ECAI 2020
G.D. Giacomo et al. (Eds.)
© 2020 The authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms
of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/FAIA200215

Simplifying Graph Attention Networks with Source-Target Separation

Hantao Guo[1] and Rui Yan[2] and Yansong Feng[3] and Xuesong Gao [4 5 6] and Zhanxing Zhu✉ [7]

Abstract. We present a novel Graph Neural Networks (GNN) architecture as an simplification of Graph Attentional Network (GAT) model with implicit computation of edge attention coefficients and shared sparse-dense matrix multiplication between heads. These improvements reduce training time and memory consumption while keeping the model capacity of GAT. On several established benchmarks, our model has a performance on par with state-of-the-art, yet with improved efficiency and scalability similar to simpler models including Graph Convolutional Network (GCN). Notably, we are able to apply the model to the large-scale Reddit social network dataset within a reasonable training time and memory constraint, which is previously infeasible for models with similar complexity including GAT.

1 INTRODUCTION

Many structured datasets can be represented as graphs, including citation networks, social networks, traffic network [16], molecule structure [7], protein-protein interaction [35], etc. A graph consists of nodes and edges, where typically nodes correspond to objects and edges describe their relationships. For example, in citation networks, nodes represents papers and edges represents citation relationship between papers. Note that even without an explicit graph structure, building a graph from the data structure by extracting relationship between entities can be beneficial, as in point cloud segmentation with k-NN graph [25] and natural language processing with graph of word relationship [30]. Multiple tasks can be defined on graph data, including graph classification [33], node classification [13], link prediction [32], etc.

Compared to some more traditional data like images with grid structure, where translation-invariant Convolutional Neural Networks [14] can be applied with decent performance, learning on irregular graph structure presents new challenges. One important group of methods on graph data is node embedding [20, 10], in which random walk or its variants are introduced, and an unsupervised skip-gram model [18] is trained on these sequences to obtain node embeddings. However, these methods can only generate embeddings on observed nodes, unable to generalize to new graphs.

Another approach is graph neural networks [34, 27, 28], which are end-to-end neural networks directly applied on graph domain. Early examples applied recurrent operations until equilibrium to obtain high-level node representations [21]. The repeated propagation limits its efficiency. Spectral graph convolution, including ChebNet [5] and Graph Convolutional Networks (GCN) [13], introduced filters on spectral domain, usually implicitly to avoid eigendecomposition of the Laplacian matrix. The learned filters rely on specific graph structure, thus these methods do not generalize to different graphs. Various different frameworks exist for graph neural networks. Inspired by convolution on images, spatial graph convolution variants [7, 1, 11] are examples of Message Passing Neural Networks [9], where a messaging function of both the source and destination node is computed and propagated to the destination, with variable performance and computational efficiency. Other frameworks include spectral graph convolutions and graph recurrent neural networks [34, 28]. Variants of graph neural networks can also be applied to various specific graph types, including directed graphs, heterogeneous graphs, graphs with edge information, and dynamic graphs [27]. Various other researches are conducted based on the graph neural network models, including graph variational autoencoders as generative models, and adversarial training aimed for generalization improvements [34].

Semi-supervised node classification is a particularly important class of graph machine learning problems. Researches show that GCN works as Laplacian smoothing, and thus suffers from over-smoothing for deep networks [15]. And for shallower models, low label rates causes inability in exploring the global graph structure [15]. To expand the labels, self-training and random walk model co-training have been applied to GCN to improve the prediction accuracy especially under low label rates [15].

Attention mechanism [2, 6] was first used in neural machine translation tasks. It is an important neural network structure for sequential data. Graph Attention Network (GAT) [23] combined spatial graph convolution and masked self-attention, where attention coefficients computed with source and destination features are used as edge weights and normalized with softmax function. GAT achieves state-of-the-art performance in various citation network and protein-protein interaction datasets, however, it is not as fast as GCN or other similar models, according to the empirical results in [26]. Meanwhile, attention coefficients and the intermediate computational results require a large amount of memory, especially when the number of edges are high.

In this paper, to improve efficiency and scalability of attention-based networks, we propose a new architecture of graph neural networks. By choosing the attention function in GAT as sum of products

[1] Center for Data Science, Peking University, China. Email: guohantao@pku.edu.cn
[2] Wangxuan Institute of Computer Technology, Peking University, China. Email: fengyansong@pku.edu.cn
[3] Wangxuan Institute of Computer Technology, Peking University, China. Email: fengyansong@pku.edu.cn
[4] College of Intelligence and Computing, Tianjin University, China
[5] Tianjin Key Laboratory of Advanced Networking (TANK)
[6] State Key Laboratory of Digital Multimedia Technology, Hisense Co., Ltd., Qingdao, China. Email: gaoxuesong@tju.edu.cn
[7] School of Mathematical Sciences and Center for Data Science, Peking University, China. Email: zhanxing.zhu@pku.edu.cn.

of source- and target-dependent factors, we can compute the attention mechanism implicitly by applying the two sets of factors before and after propagation, respectively. We name this model Separated Graph Attention Networks (SepGAT). A simplification is to keep only one source-dependent factor, yielding Simplified Graph Attention Networks (SimpGAT). In both models, the adjacency matrix is the only sparse matrix involved, and its product with other matrices can be easily batched by concatenation. This leads to a significant increase in time efficiency. Without explicit computation and storage of attention coefficients and intermediate results, both models have a lower memory requirement, thus scalable to larger datasets. Both models perform on par with state-of-the-art. SepGAT is better suited for larger training data with lower overfitting risk, while SimpGAT is faster and more robust on smaller datasets.

