GCN respectively. The same interpretation given for Fig. 2a and b holds also here, with the only difference being in the last layer, since now it has to return a k-class probability vector for each instant of the sequence. It can be seen as the composition of two FC layers: the first one working on each vertex for every instant and the next one working on all the vertices at a given instant.

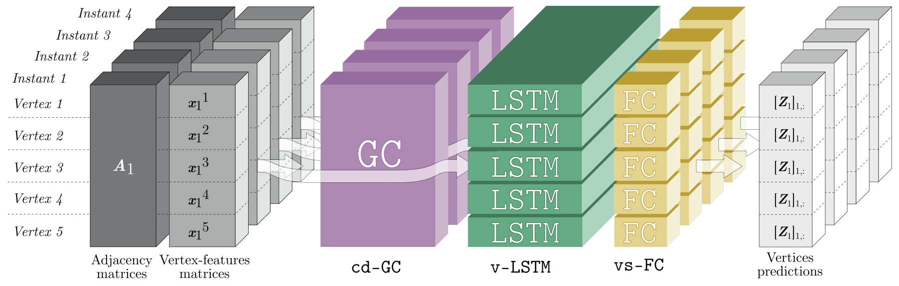
Also under this setting the training can be performed by means of gradient descent methods, with the cross entropy as loss function:

$$-\sum_{t\in\mathbb{Z}_T}\sum_{c\in\mathbb{Z}_k} [\boldsymbol{y}_t]_c \log[\boldsymbol{z}_t]_c, \tag{11}$$

with the convention $0 \times \log 0 = 0$.

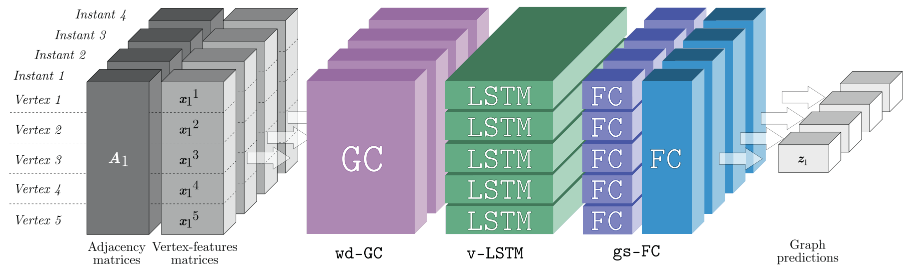


(a) `WD-GCN` for the classification of sequences of vertices.

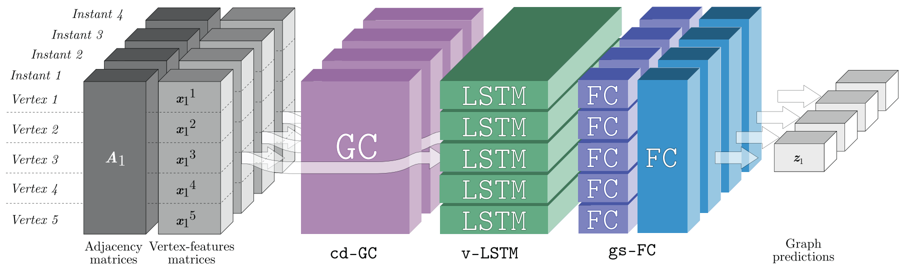


(b) `CD-GCN` for the classification of sequences of vertices.

Fig. 2. The figure shows the network architectures presented in Section 3.2 for the semi-supervised classification of sequences of vertices. In these pictorial representations, all of them work on sequences of four graphs composed of five vertices, i.e. $(\mathcal{G}_i)_{i\in\mathbb{Z}_4}$, $|\mathcal{V}| = 5$.
(a) The `wd-GC` layer acts as four copies of a regular `GC` layer, each one working on one instant of the sequence. The output of this first layer is processed by the `v-LSTM` layer that acts as five copies of the *returning sequence*-`LSTM` layer, each one working on a vertex of the graphs. The last layer, which produces the *k*-class probability vector for each vertex and for each instant of the sequence, can be seen as 5×4 copies of a `FC` layer.
(b) The `cd-GC` and the `v-LSTM` layers work as the `wd-GC` and the `v-LSTM` of the Fig. 2a, the only difference being that `v-LSTM` works both on graph convolutional features, as well as plain vertex features, due to the fact that `cd-GC` produces their concatenation.



(a) `WD-GCN` for the classification of sequences of graphs.



(b) `CD-GCN` for the classification of sequences of graphs.

Fig. 3. The figure shows the network architectures presented in Sections 3.3 for the supervised classification of sequences of graphs. All of them work on sequences of four graphs composed of five vertices, i.e. $(\mathcal{G}_i)_{i\in\mathbb{Z}_4}$, $|\mathcal{V}| = 5$. The networks of (a) and (b) differ from their counterparts of Fig. 2 only on the last `gs-FC` layer, since in this case the networks shall produce one *k*-class probability vector prediction for each graph.

3.4. Computational complexity

Considering a sparse implementation for the `wd-GC` and `cd-GC` layers, the time complexities of the proposed networks are linear in the number of graph edges and the length of the sequence. Indeed, let *F*, *G*, *L*, *K*, *T*, *E*, *V* be the number of input features, `wd-GC` nodes, `v-LSTM` nodes, output classes, time steps, maximum edges and vertices. Consider first v_wd-GC_LSTM$_{G,L,K}$, i.e. `WD-GCN` for a semi-supervised classification of a sequence of vertices. Since such network architecture is composed of wd-GC$_G$, v-LSTM$_L$ and vs-FC$_K$, the time complexity of each component is here reported:

Table 1
The table below summarizes the datasets considered in the experiments.

Task	Dataset	Nodes	Time steps	Vertex features	Prediction classes
Vertex-focused (semi-supervised classification)	DBLP	500	10	70	6
		5000	10	70	6
		20000	10	70	6
	CIAW	92	20	64	5
	Synthetic	500	20	1	2
Graph-focused (supervised classification)	CAD-120	34	1298	24	10
	HDM05	31	1654	6	19

- wd-GC$_G$: each one of the T GC units has a computational complexity of EFG [14], thus leading to a grand total of $EFGT$;
- v-LSTM$_L$: each one of the T recurrent iteration of the V LSTM building units requires L^2+LG operations for each one of the 4 LSTM internal gates, thus leading to a grand total of $4VLT(G+L)$;
- vs-FC$_K$: each one of the fully connected unit for each of the V vertices and the T steps of the sequences requires LK operations, giving $VLKT$ total operations.

Similar arguments hold also for the other architectures introduced in this paper; hence the computational complexity of the networks are given by:

- Semi-supervised classification of sequences of vertices
  - WD-GCN $\rightarrow$ $EFGT+4VLT(G+L)+VLKT$;
  - CD-GCN $\rightarrow$ $EFGT+4VLT(G+F+L)+VLKT$;
- Supervised classification of sequences of graphs
  - WD-GCN $\rightarrow$ $EFGT+4VLT(G+L)+VLKT+VKT$;
  - CD-GCN $\rightarrow$ $EFGT+4VLT(G+F+L)+VLKT+VKT$;

Since in many real world scenarios $V \leq E$, the above claim follows.

4. Experimental results

4.1. Datasets

The proposed architectures have been tested on the five datasets summarized in Table 1, namely one synthetic and two real world datasets (DBLP, CIAW) for the *vertex-focused* applications and two real world datasets for the *graph-focused* applications (CAD-120, HDM05).

DBLP. It has been considered a subset of the public available DBLP[4], a vertex-focused dataset, described in [27]. Conferences from six research communities, including artificial intelligence and machine learning, algorithm and theory, database, data mining, computer vision and information retrieval, have been taken into account. The co-author relationships from 2001 to 2010 have been considered and data belonging to each year has been organized in a graph form. Each author represents a node in the network and an edge between two nodes exists if two authors have collaborated on a paper in the considered year. Note that the resulting adjacency matrix is *unweighted*. The task to be tackled with this dataset is the prediction of each author's research community, by leveraging the dynamic evolution of the collaborations between the authors through the years in a semi-supervised fashion. This is a *vertex-focused* task.

The node features are extracted from each temporal instant using DeepWalk [23] and correspond to a 64-dimensional vector. DeepWalk uses truncated random walks to learn latent representations of vertices in a graph thus embedding the nodes in a vector space. This approach is well-known and largely employed, because it allows to build representations (features) that can help techniques that are not able to properly exploit graph structure, such as fully connected (feed-forward) neural networks or LSTM, to achieve better results. The node features have been augmented by adding the number of articles published by the authors in each of the communities, obtaining a 70-dimensional features vector.

The dataset is made of 25215 authors across the ten years under analysis. Each year 4786 authors appear on average and 114 authors appear all the years, with an average of 1594 authors appearing on two consecutive years.

The 500/5000/20000 authors with the highest number of connections during the analyzed 10 years have been considered, i.e. the 500/5000/20000 vertices among the total 25215 with the highest $\sum_{t\in\mathbb{Z}_{10}}\sum_i[\boldsymbol{A}_t]_{i,j}$, with $\boldsymbol{A}_t$ the adjacency matrix at the tth year. If one of the selected authors does not appear in the tth year, its feature vector is set to zero.

The final dataset is composed of 10 vertex-features matrices in $\mathbb{R}^{x\times 70}$ and 10 adjacency matrices belonging to $\mathbb{R}^{x\times x}$, with $x\in\{500, 5000, 20000\}$. Moreover, each vertex belongs to one of the 6 classes.[5]

CIAW. The *Contacts In A Workplace* (CIAW)[6] is a vertex-focused dataset of [35] that contains the temporal network of contacts between individuals measured in an office building in France, from June 24, to July 3, 2013. Each individual was wearing a sensor able to record the interaction with another individual within 1.5 m. Any interaction lasting more than 20 s was considered as a *contact* between the two people. For each 20 s interval between June 24, and July 3, 2013, all the contacts occurring between the surveyed individuals have been recorded. Each of the individuals is further characterized by his or her *department name*.

