Dual-Primal Graph Convolutional Networks

Federico Monti[1], Oleksandr Shchur[2], Aleksandar Bojchevski[2], Or Litany[4], Stephan Günnemann[2], and Michael Bronstein[1,3]

[1] USI Lugano
[2] Technical University of Munich
[3] Imperial College London
[4] Facebook AI Research

Abstract. In recent years, there has been a surge of interest in developing deep learning methods for non-Euclidean structured data such as graphs. In this paper, we propose Dual-Primal Graph CNN, a graph convolutional architecture that alternates convolution-like operations on the graph and its dual. Our approach allows to learn both vertex- and edge features and generalizes the previous graph attention (GAT) model. We provide extensive experimental validation showing state-of-the-art results on a variety of tasks tested on established graph benchmarks, including CORA and Citeseer citation networks as well as MovieLens, Flixter, Douban and Yahoo Music graph-guided recommender systems.

1 Introduction

Recently, there has been an increasing interest in developing deep learning architectures for data with non-Euclidean structure, such as graphs or manifolds. Such methods, known under the name *geometric deep learning* [6], have been applied in a variety of applications, from computer graphics, vision [27,5,24,38], and medicine [33,43] to chemistry [12,13] and high energy physics [17].

One of the drawbacks of graph CNNs described above is that the convolution operations are only applied to vertex features and the underlying domain is considered as fixed. In many situations, this can be a major disadvantage, as the graph can be noisy or only known approximately (e.g. in recommender systems k-NN graphs are typically computed *a priori* based on additional meta-information, which does not necessarily represent the true social relationships existing among users). Furthermore, while vertex features can be informative, there is no clean mechanism to take advantage of edge features that are more complex than scalars.

The key contribution of our paper is an extension of the graph attention mechanism to edges using the dual graph, whose vertices correspond to the edges of the original graph. The proposed Dual-Primal Graph CNN (DPGCNN) addresses the aforementioned issues and achieves superior performance on a broad range of tasks.

2 Graph Convolutional Networks

Definitions. Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathbf{A}\}$ be a given weighted undirected graph with vertices $\mathcal{V} = \{1, \dots, n\}$, edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ s.t. $(i,j) \in \mathcal{E}$ iff $(j,i) \in \mathcal{E}$, and edge weights $a_{ij} = a_{ji} \geq 0$ for $(i,j) \in \mathcal{E}$ and zero otherwise. We denote by $\mathcal{N}_i$ a neighborhood of vertex i; $\mathcal{N}_i^p$ denotes the p-hop neighborhood. The graph structure is represented by the $n \times n$ symmetric adjacency matrix $\mathbf{A} = (a_{ij})$. We define the *normalized graph Laplacian* $\mathbf{\Delta} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{D} = \mathrm{diag}(\sum_{j \neq 1} a_{1j}, \dots, \sum_{j \neq n} a_{nj})$ denotes the degree matrix. In the above setting, the Laplacian is a symmetric matrix admitting an eigendecomposition $\mathbf{\Delta} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^\top$ with orthonormal eigenvectors $\mathbf{\Phi} = (\boldsymbol{\phi}_1^\top, \dots, \boldsymbol{\phi}_n^\top)$ and non-negative eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \dots \lambda_n$ arranged into a diagonal matrix $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$.

We are interested in manipulating functions $f : \mathcal{V} \to \mathbb{R}$ defined on the vertices of the graph, which can be represented as vectors $\mathbf{f} \in \mathbb{R}^n$. The space of such functions is a Hilbert space with the standard inner product $\langle \mathbf{f}, \mathbf{g} \rangle = \mathbf{f}^\top \mathbf{g}$. The eigenvectors of the Laplacian form an orthonormal basis in the aforementioned Hilbert space, allowing a Fourier decomposition of the form $\mathbf{f} = \mathbf{\Phi}\mathbf{\Phi}^\top \mathbf{f}$, where $\hat{\mathbf{f}} = \mathbf{\Phi}^\top \mathbf{f}$ is the *graph Fourier transform* of $\mathbf{f}$. The Laplacian eigenvectors thus play the role of the standard Fourier atoms and the corresponding eigenvalues that of the respective frequencies. Finally, a convolution operation can be defined in the spectral domain by analogy to the Euclidean case as $\mathbf{f} \star \mathbf{g} = \mathbf{\Phi}(\hat{\mathbf{f}} \cdot \hat{\mathbf{g}}) = \mathbf{\Phi}(\mathbf{\Phi}^\top \mathbf{f}) \cdot (\mathbf{\Phi}^\top \mathbf{g})$.

Spectral graph CNNs. Bruna *et al.* [7] exploited the above formulation for designing graph convolutional neural networks, in which a spectral convolution operation has the following form:

$$\mathbf{f}' = \mathbf{\Phi}\hat{\mathbf{G}}\mathbf{\Phi}^\top \mathbf{f}, \tag{1}$$

where $\hat{\mathbf{G}} = \mathrm{diag}(\hat{g}_1, \dots, \hat{g}_n)$ is a diagonal matrix of spectral multipliers representing the filter and $\mathbf{f}'$ is the filter output. Here, for the sake of simplicity, we assume a scalar input, though like in classical CNNs, the basic spectral convolution operation (1) can be applied in combination with linear transformations of input and output features, non-linearities, and pooling layers (implemented as graph coarsening). Note that this formulation explicitly assumes the graph to be undirected, since the orthogonal eigendecomposition of the Laplacian requires a symmetric adjacency matrix.

