Pattern Recognition Letters 148 (2021) 15–21

Contents lists available at ScienceDirect

Pattern Recognition Letters

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

Graph-based neural network models with multiple self-supervised auxiliary tasks

Franco Manessi, Alessandro Rozza*

Strategic Analytics, lastminute.com group, Chiasso, Switzerland

ARTICLE INFO

Article history:
Received 9 December 2020
Revised 22 March 2021
Accepted 26 April 2021
Available online 5 May 2021

Keywords:
Graph neural networks
Self-supervised learning
Multi-task learning
Graph convolutional networks
Semi-supervised learning

ABSTRACT

Self-supervised learning is currently gaining a lot of attention, as it allows neural networks to learn robust representations from large quantities of unlabeled data. Additionally, multi-task learning can further improve representation learning by training networks simultaneously on related tasks, leading to significant performance improvements. In this paper, we propose three novel self-supervised auxiliary tasks to train graph-based neural network models in a multi-task fashion. Since Graph Convolutional Networks are among the most promising approaches for capturing relationships among structured data points, we use them as a building block to achieve competitive results on standard semi-supervised graph classification tasks.

1. Introduction

In the last decade, neural networks approaches that can deal with with structured data have been gaining a lot of traction [7,11,26,30,39]. Due to the prevalence of data structured in the form of graphs, the capability to explicitly exploit structural relationships among data points is particularly useful in improving the performance for a variety of tasks, e.g. in human activity detection [57] and gate recognition [5]. Graph Convolutional Networks (GCNs, [26]) stand out as a particularly successful iteration of such networks, especially for semi-supervised problems. GCNs act to encode graph structures, while being trained on a supervised target loss for all the nodes with labels. This technique is able to share the gradient information from the supervised loss through the graph adjacency matrix and to learn representations exploiting both labeled and unlabeled nodes. Although GCNs can stack multiple graph convolutional layers in order to capture high-order relations, these architectures suffer from "over-smoothing" when the number of layers increases [28], thus making difficult to choose an appropriate number of layers.

If we have a dataset with enough labels, supervised learning can usually achieve good results. Unfortunately, to label a large amount of data is an expensive task. In general, the amount of unlabelled data is substantially more than the data that has been human curated and labelled. It is therefore valuable to find ways to make use of this unlabelled data. A potential solution to this problem comes if we can get labels from unlabelled data and train unsupervised dataset in a supervised manner. Self-supervision achieves this by automatically generating additional labelled signals from the available unlabelled data, using them to learn representations. A possible approach in deep learning involves taking a complex signal, hiding part of it from the network, and then asking the network to fill in the missing information [13].

Additionally, it is found that joint learning of different tasks can improve performance over learning them individually, given that at least a subset of these tasks are related to each other [8]. This observation is at the core of multi-task learning. Precisely, given T tasks $\{\mathcal{T}_i\}_{i=1}^{T}$ where a subset of them are related, multi-task learning aims to help improve the learning of a model for $\{\mathcal{T}_i\}_{i=1}^{T}$ by using the knowledge contained in all or some of the T tasks [54].

In this paper we train neural network-based graph architectures by means of self-supervised auxiliary tasks in a multi-task framework, similarly to [51]. Considering the promising results of the GCN, we decided to experiment this framework in semi-supervised classification problems on graphs, employing GCN as a base building block. The main contribution of this paper consists of three novel auxiliary tasks for graph-based neural networks:

- a**utoencoding**: with which we aim at extracting node representations robust enough to allow both semi-supervised classification as well as vertex features reconstruction;
- c**orrupted features reconstruction**: with which we try to extract node representations that allows to reconstruct some of the vertex input features, starting from an embedding built from

* Corresponding author.
E-mail addresses: alessandro.rozza@lastminute.com, alessandro.rozza.it@ieee.org (A. Rozza).

https://doi.org/10.1016/j.patrec.2021.04.021
0167-8655/

a corrupted version of them. This auxiliary task can be seen as the graph equivalent of reconstructing one of the color channels of a RGB image using the other channels in computer vision self-supervised learning;

c**orrupted embeddings reconstruction**: with which we try to extract node representations robust to embedding corruption. This is similar to the aforementioned auxiliary task, with the difference that the reconstruction is performed on the node embeddings instead of the vertex features.

These three tasks are intrinsically self-supervised, since the labels are directly extracted from the input graph and its vertex features. These novel auxiliary tasks allow to achieve competitive results on standard datasets and to reduce the aforementioned "over-smoothing" limitation of deep GCNs.

The paper is organized as follows: in Section 2 the related works are summarized; in Section 3 we introduce the three auxiliary tasks; in Section 4 a detailed comparison against GCN on a standard public datasets is presented; Section 5 reports conclusions and future works.

2. Related works

In recent years, graph representation learning have gained a lot of attention. These techniques can be divided in three main categories: i) random walk-based; ii) factorization-based; iii) neural network-based. In the first group, *node2vec* [17] and *Deepwalk* [36] are worth mentioning. The former is an efficient and scalable algorithm for feature learning that optimizes a novel network-aware, neighborhood preserving objective function, using stochastic gradient descent. The latter 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 standard statistical models to produce state-of-the-art results.

Among the factorization based methods, [47] presents a semi-supervised factor graph model that can exploit the relationships among the nodes. In this approach, each vertex is modeled as a variable node and the various relationships are modeled as factor nodes.

In the last group, we find all the works that 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. [16] and Scarselli et al. [39] formalize a novel neural network model, the Graph Neural Network. This model maps a graph and its nodes into a D-dimensional Euclidean space in order to learn a final classification/regression model. Bruna et al. [7] approach the graph structured data by proposing two generalizations of Convolutional Neural Networks (CNNs): one based on a hierarchical clustering of the domain and another based on the spectrum of the graph (computed using the Laplacian matrix). Defferrard et al. [11] extend the spectral graph theory approach of the previous work by providing efficient numerical schemes to design fast localized convolutional filters on graphs, achieving the same computational complexity of classical CNNs. Kipf and Welling [26] build on this idea by introducing GCNs. They exploit a localized first-order approximation of the spectral graph convolutions framework [20]. Recently, [45] have applied attention mechanism to graph neural networks to improve model performance.

However, the majority of the methods belonging the three aforementioned categories require a large amount of labelled data, which can limit their applicability. On the other hand, unsupervised algorithms, such as [18,19,44] do not require any external labels, but their performances usually suffer when compared to supervised techniques.