2 Preliminaries

Notations We consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ and $\mathcal{E}$ represent node and edge set, respectively. We construct adjacency matrix $A \in \mathbb{R}^{N\times N}$ and feature matrix $X \in \mathbb{R}^{N\times F}$, where N is the number of nodes and F is the feature dimension. The i-th row $x_i \in \mathbb{R}^F$ of X is the feature of node i. The output of a layer is denoted $X' \in \mathbb{R}^{N\times F'}$, where F' is the number of output features.

2.1 Spectral graph convolution

The spectral graph convolution approach [5] considers a signal filtering on the graph Fourier domain. The method first considers graph Laplacian L, which is defined as combinatorial graph Laplacian $L = D - A$ or symmetric normalized Laplacian $L = I_N - D^{-1/2}AD^{-1/2}$, where D is the diagonal degree matrix $D_{ii} = \sum_j A_{ij}$. The orthonormal eigenvectors $\{u_l\}$ of L are known as the graph Fourier modes, and their associated eigenvalues $\{\lambda_l\}$ are defined as the frequencies. The eigendecomposition can be written in matrix form $L = U\Lambda U^\mathsf{T}$. The graph Fourier transform of a feature channel $x \in \mathbb{R}^N$ is defined $\hat{x} = U^\mathsf{T}x \in \mathbb{R}^N$, and its inverse is $x = U\hat{x}$.

Next, a spectral filtering g_θ is defined in the Fourier domain. A feature channel x is filtered by g_θ as:

$$x' = g_\theta(L)x = g_\theta(U\Lambda U^\mathsf{T})x = Ug_\theta(\Lambda)U^\mathsf{T}x. \tag{1}$$

To avoid expensive explicit eigendecomposition, g_θ is set to be a polynomial, thus $Ug_\theta(\Lambda)U^\mathsf{T}$ can be computed as a matrix polynomial $g_\theta(L)$. In ChebNet [5], the choice of g_θ is a combination of Chebyshev polynomials with rescaled frequencies $\tilde{\Lambda} = 2\Lambda/\lambda_{\max} - I_N$,

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}), \tag{2}$$

where $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), T_0(x) = 1, T_1(x) = x$. Thus, a spectral convolution layer can be written in matrix form as

$$X' = \sum_{k=0}^{K-1} T_k(\tilde{L})X\Theta_k, \tag{3}$$

where $\tilde{L}$ has the same frequency rescaling as $\tilde{\Lambda}$, that is, $\tilde{L} = 2L/\lambda_{\max} - I_N$.

Graph Convolutional Networks (GCN) [13] is a simplification of ChebNet, where the symmetric normalized Laplacian is used and assumed already in the range of $[0, 2]$, thus $\tilde{L} = L - I_N = -D^{-1/2}AD^{-1/2}$. Only two terms $k = 0, 1$ is preserved, with $\theta_0 = -\theta_1$. The graph convolution layer becomes

$$X' = X\Theta + D^{-1/2}AD^{-1/2}X\Theta. \tag{4}$$

With a renormalization trick replacing $I_N + D^{-1/2}AD^{-1/2}$ with $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, where $\tilde{A} = A + I_N$ and $D_{ii} = \sum_j \tilde{A}_{ij}$, the GCN layer is defined as

$$X' = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X\Theta. \tag{5}$$

2.2 Graph attention networks

An attention function, as in [6], is the mapping from a query and a set of key-value pairs to an output. It computes attention coefficients between a set of queries Q and a set of keys K. The attention coefficients are then normalized with softmax function and used as weights of the corresponding values V. For queries q, key k and its value v, the attention mechanism can be formulated as

$$y = \mathrm{softmax}(f(q, k))v \tag{6}$$

An attention mechanism is called self-attention when queries and keys come from the same set.

Graph Attention Networks [23] is a masked self-attention applied on graph structure, in the sense that only keys and values from the neighborhood of query node are used.

First, the node features are transformed by a weight matrix $W \in \mathbb{R}^{F\times F'}$, where F' is the output dimension. Attention coefficients between nodes i and node j is computed as a function of node i and j features. In GAT, the attention mechanism is a single-layer feedforward neural network with LeakyReLU nonlinearity, parameterized by $a \in \mathbb{R}^{2F'}$,

$$e_{ij} = f(W^\mathsf{T}X_i, W^\mathsf{T}X_j) = \eta(a^\mathsf{T}[W^\mathsf{T}x_i, W^\mathsf{T}x_j]), \tag{7}$$

where η represents the LeakyReLU function.

The attention coefficients is then normalized with a softmax function within $\mathcal{N}_i$, the neighborhood of target node i including itself,

$$\alpha_{ij} = \mathrm{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})}, \tag{8}$$

and the final output feature of each object is

$$x_i{}' = \sigma(\sum_{j\in\mathcal{N}_i} \alpha_{ij}W^\mathsf{T}x_j), \tag{9}$$

with an activation function σ for the layer.

Other works based on attention mechanisms on graphs exist. One example is Attention-based Graph Neural Network (AGNN) [22]. AGNN uses a cosine similarity function for attention, parameterized by a scaling factor $\beta \in \mathbb{R}$,

$$e_{ij} = \beta\frac{x_i^\mathsf{T}x_j}{\|x_i\|\|x_j\|}. \tag{10}$$

Other differences include linear transformation after attention mechanism instead of before, and removal of nonlinear activation functions between layers.

The cosine similarity attention in AGNN is also highly related to the inner product attention, where $e_{ij} = x_i^\mathsf{T}x_j$, which is applied in some Natural Language Processing (NLP) models [6]. Specifically, the cosine similarity function is the inner product normalized with l_2 norms of the input vectors.