Similarly to the dataset DBLP, the task is to predict each individual's department, by leveraging the historical sequence of their interactions in a semi-supervised fashion. Again, this is a *vertex-focused* task.

The interaction data have been downsampled by grouping it in 20 temporally equally spaced snapshots.[7] A sequence of 20 graphs has been built as follows: (i) the set of vertices (shared between all the graphs) is composed of the collection of all the individuals, (ii) if two individuals interact in the ith snapshot, the respective vertices are connected by an edge in the ith graph. For all the vertices of every graph of each snapshot, a 64 dimensional representation by means of DeepWalk has been built.

Each person's department has been considered as the target label for the semi-supervised classification, for a grand total of 5 classes. The dataset is made of 92 individuals across the 20 snapshots. For each snapshot, 60 individuals appear on average and 4 individuals appear in all the snapshots, with an average of 49 people appearing on two consecutive snapshots.

[4] http://dblp.uni-trier.de/xml/.

[5] Note that even if the label of a vertex does not change during the sequence, all the tested architectures have to provide a vertex label for each year in the sequence.

[6] http://www.sociopatterns.org/datasets/contacts-in-a-workplace/.

[7] Note that the downsampling is only for the purpose of experimental comparison and that other aggregations may significantly alter the results [36].

Table 2
The two configurations used to generate the synthetic dataset. *Configuration 2* has much more noise in the graph structured data than *Configuration 1*.

	$p_{\text{same dep}}$	$p_{\text{same inter}}$	p_{other}	$p_{\text{change inter}}$	p_{move}
Configuration 1	0.5	0.4	0.05	0.05	0.05
Configuration 2	0.5	0.05	0.4	0.05	0.05

The resulting dataset is made of 20 vertex-feature matrices (in $\mathbb{R}^{92\times 64}$), 20 adjacency matrices (in $\mathbb{R}^{92\times 92}$) and each vertex belongs to one of the 5 classes.[5]

Synthetic. Similarly to CIAW, the synthetic vertex-focused dataset aims to simulate the behavior of employees in a workplace environment. There are 500 employees, each one belonging to one of two possible departments. Moreover, each employee is characterized by one boolean feature describing his or her working interests: when the feature is zero it means that the employee would like to work for the first of the two departments, while, when it is equal to one, he or she would like to work for the second. Employees can interact with each other over time, as well as change working department and working interests. The task to solve is to predict the current department of each of the employees by looking at their historical interactions with the other employees and the time series of their interests.

The dataset has been created in the following way: the 500 employees have been initially assigned a department as well as a working interest uniformly at random. The graph describing their interactions has been built as follows: (i) two people belonging to the same department have a probability $p_{\text{same dep}}$ to be connected; (ii) two people not belonging to the same department but having the same working interest have a probability $p_{\text{same inter}}$ to be connected; (iii) two people, neither sharing department nor interest, have a probability p_{other} to be connected. The previous three items completely identify the initial time step in the sequence; the following ones are identified by the following sequential update rules: (i) each employee, not belonging to the department of his or her liking, has a probability $p_{\text{change inter}}$ of changing working interest into the one corresponding to its current department; (ii) each employee, not belonging to the department of his or her liking, has a probability p_{move} of changing department. Experiments have been done using the two configurations reported in Table 2, where each sequence is made of 20 steps.

The resulting dataset is made of 20 vertex-feature matrices (in $\mathbb{R}^{500\times 1}$), 20 adjacency matrices (in $\mathbb{R}^{500\times 500}$) and each vertex belongs to one of the 2 classes.

CAD-120. This graph-focused dataset[8] is made of 122 RGB-D videos corresponding to 10 high-level human activities [37]. Each video is annotated with sub-activity labels, object affordance labels, tracked human skeleton joints and tracked object bounding boxes. The 10 sub-activity labels are: *reaching, moving, pouring, eating, drinking, opening, placing, closing, scrubbing* and *null*. All the data related to the detection of sub-activities have been considered, i.e. no object affordance data have been considered. Given one video (with its annotated data) one wants to predict the sub-activity for each frame within the video. Note that detecting the sub-activities is a challenging problem as it involves complex interactions, since humans can interact with multiple objects during a single activity. One can imagine to tackle this problem by extracting graph-structured data from the tracked positions of both human skeleton joints and objects. This implies that each video can be seen as a graph sequence as presented at the beginning of Section 3.1, one graph per frame and the detection of the sub-activity for each of the frames as the prediction of a target label attached to each graph in the sequence, i.e. a *graph-focused* application.

Each one of the 10 high-level activities is characterized by one person, whose 15 joints are tracked (in position and orientation) in the 3D space for each frame of the sequence. In each high-level activity there appears a variable number of objects, for which are registered their bounding boxes in the video frame together with the transformation matrix matching extracted SIFT features [38] from the frame to the ones of the previous frame. 19 are objects involved.

A graph for each video frame has been built: the vertices are the 15 skeleton joints plus the 19 objects, while the *weighted* adjacency matrix has been derived by exploiting Euclidean distance. Precisely, for two skeleton joints, the edge weight is given by the Euclidean distance between their 3D positions; for two objects, it is the 2D distance between the centroids of their bounding boxes; for an object and a skeleton joint, it is the 2D distance between the centroid of the object bounding box and the skeleton joint projection into the 2D video frame. All the distances have been scaled between zero and one. When an object does not appear in a frame its related row and column in the adjacency matrix and its corresponding features in the vertex-features matrix are set to zero.

Each vertex is characterized by 24 features:

- 9 numbers representing the 3×3 orthonormal matrix characterizing the rotation of a joint's local coordinates with respect to a reference coordinate system;
- 1 boolean number representing the confidence value of a joint orientation measurement—i.e. it is 1 when tracking seems to work, 0 if tracking fails and the measurement apparatus is uncertain about the output;
- 3 numbers representing the position of a joint in the 3D space;
- 1 boolean number representing the confidence value of a joint position measurement;
- 6 numbers representing the transform matrix matching the SIFT features of an object to the ones belonging to previous frame;
- 4 numbers representing the coordinates of the bounding box of an object within the frame.

Note that object-related features are set to zero for joint vertices and vice versa.

Since the videos have different lengths, all the sequences have been zero-padded to match the longest one, which has 1298 frames. Moreover, the zero-padded steps are not considered both during training and testing, i.e. both the loss function evaluation and the metrics disregard the padded steps. Note that no temporal segmentation data have been used: all the experimented architecture worked directly at the frame level with no ground-truth information regarding the similarity between consecutive frames, i.e. segment of consecutive frames belonging to the same ground-truth sub-activity label.

Finally, the feature columns have been standardized. The resulting dataset is composed of 122×1298 vertex-feature matrices belonging to $\mathbb{R}^{34\times 24}$, 122×1298 adjacency matrices (in $\mathbb{R}^{34\times 34}$) and each graph belongs to one of the 10 classes.

HDM05. This graph-focused dataset[9] of Müller et al. [39] contains more than three hours of systematically recorded motion capture data. The motion sequences are performed by five non-professional actors. Most of the sequences have been performed several times by all five actors according to the guidelines fixed in a script. The script consists of five *parts*, where each part is subdivided into several *scenes*, for a grand total of 20 scenes. Since

[8] http://pr.cs.cornell.edu/humanactivities/data.php.

[9] http://resources.mpi-inf.mpg.de/HDM05/.

one of these 20 scenes has been performed only once, is has been dropped.

Similarly to the CAD−120 dataset, the task is to classify each frame according to the *scene* it represents by exploiting graph-structured data extracted from the tracked motion capture data, thus resulting in a *graph-focused* application. Each person is characterized by 31 body parts, which have been tracked both in position and orientation for each frame. The motion capture data have been downsampled by considering only one frame every 24 and a weighted graph of 31 vertices (the body parts) has been built for each frame. The graph edges have been built employing the same approach proposed for the CAD−120 joint-joint edges. The considered vertex features are the 3D position of the corresponding body parts, together with the 3 Euler angles representing their orientation, thus resulting in a 6-dimensional feature vector.

Since the videos have different lengths, the sequences have been zero-padded to match the longest one, which has 1654 frames. Note that also in this case the zero-padded steps are not considered both during training and testing. It is important to underline that also for this experiment, no temporal segmentation data have been used.

Finally, the features columns have been standardized. The resulting dataset is composed of 88×1654 vertex-feature matrices belonging to $\mathbb{R}^{31\times6}$, 88×1654 adjacency matrices (in $\mathbb{R}^{31\times31}$) and each graph belongs to one of the 19 classes.

4.2. Experimental settings

In the experiments, the results achieved by the proposed architectures have been compared with those obtained by other baseline networks (see Section 4.3 for a full description of the chosen baselines).

For the baselines that are not able to explicitly exploit sequentiality in the data, the temporal dimension of all the sequences has been flattened, thus considering the same point at two different times as two different samples.

Unless otherwise stated, the hyper-parameters[10] of all the networks (in terms of number of nodes of each layer and dropout rate[11]) have been appropriately tuned by means of a grid[12] approach. The performances have been assessed employing 10 iterations of Monte Carlo Cross-Validation[13], preserving the percentage of samples for each class. It is important to underline that to keep the experiments as fair as possible, the 10 train/test sets are generated once and then used to evaluate all the architectures. Moreover, the training phase has been performed using the Adam optimizer [40] for a maximum of 100 epochs. For each network, the epoch where the model achieved the best performance on the validation set has been selected, and the performance of corresponding trained model has been assessed on the test set.