ChebNet. Defferrard *et al.* [10] considered the spectral CNN framework with polynomial filters represented in the Chebyshev basis, $\tau_\theta(\lambda) = \sum_{j=0}^p \theta_j T_j(\lambda)$, where $T_j(\lambda) = 2\lambda T_{j-1}(\lambda) - T_{j-2}(\lambda)$ denotes the Chebyshev polynomial of degree j, with $T_1(\lambda) = \lambda$ and $T_0(\lambda) = 1$. A single filter of this form can be efficiently computed by applying powers of the graph Laplacian to the feature vector,

$$\mathbf{f}' = \mathbf{\Phi} \sum_{j=0}^p \theta_j T_j(\tilde{\mathbf{\Lambda}}) \mathbf{\Phi}^\top \mathbf{f} = \sum_{j=0}^p \theta_j T_j(\tilde{\mathbf{\Delta}}) \mathbf{f}, \tag{2}$$

thus avoiding its eigendecomposition altogether. Here $\tilde{\lambda}$ is a frequency rescaled in $[-1, 1]$, $\tilde{\mathbf{\Delta}} = 2\lambda_n^{-1}\mathbf{\Delta} - \mathbf{I}$ is the rescaled Laplacian with eigenvalues $\tilde{\mathbf{\Lambda}} = 2\lambda_n^{-1}\mathbf{\Lambda} - \mathbf{I}$.

The computational complexity thus drops from $\mathcal{O}(n^2)$ as in the case of spectral CNNs to $\mathcal{O}(|\mathcal{E}|)$, and if the graph is sparsely connected (with maximum degree of $\mathcal{O}(1)$), to $\mathcal{O}(n)$. Furthermore, the graph can now be directed, as the framework does not rely on explicit eigendecomposition.

Several follow-up works refined and extended this scheme. Kipf and Welling [22] proposed a simplification of ChebNet (referred to as Graph Convolutional Network or GCN) by limiting the order of the polynomial to $p = 1$ and using a re-normalization of the Laplacian to avoid numerical instability.

Levie *et al.* [23] replaced the polynomial filter functions by rational functions based on the complex Cayley transform (CayleyNet), allowing to achieve better spectral resolution of the filters, especially relevant for graphs with communities. Monti *et al.* [32] proposed using graph motifs [29,3] to create anisotropic kernels (MotifNet). Finally, Monti *et al.* [31] proposed an extension of ChebNet to multiple graphs (Multi-Graph CNN or MGCNN) in the context of graph-guided matrix completion and recommender systems.

Mixture Model Networks (MoNet). Monti *et al.* [30] proposed a spatial-domain Graph CNN (MoNet) generalizing the notion of 'patches' to graphs. The neighbors of each vertex i are assigned local pseudo-coordinates $\mathbf{u}_{ij} \in \mathbb{R}^d, j \in \mathcal{N}_i$. The analogue of a convolution is then defined as a Gaussian mixture in these coordinates,

$$f_i' = \sum_{m=1}^{M} w_m \sum_{j \in \mathcal{N}_i} \frac{e^{-(\mathbf{u}_{ij}-\boldsymbol{\mu}_m)^\top \boldsymbol{\Sigma}_m^{-1}(\mathbf{u}_{ij}-\boldsymbol{\mu}_m)}}{\sum_{k \in \mathcal{N}_i} e^{-(\mathbf{u}_{ik}-\boldsymbol{\mu}_m)^\top \boldsymbol{\Sigma}_m^{-1}(\mathbf{u}_{ik}-\boldsymbol{\mu}_m)}} f_j, \tag{3}$$

where $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_M \in \mathbb{R}^d$ and $\boldsymbol{\Sigma}_1, \dots, , \boldsymbol{\Sigma}_M \in \mathbb{S}_+^d$ are learnable parameters of the Gaussians. The Gaussians define local weights extracting a local representation of $\mathbf{f}$ around i that generalizes the notion of a 'patch'; the additional learnable parameters $w_1, \dots, w_M$ correspond to filter coefficients in classical convolution.

Graph Attention Networks (GAT). Veličkovič *et al.* [37] proposed an attention mechanism for learning the relevance of each neighbor. The convolution operation with attention has the form:

$$f_i' = \sum_{j \in \mathcal{N}_i^1} \alpha_{ij} f_j, \qquad \alpha_{ij} = \frac{e^{\eta(a([f_i,\, f_j]))}}{\sum_{k \in \mathcal{N}_i^1} e^{\eta(a([f_i,\, f_k]))}} \tag{4}$$

where η denotes the Leaky ReLU, and $a([f_i,\, f_j])$ is some transformation of the concatenated features at vertices i and j, implemented in [37] as a fully connected (linear) layer.

Performing this process multiple times with different transformations (multiple heads) produces filters capable of focusing on different classes of vertices in a neighborhoods. We note that GAT can be considered as a particular instance of MoNet (3), where the pseudo-coordinates $\mathbf{u}_{ij}$ are just the features of the nodes i and j.

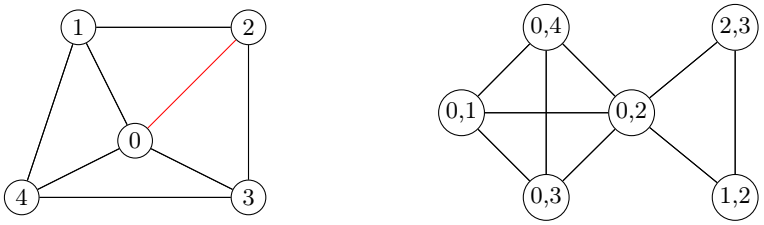

Fig. 1. Primal (left) and dual (right) graphs. For clarity, only the ego graph of primal edge $(0, 2)$ is shown for the dual.