Self-supervised learning can be considered a branch of unsupervised learning, where virtually unlimited supervised signals are generated from the available data and used to learn representations. This learning framework finds many applications, ranging from language modeling [31,38,46], to robotics [22], and computer vision [12,27,34,35,52,53]. Applied to graph representation learning, [42] proposed a multi-stage self-supervised framework, called M3S, showing some empirical success.

Multi-task learning approaches can be divided in five many categories: i) feature learning; ii) low-rank approaches; iii) task clustering; iv) task relation learning; v) decomposition [54]. In the feature learning approach, it is assumed that different tasks share a common feature representation based on the original features. In the Multi-Task Feature Learning method, task specific hidden representations within a shallow network are obtained by learning the feature covariance for all the tasks, in turn allowing to decouple the learning of the different tasks [2,3]. A common approach applied in the deep learning setting is to have the different tasks share the first several hidden network layers, including task-specific parameters only in the subsequent layers [29,33,55]. A more complex approach in deep learning is the cross-stitch network, proposed by Misra et al. [32], in which each task has its own independent hidden layers that operate on learned linear combinations of the activation maps of the previous layers.

The low-rank approaches assume that the model parameters of different tasks share a low-rank subspace [1]. Pong et al. [37] propose to regularize the model parameters by means of the trace norm regularizer, in order to exploit the property of the trace norm to induce low rank matrices. The same idea has been applied in deep learning by Yang and Hospedales [49].

Another approach is to assume that different tasks form several clusters, each of which consists of similar tasks. This can be thought of as clustering algorithms on the task level, while the conventional clustering algorithms operate on the data level. Thrun and O'Sullivan [43] introduced the first implementation of this idea for binary classification tasks that are defined over the same input space. Bakker and Heskes [4] followed the idea to recast the neural networks used in the feature learning approach in a Bayesian settings, where the weights of the task specific final layers are assumed to have a Gaussian mixture as a prior. Similarly, [48] build on the previous idea by changing the prior to a Dirichlet process.

In the task relation based approaches, the task relatedness (e.g. task correlation or task covariance) is used to drive the joint training of multiple tasks. In the early works, these relations are assumed to be known in advance. They are used to design regularizers to guide the learning of multiple tasks, so that the more similar two tasks are, the closer the corresponding model parameters are expected to be [14,23]. However, task relations are often not available and need to be automatically estimated from data. [6] go into this direction by exploiting Gaussian processes and defining a multivariate normal prior on the functional output of all the task outputs, whose covariance is trained from data and represents the relation between the tasks.

In the decomposition approach, it is assumed that the matrix whose row vectors are the weights of each of the tasks can be decomposed as a linear combination of two or more sub-matrices, where each sub-matrix is suitably regularized [9,21,56].

The first attempt to combine self-supervision and multi-task learning on top of GCNs can be found in [51]. The authors compare the direct usage of self-supervision against self-supervision by means of multi-task learning, showing that the latter approach achieves better results. In their paper, they introduce three self-supervised auxiliary tasks, i.e. node clustering, graph partitioning, and graph completion. It is important to notice that these auxiliary tasks are very different with respect to the ones introduced in this paper.

3. Methods

In this section, we introduce the formalization of a multi-task self-supervised GCN for semi-supervised classification. We will first give some preliminary definitions, including of a Graph Convolutional (GC) layer and multi-task target loss. We then proceed by showing the auxiliary tasks that can be learned jointly with the semi-supervised classification loss. Finally, we introduce the overall architecture we used in our experiments.

3.1. Preliminaries

Let $Y_{i,j}$ be the i-th row, j-th column element of the matrix $\boldsymbol{Y}$. $\boldsymbol{I}_d$ is the identity matrix in $\mathbb{R}^d$; $softmax$ and $ReLU$ are the *softmaximum* and the *rectified linear unit* activation functions [15]. Note that all the activation functions act element-wise when applied to a matrix.

An *undirected* graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by its set of the nodes (or vertices), $\mathcal{V}$, and set of the edges, $\mathcal{E}$. For each vertex $v_i \in \mathcal{V}$ let $\boldsymbol{v}_i \in \mathbb{R}^d$ be the corresponding feature vector. Moreover, let $\boldsymbol{A}$ be the adjacency matrix of the graph $\mathcal{G}$; namely, $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ where $A_{i,j} = A_{j,i} = w_{ij}$ if and only if there is an edge between the i-th and j-th vertices and the edge has weight w_{ij}. In the case of an unweighted graph, $w_{ij} = 1$. The symbol $\boldsymbol{X}$ will denote instead the vertex-features matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}|\times d}$, i.e. the matrix whose row vectors are the $\boldsymbol{x}_i$.

The mathematics of the GC layer [26] is here briefly recalled, since it is a basic building block of the following network architectures. 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 (also called channels) and $\boldsymbol{B} \in \mathbb{R}^{d\times M}$ weight matrix is defined as the function GC_M from $\mathbb{R}^{|\mathcal{V}|\times d}$ to $\mathbb{R}^{|\mathcal{V}|\times M}$ such as $\mathrm{GC}_M(\boldsymbol{X}) := \hat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{B}$, where $\hat{\boldsymbol{A}}$ is the re-normalized adjacency matrix, i.e. $\hat{\boldsymbol{A}} := \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$ with $\tilde{\boldsymbol{A}} := \boldsymbol{A} + \boldsymbol{I}_{|\mathcal{V}|}$ and $\tilde{D}_{kk} := \sum_l \tilde{A}_{kl}$. Note that the GC layer can be seen as localized first-order approximation of spectral graph convolution [11], with the additional *renormalization trick* in order to improve numerical stability [26].