To assess the performances of all the considered architectures, the metrics *Accuracy* and *Unweighted F1 Measure*[14] have been employed. Note that two ways of assessing the networks performances have been used for the DBLP and the CIAW datasets. Since in this case not all the vertices appear all the time in the graph sequences (i.e. in DBLP only 114 authors out of 25215 appear all the years, while in CIAW 4 people out of 60 appear in all the snapshots), the two aforementioned metrics on the test set have been evaluated in the following ways: (i) taking into account all the snapshots in the label sequences for all the vertices, regardless of whether or not the vertices appear in the snapshot taken into account; (ii) taking into account for each snapshot in the label sequences just the vertices appearing in the snapshot.

Finally, the *training time* of each architecture with the best hyper-parameter configurations has been analyzed.[15] The training time evaluation has been performed on 2×10-core Intel Xeon 4114 @ 2.20 GHz, 64 GB RAM, with a single Nvidia Tesla P100 GPU. All the architectures have been implemented in Keras 1.2.2 with Tensorflow 1.11.0 as a backend.

4.3. Results

Since the techniques presented in this work are, to the best of the authors knowledge, the first attempts to provide general purpose neural architectures to address classification problems in the context of dynamic graph datasets, other general purpose techniques based on neural networks have been chosen. The baselines feature all the base elements of the proposed architectures.

In particular, for all the datasets one neural network for each of the following groups has been considered: (i) simple networks made of two fully connected layers; (ii) graph convolutional networks; (ii) networks made of a LSTM followed by a fully connected layer; (iv) networks made of a fully connected layer followed by a LSTM and another fully connected layer. In particular, (ii) and (iii) allow us to assess the performance of our new architectures against their building blocks, while (iv) allows to ascertain whether it is the increased depth of WD-GCN and CD-GCN the responsible for better results when compared to (iii).

DBLP. The approaches proposed in Section 3.2 (WD-GCN and CD-GCN) have been compared against the following baseline methodologies: (i) a GCN composed of two layers, (ii) a network made of two FC layers, (iii) a network composed of LSTM+FC, (iv) and a deeper architecture made of FC+LSTM+FC. Note that the FC is a Fully Connected layer; when it appears as the first layer of a network it employs a ReLU activation, a softmax activation is used when it is the last layer of a network, in order to get as output a k-class probability vector.

First of all, the 500 vertices version of the DBLP dataset has been considered. The test set contains 30% of the 500 vertices. Moreover, 20% of the remaining vertices have been used for validation purposes. It is important to underline that an unlabeled vertex remains unlabeled for all the years in the sequence, i.e. considering Definition 6, $\boldsymbol{P}_i^{\text{Lab}} = \boldsymbol{P}^{\text{Lab}}, \ \forall i \in \mathbb{Z}_T$.

Tables 3 and 4 show the best hyper-parameter configurations, together with the test results of all the evaluated architectures for the 500 vertices DBLP. In particular, Table 3 shows the results in which the metrics have been evaluated considering all the years in the label sequences for all the vertices, while Table 4 takes into account only the years of the label sequences where each vertex appears.

Figs. 4 and 5 show the confusion matrices for each of the best configurations of Tables 3 and 4 respectively. By looking at the confusion matrices, it is clear that both WD−GCN and CD−GCN achieved either equivalent or better results that the baseline architectures for each of the target classes, thus resulting in better

[10] A hyper-parameter is a parameter whose value is set before the learning process begins and with respect to which the loss function is not differentiable.

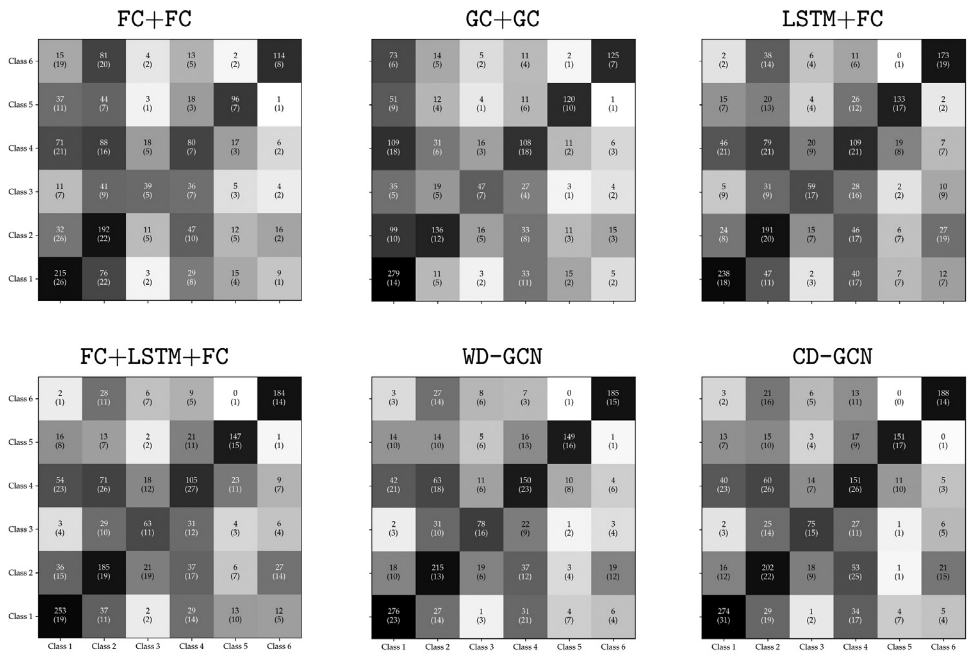
[11] Dropout is a regularization technique to reduce overfitting in a neural network. The term “dropout” refers to dropping out nodes in a neural network. The *dropout rate* is the percentage of nodes that are randomly dropped out.

[12] A possible approach to perform hyper-parameter optimization is grid search. This methodology consists in an exhaustive searching through a manually specified subset of the hyper-parameter space of the employed learning algorithm.

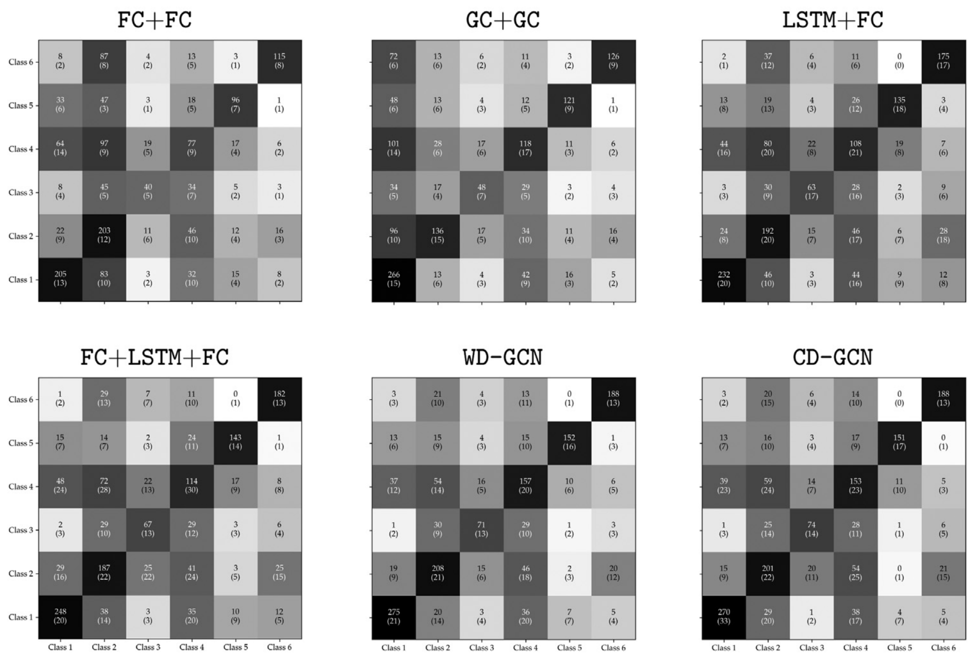
[13] This approach randomly selects (without replacement) some fraction of the data to build the training set and it assigns the rest of the samples to the test set. This process is repeated multiple times, generating (at random) new training and test partitions each time. Note that in our experiments, the training set is further split into training and validation.

[14] The Unweighted F1 Measure evaluates the F1 scores for each class and find their mean: $\frac{1}{k}\sum_{c\in\mathbb{Z}_k}\frac{2p_c r_c}{p_c+r_c}$, where p_c and r_c are the *precision* and the *recall* of the class c.

[15] The best configuration is the hyper-parameter configuration that generates the best results in terms of either Accuracy or Unweighted F1 Measure.



(a) Best configurations according to Accuracy



(b) Best configurations according to unweighted F1 measure

Fig. 4. Confusion matrices for the best configuration of each architecture of Table 3. Rows represent real target classes, while columns are the predicted ones. Each entry shows the number of vertices falling into the relative true class vs predicted class, averaged across the full 10-fold Monte Carlo cross-validation and rounded to the unit. In brackets, the corresponding standard deviation is shown.

predictions not only for the minority class (i.e. *Class 3*) but also for the ones bigger in size.

On the 500 vertices DBLP dataset, all the networks have also been tested by changing the ratio of the labeled vertices—i.e. the size of the training set—as follows: 20%, 30%, 40%, 50%, 60%, 70%, 80%. To obtain robust estimations, the performances have been averaged by means of 10 iterations of Monte Carlo Cross-Validation. Fig. 6 reports the results of these experiments, considering the best configuration for each of the architectures in Tables 3 and 4 and evaluating the performance metrics by: (a) taking into account all the years in the label sequences for all the vertices, regardless of whether or not the vertices appear in the year taken into account; (b) taking into account for each year in the label sequences just the vertices appearing in the year taken into account.

Finally, employing the hyper-parameters of the best configuration for each of the architectures in Tables 3 and 4, all the networks performance have been assessed on the much bigger 5000 and 20000 vertices versions of the DBLP dataset. Also in this case,

Table 3
Results of the evaluated architectures on a semi-supervised classification of a sequence of vertices employing the 500 vertices version of the DBLP dataset. Metrics have been evaluated considering all the years in the label sequences for all the vertices, regardless of whether or not the vertices appear in the year taken into account. A *Wilcoxon Test* shows a *p*-value $<0.6\%$ when comparing WD-GCN and CD-GCN against all the baselines for both the scores.