3 Learning on dual/primal graphs

Dual graphs. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a given directed graph, to which we refer as the *primal graph*. The *dual* (also known in graph theory as the *line (di)graph* or *adjoint graph*) of $\mathcal{G}$, denoted by $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}} = \mathcal{E}, \tilde{\mathcal{E}})$, is constructed as follows [16]: each dual vertex $(i, j) \in \tilde{\mathcal{V}}$ corresponds to a primal edge $(i, j) \in \mathcal{E}$, two dual vertices $(i, j), (i', j') \in \tilde{\mathcal{V}}$ are connected by an edge in $\tilde{\mathcal{G}}$ if they share direction and at least an endpoint in $\mathcal{G}$. Figure 1 provides an illustration of this construction for an undirected graph.

We summarize the properties of dual graphs and refer the reader to [16,15,14] for additional details. If the primal graph $\mathcal{G}$ is connected, so is its dual $\tilde{\mathcal{G}}$. The dual graph has $\tilde{n} = |\tilde{\mathcal{V}}| = |\mathcal{E}|$ vertices. If $\mathcal{G}$ is undirected, the number of dual edges is $|\tilde{\mathcal{E}}| = \frac{1}{2}\sum_{i=1}^{n} d_i^2 - |\mathcal{E}|$, where d_i denotes the degree of primal vertex i. If $\mathcal{G}$ is directed, the dual contains $|\tilde{\mathcal{E}}| = \sum_{i=1}^{n} d_i^{\text{in}} d_i^{\text{out}} - |\mathcal{E}|$ edges [1], where d^{in} and d^{out} denote the in- and out-degrees, respectively. The complexity of constructing the dual graph is $\mathcal{O}(|\mathcal{E}| d_{\max}^{out})$, where $d_{\max}^{out} = \max_{i=1,\dots,n} d_i^{\text{out}}$ is the maximum vertex out-degree in the primal graph. While the worst-case complexity is $\mathcal{O}(n^3)$ for fully-connected graphs, for sparsely-connected graphs encountered in practice the cost is linear in n.

In a concurrent work, [9] use dual graphs in GNNs for supervised community detection, which is different from the tasks considered in this paper (classification & matrix completion).

3.1 Dual-Primal GCNN

We propose a *Dual-Primal Graph CNN* (DPGCNN) architecture, which alternates between *dual* and *primal convolutional layers*. The dual convolutional layer applies a GAT on the dual graph to produce features on the edges of the primal graph. These primal edge features are used in the primal convolutional layer to compute attention scores for another GAT, producing primal vertex features. The implementation of both layers are detailed in the following.

Dual convolution. Let $\mathbf{F}$ denote the $n \times q$ matrix of input *primal vertex features*, where each row corresponds to a vertex in the primal graph $\mathcal{G}$. The *dual vertex features* (or equivalently, *primal edge features*) $\tilde{\mathbf{f}}_{ij} = [\mathbf{f}_i, \mathbf{f}_j]$ are constructed

by concatenating the respective primal vertex features (row vectors $\mathbf{f}_i, \mathbf{f}_j$), for each $(i,j) \in \mathcal{E}$. We denote by $\tilde{\mathbf{F}}$ the $\tilde{n} \times 2q$ matrix of all the dual vertex features arranged row-wise. To avoid ambiguity, for undirected graphs we construct two nodes in the dual for every undirected edge $\{i,j\}$, namely $(i,j),(j,i) \in \tilde{\mathcal{V}}$, and we connect a dual node (i,j) to all the nodes corresponding to edges pointing to i or departing from j. This avoids establishing an order among vertices which otherwise would be required to define edge features (if one edge (i,j) is represented by one dual node, one needs to define whether the features of i or j comes first in the concatenation).

Applying GAT to the dual graph $\tilde{\mathcal{G}}$ with features $\tilde{\mathbf{F}}$ has the form

$$\tilde{\mathbf{f}}'_{ij} = \xi_{\mathrm{d}} \left(\sum_{r \in \mathcal{N}_i} \tilde{\alpha}_{ij,ir} \tilde{\mathbf{f}}_{ir} \tilde{\mathbf{W}} + \sum_{t \in \mathcal{N}_j} \tilde{\alpha}_{ij,tj} \tilde{\mathbf{f}}_{tj} \tilde{\mathbf{W}} \right), \tag{5}$$

$$\tilde{\alpha}_{ij,ik} = \frac{e^{\eta(\tilde{a}([\tilde{\mathbf{f}}_{ij}\tilde{\mathbf{W}}, \tilde{\mathbf{f}}_{ik}\tilde{\mathbf{W}}]))}}{\sum_{r \in \mathcal{N}_i} e^{\eta(\tilde{a}([\tilde{\mathbf{f}}_{ij}\tilde{\mathbf{W}}, \tilde{\mathbf{f}}_{ir}\tilde{\mathbf{W}}]))} + \sum_{t \in \mathcal{N}_j} e^{\eta(\tilde{a}([\tilde{\mathbf{f}}_{ij}\tilde{\mathbf{W}}, \tilde{\mathbf{f}}_{tj}\tilde{\mathbf{W}}]))}} \tag{6}$$

where $\tilde{\mathbf{f}}'_{ij}$ denotes the $\tilde{q}$-dimensional output feature of a dual vertex (*i.e.*, primal edge) (i,j), $\tilde{\alpha}_{ij,ik}$ are the dual attention scores, $\tilde{\mathbf{W}}$ is a $2q \times \tilde{q}$ learnable weight matrix, $\tilde{a}$ is a linear layer mapping $2\tilde{q}$-dimensional input to a scalar output, ξ_{d} is the dual layer activation function (typically, a ReLU) and η is the Leaky ReLU.