Consider now a multi-task problem, made of T tasks, indexed by $t = 1, \dots, T$. All the tasks share the input space $\mathcal{X}$ and have the task-specific output spaces $\mathcal{Y}_t$. We suppose that each task t is associated to a parametric hypothesis class f_t (e.g. a neural network architecture) such that $f_t(\boldsymbol{x}; \vartheta_{\mathrm{sh}}, \vartheta_t) = \hat{y}^t$, where $\boldsymbol{x} \in \mathcal{X}$, $y^t \in \mathcal{Y}_t$, ϑ_{sh} is a parameter vector shared among the hypothesis classes of different tasks, and ϑ_t is task-specific. The joint training of each of the f_t is achieved by means of empirical risk minimization:

$$\mathrm{argmin}_{\vartheta_{\mathrm{sh}},\vartheta_1,\dots,\vartheta_T} \sum_{t=1}^{T} w_t \mathcal{R}_t(\vartheta_{\mathrm{sh}}, \vartheta_t), \tag{1}$$

where $w_t \in \mathbb{R}^+$ and $\mathcal{R}_t(\vartheta_{\mathrm{sh}}, \vartheta_t)$ are the task-specific empirical risks. Precisely, $\mathcal{R}_t(\vartheta_{\mathrm{sh}}, \vartheta_t) := \frac{1}{N}\sum_i \mathcal{L}_t(f_t(\boldsymbol{x}_i; \vartheta_{\mathrm{sh}}, \vartheta_t), y_i^t)$ with $\mathcal{L}_t$ the task-specific loss function, $\boldsymbol{x}_i$ the feature vectors of the i-th training sample, y_i^t the target variable of the i-th training sample corresponding to the t-th task, and N the total number of training samples. Roughly speaking, the multi-task objective of Eq. (1) is the conic combination with weights w_t of the empirical risk of each task. A basic justification for taking the weighted combination is due to the fact that it is not possible to define global optimality in the multi-task setting. Indeed, consider two sets of solutions $(\vartheta_{\mathrm{sh}}, \vartheta_1, \vartheta_2)$ and $(\bar{\vartheta}_{\mathrm{sh}}, \bar{\vartheta}_1, \bar{\vartheta}_2)$ such that $\mathcal{R}_1(\vartheta_{\mathrm{sh}}, \vartheta_1) < \mathcal{R}_1(\bar{\vartheta}_{\mathrm{sh}}, \bar{\vartheta}_1)$ and $\mathcal{R}_2(\vartheta_{\mathrm{sh}}, \vartheta_2) > \mathcal{R}_2(\bar{\vartheta}_{\mathrm{sh}}, \bar{\vartheta}_2)$, i.e. $(\vartheta_{\mathrm{sh}}, \vartheta_1, \vartheta_2)$ is the best solution for the first task, while $(\bar{\vartheta}_{\mathrm{sh}}, \bar{\vartheta}_1, \bar{\vartheta}_2)$ reaches optimality in the second task. It is not possible to compare these two solutions without a pairwise measure. A way to put them on the same footing is by mean of Eq. (1).

The weights w_t will be considered as static hyper-parameters of the training procedure in the remaining of the paper. It is worth mentioning that also other approaches exist in which the weights are dynamically computed or obtained through an heuristic [10,24].

It is worth noting that the framework we are considering is usually called *hard parameter sharing*, i.e. there are some parameters ϑ_{sh} that are shared among all the tasks. On the other hand, in *soft parameter sharing*, all parameters are task-specific but they are jointly constrained by means of regularization.

3.2. The tasks

This section is organized as follows: in 3.2.1 the main task is defined; in 3.2.2, 3.2.3, 3.2.4 the auxiliary tasks are formalized.

3.2.1. The main task

As mentioned before, we will consider the semi-supervised classification of graph nodes as our main task. However, what follows can easily be extended to other main tasks as well.

Let's consider a K-class semi-supervised classification problem; thus the output space of the main task can be written as $\mathcal{Y}_{\mathrm{main}} := \{\boldsymbol{y} \in \mathbb{R}^K \mid y_k \in \{0, 1\},\ \sum_k y_k = 1\}$, i.e. the space of one-hot encoded K-class vectors. By denoting with $\mathcal{V}_{\mathrm{l}} \subseteq \mathcal{V}$ the subset of the labeled nodes of the graph $\mathcal{G}$, the empirical risk $\mathcal{R}_{\mathrm{main}}$ of the main task, corresponding to a cross-entropy loss, can be written as:

$$\mathcal{R}_{\mathrm{main}} := -\frac{1}{|\mathcal{V}_{\mathrm{l}}|}\sum_{i\in\mathcal{V}_{\mathrm{l}}}\sum_{k=1}^{K} y_k \log f_{\mathrm{main}}(\boldsymbol{x}_i; \vartheta_{\mathrm{sh}}, \vartheta_{\mathrm{main}}),$$

with $0 \times \log 0 = 0$. We make the assumption that $f_{\mathrm{main}} := g_{\mathrm{sh}} \circ g_{\mathrm{main}}$, with $\partial g_{\mathrm{sh}}/\partial\vartheta_{\mathrm{main}} = \partial g_{\mathrm{main}}/\partial\vartheta_{\mathrm{sh}} = 0$, namely, f_{main} can be seen as the function composition of a vertex feature embedding function g_{sh} parameterized only by ϑ_{sh}, followed by a task specific classification head g_{main} parameterized by $\vartheta_{\mathrm{main}}$ only. As we will see later, g_{sh} is shared with the auxiliary tasks. Finally, g_{sh}, g_{main}, and all the functions we will discuss further ahead are considered differentiable almost everywhere.

3.2.2. Autoencoding

The objective in the autoencoding task (also called AE) is to reconstruct the graph vertex features from an encoding thereof. Using the mean squared error reconstruction loss, the corresponding empirical risk $\mathcal{R}_{\mathrm{AE}}$ can be written as:

$$\mathcal{R}_{\mathrm{AE}} := \frac{1}{|\mathcal{V}_{\mathrm{AE}}|}\sum_{\substack{i\in\mathcal{V}_{\mathrm{AE}}\\ \mathcal{V}_{\mathrm{AE}}\subseteq\mathcal{V}}} \|\boldsymbol{x}_i - f_{\mathrm{AE}}(\boldsymbol{x}_i; \vartheta_{\mathrm{sh}}, \vartheta_{\mathrm{AE}})\|_2^2, \tag{2}$$

with $f_{\mathrm{AE}} := g_{\mathrm{sh}} \circ g_{\mathrm{AE}}$, $\partial g_{\mathrm{AE}}/\partial\vartheta_{\mathrm{sh}} = 0$. Namely, the autoencoder is made of the encoder function g_{sh} also present in the main task, and a decoder component specified by g_{AE} that depends on the task specific parameters ϑ_{AE} only.

3.2.3. Corrupted features reconstruction

The aim of this task (also called FR) is to reconstruct the graph vertex features from an encoding of a corrupted version of them. Namely, the goal is to train an autoencoder that it is able to restore vertex features starting from a vertex-feature matrix $\boldsymbol{X}$ that has some columns zeroed out, i.e. corrupted.