Network	Hyper-params	Grid	Accuracy		Unweighted F1 Measure	
			Best config.	Performance mean ± std	Best config.	Performance mean ± std
FC+FC	1st FC nodes:	{150, 200, 250, 300, 350, 400}	250	49.1% ± 1.1%	250	48.4% ± 1.1%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	50%		40%	
GC+GC	1st GC nodes:	{150, 200, 250, 300, 350, 400}	350	54.4% ± 1.2%	350	54.0% ± 1.7%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	50%		10%	
LSTM+FC	LSTM nodes:	{100, 150, 200, 300, 400}	100	60.2% ± 2.1%	100	60.2% ± 2.5%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	0%		0%	
FC+LSTM+FC	FC nodes:	{100, 200, 300, 400}	300	62.5% ± 2.3%	300	62.6% ± 2.5%
	LSTM nodes:	{100, 200, 300, 400}	300		300	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	50%		50%	
WD-GCN	wd-GC nodes:	{100, 200, 300, 400}	300	***70.1% ± 3.1%***	400	***69.8% ± 3.1%***
	v-LSTM nodes:	{100, 200, 300, 400}	300		300	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	50%		0%	
CD-GCN	cd-GC nodes:	{100, 200, 300, 400}	200	69.3% ± 3.4%	200	69.3% ± 3.1%
	v-LSTM nodes:	{100, 200, 300, 400}	100		100	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	50%		50%	

Table 4
Results of the evaluated architectures on a semi-supervised classification of a sequence of vertices employing the 500 vertices version of the DBLP dataset. Metrics have been evaluated considering for each year in the label sequences just the vertices appearing in the year taken into account. A *Wilcoxon Test* shows a *p*-value $\approx 0.5\%$ when comparing WD-GCN and CD-GCN against all the baselines for both the scores.

Network	Hyper-params	Grid	Accuracy		Unweighted F1 Measure	
			Best config.	Performance mean ± std	Best config.	Performance mean ± std
FC+FC	1st FC nodes:	{150, 200, 250, 300, 350, 400}	250	53.5% ± 1.2%	250	50.9% ± 1.5%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	40%		50%	
GC+GC	1st GC nodes:	{150, 200, 250, 300, 350, 400}	300	68.4% ± 2.0%	250	66.6% ± 3.4%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	50%		20%	
LSTM+FC	LSTM nodes:	{100, 150, 200, 300, 400}	100	62.5% ± 2.5%	100	62.2% ± 3.5%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	0%		0%	
FC+LSTM+FC	FC nodes:	{100, 200, 300, 400}	200	64.6% ± 2.4%	100	63.9% ± 2.7%
	LSTM nodes:	{100, 200, 300, 400}	100		200	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	50%		50%	
WD-GCN	wd-GC nodes:	{100, 200, 300, 400}	100	***77.4% ± 3.4%***	100	***76.9% ± 3.4%***
	v-LSTM nodes:	{100, 200, 300, 400}	300		300	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	40%		40%	

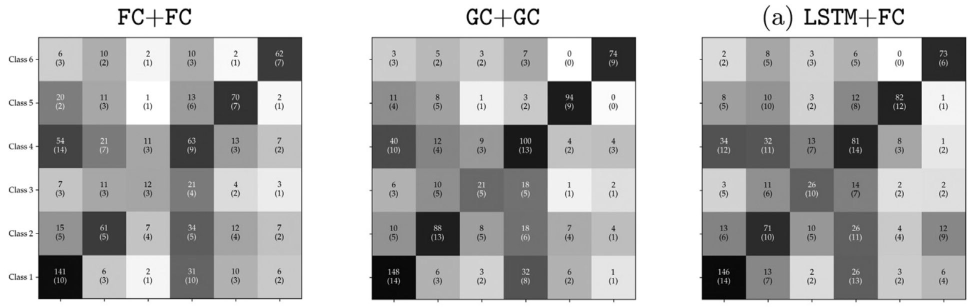
Table 5
Results of the best configuration for each architecture of Table 3 on semi-supervised classification of sequence of vertices employing the 500, 5000 and 20000 vertices versions of the DBLP dataset. Metrics have been evaluated considering all the years in the label sequences for all the vertices, regardless of whether or not the vertices appear in the year taken into account. A *Wilcoxon Test* shows *p*-value $<0.6\%$ when comparing WD-GCN and CD-GCN against all the baselines for both the scores.

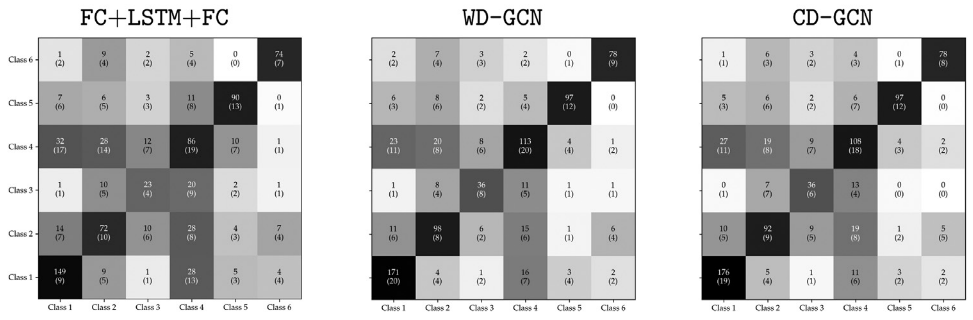
Network	Accuracy (mean ± std)			Unweighted F1 Measure (mean ± std)		
	DBLP 500	DBLP 5000	DBLP 20000	DBLP 500	DBLP 5000	DBLP 20000
FC+FC	49.1% ± 1.1%	38.1% ± 0.3%	31.7% ± 0.1%	48.4% ± 1.1%	34.6% ± 0.4%	24.5% ± 0.2%
GC+GC	54.4% ± 1.2%	41.4% ± 0.3%	27.2% ± 0.1%	54.0% ± 1.7%	39.8% ± 0.5%	24.3% ± 0.2%
LSTM+FC	60.2% ± 2.1%	56.7% ± 0.7%	50.2% ± 0.3%	60.2% ± 2.5%	56.0% ± 0.9%	49.6% ± 0.4%
FC+LSTM+FC	62.5% ± 2.3%	57.8% ± 0.9%	50.8% ± 0.4%	62.6% ± 2.5%	57.4% ± 0.9%	50.4% ± 0.5%
WD-GCN	***70.1% ± 3.1%***	***64.7% ± 1.1%***	***56.3% ± 0.7%***	***69.8% ± 3.1%***	***64.9% ± 1.3%***	***56.9% ± 0.9%***
CD-GCN	69.3% ± 3.4%	64.1% ± 0.7%	55.5% ± 0.3%	69.3% ± 3.1%	64.5% ± 0.7%	56.0% ± 0.4%

the test set is made of 30% of the vertices and the validation one is made of the remaining 20%. The results have been averaged by means of 10 iterations of Monte Carlo Cross-Validation. Tables 5 and 6 show the results of this experiment.

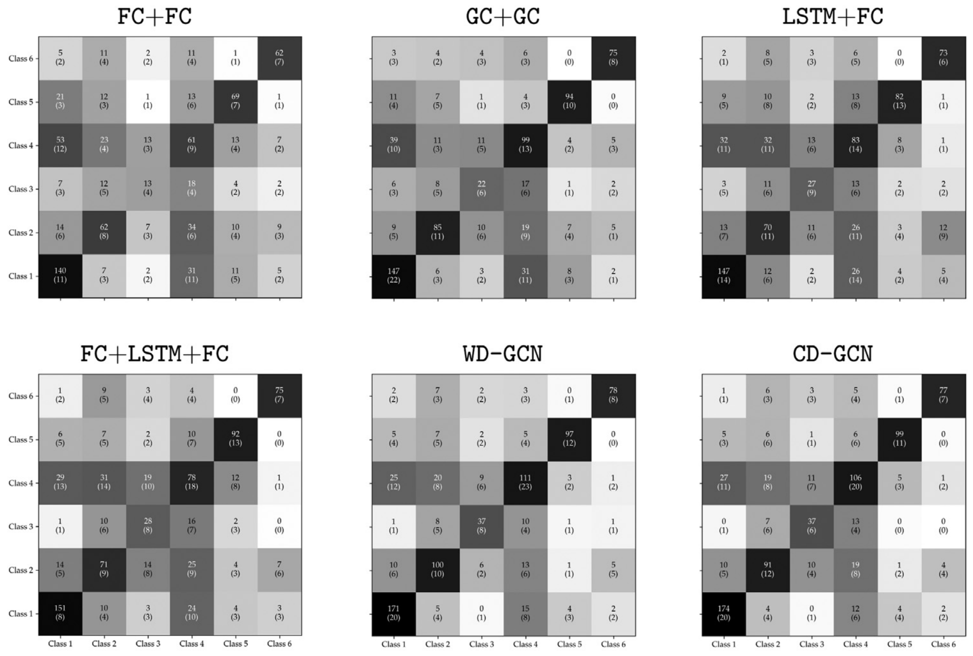
Both the proposed architectures have overcome the results achieved by the considered baselines, both when evaluating the performance metrics by (a) taking into account all the years in the label sequences for all the vertices, regardless of whether or not the vertices appear in the year taken into account; (b) taking into account, for each year in the label sequences, just the vertices appearing in the year taken into account.

Moreover, the WD-GCN and the CD-GCN performances are roughly equivalent in terms of Accuracy and Unweighted F1 Measure. Architectures such as GCNs and LSTMs are mostly likely limited for their inability to jointly exploit graph structure and long short-term dependencies (see also the part discussing the synthetic dataset within the Section 4.3). Note that the structure of the graphs appearing in the sequence is not exclusively conveyed by the DeepWalk vertex-features and it is effectively captured by the GC units. Indeed, the two layers-GCN has obtained better results with respect to those achieved by the two FC layers.