The dual convolution on $\tilde{\mathcal{G}}$ is equivalent to exchanging information across primal edges which share common directions. In particular, primal edge (i,j) exchanges information only with edges that income to primal vertex i or outgo from from primal vertex j. This additional diffusion naturally allows to better characterize the behavior of each single connection, as every edge (i,j) is now not only represented by the features associated with the corresponding incident vertices but also by an aggregated representation of all the edges that present common spreading patterns (i.e., that bring information to i or spread information from j). This naturally allows to predict better attention scores. Note that such description could not be achieved with two GAT layers as our aggregation step in equations (5)-(6) depends on dual connectivity plus the concatenation of the features of incident vertices and not on the features of the single vertices themselves. For a concrete example, consider Figure 2. Since vertices 2 and 3 have the same attribute vectors ($\mathbf{f}_2 = \mathbf{f}_3$), primal GAT will produce the same attention scores for edges $(1,2)$ and $(1,3)$. In contrast, dual GAT (our dual convolutional layer) is able to differentiate between them by aggregating different edge features for the two different edges.

Primal convolution. The convolution on the primal graph is applied using a GAT on the primal vertex features $\mathbf{F}$. The key difference compared to the simple GAT (4) is that primal attention scores are computed using the dual

vertex features $\tilde{\mathbf{F}}'$ produced by the dual convolution,

$$\mathbf{f}'_i = \xi_{\mathrm{p}}\left(\sum_{r\in\mathcal{N}_i}\alpha_{ij}\mathbf{f}_i\mathbf{W}\right), \qquad \alpha_{ij} = \frac{e^{\eta(a(\tilde{\mathbf{f}}'_{ij}))}}{\sum_{k\in\mathcal{N}_i} e^{\eta(a(\tilde{\mathbf{f}}'_{ik}))}}, \tag{7}$$

where $\mathbf{f}'_i$ denotes the q'-dimensional output features at primal vertex i, α_{ij} are the primal attention scores, ξ_{p} is the primal layer activation function, $\mathbf{W}$ is a $q \times q'$ learnable weight matrix, and a is a linear layer mapping $\tilde{q}$-dimensional input to a scalar output.

3.2 Architecture Variants

GAT as an instance of DPGCNN. The primal and dual convolutional layers can be used as building blocks of graph CNNs. The dual convolution precedes one or more primal convolutional layers; multiple layers can be used to obtain deep neural networks. GAT is a particular setting of DPGCNN obtained by setting the dual attention scores $\tilde{\alpha}_{ij,ik} = 0$ for $k \neq j$ and 1 otherwise.

DPGCNN with polynomial filters. Convolution with polynomials of normalized adjacency matrices are obtained with DPGCNN by computing different attention scores for different orders:

$$\mathbf{f}'_i = \xi\left(\sum_{l=0}^{p}\mathbf{f}_i^{(l)}\mathbf{\Theta}_l\right), \tag{8}$$

$$\mathbf{f}_i^{(k)} = \sum_{j\in\mathcal{N}_i}\alpha_{ij}^{(k)}\mathbf{f}_j^{(k-1)}, \quad \mathbf{f}_i^{(0)} = \mathbf{f}_i \tag{9}$$

where $\mathbf{\Theta}_l$ denote polynomial coefficients and the recursive definition is similar to [32]. $\alpha_{ij}^{(k)}$ is obtained as described in Eq. 6 or 7 if operating on the primal or dual respectively. Such an approach can be exploited in Eq. 5 or 7 to further enrich the filters on both primal and dual graphs.

Arbitrary edge features. We have so far assumed for simplicity that the edge features are derived from vertex features, $\mathbf{f}_{ij} = [\mathbf{f}_i, \mathbf{f}_j]$. Our framework naturally allows to apply dual convolutional layers to arbitrary edge features, both vector- or scalar-valued.

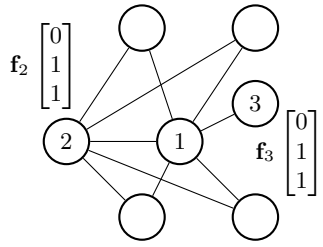

Fig. 2. Unlike GAT, DPGCNN is able to distinguish edges $(1, 2)$ and $(1, 3)$, even though nodes (1) and (2) have the same feature vectors $\mathbf{f}_2 = \mathbf{f}_3$.

4 Experiments

4.1 Citation Networks

Vertex Classification. The first task we consider is a semi-supervised (transductive) learning problem on two citation networks (CORA and Citeseer [36]), following the experimental setup from [41,21,30,37]. The vertices of the citation graph represent scientific papers and edges are citations. The task is to classify each vertex in the graph according to its publication field. CORA contains 2708 vertices, 5429 edges, 7 different categories and 1433 binary features per vertex; Citeseer contains 3327 vertices, 4732 edges, 6 different classes and 3703 features per vertex.