We distinguish two methods: i) *partial reconstruction*, where we aim at outputting the restored features only; ii) *full reconstruction*, where we aim at outputting also the non-corrupted ones.

Let $\mathcal{M}$ be the subset $\mathcal{M} \subset \{1, \dots, d\}$, and $\boldsymbol{P}_{\mathcal{M}} \in \mathbb{R}^{d\times d}$ the diagonal matrix such as its i-th diagonal elements are 1 for all $i \notin \mathcal{M}$, and 0 otherwise, i.e. $\boldsymbol{P}_{\mathcal{M}}$ is the identity matrix with some elements

equal to zero. When applied to a column vector $\boldsymbol{v} \in \mathbb{R}^d$, such a matrix has the property of zero-ing out all the vector elements corresponding to the indexes belonging to $\mathcal{M}$.

Thanks to $\boldsymbol{P}_{\mathcal{M}}$, and considering the mean squared error reconstruction loss, the empirical risk $\mathcal{R}^{\text{f}}_{\text{FR}}$ corresponding to the corrupted *full* features reconstruction can be written as:

$$\mathcal{R}^{\text{f}}_{\text{FR}} := \frac{1}{|\mathcal{V}_{\text{FR}}|} \sum_{\substack{i \in \mathcal{V}_{\text{FR}} \\ \mathcal{V}_{\text{FR}} \subseteq \mathcal{V}}} \|\boldsymbol{x}_i - f^{\text{f}}_{\text{FR}}(\boldsymbol{P}_{\mathcal{M}}\boldsymbol{x}_i; \vartheta_{\text{sh}}, \vartheta^{\text{f}}_{\text{FR}})\|_2^2, \tag{3}$$

with $f^{\text{f}}_{\text{FR}} := g_{\text{sh}} \circ g^{\text{f}}_{\text{FR}}$, $\partial g^{\text{f}}_{\text{FR}} / \partial \vartheta_{\text{sh}} = 0$. Namely, f^{f}_{FR} acts as a *denoising* autoencoder, with the input corrupted by the matrix $\boldsymbol{P}_{\mathcal{M}}$ (for some arbitrary chosen $\mathcal{M}$), and as encoder the function g_{sh}. The decoder component is specified by g^{f}_{FR}, that depends on task specific parameters $\vartheta^{\text{f}}_{\text{FR}}$ only.

Now, we will consider the partial features reconstruction. Let $\boldsymbol{I}_{\mathcal{M}} \in \mathbb{R}^{|\mathcal{M}| \times d}$ be a rectangular matrix whose i-th row is a zero vector, with only a 1 at the j-th position, with $j \in \mathcal{M}$. When applied to a column vector $\boldsymbol{v} \in \mathbb{R}^d$, such a matrix has the property of selecting the vector elements corresponding to the indexes belonging to $\mathcal{M}$. Leveraging $\boldsymbol{I}_{\mathcal{M}}$, the empirical risk $\mathcal{R}^{\text{p}}_{\text{FR}}$ corresponding to the corrupted *partial* features reconstruction can be written similarly as:

$$\mathcal{R}^{\text{p}}_{\text{FR}} := \frac{1}{|\mathcal{V}_{\text{FR}}|} \sum_{\substack{i \in \mathcal{V}_{\text{FR}} \\ \mathcal{V}_{\text{FR}} \subseteq \mathcal{V}}} \|\boldsymbol{I}_{\mathcal{M}}\boldsymbol{x}_i - f^{\text{p}}_{\text{FR}}(\boldsymbol{P}_{\mathcal{M}}\boldsymbol{x}_i; \vartheta_{\text{sh}}, \vartheta^{\text{p}}_{\text{FR}})\|_2^2, \tag{4}$$

with $f^{\text{p}}_{\text{FR}} := g_{\text{sh}} \circ g^{\text{p}}_{\text{FR}}$, $\partial g^{\text{p}}_{\text{FR}} / \partial \vartheta_{\text{sh}} = 0$.

3.2.4. Corrupted embeddings reconstruction

Similarly to the previous task, the aim is to reconstruct "something" from a corrupted version of it. In this case (also called ER), the goal is to reconstruct the embeddings produced by some encoder, in order to make the embeddings resilient to noise. Also in this case, the corruption is achieved by zero-ing out some entries, distinguishing two methods: i) *partial reconstruction*, where we aim at outputting the restored embeddings only; ii) *full reconstruction*, where we aim at outputting the restored embeddings as well as the non-corrupted ones.

Considering the full reconstruction case, the mean squared error loss, and $\mathcal{N}$ as the set containing the corrupted embedding index, we can write the empirical risk $\mathcal{R}^{\text{f}}_{ER}$ corresponding to the corrupted *full* embeddings reconstruction can be written as:

$$\mathcal{R}^{\text{f}}_{\text{ER}} := \frac{1}{|\mathcal{V}_{\text{ER}}|} \sum_{\substack{i \in \mathcal{V}_{\text{ER}} \\ \mathcal{V}_{\text{ER}} \subseteq \mathcal{V}}} \|g_{\text{sh}}(\boldsymbol{x}_i) - f^{\text{f}}_{\text{ER}}(\boldsymbol{x}_i; \vartheta_{\text{sh}}, \vartheta^{\text{f}}_{\text{ER}})\|_2^2, \tag{5}$$

with $f^{\text{f}}_{\text{ER}} := g_{\text{sh}} \circ \boldsymbol{P}_{\mathcal{N}} \circ g^{\text{f}}_{\text{ER}}$, $\partial g^{\text{f}}_{\text{ER}} / \partial \vartheta_{\text{sh}} = 0$. Namely, we use the function g_{sh} also present in the main task as our encoder producing vertex embeddings, we corrupt them by zero-ing out some of them with $\boldsymbol{P}_{\mathcal{N}}$, and we try to reconstruct them with the decoder defined by g^{f}_{ER}.

The *partial* reconstruction version can be obtained by leveraging $\boldsymbol{I}_{\mathcal{N}}$ as made in Section 3.2.3.

3.3. The final network

Our overall network, is composed of a shared encoder g_{sh}, and four output heads g_{main}, g_{AE}, g^{f}_{FR}, g^{f}_{ER}, one per task. Note that we are going to present explicitly only the *full reconstruction* variant of the network, since the *partial reconstruction* version can be easily derived. In the rest of the paper, we will restrict our analysis by using as a foundation block the GC layer [26].