(b) Best configurations according to Accuracy



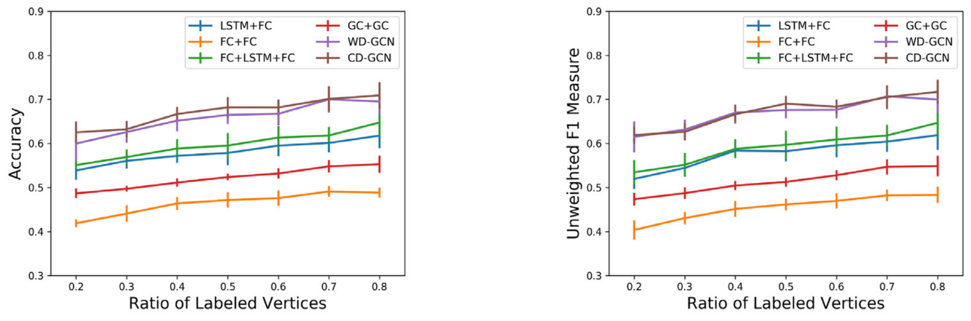
(c) Best configurations according to unweighted F1 measure

Fig. 5. Confusion matrices for the best configuration of each architecture of Table 4. Rows represent real target classes, while columns are the predicted ones. Each entry shows the number of vertices falling into the relative true class vs predicted class, averaged across the full 10-fold Monte Carlo cross-validation and rounded to the unit. In brackets, the corresponding standard deviation is shown.

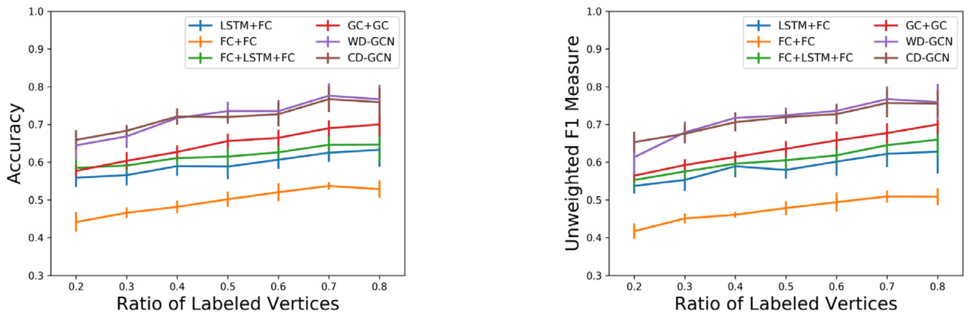
It is important to underline that the WD−GCN and the CD−GCN have achieved better performances with respect to the baselines not because they exploit a greater number of parameters, or since they are deeper, rather:

- The baseline architectures have achieved their best performances without employing the maximum of the allowed number of nodes, thus showing that their performance is unlikely to become better with an even greater number of nodes;
- The number of parameters of our approaches is significantly lower than the number of parameters of the biggest employed network: i.e. the best WD−GCN and CD−GCN have, respectively, 872206 and 163406 parameters, while the largest tested network is the FC+LSTM+FC with 1314006 parameters;
- The FC+LSTM+FC network has a comparable depth with respect to our approaches, but it has achieved lower performance.

Moreover, WD−GCN and CD−GCN have shown little sensitivity to the labeling ratio, consistently outperforming the baselines



(a) Best configurations of Table 3



(b) Best configurations of Table 4

Fig. 6. The figure shows the performances of the tested approaches (on the 500 vertices DBLP dataset) varying the ratio of the labeled vertices—i.e. size of the training set divided by the number of total samples. The vertical bars represent the standard deviation of the performances achieved in the 10 iterations of the Monte Carlo cross-validation. The top plots show Accuracy and Unweighted F1 Measure for the best configuration of each architecture of Table 3, taking into account all the years in the label sequences for all the vertices, regardless of whether or not the vertices appear in the year taken into account. The bottom plots show the same metrics for Table 4, this time evaluated by taking into account for each year in the label sequences just the vertices appearing in the year taken into account.

Table 6
Results of the best configuration for each architecture of Table 4 on semi-supervised classification of sequence of vertices employing the 500, 5000 and 20000 vertices versions of the DBLP dataset. Metrics have been evaluated considering for each year in the label sequences just the vertices appearing in the year taken into account. A *Wilcoxon Test* shows a *p*-value $\lesssim$0.5% when comparing WD-GCN and CD-GCN against all the baselines for both the scores.

Network	Accuracy (mean ± std)			Unweighted F1 Measure (mean ± std)		
	DBLP 500	DBLP 5000	DBLP 20000	DBLP500	DBLP 5000	DBLP 20000
FC+FC	53.5% ± 1.2%	58.4% ± 0.9%	62.5% ± 0.5%	50.9% ± 1.5%	57.0% ± 1.0%	61.0% ± 0.4%
GC+GC	68.4% ± 2.0%	71.9% ± 1.1%	73.7% ± 0.4%	66.6% ± 3.4%	71.8% ± 0.8%	73.9% ± 0.5%
LSTM+FC	62.5% ± 2.5%	68.4% ± 1.2%	69.7% ± 0.4%	62.2% ± 3.5%	67.6% ± 1.1%	68.9% ± 0.5%
FC+LSTM+FC	64.6% ± 2.4%	70.8% ± 1.2%	71.2% ± 0.4%	63.9% ± 2.7%	70.0% ± 1.0%	70.1% ± 0.5%
WD-GCN	***77.4% ± 3.4%***	79.1% ± 0.7%	77.5% ± 0.6%	***76.9% ± 3.4%***	78.7% ± 0.8%	76.9% ± 0.6%
CD-GCN	76.6% ± 3.4%	***80.2% ± 0.9%***	***80.5% ± 0.5%***	75.6% ± 3.8%	***79.7% ± 1.1%***	***80.2% ± 0.4%***

by more than 5 percentage points in both Accuracy and Unweighted F1 Measure, regardless of its value. Furthermore, they have achieved better performance than the baselines when tested also on bigger graphs, further demonstrating the robustness of our methods.

Finally, Fig. 7 represents the wall-clock training time cost in seconds for all the best architectures of Table 3 on the 500, 5000 and 20000 vertices versions of the DBLP dataset. It can be seen that both the WD-GCN and CD-GCN training times are comparable to these baselines. Moreover, when increasing the graph size, CD-GCN, which achieves results very similar to the WD-GCN, can be trained faster than FC+LSTM+FC, which is the best performer among the baselines.

CIAW. The performance of WD-GCN and CD-GCN has been measured against the same baseline as in the DBLP dataset. Again, the test set contains 30% of the vertices and 20% of the remaining ones have been used for validation purposes.

Tables 7 and 8 show the best hyper-parameter configurations together with the test results of all the evaluated architectures for the CIAW dataset. In particular, Table 7 shows the results in which the metrics have been evaluated considering all the snapshots in the label sequences for all the vertices, regardless of whether or not the vertices appear in the snapshot taken into account. On the other hand, Table 8 takes into account, for each snapshot in the label sequences, just the vertices appearing in the snapshot taken into account. Finally, Fig. 8 represents the wall-clock training time cost in seconds for all the best architectures of Table 7.

The considerations are similar to those made for the DBLP dataset: our techniques have outperformed all the baselines, with CD-GCN achieving statistically significative better results than the other architectures. Moreover, it can be seen that the training time of both of our techniques is very similar to the one of FC+LSTM+FC, which is the best performer among the baselines and even faster than the slower baseline networks, i.e. GC+GC.

Synthetic dataset. The performance of the WD-GCN and CD-GCN has been measured against the same baseline as in the DBLP dataset. Again, the test set contains 30% of the vertices and 20% of the remaining ones have been used for validation purposes.

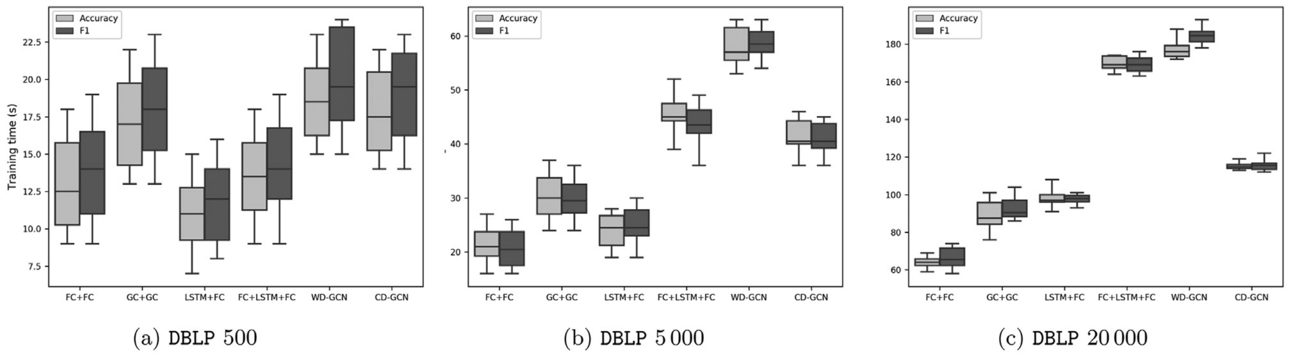


Fig. 7. The box plots represent the *training time* in seconds taken by the best configuration of each architecture of Table 3 on semi-supervised classification of sequence of vertices employing the 500, 5000 and 20000 vertices versions of the DBLP dataset.

Table 7
Results of the evaluated architectures on a semi-supervised classification of a sequence of vertices employing the CIAW dataset. Metrics have been evaluated considering all the snapshots in the label sequences for all the vertices, whether or not the vertices appear in the snapshot taken into account. A *Wilcoxon Test* shows *p*-value $<3\%$ for the Accuracy score and $<5\%$ for the Unweighted F1 Measure when comparing the best of the proposed architecture against all the baselines.