For each dataset we repeat verbatim the experiments presented in [21,30,37]. As training set, we use 140 vertices sampled from CORA and 120 from Citeseer, using the split from [41]. We use the architecture of [37] (i.e. 2 convolutional layers, 8 heads for first layer with 8 features as output per head, 1 head for the second layer with # classes as output) with one head and 32 features as output when convolving on the dual graph. Training settings, including dropout, weight decay and learning rate are as in [37]. Table 1 summarizes the results, averaged over 100 runs to account for different random initializations. DPGCNN beats all the competing architectures, albeit by a small margin.

Table 1. Vertex classification accuracy on CORA and Citeseer citation networks, averaged over 100 runs.

Method	Cora	Citeseer
MLP	51.1%	46.5%
ManiReg [2]	59.5%	60.1%
SemiEmb [39]	59.0%	59.6%
LP [42]	68.0%	45.3%
DeepWalk [34]	67.2%	43.2%
ICA [25]	75.1%	69.1%
Planetoid [41]	75.7%	64.7%
ChebNet [10]	81.2%	69.8%
GCN [21]	81.5%	70.3%
MoNet [30]	81.7 ± 0.5%	–
GAT [37]	83.0 ± 0.7%	72.5 ± 0.7%
DPGCNN	**83.3** ± 0.5%	**72.6** ± 0.8%

We further reproduced a different setting of the same experiment reported in [23], in which 500 vertices were sampled from CORA for training[5]. We used

[5] Training/validation/test indices together with CayleyNet performance have been obtained by the authors of the paper, scaled unnormalized laplacian has been used for the reported CayleyNet's accuracies.

an architecture with two convolutional layers and 16 features as output from convolutional layers on both primal and dual graphs (realized as described in equation (8) with monomial bases and attention). Dual convolution was applied only in the second layer, in order to reduce the overall number of parameters. Attention with one head was used on both primal and dual to provide a fair comparison with CayleyNet [23]. Dropout, weight decay and learning rate were as in [23]. We compare our results to CayleyNet [23] and GAT [37] with the same architecture, using polynomial filters of different order (GAT with polynomial filters was implemented according to equation 8, using primal graph only). Table 2 reports the performance averaged on 20 runs. Our DPGCNN beats the competing architectures.

Table 2. Number of parameters / Vertex classification accuracy on CORA citation network (500 training samples) using polynomial filters of different order.

Order p	CayleyNet	GAT	**DPGCNN**
1	46K / 88.1 ± 0.6%	46K / 88.65 ± 0.58%	47K / **88.92** ± 0.51%
2	69K / 88.0 ± 0.5%	69K / 88.00 ± 0.39%	71K / **88.22** ± 0.41%
3	92K / 87.6 ± 0.6%	92K / 87.54 ± 0.52%	95K / **87.69** ± 0.42%
4	115K / 86.4 ± 0.8%	115K / 87.06 ± 0.42%	118K / **87.30** ± 0.50%
5	138K / 86.5 ± 0.8%	138K / 86.67 ± 0.52%	142K / **86.68** ± 0.74%
6	161K / 86.7 ± 0.7%	161K / 86.38 ± 0.57%	165K / **86.50** ± 0.61%

Link Direction Prediction. The second task we address is to predict the direction of links, which we cast as a semi-supervised classification problem on the dual graph. For this experiment, we used a directed version of the CORA graph [32,4], of which we took a subset containing 1118 vertices (each represented by a 8710-dimensional feature vector) and 4155 directed edges. All the edges were turned into undirected; given an undirected edge $\{i, j\}$, the goal was to predict the direction of the original edge. The dual graph contains every edge in the two possible directions i.e. $(i, j), (j, i) \in \tilde{\mathcal{V}}$, link prediction in the primal is thus a binary classification problem on the dual. 10% of edges' directions were used for training, 10% for validation, and 10% for testing.

Three different architectures were tested: GAT operating only on the primal graph (Primal GAT), GAT operating only on the dual graph (Dual GAT), and a DPGCNN operating on both. Three convolutional layers and a final fully connected layer followed by *softmax* were used in all the three architectures. The final FC layer was applied (i) on the concatenation of the features of nodes i and j for Primal GAT since outputs only node features, (ii) on the features of edge (i, j) for Dual GAT, and (iii) on the concatenation of the two for DPGCNN. In Dual GAT, we had an additional initial dimensionality reduction layer to assure approximately equal overall number of parameters in all the three models for a fair comparison. DPGCNN uses primal and dual convolution in every layer. Each dual convolutional layer receives as input for each edge the refined edge features

concatenated with the refined vertex features of its incident nodes produced by the previous dual/primal convolutional layer. Edge features were initialized with the features of incident nodes for both Dual GAT and DPGCNN. Mean cross-entropy, dropout with keep probability of 0.9, and learning rate of 10^{-2} were used for all models. Table 4.1 presents the link prediction results averaged on 100 runs, showing that DPGCNN outperforms both competitors.

Table 3. Link direction prediction accuracy on directed CORA, averaged over 100 runs.