The shared encoder g_{sh} is the same used in [26], i.e. a dropout layer [41] with 50% dropout rate followed by GC layer made of 16 units and a *ReLU* activation function: $g_{\text{sh}} = \text{Dropout}(0.5) \circ \text{GC}_{16} \circ \textit{ReLU}$. The main task classification head g_{main} is made of a dropout layer followed by a GC layer and a $softmax$ activation, where the units of the GC layer depends on the number of classification classes ($g_{\text{main}} = \text{Dropout}(0.5) \circ \text{GC} \circ softmax$).

All the auxiliary task heads g_{AE}, g^{f}_{FR}, g^{f}_{ER} are made of a dropout layer followed by a GC layer made of 16 units, a *ReLU* activation function, another dropout layer, and a final GC layer with no activation function: $\text{Dropout}(0.5) \circ \text{GC}_{16} \circ \textit{ReLU} \circ \text{Dropout}(0.5) \circ \text{GC}$. Note that the number of nodes of the last GC layer depends on the dimension of the vector that we have to reconstruct.

The resulting 4-head network is represented in Fig. 1, and it is trained by minimization of the empirical risk given by $w_{\text{main}}\mathcal{R}_{\text{main}} + w_{\text{AE}}\mathcal{R}_{\text{AE}} + w_{\text{FR}}\mathcal{R}^{f}_{\text{FR}} + w_{\text{ER}}\mathcal{R}^{f}_{\text{ER}}$. The parameters ϑ_{sh}, ϑ_{main}, ϑ_{AE}, $\vartheta^{\text{f}}_{\text{FR}}$, $\vartheta^{\text{f}}_{\text{ER}}$ are trained by stochastic gradient descent. As in previous works [26], the gradient update is performed batch-wise, using the full dataset for every training iteration. The proposed framework inherits memory and time complexity from the underlying layers we chose to use for each of the functions g_{sh}, g_{main}, g_{AE}, g^{f}_{FR}, g^{f}_{ER}. Thus, for the architecture here presented, it means a memory complexity linear in the number of edges for a sparse representation of the adjacency matrix $\boldsymbol{A}$, and a time complexity linear in the number of edges [26].

Note that by forcing some of w_{AE}, w^{f}_{FR}, and w^{f}_{ER} to be identically equal to zero, we can achieve with the same network architecture a settings where some of the auxiliary tasks are effectively deactivated, and in the limit case where all of them are equal to zero we recover the standard GCN.

4. Experimental Results

4.1. Datasets and Experimental Setup

We test our models on semi-supervised classification using the standard datasets Citeseer, Cora, and Pubmed [40]. These are citation networks, where graph vertexes correspond to documents and (undirected) edges to citations. The vertex features are a bag-of-words representation of the documents. Each node is associated to a class label. The Cora dataset contains 2.708 nodes, 5.429 edges, 7 classes and 1.433 features per node. The Citeseer dataset contains 3.327 nodes, 4.732 edges, 6 classes and 3.703 features per node. The Pubmed dataset contains 19.717 nodes, 44.338 edges, 3 classes and 500 features per node.

For the training phase, we used the same setting adopted by Kipf and Welling [26], which in turns follows the experimental setup of [50]. We allow for only 20 nodes per class to be used for training. However, since this is a semi-supervised setting, the training procedure can use the features of all the nodes. The test and validation sets comprise 1.000 and 500 nodes respectively. All the train/test/splitting used in the experiments are the ones used by Kipf and Welling [26].

Since as already mentioned in Section 3.3 we opted to use the GC layers as foundational building blocks, the fair baseline comparison is with GCNs. We conducted three kinds of experiments. In the first round of tests, we focused on one-hidden layer architectures with 16 units, so that we could compare these results directly with the one presented in [26]. For each of the three datasets, the networks that we compared against the baseline and against each other are the ones showed in Table 1. Each of them are instances of the multi-head architecture presented in Section 3.3, with some of the heads deactivated. We used the same amount of dropout, L2 regularization, optimizer [25], and learning rate used by Kipf and Welling [26], unless otherwise stated. The sub-tasks weights appearing in Eq. (1) have been tuned for all the networks by means of grid search. For the network *main* + FR, we tuned whether it is better the full or the partial reconstruction and we found the optimal number of reconstructed features by searching

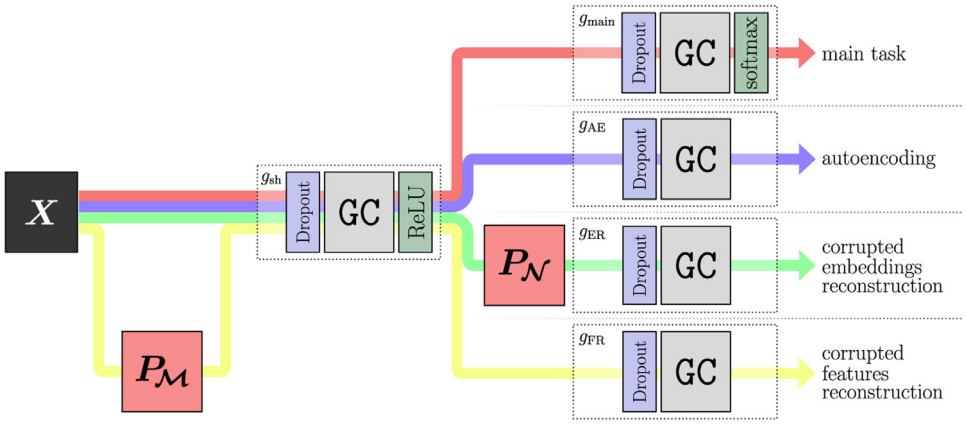

Fig. 1. The drawing shows the architecture of the network described in Section 3.3, which is the one used in our experiments. The network is made of a shared encoder g_{sh}, followed by four heads, one devoted to the main task g_{main}, and the other three to each of the auxiliary tasks g_{AE}, g_{FR}, g_{ER}.

Table 1
The table shows the mean classification accuracy achieved on the test sets and the standard error of the mean for the one-hidden layer networks on Citeseer, Cora, and Pubmed. The acronyms AE, FR, and ER refer to the three auxiliary tasks described in Section 3.2. For completeness, we show also the results reported by Kipf and Welling [26], which however do not report the standard errors.