Network	Hyper-params	Grid	Accuracy		Unweighted F1 Measure	
			Best config.	Performance mean ± std	Best config.	Performance mean ± std
FC+FC	1st FC nodes:	{150, 200, 250, 300, 350, 400}	150	62.5% ± 3.6%	150	52.0% ± 2.4%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	20%		0%	
GC+GC	1st GC nodes:	{150, 200, 250, 300, 350, 400}	400	64.7% ± 3.5%	150	54.6% ± 3.9%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	0%		40%	
LSTM+FC	LSTM nodes:	{100, 150, 200, 300, 400}	100	79.3% ± 4.1%	100	66.7% ± 5.2%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	40%		20%	
FC+LSTM+FC	FC nodes:	{100, 200, 300, 400}	400	80.7% ± 3.5%	400	66.8% ± 5.4%
	LSTM nodes:	{100, 200, 300, 400}	300		30%	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	300		10%	
WD-GCN	wd-GC nodes:	{100, 200, 300, 400}	400	81.3% ± 2.7%	100	68.5% ± 9.0%
	v-LSTM nodes:	{100, 200, 300, 400}	200		100	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	10%		10%	
CD-GCN	cd-GC nodes:	{100, 200, 300, 400}	400	***81.9% ± 4.0%***	400	***69.0% ± 5.6%***
	v-LSTM nodes:	{100, 200, 300, 400}	200		200	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	30%		20%	

Table 8
Results of the evaluated architectures on a semi-supervised classification of a sequence of vertices employing the CIAW dataset. Metrics have been evaluated considering for each snapshot in the label sequences just the vertices appearing in the snapshot taken into account. A *Wilcoxon Test* shows *p*-value $<5\%$ for both the metrics when comparing the best of the proposed architecture against all the baselines.

Network	Hyper-params	Grid	Accuracy		Unweighted F1 Measure	
			Best config.	Performance mean ± std	Best config.	Performance mean ± std
FC+FC	1st FC nodes:	{150, 200, 250, 300, 350, 400}	150	77.1% ± 2.0%	300	64.6% ± 3.9%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	0%		10%	
GC+GC	1st GC nodes:	{150, 200, 250, 300, 350, 400}	200	80.8% ± 2.1%	400	70.0% ± 5.5%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	20%		10%	
LSTM+FC	LSTM nodes:	{100, 150, 200, 300, 400}	150	84.6% ± 3.4%	150	68.8% ± 2.4%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	40%		40%	
FC+LSTM+FC	FC nodes:	{100, 200, 300, 400}	300	86.2% ± 2.6%	400	71.6% ± 3.1%
	LSTM nodes:	{100, 200, 300, 400}	300		300	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	0%		20%	
WD-GCN	wd-GC nodes:	{100, 200, 300, 400}	400	86.6% ± 3.7%	300	72.6% ± 3.4%
	v-LSTM nodes:	{100, 200, 300, 400}	100		100	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	40%		30%	
CD-GCN	cd-GC nodes:	{100, 200, 300, 400}	400	***86.7% ± 2.6%***	400	***72.1% ± 3.0%***
	v-LSTM nodes:	{100, 200, 300, 400}	200		400	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	30%		40%	

Table 9 shows the best hyper-parameter configurations together with the test results of all the architectures, where the synthetic dataset is the one called *Configuration 1* in Table 2. It can be seen that with respect to FC+FC, GC+GC and the LSTM-based baselines can achieve better results, the former exploiting the graph structure, while the second ones leveraging the sequence of data. Both WD-GCN and CD-GCN can exploit both graph structure information and sequential correlations at the same time, thus significantly increasing the final network performance.

To confirm the previous interpretation of the results, the experiment has been repeated for the best configurations of Table 9 on a new synthetic dataset with a noisier graph structure, namely *Con-*

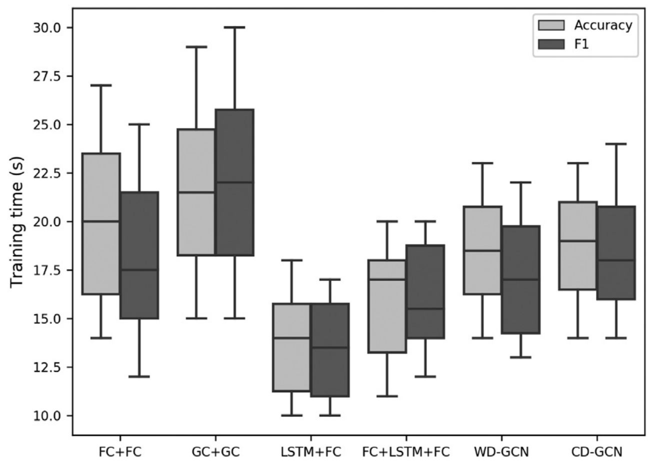


Fig. 8. The box plots represent the *training time* in seconds taken by the best configuration of each architecture of Table 7 on semi-supervised classification of sequence of vertices employing the CIAW dataset.

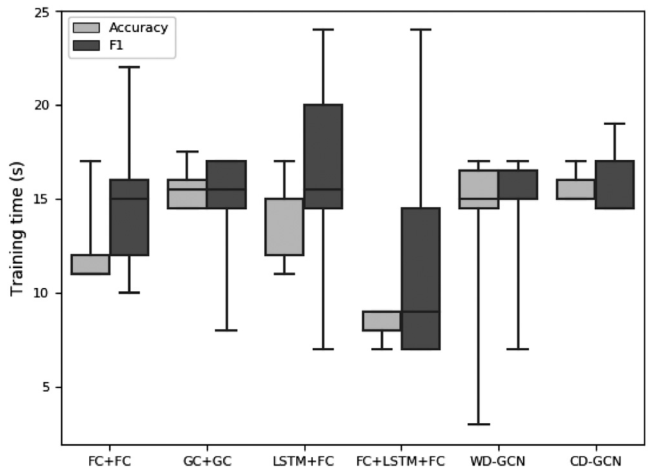


Fig. 9. The box plots represent the *training time* in seconds taken by the best configuration for each architecture of Table 9 on semi-supervised classification of sequence of vertices employing the synthetic dataset.

figuration 2 of Table 2. The results presented in Table 10 show that the added noise greatly reduces the capability of the GC units to exploit the structured information within the graph, which in turns implies a decrease in the performance for all GC+GC, WD-GCN and CD-GCN. Nevertheless, CD-GCN is less affected than WD-GCN thanks to the skip-connection built within it, thus allowing to achieve better performance with respect to all the baselines.

Finally, Fig. 9 represents the wall-clock training time cost in seconds for all the best architectures of Table 9. Also in this case, the training time of our techniques is comparable to the one of the slower baseline network (LSTM+FC) and still in the same order of magnitude as the best and faster baseline performer, the FC+LSTM+FC.

CAD-120. The approaches proposed in Section 3.3 have been compared against a GC+gs-FC network, a vs-FC+gs-FC architecture, a v-LSTM+gs-FC network and a deeper architecture made of vs-FC+v-LSTM+gs-FC. Note that for these architectures, the vs-FCs are used with a ReLU activation, instead of a softmax, since they appear as the first layer of the network.

Of the videos, 10% have been selected for testing the performances of the model and 10% of the remaining videos have been employed for validation.

Table 11 shows the results of this experiment, while Fig. 10a represents the wall-clock training time cost in seconds for all the best architectures of the same table. The obtained results have shown that only CD-GCN has outperformed the baseline, while WD-GCN has reached performances similar to those obtained by the baseline architectures. This difference may be due to the low number of vertices in the sequence of graphs. Under this setting, the predictive power of the graph convolutional features is less effective and the CD-GCN approach, which augments the plain vertex-features with the graph convolutional ones, provides an advantage. Hence, one can further suppose that while WD-GCN and CD-GCN are suitable to effectively exploit structure in graphs with high vertex-cardinality, only the latter can deal with dataset with a limited amount of nodes. It is worth noting that although all the experiments have shown a high variance in their performances, the Wilcoxon Test has shown that CD-GCN is statistically better than the baselines with a p-value $< 5\%$ for the Unweighted F1 Measure and $< 10\%$ for the Accuracy. This reveals that in almost every iteration of the Monte Carlo Cross-Validation, the CD-GCN has performed better than the baselines.

Finally, the same considerations presented for the DBLP dataset regarding the depth and the number of parameters are valid also

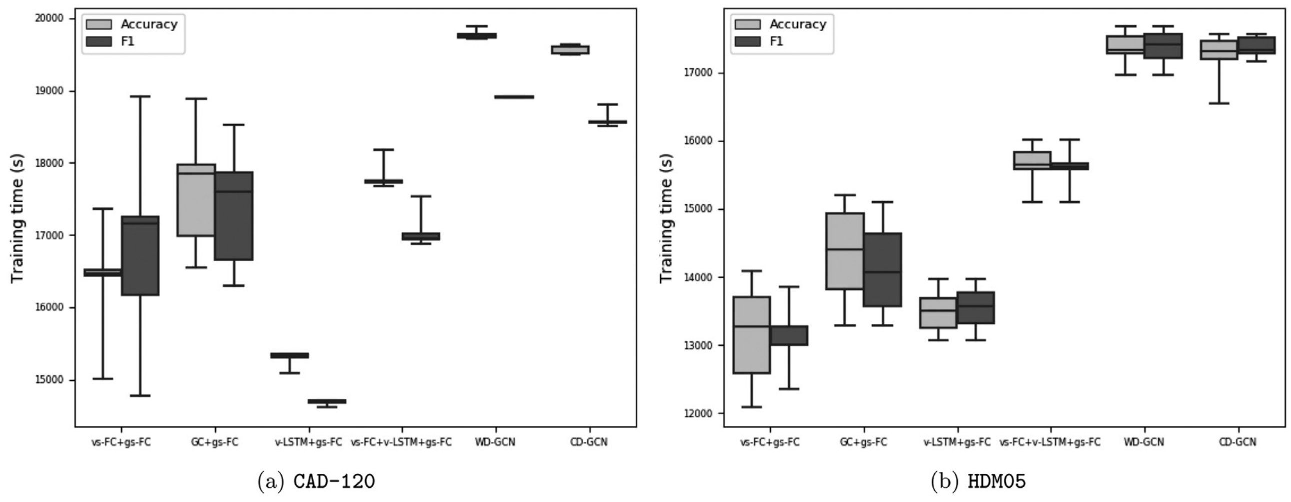


Fig. 10. The box plots represent the *training time* in seconds taken by the best configuration of each architecture of Tables 11 and 12 on a supervised classification of a sequence of graphs on the CAD-120 and HDM05 dataset, respectively.