2.3 Scalable variants of graph neural networks

To cope with large graphs impossible to fit in the memory and/or impractically time-consuming for computation, various approaches have been proposed.

GraphSAGE [11] is an extension of Message Passing Neural Networks [9, 34] into a mini-batched setting. Neighborhood up to order K is sampled with fixed size for a given set of input nodes, and only the node representation computation required by the center node is performed. In each layer, for each node in the sampled subgraph, the neighborhood features are aggregated with mean, max, or LSTM functions to obtain a new hidden state for the next layers. The model can be trained in an unsupervised setting, where the final embedding is used to predict the co-occurrence in random walks. Supervised training with downstream loss function is also possible.

GaAN [31] samples nodes like GraphSAGE, and applies a multi-head attention mechanism as aggregator. The attention heads are additionally weighted with a sigmoid gate to assign different importance weights to each head.

FastGCN [3] views graph convolutions as integral transforms of embedding functions under probability measures. It involves sampling nodes for each layer from a distribution $q(u)$, and connections between nodes in adjacent layers is considered as a sampling of all edges. During training, the features of sampled nodes in this layer is propagated to sampled nodes in next layer in a way similar to GCN, and backward propagation can be done by applying the chain rule straightforwardly. The learned weights can be used in a full-scale GCN or similarly sampling model for inference.

Instead of sampling, Simple Graph Convolution (SGC) [26] removes the activations between layers in GCN and combines all weights into a single matrix,

$$\begin{aligned} Y &= \mathrm{softmax}(S \dots SSXW^{(1)}W^{(2)} \dots W^{(k)}) \\ &= \mathrm{softmax}(S^K XW). \end{aligned} \tag{11}$$

$S^K X$ can be pre-computed with K sparse-dense matrix multiplications, and the following model is a logistic regression which can be trained fast with various method including SGD and second order methods.

3 OUR METHODS

Graph Neural Networks in the propagation and aggregation scheme typically involve a time consuming sparse matrix multiplication. It is thus beneficial to the computational efficiency to reduce the amounts of sparse matrix multiplications. Additionally, Graph Attention Networks (GAT) suffer from high memory cost due to computing and storing attention coefficient on all edges, especially with a large number of edges. Here, we will present a novel graph neural network architecture with similar representation capacity compared to complex models like GAT, but without memory-costly explicit attention coefficient computation. The attention heads can be easily batched with one sparse matrix multiplication per layer, thus giving the model a similar efficiency to GCN. A further simplification is provided to shorten the training time and reduce overfitting.

3.1 Limitations of GAT

GAT has a high model capacity by weighting different edges with attention mechanism. It is the former state-of-the-art on node classification task, and still has decent performance compared to various recent methods. However, one limitation of GAT is its efficiency, even though its time complexity per layer is $\mathcal{O}(K|\mathcal{V}|FF' + K|\mathcal{E}|F')$ with K head, similar to GCN. Multiple issues hinder the efficiency of GAT.

First, multi-head graph attention requires multiple sparse-dense matrix multiplication (SDMM) between different weighted adjacency matrices and transformed features, which can be heavily time-consuming. This is especially evident with GPU training, where dense matrix operations are more easily optimized. Ideally, these SDMM operations can be batched to reduce time, however, no known deep learning frameworks have implemented sparse tensor multiplication with rank above 2 to the best of our knowledge.

Second, computation of general attention functions requires $\mathcal{O}(|\mathcal{E}|F')$ time complexity, comparable to previously described SDMM operations, and it is also tricky to parallelize. One way to deal with this issue is to gather both node features into the edges, and compute the attention functions with two $|\mathcal{E}| \times F'$ matrices. Known as gather/scatter [8], this method is well parallelized, but it requires $\mathcal{O}(|\mathcal{E}|F')$ memory per head, which can be prohibitive on larger or denser datasets. To avoid this problem, the attention functions are limited to specific forms, hindering capacity of the model. In GAT, a single-layer feedforward neural network with LeakyReLU nonlinearity is used, which can be computed as

$$e_{ij} = \eta(a^\mathsf{T}[W^\mathsf{T}x_i, W^\mathsf{T}x_j]) = \eta(a_1^\mathsf{T}W^\mathsf{T}x_i + a_2^\mathsf{T}W^\mathsf{T}x_j), \tag{12}$$

where all instances of $a_1^\mathsf{T}W^\mathsf{T}x_i$ and $a_2^\mathsf{T}W^\mathsf{T}x_j$ can be computed beforehand with $\mathcal{O}(|\mathcal{V}|F')$ time complexity, followed by element-wise operations on the sparse adjacency matrix of $\mathcal{O}(|\mathcal{E}|)$.

Third, attention function values, intermediate results, and gradients are all large sparse matrices with $\mathcal{O}(|\mathcal{E}|)$ memory occupation. For K heads, this requires $\mathcal{O}(K|\mathcal{E}|)$ memory with a large constant coefficient. This high memory complexity hinders the scalability of GAT model.

As for another attention-based approach, AGNN, the cosine similarity function in the attention mechanism involves the inner product between x_i and x_j, which cannot be represented as operations between scalar functions of x_i and x_j. Thus, the circumvention as in GAT cannot be applied, and the gather/scatter approach is the only reasonably efficient algorithm for AGNN. As described above, this approach has a large $\mathcal{O}(|\mathcal{E}|F)$ memory complexity, which limits its scalability. Here, we replace F' with F, since the attention mechanism in AGNN is performed before the linear transformation.