	Accuracy		
Network	**Citeseer**	**Cora**	**Pubmed**
GCN [26]	70.3%	81.5%	79.0%
GCN (our 10 runs)	$69.84 \pm 0.22\%$	$81.13 \pm 0.13\%$	$78.63 \pm 0.41\%$
main + AE	$71.06 \pm 0.16\%$	$\mathbf{82.17 \pm 0.09}\%$	$78.97 \pm 0.14\%$
main + FR	$70.94 \pm 0.13\%$	$82.07 \pm 0.15\%$	$78.91 \pm 0.19\%$
main + ER	$70.42 \pm 0.20\%$	$81.83 \pm 0.12\%$	$\mathbf{79.33 \pm 0.07}\%$
main + AE + FR	$\mathbf{71.14 \pm 0.12}\%$	$82.13 \pm 0.10\%$	$78.92 \pm 0.14\%$
main + AE + ER	$69.95 \pm 0.14\%$	$82.05 \pm 0.14\%$	$79.15 \pm 0.13\%$
main + AE + FR + ER	$70.09 \pm 0.20\%$	$81.96 \pm 0.13\%$	$79.15 \pm 0.13\%$

through the values 100, 200, 400, 800 for the Citeseer and Cora, and through the values 50, 100, 200 for Pubmed.[1] The resulting optimal values have been used also for the networks *main* + AE + FR and *main* + AE + FR + ER. Similarly, for the network *main* + FR we tuned whether it is better the full or the partial reconstruction and we found the optimal number of reconstructed embeddings by searching through the values 2, 4, 8 for all the datasets. The resulting optimal values have been used also for the networks *main* + AE + ER and *main* + AE + FR + ER as well. Finally, we tuned the sets $\mathcal{V}_{AE}$, $\mathcal{V}_{FR}$, $\mathcal{V}_{ER}$ by comparing the two limiting cases $\mathcal{V}_{AE} = \mathcal{V}_{FR} = \mathcal{V}_{ER} = \mathcal{V}_{I}$ and $\mathcal{V}_{AE} = \mathcal{V}_{FR} = \mathcal{V}_{ER} = \mathcal{V}$.

With Citeseer and Cora we trained for 5.000 epochs, while with Pubmed we trained for 2.500 epochs. During the training, the learning rate was reduced by a factor 10 if the multi-task loss on the validation set did not improve for 40 epochs in a row. In all the cases, we selected the best performing epoch on the validation set to assess the final performance on the test set. Each network has been trained and tested 10 times, each time with a different random weights initialization, and randomized reconstruction features and embeddings (if applicable).

In the second rounds of experiments we focused on the Cora dataset and we compared one, two and five hidden layers architectures. The goal was to assess if the proposed techniques allow to achieve good results even when increasing the network depth, thus making less relevant to tune the number of hidden layers in GCN architectures in order to reduce the "over-smoothing" phenomenon whenever the networks become deeper. The experimental setup is the same as in the first round of experiments, and each hidden layer is a 16 units GC layer.

Finally, in the third rounds of experiment, we compared our best results with the best results achieved by You et al. [51], who employed an approach based on self-supervision and multi-task learning, and those obtained by M3S [42].

4.2. Results

Table 1 shows the results for the one-hidden layer architectures. It can be seen that all the proposed self-supervised multi-task architectures achieve better mean accuracy than a plain GCN. Moreover, the best performing network in each dataset is always one of ours, with a corresponding improvement in performance ranging from 0.70 to 1.30 percentage points. Interestingly, the best performing architecture always shows a smaller standard error compared to the plain GCN, thus exhibiting a more stable performance at different random weights initialization. It is worth noticing that the best performing architectures varies depending on the dataset at hand. Specifically, *main* + AE + FR, *main* + AE, and *main* + ER turned out to be the best performing architectures with the datasets Citeseer, Cora, and Pubmed, respectively. Note that the full network with 4 active heads was never the best performing candidate model (but still better than the baseline). Table 2 shows the hyper-parameters configuration for each of the tested architectures.

The better performance verified with our one-hidden layer networks are confirmed in the second rounds of experiments (see Table 3). Also in this case, the best performing architectures are those proposed in this paper, which additionally show a reduced variance compared to GCN. These results suggest that the proposed architectures help to alleviate the "over-smoothing" problem affecting deep GCNs.

Finally, in Table 4 we compare our results with those obtained by the best architectures proposed in [51]. Since the base building blocks in [51] achieve performance different than ours, and considering that we used the same building block (i.e. GC layer), Table 4 shows the increase (i.e. delta) in accuracy with respect to the building block, to keep the comparison fair.

The same table lists also the performance of M3S [42], which is chosen as a baseline by You et al. [51]. It is possible to notice that our architectures produced a larger or comparable increase in accuracy.

[1] As an example, considering Citeseer, which has 3.703 dimensions, we searched through 100, 200, 400, 800 reconstructed features, meaning that we zero-ed out 3.603, 3.503, 3.303, 2.903 number of features, respectively.

Table 2
The table shows the hyper-parameters for all the configurations listed in Table 1. w_{AE}, w_{FR}, w_{ER} are the weights of the loss function of Eq. (1) corresponding to the autoencoding, feature reconstruction, and embedding reconstruction auxiliary tasks, respectively. $|\mathcal{M}|$ and $|\mathcal{N}|$ represent the number of reconstructed features and embeddings, respectively. $\mathcal{V}_{AE,FR,ER}$ represents instead the subset of nodes used by the auxiliary tasks loss, as in Eqs. (2), (3), and (5). $\mathcal{V}_l$ represents the set of labeled nodes in the datasets. Finally, the last two columns indicate if the experiments have been performed using a full reconstruction for the feature and the embedding reconstruction tasks, respectively.

Dataset/Network	Hyper-Parameters							
	w_{AE} $\lvert\mathcal{N}\rvert$	w_{FR} $\mathcal{V}_{AE/FR/ER}$	w_{ER} Full FR	$\lvert\mathcal{M}\rvert$ Full ER				
Citeseer								
main + AE	10^5	–	–	–	–	–	–	–
main + FR	–	10^3	–	800	–	$\mathcal{V}_l$	Yes	–
main + ER	–	–	10^3	–	1	$\mathcal{V}_l$	–	Yes
main + AE + FR	10^5	10^3	–	800	–	$\mathcal{V}_l$	Yes	–
main + AE + ER	10^5	–	10^3	–	1	$\mathcal{V}_l$	–	Yes
main + AE + FR + ER	10^5	10^3	10^3	800	1	$\mathcal{V}_l$	Yes	Yes
Cora								
main + AE	10^5	–	–	–	–	–	–	–
main + FR	–	10^3	–	200	–	$\mathcal{V}_l$	No	–
main + ER	–	–	10^3	–	4	$\mathcal{V}_l$	–	Yes
main + AE + FR	10^5	10^3	–	200	–	$\mathcal{V}_l$	No	–
main + AE + ER	10^5	–	10^3	–	4	$\mathcal{V}_l$	–	Yes
main + AE + FR + ER	10^5	10^3	10^3	200	4	$\mathcal{V}_l$	No	Yes
Pubmed								
main + AE	10^5	–	–	–	–	–	–	–
main + FR	–	10^3	–	100	–	$\mathcal{V}_l$	Yes	–
main + ER	–	–	10^3	–	2	$\mathcal{V}_l$	–	Yes
main + AE + FR	10^5	10^3	–	200	–	$\mathcal{V}_l$	No	–
main + AE + ER	10^5	–	10^3	–	2	$\mathcal{V}_l$	–	Yes
main + AE + FR + ER	10^5	10^3	10^3	200	2	$\mathcal{V}_l$	No	Yes