Table 9
Results of the evaluated architectures on a semi-supervised classification of a sequence of vertices employing the *Configuration 1* of Table 2 for the synthetic dataset. A *Wilcoxon Test* shows a *p*-value $<5\%$ for both the Accuracy score and the Unweighted F1 Measure when comparing the best of the proposed architecture against all the baselines.

Network	Hyper-params	Grid	Accuracy		Unweighted F1 Measure	
			Best config.	Performance mean ± std	Best config.	Performance mean ± std
FC+FC	1st FC nodes:	{150, 200, 250, 300, 350, 400}	150	65.6% ± 3.0%	150	65.4% ± 3.2%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	40%		40%	
GC+GC	1st GC nodes:	{150, 200, 250, 300, 350, 400}	400	73.0% ± 3.6%	300	72.8% ± 3.3%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	20%		0%	
LSTM+FC	LSTM nodes:	{100, 150, 200, 300, 400}	100	73.2% ± 1.4%	100	73.1% ± 1.8%
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	0%		10%	
FC+LSTM+FC	FC nodes:	{100, 200, 300, 400}	100	73.8% ± 1.4%	300	73.8% ± 1.7%
	LSTM nodes:	{100, 200, 300, 400}	100		100	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	20%		20%	
WD-GCN	wd-GC nodes:	{100, 200, 300, 400}	400	95.1% ± 0.8%	400	95.1% ± 0.8%
	v-LSTM nodes:	{100, 200, 300, 400}	100		100	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	40%		40%	
CD-GCN	cd-GC nodes:	{100, 200, 300, 400}	400	***96.3% ± 0.7%***	400	***96.3% ± 0.7%***
	v-LSTM nodes:	{100, 200, 300, 400}	100		100	
	dropout:	{0%, 10%, 20%, 30%, 40%, 50%}	40%		40%	

Table 10
Results of the best configuration for each architecture of Table 9 on a semi-supervised classification of a sequence of vertices, both for the datasets generated using *Configuration 1* and *Configuration 2* of Table 2.

Network	Accuracy (mean ± std)		Unweighted F1 Measure (mean ± std)	
	Configuration one	Configuration two	Configuration one	Configuration two
FC+FC	65.6% ± 3.0%	65.6% ± 3.0%	65.4% ± 3.2%	65.4% ± 3.2%
GC+GC	73.0% ± 3.6%	69.4% ± 2.5%	72.8% ± 3.3%	69.3% ± 2.7%
LSTM+FC	73.2% ± 1.4%	73.2% ± 1.4%	73.1% ± 1.8%	73.1% ± 1.8%
FC+LSTM+FC	73.8% ± 1.4%	73.8% ± 1.4%	73.8% ± 1.7%	73.8% ± 1.7%
WD-GCN	95.1% ± 0.8%	73.1% ± 2.9%	95.1% ± 0.8%	72.9% ± 2.9%
CD-GCN	***96.3% ± 0.7%***	***79.6% ± 2.5%***	***96.3% ± 0.7%***	***79.5% ± 2.5%***

Table 11
Results of the evaluated architectures on a supervised classification of a sequence of graphs employing the CAD-120 dataset. CD-GCN is the only technique comparing favorably to all the baselines, resulting in a Wilcoxon Test with a *p*-value lower than 5% for the Unweighted F1 Measure and lower than 10% for the Accuracy.

Network	Hyper-params	Grid	Accuracy		Unweighted F1 Measure	
			Best config.	Performance mean ± std	Best config.	Performance mean ± std
vs-FC+gs-FC	1st vs-FC nodes:	{100, 200, 250, 300}	100	49.9% ± 5.2%	200	48.1% ± 7.2%
	dropout:	{0%, 20%, 30%, 50%}	20%		20%	
GC+gs-FC	1st GC nodes:	{100, 200, 250, 300}	250	46.2% ± 3.0%	250	36.7% ± 7.9%
	dropout:	{0%, 20%, 30%, 50%}	30%		50%	
v-LSTM+gs-FC	LSTM nodes:	{100, 150, 200, 300}	150	56.8% ± 4.1%	150	53.0% ± 9.9%
	dropout:	{0%, 20%, 30%, 50%}	0%		0%	
vs-FC+v-LSTM+gs-FC	vs-FC nodes:	{100, 200, 250, 300}	200	58.7% ± 1.5%	200	57.5% ± 2.9%
	v-LSTM nodes:	{100, 150, 200, 300}	150		150	
	dropout:	{0%, 20%, 30%, 50%}	20%		20%	
WD-GCN	wd-GC nodes:	{100, 200, 250, 300}	250	54.3% ± 2.6%	250	50.6% ± 6.3%
	v-LSTM nodes:	{100, 150, 200, 300}	150		150	
	dropout:	{0%, 20%, 30%, 50%}	30%		30%	
CD-GCN	cd-GC nodes:	{100, 200, 250, 300}	250	***60.7% ± 8.6%***	250	***61.0% ± 5.3%***
	v-LSTM nodes:	{100, 150, 200, 300}	150		150	
	dropout:	{0%, 20%, 30%, 50%}	30%		30%	

for this set of data. Also in this case, the training time of our techniques is comparable to the one of the baseline networks, always in the same order of magnitude of the one of the best baseline performer, i.e. vs-FC+v-LSTM+gs-FC. Note that in this dataset all the networks have much longer training time with respect to the previous vertex-focused datasets due to the length of the sample sequence, i.e. 1298 compared to 10, 20 and 20 for DBLP, CIAW and the synthetic datasets respectively.

HDM05. Finally, the same architectures employed for the CAD-120 dataset has been compared. 10% of the data has been used as a test set and 10% of the remaining data for validation.

Table 12 shows the results of this experiment and Fig. 10b represents the wall-clock training time cost in seconds for all the best architectures of the same table. The obtained results have shown that CD-GCN has outperformed the baseline, while WD-GCN has reached performances similar to those obtained by the baseline architectures. It is important to note that under these settings the performances of GC+gs-FC are worse than vs-FC+gs-FC, showing that the graph information is limited in this case. However, the graph-convolutional features extracted by the GC units, when considered together with their sequentiality (as in CD-GCN), helps in overcoming the performances of a network that deals only

Table 12
Results of the evaluated architectures on a supervised classification of a sequence of graphs employing the HDM05 dataset. CD-GCN compares favorably to all the baselines, resulting in a Wilcoxon test with a *p*-value lower than 2% for Accuracy and Unweighted F1 Measure.

Network	Hyper-params	Grid	Accuracy		Unweighted F1 Measure	
			Best config.	Performance mean ± std	Best config.	Performance mean ± std

vs-FC+gs-FC	1st vs-FC nodes:
	dropout:
		{100, 200, 300}
		{20%}
			200
			20%
				33.7% ± 5.3%	300
					20%
						30.0% ± 4.0%

GC+gs-FC	1st GC nodes:
	dropout:
		{100, 200, 300}
		{20%}
			200
			20%
				30.8% ± 3.9%	200
					20%
						26.5% ± 4.2%

v-LSTM+gs-FC	LSTM nodes:
	dropout:
		{100, 200, 300}
		{20%}
			200
			20%
				77.6% ± 8.5%	200
					20%
						73.9% ± 8.1%

vs-FC+v-LSTM+gs-FC	(vs-FC, v-LSTM) nodes:
	dropout:
		{(100, 100), (200, 150), (300, 200)}
		{20%}
			(100, 100)
			20%
				84.36% ± 2.0%	(100,100)
					20%
						80.0% ± 4.0%

WD-GCN	(wd-GC, v-LSTM) nodes: dropout:	{(100, 100), (200, 150), (300, 200)} 20%	(100, 100) 20%	80.4% ± 3.8%	(100,100) 20%	76.7% ± 5.1%

CD-GCN	(cd-GC, v-LSTM) nodes:
	dropout:
		{(100, 100), (200, 150), (300, 200)}
		{20%}
			(200, 150)
			20%
				85.1% ± 1.9%	(200, 150)
					20%
						81.5% ± 3.8%

with vertex features without exploiting the graph structure (as in v-LSTM+gs-FC or vs-FC+v-LSTM+gs-FC).

Moreover, the same considerations presented for the DBLP dataset regarding the depth and the number of parameters still hold. Additionally, also under this setting, the training time of our techniques is comparable to the one of the baseline networks and always in the same order of magnitude of the one of the best baseline performer, i.e. vs-FC+v-LSTM+gs-FC. Note that in this dataset, all the networks have much longer training time with respect to the previous vertex-focused datasets due to the length of the sample sequence, i.e. 1654 compared to 10, 20 and 20 for DBLP, CIAW and the synthetic datasets, respectively.

5. Conclusions and future works

Two novel neural network architectures have been introduced. They can deal with the semi-supervised classification of sequences of vertices and the supervised classification of sequences of graphs. Both the techniques satisfy the following two properties: (i) they are neural architectures that can be trained by means of gradient descent; (ii) they are general purpose, i.e. not tied to a specific application.