Method	Accuracy	#Param
Dual GAT	72.94 ± 1.12%	140K
Primal GAT	74.95 ± 1.38%	140K
DPGCNN	**76.45** ± 1.07%	142K

4.2 Graph-Guided Matrix Completion

In our final experiment, we address the problem of item recommendation, formulated as matrix completion problem on user and item graphs [30]. Such problems are also known under the name of *geometric-* or *graph-guided matrix completion*. The task is, given a sparsely sampled matrix of scores assigned by users (columns) to items (rows), to fill in the missing scores. The similarities between users and items are given in the form of column- and row graphs, respectively. Monti *et al.* [30] approached this problem as learning with a separable recurrent Multi-Graph CNN (MGCNN) architecture, using an extension of ChebNets [10] to matrices defined on multiple graphs in order to extract spatial features from the score matrix; these features are then fed into an RNN producing a sequential estimation of the missing scores. We repeated verbatim the experiment of [30,23] on several standard datasets used in the recommender systems literature (MovieLens [28], Flixster [19], Douban [26], and YahooMusic [11]), using different convolutional layers inside RMGCNN (Chebyshev [10], Cayley [23], GAT [37] and the proposed Dual-Primal convolution). For reference, we also report the results of some standard matrix completion methods that are not learning-based. For Douban and Yahoo Music datasets, only a single user/items graph was used, as described in [31].

Polynomial filters (Eq. 8) of degree $p = 4$ were used for convolution on the primal graph; different attention scores were computed according to (Eq. 7) for every order and for every diffusion iteration. We used 4 heads on the dual for every dataset besides Douban where just one head has been exploited because of overfitting. Eight features have been produced as output for each head on the dual. GAT hyperparametrs were determined by cross-validation for Flixster and Yahoo Music; the same hyperparameters were used for DPGCNN. For the remaining datasets, we used hyperparameters from [31]. To make the problem more tractable with classic GPUs (in our experiments we used Nvidia Titan X

with 12GB RAM), randomly sparsified versions of the dual graph were used with DPGCNN for Movielens (100 neighbors in the dual), Flixster (18 neighbors) and Yahoo Music (30 neighbors). Such sparsification was pre-computed and fixed throughout the entire learning and testing process. Table 4 shows the results for different architectures. DPGCNN outperforms all the competing Graph CNN architectures on all the considered datasets, and beats by a significant margin the standard recommendation systems. Table 5 compares the number of parameters required by MGCNN implemented with GAT and DPGCNN.

Table 4. Performance (RMSE) of several matrix completion methods on the MovieLens, Flixster, Douban and Yahoo Music datasets (– indicates that the result was not reported in the original paper). GAT's performance has been computed in this work.

	Method	MovieLens	Flixster	Douban	Yahoo
	IMC [18,40]	1.653	–	–	–
	GMC [20]	0.996	–	–	–
	MC [8]	0.973	–	–	–
	GRALS [35]	0.945	1.245	0.833	38.042
MGCNN	Chebyshev [31]	0.929	0.926	0.801	22.415
MGCNN	Cayley [23]	0.922	–	–	–
MGCNN	GAT [37]	0.929	0.931	0.791	22.102
MGCNN	**Dual/Primal**	**0.915**	**0.902**	**0.789**	**21.970**

Table 5. Number of parameters for MGCNN with GAT layers and our DPGCNN. Note how both solutions present same number of parameters for all considered datasets. The increased number of parameters for Douban and Yahoo is due to the single graph used for implementing MGCNN.

Method	MovieLens	Flixster	Douban	Yahoo
GAT-MGCNN [37]	23K	22K	41K	41K
Dual/Primal-MGCNN	25K	24K	42K	42K

5 Conclusions

We presented a Dual-Primal Graph Convolutional Network able to realize rich convolutional filters by operating on both the primal and the dual graph. Our architecture achieves state-of-the-art performance on vertex classification, link prediction and matrix completion problems by requiring a small amount of additional parameters. In future works we plan to further investigate the importance of the dual graph by exploring applications to Computer Vision and Graphics (e.g. point clouds and meshes) as well as analyzing datasets where edge features are available.

References

1. Aigner, M.: On the linegraph of a directed graph. Mathematische Zeitschrift **102**(1), 56–61 (1967)
2. Belkin, M., Niyogi, P., Sindhwani, V.: Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. JMLR **7**, 2399–2434 (2006)
3. Benson, A.R., Gleich, D.F., Leskovec, J.: Higher-order organization of complex networks. Science **353**(6295), 163–166 (2016)
4. Bojchevski, A., Günnemann, S.: Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. arXiv:1707.03815 (2017)
5. Boscaini, D., Masci, J., Rodolà, E., Bronstein, M.M.: Learning shape correspondence with anisotropic convolutional neural networks. In: Proc. NIPS (2016)
6. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine **34**(4), 18–42 (2017)
7. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. arXiv:1312.6203 (2013)
8. Candès, E., Recht, B.: Exact Matrix Completion via Convex Optimization. Foundations of Computational Mathematics **9(6)**, 717–772 (2009)
9. Chen, Z., Li, L., Bruna, J.: Supervised community detection with line graph neural networks. ICLR (2018)
10. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Proc. NIPS (2016)
11. Dror, G., Koenigstein, N., Koren, Y., Weimer, M.: The Yahoo! music dataset and KDD-Cup'11. In: KDD Cup (2012)
12. Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P.: Convolutional networks on graphs for learning molecular fingerprints. In: Proc. NIPS (2015)
13. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. arXiv:1704.01212 (2017)
14. Gross, J.L., Yellen, J.: Graph theory and its applications. CRC press (2005)
15. Harary, F.: Graph theory. Addison-Wesley (1969)
16. Harary, F., Norman, R.Z.: Some properties of line digraphs. Rendiconti del Circolo Matematico di Palermo **9**(2), 161–168 (1960)
17. Henrion, I., Brehmer, J., Bruna, J., Cho, K., Cranmer, K., Louppe, G., Rochette, G.: Neural message passing for jet physics (2018)
18. Jain, P., Dhillon, I.S.: Provable inductive matrix completion. arXiv:1306.0626 (2013)
19. Jamali, M., Ester, M.: A matrix factorization technique with trust propagation for recommendation in social networks. In: Proc. Recommender Systems (2010)
20. Kalofolias, V., Bresson, X., Bronstein, M.M., Vandergheynst, P.: Matrix completion on graphs. arXiv:1408.1717 (2014)
21. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks (2017)
22. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv:1609.02907 (2016)
23. Levie, R., Monti, F., Bresson, X., Bronstein, M.M.: Cayleynets: Graph convolutional neural networks with complex rational spectral filters. arXiv:1705.07664 (2017)
24. Litany, O., Bronstein, A., Bronstein, M., Makadia, A.: Deformable shape completion with graph convolutional autoencoders. CVPR (2018)
25. Lu, Q., Getoor, L.: Link-based classification. In: Proc. ICML (2003)