Table 3
The table shows the mean classification accuracy achieved on the test sets and the standard error of the mean for one/two/five-hidden layer networks on Cora. In parenthesis we show the increase in performance measured in percentage points (pp) with respect to baseline GCN architectures. The acronyms AE, FR, and ER are the same as in Table 1.

	Accuracy on Cora		
Network	**1 hidden layer**	**2 hidden layers**	**5 hidden layers**
GCN (our 10 runs)	$81.13 \pm 0.13\%$	$79.74 \pm 0.54\%$	$16.47 \pm 2.19\%$
main + AE	$\mathbf{82.17 \pm 0.09}\%$ (+1.04pp)	$\mathbf{81.10 \pm 0.28}\%$ (+1.36pp)	$49.57 \pm 5.50\%$ (+33.10pp)
main + FR	$82.07 \pm 0.15\%$ (+0.94pp)	$80.89 \pm 0.29\%$ (+1.15pp)	$\mathbf{51.05 \pm 4.32}\%$ (+34.58pp)
main + ER	$81.83 \pm 0.12\%$ (+0.70pp)	$80.37 \pm 0.20\%$ (+0.63pp)	$34.01 \pm 4.56\%$ (+17.54pp)
main + AE + FR	$82.13 \pm 0.10\%$ (+1.00pp)	$80.85 \pm 0.25\%$ (+1.11pp)	$49.51 \pm 5.58\%$ (+33.04pp)
main + AE + ER	$82.05 \pm 0.14\%$ (+0.92pp)	$79.76 \pm 0.23\%$ (+0.02pp)	$25.21 \pm 3.41\%$ (+8.74pp)
main + AE + FR + ER	$81.96 \pm 0.13\%$ (+0.83pp)	$79.45 \pm 0.13\%$ (-0.29pp)	$26.23 \pm 3.40\%$ (+9.76pp)

Table 4
The table shows the delta increase in accuracy (measured in percentage points) between the GC building block, the best architectures of Table 1, and those proposed by You et al. [51], including the baseline M3S [42]. The M3S performance listed in the table is the one reported by You et al. [51]. It can be seen that our architectures produced a larger or comparable increase in accuracy.

	Increase in accuracy		
Network	**Citeseer**	**Cora**	**Pubmed**
Best of ours	**1.30 pp**	**1.04 pp**	0.70 pp
M3S	1.09 pp	0.60 pp	0.18 pp
Best of [51]	0.81 pp	0.83 pp	**0.90 pp**

5. Conclusion

We introduced three self-supervised auxiliary tasks to improve semi-supervised classification performance on graph structured data by training them in a multi-task framework. Precisely, i.e. i) vertex features autoencoding; ii) corrupted vertex features reconstruction; iii) corrupted vertex embeddings reconstruction.

The experiments we performed on standard datasets showed better performance with respect to GCNs. Moreover, we compared our results with those achieved by You et al. [51] and M3S [42], showing a larger or comparable increase in accuracy with respect to the base building block.

The two/five-hidden layers scenarios showed that the proposed architectures are more stable and can achieve better results compared to the GCN baselines. These considerations suggest that the proposed architectures help to alleviate the "over-smoothing" problem affecting deep GCNs [28]

A possible future work could be to replace the GC layer with other neural layers devoted to deal with graph structured data (e.g. Graph Attention Networks [45]) to analyze advantages and drawbacks.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank Adam Elwood for his helpful and constructive comments that contributed to improve the work.