Our models are based on modified GC layers connected with a modified version of an LSTM, the former sharing their weights across the time steps of the data sequence, the latter sharing them between the graph vertices. This allows the proposed techniques to have a time cost that is linear in both number of edges and sequence length, as well as the GC and LSTM building blocks.

WD-GCN performance and CD-GCN performance have been assessed against neural network baselines based on FC, GC, and LSTM layers on four real world datasets and a synthetic one. The experiments showed the superiority of the CD-GCN for both the semi-supervised classification of sequences of vertices and the supervised classification of sequences of graphs, with the WD-GCN achieving good results on datasets with more complex graph structures. Experiments performed on the synthetic dataset showed that the CD-GCN can cope better with noise in the graph structure, allowing it to achieve significantly better results than the baselines even when graph information is limited. It is hypothesized that this is due to the feature augmentation approach guaranteed by the skip connection that is present in the CD-GCN architecture, which allows for the v-LSTM building block to work on vertex features when uncorrupted graph features are not available due to the noise in the graph structure.

The experiments showed that WD-GCN and CD-GCN scale well with the graph size both considering the computational and the prediction performance. Furthermore, wall-clock training time is always comparable with the considered baselines, thus making WD-GCN and CD-GCN viable techniques.

Interesting extensions of this work may consist in:

- The usage of alternative recurrent units to replace the LSTM, such as a Gated Recurrent Unit [29], which is known to achieve performance comparable to an LSTM at a much lower computational cost;
- Further extensions of the GC unit, e.g. replacing the GC unit within WD-GCN and CD-GCN with higher order approximations of the spectral graph convolution [34];
- Exploring the performance of deeper architectures that combine the layers proposed in this work;
- Identifying which statistical properties the data have to fulfill in order to have some theoretical guarantees for WD-GCN and CD-GCN generalization error; e.g. a good candidate might be *ergodicity* of the dynamic process producing the data, since it would intuitively justify the weight sharing among vertices and time steps as the key enabler allowing the proposed architectures to achieve much better results than the considered neural network baselines;
- Reducing the number of parameters of the proposed architectures by means of compression techniques (such as [41]) thus to speed up model inference.

Acknowledgments

The authors would like to thank Giorgio Patrini and Adam Elwood for their helpful and constructive comments that contributed to improve the work. Moreover, the authors would like to thank Pei et al. [27] that provided the code and their version of the DBLP dataset employed in their experiment.

References

[1] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2009) 61–80.
[2] M. Bianchini, M. Maggini, L. Sarti, F. Scarselli, Recursive neural networks learn to localize faces, Pattern Recognit. Lett. 26 (12) (2005) 1885–1895.
[3] J. Liu, M. Li, Q. Liu, H. Lu, S. Ma, Image annotation via graph learning, Pattern Recognit. 42 (2) (2009) 218–228.
[4] A. Srinivasan, S. Muggleton, R.D. King, M.J. Sternberg, Mutagenesis: ILP experiments in a non-determinate biological domain, in: Proceedings of the 4th International Workshop on Inductive Logic Programming, vol. 237, Citeseer, 1994, pp. 217–232.
[5] A. Jain, A.R. Zamir, S. Savarese, A. Saxena, Structural-RNN: deep learning on spatio-temporal graphs, in: CVPR, IEEE, 2016, pp. 5308–5317.
[6] Y. Yuan, X. Liang, X. Wang, D.-Y. Yeung, A. Gupta, Temporal dynamic graph LSTM for action-driven video object detection, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1801–1810.
[7] A. Rozza, M. Manzo, A. Petrosino, A novel graph-based fisher kernel method for semi-supervised learning, in: ICPR, 2014, pp. 3786–3791.

[8] H. Xu, Y. Yang, L. Wang, W. Liu, Node classification in social network via a factor graph model, in: PAKDD, 2013, pp. 213–224.
[9] Y. Zhao, G. Wang, P.S. Yu, S. Liu, S. Zhang, Inferring social roles and statuses in social networks, in: ACM SIGKDD, ACM, 2013, pp. 695–703.
[10] Y. Dong, Y. Yang, J. Tang, Y. Yang, N.V. Chawla, Inferring user demographics and social strategies in mobile social networks, in: KDD '14, ACM, 2014, pp. 15–24.
[11] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, ICLR, 2013.
[12] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, NIPS, 2016.
[13] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, NIPS, 2015.
[14] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, ICLR, 2017.
[15] Y. Li, D. Tarlow, M. Brockschmidt, R.S. Zemel, Gated graph sequence neural networks, ICLR, 2016.
[16] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[17] L.C. Jain, L.R. Medsker, Recurrent Neural Networks: Design and Applications, 1st ed., CRC Press, Inc., 1999.
[18] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in: Proceedings of the IEEE, 1998, pp. 2278–2324.
[19] X. Zhu, Z. Ghahramani, J. Lafferty, et al., Semi-supervised learning using gaussian fields and harmonic functions, in: ICML, vol. 3, 2003, pp. 912–919.
[20] Y. Boykov, O. Veksler, R. Zabih, Fast approximate energy minimization via graph cuts, IEEE Trans. Pattern Anal. Mach. Intell. 23 (11) (2001) 1222–1239.
[21] B. Wang, J. Tsotsos, Dynamic label propagation for semi-supervised multi-class multi-label classification, Pattern Recognit. 52 (2016) 75–84.
[22] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: ACM SIGKDD, ACM, 2016, pp. 855–864.
[23] B. Perozzi, R. Al-Rfou, S. Skiena, DeepWalk: online learning of social representations, in: ACM SIGKDD, ACM, 2014, pp. 701–710.
[24] F.B. Silva, R.d.O. Werneck, S. Goldenstein, S. Tabbone, R.d.S. Torres, Graph-based bag-of-words for classification, Pattern Recognit. 74 (2018) 266–285.
[25] K. Li, S. Guo, N. Du, J. Gao, A. Zhang, Learning, analyzing and predicting object roles on dynamic networks, in: IEEE ICDM, 2013, pp. 428–437.
[26] Y. Yao, L. Holder, Scalable SVM-based classification in dynamic graphs, in: IEEE ICDM, 2014, pp. 650–659.
[27] Y. Pei, J. Zhang, G.H. Fletcher, M. Pechenizkiy, Node classification in dynamic social networks, in: AALTD 2016: 2nd ECML-PKDD International Workshop on Advanced Analytics and Learning on Temporal Data, 2016, pp. 54–93.
[28] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, in: Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, vol. 2, 2005, pp. 729–734.
[29] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: EMNLP, 2014, pp. 1724–1734.
[30] D.K. Hammond, P. Vandergheynst, R. Gribonval, Wavelets on graphs via spectral graph theory, Appl. Comput. Harmon. Anal. 30 (2) (2011) 129–150.
[31] Y. Seo, M. Defferrard, P. Vandergheynst, X. Bresson, Structured sequence modeling with graph convolutional recurrent networks, in: International Conference on Neural Information Processing, Springer, 2018, pp. 362–373.
[32] F. Monti, M. Bronstein, X. Bresson, Geometric matrix completion with recurrent multi-graph neural networks, in: NIPS 30, 2017, pp. 3697–3707.
[33] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[34] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: NIPS, 2016, pp. 3844–3852.
[35] M. Génois, C.L. Vestergaard, J. Fournet, A. Panisson, I. Bonmarin, A. Barrat, Data on face-to-face contacts in an office building suggest a low-cost vaccination strategy based on community linkers, Netw. Sci. 3 (2015) 326–347.
[36] M. Hashemian, W. Qian, K.G. Stanley, N.D. Osgood, Temporal aggregation impacts on epidemiological simulations employing microcontact data, BMC Med. Inf. Decis. Making 12 (1) (2012) 132.
[37] H.S. Koppula, R. Gupta, A. Saxena, Learning human activities and object affordances from RGB-D videos, Int. J. Rob. Res. 32 (8) (2013) 951–970.
[38] D.G. Lowe, Object recognition from local scale-invariant features, in: ICCV, IEEE, 1999, pp. 1150–1157.
[39] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, A. Weber, Documentation Mocap Database HDM05, Technical Report, Universität Bonn, 2007.
[40] D. Kingma, J. Ba, Adam: a method for stochastic optimization, ICLR, 2015.
[41] F. Manessi, A. Rozza, S. Bianco, P. Napoletano, R. Schettini, Automated pruning for deep neural network compression, in: Proceedings of the 24th International Conference on Pattern Recognition, 2018, pp. 657–664.

Franco Manessi He obtained his master degree cum Laude, from the Department of Theoretical Physics of Universitá degli Studi di Pavia in the 2011. He received the Ph.D. degree in Theoretical Physics in January 2015 from the Department of Theoretical Physics, Universitá degli Studi di Pavia. From 2015 to 2016 he was Data Scientist at a consulting firm. In 2017 he was a member of Waynaut's research team, a startup working in the on-line travel industry later acquired by the lastminute.com group. He is now a data scientist in the lastminute.com group. His research interests include machine learning and its applications.

Alessandro Rozza He obtained his master degree cum Laude, from the Department of Computer Science of Universitá degli Studi di Milano/Bicocca in the 2006. He received the Ph.D. degree in Computer Science in March 2011 from the Department of Scienze dell'Informazione, Universitá degli Studi di Milano. From 2012 to 2014 he was Assistant Professor at Universitá degli Studi di Napoli-Parthenope. From 2015 to 2017, he was head of research at Waynaut. Nowadays, he is the Chief Scientist of lastminute.com group. His research interests include machine learning and its applications.

Mario Manzo He obtained his master degree cum Laude, from the Department of Computer Science of Universitá degli Studi di Napoli "Parthenope" in the 2008. He received the Ph.D. degree in Computer Science in 2013 from the Department of Scienze dell'Informazione, Universitá degli Studi di Milano. His research interests include machine learning, structured data, and its applications.