3.2 Separated graph attention mechanism

To handle those issues, we deliver a novel method allowing the attention outputs to be obtained implicitly, which involves no large sparse matrices other than the same input adjacency matrix shared by all heads.

We can reformulate the masked attention in GAT into a weighted attention representation,

$$\alpha_{ij} = \frac{A_{ij}\exp(e_{ij})}{\sum_{k\in\mathcal{V}} A_{ik}\exp(e_{ik})}, \tag{13}$$

$$x_i{}' = \sum_{j\in\mathcal{V}} \frac{A_{ij}\exp(e_{ij})}{\sum_{k\in\mathcal{V}} A_{ik}\exp(e_{ik})} W^\mathsf{T}x_j, \tag{14}$$

and we can compute $\hat{x}'_i = \sum_{j\in\mathcal{V}} A_{ij}\exp(e_{ij})W^\mathsf{T}x_j$ and $d_i = \sum_{k\in\mathcal{V}} A_{ik}\exp(e_{ik})$ respectively, then $x_i{}' = \hat{x}'_i/d_i$. Here, $\hat{x}'_i$ is the unnormalized attention mechanism, and d_i is the sum of row i in the matrix defined by $\hat{A}_{ij} = A_{ij}\exp(e_{ij})$.

Given some specific formulation of e_{ij} from $W^\mathsf{T}x_i$ and $W^\mathsf{T}x_j$, we can separate the computation into source node j-related part, multiplication with A_{ij}, and target node i-related part.

Specifically, we let $\exp(e_{ij})$ be a sum of M products of i- and j-dependent factors

$$\exp(e_{ij}) = \sum_{m=1}^{M} f_m(W^\mathsf{T}x_i)g_m(W^\mathsf{T}x_j), \tag{15}$$

and the form of e_{ij} is thus required to be

$$e_{ij} = \log\left(\sum_{m=1}^{M} f_m(W^\mathsf{T}x_i)g_m(W^\mathsf{T}x_j)\right), \tag{16}$$

where the sum $\sum_{m=1}^{M} f_m(W^\mathsf{T}x_i)g_m(W^\mathsf{T}x_j)$ is required to be larger than zero. Then, the unnormalized attention mechanism $\hat{x}'_i$ and rowsum d_i becomes

$$\hat{x}'_i = \sum_{m=1}^{M} f_m(W^\mathsf{T}x_i) \sum_{j\in|\mathcal{V}|} A_{ij}g_m(W^\mathsf{T}x_j)W^\mathsf{T}x_j, \tag{17}$$

$$d_i = \sum_{m=1}^{M} f_m(W^\mathsf{T}x_i) \sum_{j\in|\mathcal{V}|} A_{ij}g_m(W^\mathsf{T}x_j), \tag{18}$$

and we can first compute all $g_m(W^\mathsf{T}x_j)W^\mathsf{T}x_j$ for each source node j in a batched manner, propagate to each target node i with adjacency matrix A, and further multiply $f_m(W^\mathsf{T}x_i)$ and do summation. This can be rewritten into the matrix form

$$\hat{X}' = \sum_{m=1}^{M} \mathrm{diag}(f_m(XW))A\mathrm{diag}(g_m(XW))XW, \tag{19}$$

and for normalization propose, the row sum of the operator $\sum_{m=1}^{M} \mathrm{diag}(f_m(XW))A\mathrm{diag}(g_m(XW))$ becomes

$$d = \sum_{m=1}^{M} \mathrm{diag}(f_m(XW))A\mathrm{diag}(g_m(XW))\mathbf{1}_N. \tag{20}$$

where f_m and g_m are row-wise applied, and $\mathrm{diag}(\cdot)$ represents the diagonal matrix constructed from the corresponding vector.

With an activation function afterward, the normalized separable weighted attention head can be represented as

$$X' = \sigma(\mathrm{diag}(d)^{-1}\hat{X}'). \tag{21}$$

We name the resulted model as Separated Graph Attention Networks (**SepGAT**). Each attention head has the time complexity $\mathcal{O}(|\mathcal{V}|FF' + M(|\mathcal{V}| + |\mathcal{E}|)F')$. Even though the time complexity is larger than GAT, efficient batching of SDMM allows the model to be still faster than GAT. Additionally, this approach allows certain attention functions with higher capacity to be applied, for which explicit computation may require a similarly high complexity on its own anyway.

We can also consider the vectors

$$f(W^\mathsf{T}x_i) = [f_1(W^\mathsf{T}x_i), f_2(W^\mathsf{T}x_i), ..., f_M(W^\mathsf{T}x_i)]^\mathsf{T} \tag{22}$$

$$g(W^\mathsf{T}x_j) = [g_1(W^\mathsf{T}x_j), g_2(W^\mathsf{T}x_j), ..., g_M(W^\mathsf{T}x_j)]^\mathsf{T} \tag{23}$$

as transformed input features, then the attention mechanism can be expressed as

$$e_{ij} = \log\left(f(W^\mathsf{T}x_i)^\mathsf{T}g(W^\mathsf{T}x_j)\right), \tag{24}$$

on which the masked softmax is then performed. Thus, just like AGNN, SepGAT can also be viewed as a variant of inner-product attention mechanism, where the input features are first transformed with functions $f, g \in \mathbb{R}^F \to \mathbb{R}^M$, and instead of normalization with l_2 norm, a logarithm nonlinearity is applied after the inner product. One particular choice of the functions f and g is single-layer feed-forward neural networks described later.