26. Ma, H., Zhou, D., Liu, C., Lyu, M., King, I.: Recommender systems with social regularization. In: Proc. Web Search and Data Mining (2011)
27. Masci, J., Boscaini, D., Bronstein, M., Vandergheynst, P.: Geodesic convolutional neural networks on riemannian manifolds. In: Proc. 3dRR (2015)
28. Miller, B.N., et al.: MovieLens unplugged: experiences with an occasionally connected recommender system. In: Proc. Intelligent User Interfaces (2003)
29. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: simple building blocks of complex networks. Science **298**(5594), 824–827 (2002)
30. Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model CNNs. In: Proc. CVPR (2017)
31. Monti, F., Bronstein, M.M., Bresson, X.: Geometric matrix completion with recurrent multi-graph neural networks. In: Proc. NIPS (2017)
32. Monti, F., Otness, K., Bronstein, M.M.: Motifnet: a motif-based graph convolutional network for directed graphs. arXiv:1802.01572 (2018)
33. Parisot, S., Ktena, S.I., Ferrante, E., Lee, M., Moreno, R.G., Glocker, B., Rueckert, D.: Spectral graph convolutions for population-based disease prediction. In: Proc. MICCAI (2017)
34. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: Online learning of social representations. In: Proc. KDD (2014)
35. Rao, N., Yu, H.F., Ravikumar, P.K., Dhillon, I.S.: Collaborative filtering with graph information: Consistency and scalable methods. In: Proc. NIPS (2015)
36. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI Magazine **29**(3), 93 (2008)
37. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. arXiv:1710.10903 (2017)
38. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph CNN for learning on point clouds. arXiv:1801.07829 (2018)
39. Weston, J., Ratle, F., Mobahi, H., Collobert, R.: Deep learning via semi-supervised embedding. In: Neural Networks: Tricks of the Trade, pp. 639–655 (2012)
40. Xu, M., Jin, R., Zhou, Z.H.: Speedup matrix completion with side information: Application to multi-label learning. In: Proc. NIPS (2013)
41. Yang, Z., Cohen, W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: Proc. ICML (2016)
42. Zhu, X., Ghahramani, Z., Lafferty, J., et al.: Semi-supervised learning using gaussian fields and harmonic functions. In: Proc. ICML (2003)
43. Zitnik, M., Agrawal, M., Leskovec, J.: Modeling polypharmacy side effects with graph convolutional networks. arXiv:1802.00543 (2018)