References

[1] R.K. Ando, T. Zhang, A framework for learning predictive structures from multiple tasks and unlabeled data, J. Mach. Learn. Res. 6 (2005) 1817–1853.
[2] A. Argyriou, T. Evgeniou, M. Pontil, Multi-task feature learning, in: Advances in Neural Information Processing Systems, 2007, pp. 41–48.
[3] A. Argyriou, T. Evgeniou, M. Pontil, Convex multi-task feature learning, Mach. Learn. 73 (2008) 243–272.
[4] B. Bakker, T. Heskes, Task clustering and gating for bayesian multitask learning, J. Mach. Learn. Res. 4 (2003) 83–99.
[5] F. Battistone, A. Petrosino, TGLSTM: a time based graph deep learning approach to gait recognition, Pattern Recognit. Lett. 126 (2019) 132–138, doi:10.1016/j.patrec.2018.05.004. Robustness, Security and Regulation Aspects in Current Biometric Systems, https://www.sciencedirect.com/science/article/pii/S0167865518301703
[6] E.V. Bonilla, K.M. Chai, C. Williams, Multi-task Gaussian process prediction, in: Advances in Neural Information Processing Systems, 2008, pp. 153–160.
[7] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, ICLR, 2013.
[8] R. Caruana, Multitask learning, Mach. Learn. 28 (1997) 41–75.
[9] J. Chen, J. Liu, J. Ye, Learning incoherent sparse and low-rank patterns from multiple tasks, ACM Trans. Knowl. Discov. Data (TKDD) 5 (2012) 1–31.
[10] Z. Chen, V. Badrinarayanan, C.Y. Lee, A. Rabinovich, GradNorm: gradient normalization for adaptive loss balancing in deep multitask networks, in: International Conference on Machine Learning, 2018, pp. 794–803.
[11] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: NIPS, 2016, pp. 3844–3852.
[12] C. Doersch, A. Gupta, A.A. Efros, Unsupervised visual representation learning by context prediction, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1422–1430.
[13] C. Doersch, A. Zisserman, Multi-task self-supervised visual learning, in: IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017, IEEE Computer Society, 2017, pp. 2070–2079.
[14] T. Evgeniou, C.A. Micchelli, M. Pontil, Learning multiple tasks with kernel methods, J. Mach. Learn. Res. 6 (2005) 615–637.
[15] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep Learning, MIT Press, 2016. 1
[16] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, in: Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., 2, 2005, pp. 729–734vol. 2.
[17] A. Grover, J. Leskovec, Node2vec: scalable feature learning for networks, in: ACM SIGKDD, ACM, 2016, pp. 855–864.
[18] A. Grover, A. Zweig, S. Ermon, Graphite: Iterative Generative Modeling of Graphs, PMLR, Long Beach, California, USA, 2019. 2434–2444
[19] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 1024–1034.
[20] D.K. Hammond, P. Vandergheynst, R. Gribonval, Wavelets on graphs via spectral graph theory, Appl. Comput. Harmonic Anal. 30 (2) (2011) 129–150.
[21] A. Jalali, S. Sanghavi, C. Ruan, P.K. Ravikumar, A dirty model for multi–task learning, in: Advances in Neural Information Processing Systems, 2010, pp. 964–972.
[22] E. Jang, C. Devin, V. Vanhoucke, S. Levine, Grasp2Vec: learning object representations from self-supervised grasping, in: Conference on Robot Learning, 2018, pp. 99–112.
[23] T. Kato, H. Kashima, M. Sugiyama, K. Asai, Multi-task learning via conic programming, in: Advances in Neural Information Processing Systems, 2008, pp. 737–744.
[24] A. Kendall, Y. Gal, R. Cipolla, Multi-task learning using uncertainty to weigh losses for scene geometry and semantics, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7482–7491.
[25] D. Kingma, J. Ba, Adam: a method for stochastic optimization, ICLR, 2015.
[26] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, ICLR, 2017.
[27] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al., Photo-realistic single image super-resolution using a generative adversarial network, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4681–4690.
[28] Q. Li, Z. Han, X.M. Wu, Deeper insights into graph convolutional networks for semi-supervised learning, arXiv preprint arXiv:1801.07606(2018).
[29] S. Li, Z.Q. Liu, A.B. Chan, Heterogeneous multi-task learning for human pose estimation with deep convolutional neural network, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014, pp. 482–489.
[30] F. Manessi, A. Rozza, M. Manzo, Dynamic graph convolutional networks, Pattern Recognit. 97 (2020) 107000.
[31] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv e-prints arXiv:1301.3781 (2013).
[32] I. Misra, A. Shrivastava, A. Gupta, M. Hebert, Cross-stitch networks for multi–task learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 3994–4003.
[33] N. Mrkšić, D.O. Séaghdha, B. Thomson, M. Gasic, P.H. Su, D. Vandyke, T.H. Wen, S. Young, Multi-domain dialog state tracking using recurrent neural networks, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), 2015, pp. 794–799.
[34] M. Noroozi, P. Favaro, Unsupervised learning of visual representations by solving jigsaw puzzles, in: European Conference on Computer Vision, 2016, pp. 69–84.
[35] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, A.A. Efros, Context encoders: feature learning by inpainting, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2536–2544.
[36] B. Perozzi, R. Al-Rfou, S. Skiena, DeepWalk: online learning of social representations, in: ACM SIGKDD, ACM, 2014, pp. 701–710.
[37] T.K. Pong, P. Tseng, S. Ji, J. Ye, Trace norm regularization: reformulations, algorithms, and multi-task learning, SIAM J. Optim. 20 (2010) 3465–3489.
[38] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving language understanding by generative pre-training, 2018,
[39] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (2009) 61–80.
[40] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, T. Eliassi-Rad, Collective classification in network data, AI Mag. 29 (2008). 93–93
[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The J. Mach. Learn. Res. 15 (2014) 1929–1958.
[42] K. Sun, Z. Lin, Z. Zhu, Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes, in: AAAI, 2020, pp. 5892–5899.
[43] S. Thrun, J. O'Sullivan, Discovering structure in multiple learning tasks: the tc algorithm, in: ICML, 1996, pp. 489–497.
[44] P. Velickovic, W. Fedus, W.L. Hamilton, P. Liò, Y. Bengio, R.D. Hjelm, Deep graph infomax, ICLR (Poster), 2019.
[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lió, Y. Bengio, Graph attention networks, in: International Conference on Learning Representations, 2018.
[46] J. Wu, X. Wang, W.Y. Wang, Self-supervised dialogue learning, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 3857–3867.
[47] H. Xu, Y. Yang, L. Wang, W. Liu, Node classification in social network via a factor graph model, in: PAKDD, 2013, pp. 213–224.
[48] Y. Xue, X. Liao, L. Carin, B. Krishnapuram, Multi-task learning for classification with Dirichlet process priors, J. Mach. Learn. Res. 8 (2007) 35–63.
[49] Y. Yang, T.M. Hospedales, Trace norm regularised deep multi-task learning, arXiv preprint arXiv:1606.04038(2016).
[50] Z. Yang, W. Cohen, R. Salakhudinov, Revisiting semi-supervised learning with graph embeddings, in: International conference on machine learning, 2016, pp. 40–48.
[51] Y. You, T. Chen, Z. Wang, Y. Shen, When does self-supervision help graph convolutional networks? ICML, 2020.
[52] R. Zhang, P. Isola, A.A. Efros, Colorful image colorization, in: European Conference on Computer Vision, 2016, pp. 649–666.
[53] R. Zhang, P. Isola, A.A. Efros, Split-brain autoencoders: unsupervised learning by cross-channel prediction, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1058–1067.
[54] Y. Zhang, Q. Yang, A survey on multi-task learning, arXiv preprint arXiv:1707.08114(2017).
[55] Z. Zhang, P. Luo, C.C. Loy, X. Tang, Facial landmark detection by deep multi-task learning, Springer, 2014. European Conference on Computer Vision, 94–108
[56] W. Zhong, J. Kwok, Convex multitask learning with flexible task clusters, arXiv preprint arXiv:1206.4601(2012).
[57] G. Zhu, L. Zhang, H. Li, P. Shen, S.A.A. Shah, M. Bennamoun, Topology-learnable graph convolution for skeleton-based action recognition, Pattern Recognit. Lett. 135 (2020) 286–292, doi:10.1016/j.patrec.2020.05.005. https://www.sciencedirect.com/science/article/pii/S0167865520301756