3.3 Simplified graph attention mechanism

To further improve the efficiency, we consider the special case of $m = 1$, where the unnormalized attention mechanism becomes $\exp(e_{ij}) = f(W^\mathsf{T}x_i)g(W^\mathsf{T}x_j)$. In this case, we found that $f(W^\mathsf{T}x_i)$ occurs in both the numerator and the denominator in the separable weighted attention representation, so we can simplify it into

$$\alpha_{ij} = \frac{A_{ij}g(W^\mathsf{T}x_j)}{\sum_{k\in\mathcal{V}} A_{ik}g(W^\mathsf{T}x_k)}. \tag{25}$$

That is, the attentional coefficient is only related to the neighborhood but not the target node. We denote this model as Simplified Graph Attention Networks (SimpGAT).

The physical explanation of this simplification is that we deduct the importance of source node j towards other nodes based on features of j but not its target nodes, and multiple importance weights of the neighborhood of a center node i is normalized with a sum of one.

3.4 Choice of functions f and g

A simple choice of function e_{ij} in SimpGAT is a single-layer feed-forward neural network with an activation function η, that is, $e_{ij} = \eta(a^\mathsf{T}W^\mathsf{T}x_j)$ and $\exp(e_{ij})$ is a function of $W^\mathsf{T}x_j$.

However, the computation of separable weighted attention mechanism requires an explicit computation of $\exp(e_{ij})$, which can lead to numerical instability, e.g. overflow or underflow when $|e_{ij}|$ is large. In (optionally masked) softmax in traditional mechanism, where subtracting the maximum e_{ij} for each i reduces all exponentials into at most one. It is not applicable in our scheme though, since the maximum of e_{ij} for all j may not correspond to an edge for some i, thus $\max_{j\in\mathcal{N}_i}(e_{ij}) - \max_{k\in\mathcal{V}}(e_{ik})$ can be negative enough for some i to cause underflow in the exponential. Finding row-wise maximum of e_{ij} defeats out propose, since it computes an edge-wise intermediate result, increasing time and memory consumption.

Instead, we choose the inverse hyperbolic sine function as the nonlinearity η, since $\sinh^{-1}(x) = \log(x + \sqrt{1+x^2})$, and $\exp\sinh^{-1}(x) = x + \sqrt{1+x^2}$. As a result, $\exp\sinh^{-1}(x) \to -\frac{1}{2|x|}$ when $x \to -\infty$, and $\exp\sinh^{-1}(x) \to 2x$ when $x \to +\infty$, which has much better numerical stability than exponential function alone. Also, it only involves arithmetic and square root operations to compute, with no difficulties for implementation or computation.

With this choice, the unnormalized attention mechanism and its rowsum becomes

$$\hat{X}' = A\mathrm{diag}(\phi(XWb))XW, \tag{26}$$

$$d = A\mathrm{diag}(\phi(XWb))\mathbf{1}_N, \tag{27}$$

where $\phi(x) = \exp\sinh^{-1}(x)$.

For similar reasons, we choose all f_m and g_m as single-layer feed-forward neural networks with exponential of inverse hyperbolic sine

nonlinearity, thus

$$\hat{X}' = \sum_{m=1}^{M} \mathrm{diag}(\phi(XWa_m))A\mathrm{diag}(\phi(XWb_m))XW, \quad (28)$$

$$d = \sum_{m=1}^{M} \mathrm{diag}(\phi(XWa_m))A\mathrm{diag}(\phi(XWb_m))\mathbf{1}_N. \quad (29)$$

3.5 SepGAT and SimpGAT models

Inspired by GAT, we apply a multi-head attention mechanism in both SepGAT and SimpGAT to stabilize the learning process. We concatenate the outputs of K independent simplified attention mechanism described above to get the final output of the multi-head attention layer

$$X' = \Big\Vert_{k=1}^{K} \sigma(\mathrm{diag}(d_k)^{-1}\hat{X}'_k), \quad (30)$$

where $\Big\Vert$ denotes the concatenation operation, and $\hat{X}'_k$ and d_k for each head are computed from independent parameters W_k and a_k.

Computations for all heads can easily batched together with tensor operations, and the computation of $\hat{X}'_k$ and d_k can also be combined, thus requiring only one sparse matrix multiplication for the entire layer, greatly reducing the running time.

We apply multiple layers of multi-head attention described above to form a neural network. For the last layer in classification, the output dimension is required to be the number of classes for softmax or sigmoid classification. Like GAT, we employ averaging instead of concatenation and apply the softmax function afterward.

We employs dropout mechanism to increase the robustness of the model, especially when the label rate is low. Specifically, we apply node feature dropouts before and after linear transformation in each layer. We also implemented edge dropping inspired by [4], which is directly applied to the adjacency matrix. To keep the computation minimal, we use the same dropped edges in all heads within the same layer. A desirable side effect of edge dropping is the reduced number of edges making computation of the sparse matrix multiplication faster. Note that when edge dropout is used, all edges around a single node can be dropped together, resulting a division by zero error during simplified attention computation. Thus, a small constant ϵ is added to the row sums of all nodes.

The time complexity of the K-head simplified weighted attention layer per epoch is $\mathcal{O}(K|\mathcal{V}|FF' + KM(|\mathcal{V}| + |\mathcal{E}|)F')$ for SepGAT and $\mathcal{O}(K|\mathcal{V}|FF' + K|\mathcal{E}|F')$ for SimpGAT. The time complexity of SimpGAT is same as GAT, and also GCN with the same combined output dimension, that is, $F'_{\mathrm{GCN}} = KF'_{\mathrm{GAT}}$. The time complexities for our approaches along with some important baselines are summarized in Table 1. However, the combination of sparse matrix multiplication in all heads greatly reduces our method's time consumption compared to GAT, giving a per-epoch training time similar to GCN. Even SepGAT with higher complexity can be made faster than GAT with batching on both heads and attention terms, since a larger SDMM in this case is actually faster than multiple smaller SDMM in GAT. Meanwhile, both models retain the superior model capacity of GAT by weighting the nodes differently. Also, without explicit attention coefficient storage for forward and backward, our models require significantly less memory compared to GAT, allowing training on larger graphs including Reddit dataset.