A Visualizations

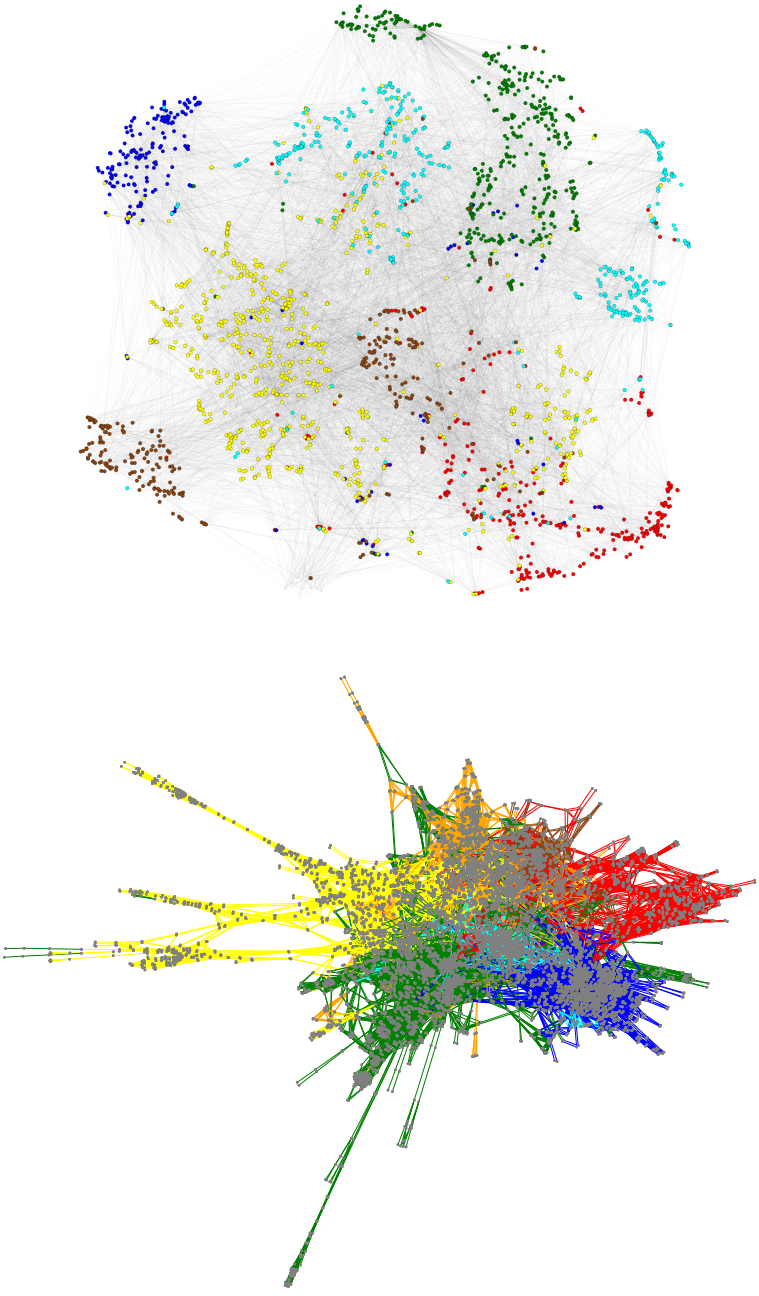

Fig. 3. Top: primal graph of the CORA citation network. Vertex colors code the groundtruth classes. Vertex positions are the learned primal vertex features, mapped to the plane using tSNE. Edge thickness represents the edge attention scores. Bottom: dual graph of CORA. Edge colors represent the groundtruth classes.

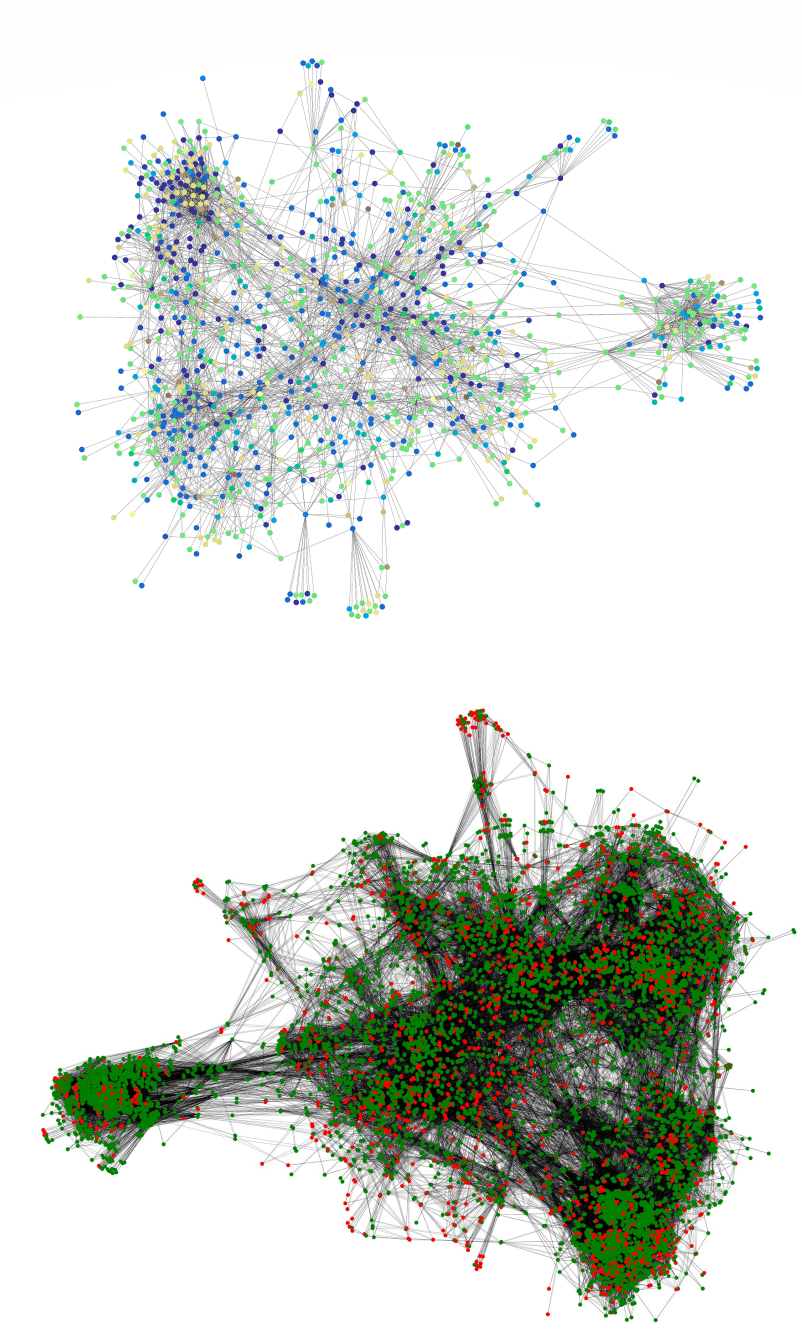

Fig. 4. Top: primal graph of the portion of directed CORA used for link prediction. Vertex colors code the ground-truth classes. Bottom: corresponding dual graph. In green/red are dual vertices (primal edges) that were correctly/wrongly classified, respectively.