Table 1. Comparison of time complexity of multiple models

Model	Time Complexity Per Layer
GCN (Dense)	$\mathcal{O}(\|\mathcal{V}\|FF' + \|\mathcal{V}\|^2F')$
GCN (Sparse)	$\mathcal{O}(\|\mathcal{V}\|FF' + \|\mathcal{E}\|F')$
GAT (Dense)	$\mathcal{O}(K\|\mathcal{V}\|FF' + K\|\mathcal{V}\|^2F')$
GAT (Sparse)	$\mathcal{O}(K\|\mathcal{V}\|FF' + K\|\mathcal{E}\|F')$
SepGAT	$\mathcal{O}(K\|\mathcal{V}\|FF' + KM(\|\mathcal{V}\| + \|\mathcal{E}\|)F')$
SimpGAT	$\mathcal{O}(K\|\mathcal{V}\|FF' + K\|\mathcal{E}\|F')$

4 EXPERIMENTS

We evaluate our SepGAT and SimpGAT models on the well established benchmarks of citation networks and social networks, and compare our results with existing works, including several strong baselines. Additionally, we compare the training time for our methods and several related approaches to better demonstrate the efficiency of our models.

4.1 Datasets

Cora, Citeseer, and Pumbed are three citation network datasets. Nodes correspond to documents and edges to citations. We use undirected edge in this work, same as all baselines. Node features are bag-of-words representation of documents, zero-one in Cora and Citeseer, and TF-IDF in Pubmed. Each node has a class label based on its subject. We follow the transductive setting of [29], where all node features are available during training.

Reddit is a social network dataset with 232,965 nodes and 11,606,919 edges, which is much larger than the citation networks. The task is to classify Reddit posts as belonging to different communities. Nodes correspond to posts and edges to comments from the same user. The first 20 days of the month are used for training and the remaining days are used for testing [11]. We apply the inductive settings as in [3] for our SepGAT and SimpGAT models, and a transductive setting for GCN baseline.

Dataset statistics are summarized in Table 2.

Table 2. Dataset Statistics

Dataset	Cora	Citeseer	Pubmed	Reddit
Nodes	2,708	3,327	19,717	232,965
Edges	5,429	4,732	44,338	11,606,919
Features	1,433	3,703	500	602
Classes	7	6	3	41
Training	140	120	60	152,410
Validation	500	500	500	23,699
Test	1,000	1,000	1,000	55,334

4.2 Experimental setup

Both SepGAT and SimpGAT are implemented in PyTorch [19]. Both models have two layers for all datasets. On citation networks, we choose the architecture based on GAT, with the first layer consisting of $K = 8$ attention heads computing $F' = 8$ features each. The second layer has $K = 1$ head for Cora and Citeseer, and $K = 8$ for Pubmed. In SepGAT, we use $M = 8$ terms in the attention formulation. We apply a dropout with $p = 0.6$ both before and after linear transformation. The edge dropping rate is 0.6 for SepGAT and 0.3 for SimpGAT on all three citation networks. Implementation of

GCN and GAT baselines are provided by their respective authors. We choose the same hyperparameter setting as their corresponding papers for GCN and GAT baselines. Additionally, a GCN model with 64 hidden units is trained for a fair comparison to GAT, SepGAT, and SimpGAT, each with total hidden units of 64.

On Reddit dataset, the larger number of classes (41) requires a larger amount of hidden units. We empirically find that $K = 16$ attention heads with $F' = 16$ features each on the first layer works best. For SepGAT, we use only $M = 4$ terms in the attention formulation due to memory limitations. The second layer still has $K = 1$ head. The larger amount of training data requires less regularization, thus we apply a dropout with $p = 0.3$ only before linear transformation, and edge dropping is not used. For GCN baseline, 64 hidden units and $p = 0.2$ dropout is used. We also experimented GCN with 256 hidden units in our PyTorch implementation.

To our best of our knowledge, no GAT implementation can fit in our GPU memory, and training on CPU does not make a fair comparison on computation time. As for AGNN, the gather/scatter approach is the only known reasonably memory-efficient algorithm, since it preserves the sparsity of the graph. Yet, with $\mathcal{O}(|\mathcal{E}|F)$ memory requirement, gathering the original input features into the edges is still highly impractical, and the storage requirement of the attention coefficient and its gradient is also prohibitive. As a result, no AGNN implementation fit in our GPU memory, similar to the GAT case.

For all models, we use Adam optimizer [12], with learning rate 0.01 for GCN and 0.005 for other models. L2 regularization of $\lambda = 5 \times 10^{-4}$ is also applied. We repeat 10 times for each experiment.

Additionally, we select several results from various works for a more complete comparison [13, 23, 22, 17, 20, 24, 26, 31, 11, 3]. The results for citation networks and Reddit are summarized in Table 3 and Table 4, respectively.

Table 3. Classification accuracy on Planetoid datasets

Model	Cora	Citeseer	Pubmed
Literature:			
GCN	81.5	70.3	79.0
GAT	83.0 ± 0.7	$\mathbf{72.5 \pm 0.7}$	79.0 ± 0.3
GLN	81.2 ± 0.1	70.9 ± 0.1	78.9 ± 0.1
AGNN	83.1 ± 0.1	71.7 ± 0.1	$\mathbf{79.9 \pm 0.1}$
LNet	79.5 ± 1.8	66.2 ± 1.9	78.3 ± 0.3
AdaLNet	80.4 ± 1.1	68.7 ± 1.0	78.1 ± 0.4
DeepWalk	70.7 ± 0.6	51.4 ± 0.5	76.8 ± 0.6
DGI	82.3 ± 0.6	71.8 ± 0.7	76.8 ± 0.6
SGC	81.0 ± 0.0	71.9 ± 0.1	78.9 ± 0.0
Our experiments:			
GCN16	82.0 ± 0.6	71.1 ± 0.9	79.1 ± 0.4
GCN64	81.6 ± 0.3	71.0 ± 0.3	79.1 ± 0.3
GAT	$\mathbf{83.4 \pm 0.3}$	72.3 ± 0.7	78.1 ± 0.6
SepGAT	83.1 ± 0.6	71.6 ± 0.5	78.2 ± 0.3
SimpGAT	$\mathbf{83.4 \pm 0.4}$	71.8 ± 0.5	79.0 ± 0.3

4.3 Results

It is easily observed that our SimpGAT model achieves the best accuracy score on Cora benchmarks, and has performance comparable to state-of-the-art on Citeseer and Pubmed. We should note that the best models on both Citeseer and Pubmed are based on attention mechanism, thus their time and memory efficiency is severely limited. Our SimpGAT model is much faster while still having competitive performance. SepGAT performs slightly worse, since it has more trainable parameters, it suffers more from overfitting.

Table 4. Classification micro F1 on Reddit dataset

Model	Micro F1
Literature:	
GaAN	**96.4**
SAGE-mean	95.0
SAGE-LSTM	95.4
SAGE-GCN	93.0
SAGE-mean (Unsupervised)	89.7
SAGE-LSTM (Unsupervised)	90.7
SAGE-GCN (Unsupervised)	90.8
FastGCN	93.7
DGI	94.0
SGC	94.9
Our experiments:	
GCN64	94.94 ± 0.03
GCN256	94.98 ± 0.02
GAT	OOM
AGNN	OOM
SepGAT	$\mathbf{96.16 \pm 0.05}$
SimpGAT	95.71 ± 0.05

On Reddit dataset, our SepGAT model is able to beat all baselines except GaAN [31]. GaAN is a node sampling model, allowing larger models to be trained on a reasonable device. However, with a large number of parameters, GaAN has not yet proven itself to be time efficient. SepGAT performs better than most sampling methods, giving the insight that using the entire graph provides more useful information compared to sampling subgraphs. When SepGAT is compared to our SimpGAT and other simpler methods, we find that complex formulations of attention function improves the capacity of models, giving a better performance when overfitting is not an issue.

4.4 Efficiency Comparison

We also compare the training time between SepGAT, SimpGAT, GAT and GCN. All timing experiments are done on a NVIDIA GTX 1080 Ti GPU. The hyperparameters are the same as in respective performance experiments. SepGAT and SimpGAT use inductive training on Reddit, which has a smaller training size than transductive GCN, so transductive versions of SepGAT and SimpGAT are also timed for fair comparison. The results are listed in Table 5.

Table 5. Training time (s) for selected methods. (i): inductive training, (t): transductive training, OOM: Out of memory.

Model	Cora	Citeseer	Pubmed	Reddit
Time Per Epoch:				
GCN16	0.0091	0.0140	0.1517	—
GCN64	0.0099	0.0157	0.1592	2.8803
GCN256	—	—	—	5.5227
GAT	0.0615	0.1764	0.4054	OOM
SepGAT	0.0135	0.0153	0.0527	4.4669(i)
				7.7428(t)
SimpGAT	0.0105	0.0114	0.0288	3.1949(i)
				6.0239(t)
Total Time:				
GCN16	1.91	2.93	24.1	—
GCN64	1.53	2.28	17.2	593
GCN256	—	—	—	1125
GAT	60.6	158	235	OOM
SepGAT	21.2	25.6	51.1	7338(i)
				14034(t)
SimpGAT	12.2	17.2	23.8	4814(i)
				8242(t)

The training time per epoch of both SepGAT and SimpGAT are

similar to that of GCN, while GAT is much slower. Even SepGAT with a higher theoretical complexity is faster than GAT, meaning that the amount rather than the dense size of the SpMM operations are critical in the computational time, and batching of the heads and attention terms are crucial for high efficiency. The advantage of SepGAT and SimpGAT on Pubmed dataset is probably due to implementation difference.

As for the total training time, SepGAT and SimpGAT are slower than GCN but still faster than GAT. This is due to the extra parameters in the attention mechanism increasing the model complexity, making the model harder to optimize. This is a necessary trade-off for SepGAT and SimpGAT to perform better than GCN, which is most evident for SepGAT on Reddit.

Also, by using the same adjacency matrix in all SpMM operations involved in attention computation, our models are more scalable in the sense of memory efficiency, allowing us to train our models on larger datasets like Reddit while GAT cannot.

5 CONCLUSION

We have presented Separated Graph Attention Networks (SepGAT) and Simplified Graph Attention Networks (SimpGAT), which are novel graph neural network architectures inspired by Graph Attention Networks. Both models have performance on par with state-of-the-art, while still roughly as fast as simpler methods including Graph Convolutional Networks. SimpGAT is faster than SepGAT, and performs better on smaller datasets with lower risk of overfitting. Although SepGAT has a larger time complexity than GAT, it can be trained faster with efficient batching of heads and attention terms. SepGAT has a higher model capacity by introducing more complex attention functions, which is beneficial for its performance on datasets with surplus training data like Reddit.

ACKNOWLEDGEMENTS

We would like to thank the referees for their comments, which helped improve this paper considerably. This work is supported by National Natural Science Foundation of China (No.61806009), Beijing Academy of Artificial Intelligence (BAAI) and Intelligent Manufacturing Action Plan of Industrial Solid Foundation Program (No.JCKY2018204C004).

REFERENCES

[1] James Atwood and Don Towsley, 'Diffusion-convolutional neural networks', in *Advances in Neural Information Processing Systems 29*, eds., D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, 1993–2001, Curran Associates, Inc., (2016).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, 'Neural machine translation by jointly learning to align and translate', in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, eds., Yoshua Bengio and Yann LeCun, (2015).

[3] Jie Chen, Tengfei Ma, and Cao Xiao, 'FastGCN: Fast learning with graph convolutional networks via importance sampling', in *International Conference on Learning Representations*, (2018).

[4] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song, 'Adversarial attack on graph structured data', in *Proceedings of the 35th International Conference on Machine Learning*, eds., Jennifer Dy and Andreas Krause, volume 80 of *Proceedings of Machine Learning Research*, pp. 1115–1124, Stockholmsmässan, Stockholm Sweden, (10–15 Jul 2018). PMLR.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, 'Convolutional neural networks on graphs with fast localized spectral filtering', in *Advances in Neural Information Processing Systems 29*, eds., D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, 3844–3852, Curran Associates, Inc., (2016).

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, 'Convolutional neural networks on graphs with fast localized spectral filtering', in *Advances in Neural Information Processing Systems 29*, eds., D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, 3844–3852, Curran Associates, Inc., (2016).

[7] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams, 'Convolutional networks on graphs for learning molecular fingerprints', in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, eds., Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, pp. 2224–2232, (2015).

[8] Matthias Fey and Jan E. Lenssen, 'Fast graph representation learning with PyTorch Geometric', in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, (2019).

[9] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl, 'Neural message passing for quantum chemistry', in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, eds., Doina Precup and Yee Whye Teh, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, (2017).

[10] Aditya Grover and Jure Leskovec, 'node2vec: Scalable feature learning for networks', in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, eds., Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, pp. 855–864. ACM, (2016).

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec, 'Inductive representation learning on large graphs', in *Advances in Neural Information Processing Systems 30*, eds., I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, 1024–1034, Curran Associates, Inc., (2017).

[12] Diederik P. Kingma and Jimmy Ba, 'Adam: A method for stochastic optimization', in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, eds., Yoshua Bengio and Yann LeCun, (2015).

[13] Thomas N. Kipf and Max Welling, 'Semi-supervised classification with graph convolutional networks', in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, (2017).

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, **86**(11), 2278–2324, (Nov 1998).

[15] Qimai Li, Zhichao Han, and Xiao-Ming Wu, 'Deeper insights into graph convolutional networks for semi-supervised learning', *CoRR*, **abs/1801.07606**, (2018).

[16] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu, 'Diffusion convolutional recurrent neural network: Data-driven traffic forecasting', in *International Conference on Learning Representations*, (2018).

[17] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard Zemel, 'Lanczosnet: Multi-scale deep graph convolutional networks', in *International Conference on Learning Representations*, (2019).

[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, 'Efficient estimation of word representations in vector space', in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, eds., Yoshua Bengio and Yann LeCun, (2013).

[19] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, 'Automatic differentiation in PyTorch', in *NIPS Autodiff Workshop*, (2017).

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena, 'Deepwalk: online learning of social representations', in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, eds., Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, pp. 701–710. ACM, (2014).

[21] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfar-

dini, 'The graph neural network model', *IEEE Transactions on Neural Networks*, **20**(1), 61–80, (Jan 2009).
[22] Kiran K. Thekumparampil, Sewoong Oh, Chong Wang, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning, 2018.
[23] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio, 'Graph attention networks', in *International Conference on Learning Representations*, (2018).
[24] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm, 'Deep graph infomax', in *International Conference on Learning Representations*, (2019).
[25] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds, 2018.
[26] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger, 'Simplifying graph convolutional networks', in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, eds., Kamalika Chaudhuri and Ruslan Salakhutdinov, volume 97 of *Proceedings of Machine Learning Research*, pp. 6861–6871. PMLR, (2019).
[27] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks, 2019.
[28] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu, 'A comprehensive survey on graph neural networks', *CoRR*, **abs/1901.00596**, (2019).
[29] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov, 'Revisiting semi-supervised learning with graph embeddings', in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 40–48. JMLR.org, (2016).
[30] Liang Yao, Chengsheng Mao, and Yuan Luo, 'Graph convolutional networks for text classification', in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pp. 7370–7377. AAAI Press, (2019).
[31] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung, 'Gaan: Gated attention networks for learning on large and spatiotemporal graphs', in *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, eds., Amir Globerson and Ricardo Silva, pp. 339–349. AUAI Press, (2018).
[32] Muhan Zhang and Yixin Chen, 'Link prediction based on graph neural networks', in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, p. 5171–5181, Red Hook, NY, USA, (2018). Curran Associates Inc.
[33] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen, 'An end-to-end deep learning architecture for graph classification', in *AAAI*, (2018).
[34] Ziwei Zhang, Peng Cui, and Wenwu Zhu, 'Deep learning on graphs: A survey', *CoRR*, **abs/1812.04202**, (2018).
[35] Marinka Zitnik and Jure Leskovec, 'Predicting multicellular function through multi-layer tissue networks', *Bioinformatics*, **33**(14), i190–i198, (Jul 2017).