Just Jump: Dynamic Neighborhood Aggregation in Graph Neural Networks

Matthias Fey
Department of Computer Graphics
TU Dortmund University
44227 Dortmund, Germany
`matthias.fey@udo.edu`

Abstract

We propose a *dynamic neighborhood aggregation* (DNA) procedure guided by (multi-head) attention for representation learning on graphs. In contrast to current graph neural networks which follow a simple neighborhood aggregation scheme, our DNA procedure allows for a selective and node-adaptive aggregation of neighboring embeddings of potentially differing locality. In order to avoid overfitting, we propose to control the channel-wise connections between input and output by making use of grouped linear projections. In a number of transductive node-classification experiments, we demonstrate the effectiveness of our approach.

1 Introduction and Related Work

Graph neural networks (GNNs) have become the de facto standard for representation learning on relational data (Bronstein et al., 2017; Gilmer et al., 2017; Battaglia et al., 2018). GNNs follow a simple neighborhood aggregation procedure motivated by two major perspectives: The generalization of classical CNNs to irregular domains (Shuman et al., 2013), and their strong relations to the Weisfeiler & Lehman (1968) algorithm (Xu et al., 2019; Morris et al., 2019). Many different graph neural network variants have been proposed and significantly advanced the state-of-the-art in this field (Defferrard et al., 2016; Kipf & Welling, 2017; Monti et al., 2017; Gilmer et al., 2017; Hamilton et al., 2017; Veličković et al., 2018; Fey et al., 2018).

Most of these approaches focus on novel kernel formulations, however, deeply stacking those layers usually result in gradually decreasing performance despite having, in principal, access to a wider range of information (Kipf & Welling, 2017). Xu et al. (2018) blame the strongly varying speed of expansion on this phenomenon, caused by locally differing graph structures, and hence propose to node-adaptively *jump* back to earlier representations if those fit the task at hand more precisely.

Inspired by these so-called Jumping Knowledge networks (Xu et al., 2018), we explore a highly dynamic neighborhood aggregation (DNA) procedure based on scaled dot-product attention (Vaswani et al., 2017) which is able to aggregate neighboring node representations of differing locality. We show that this approach, when additionaly combined with grouped linear projections, outperforms traditional stacking of GNN layers, even when those are enhanced by Jumping Knowledge.

We briefly give a formal overview of the related work before we propose our method in more detail:

Graph Neural Networks (GNNs) operate over graph structured data $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and iteratively update node features $\vec{h}_v^{(t-1)}$ of node $v \in \mathcal{V}$ in layer $t-1$ by aggregating localized information via

$$\vec{h}_v^{(t)} = f_{\boldsymbol{\Theta}}^{(t)}\left(\vec{h}_v^{(t-1)}, \left\{\vec{h}_w^{(t-1)}\right\}_{w \in \mathcal{N}(v)}\right), \quad e.g., \quad \vec{h}_v^{(t)} = \sigma\left(\boldsymbol{\Theta}^{(t)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} C_{v,w}^{(t-1)} \vec{h}_w^{(t-1)}\right), \quad (1)$$

from the neighbor set $\mathcal{N}(\cdot)$ through a differentiable function $f_{\boldsymbol{\Theta}}^{(t)}$ parametrized by weights $\boldsymbol{\Theta}^{(t)}$. In current implementations, $C_{v,w}^{(t-1)}$ is either defined to be static (Xu et al., 2019), structure- (Kipf & Welling, 2017; Hamilton et al., 2017) or data-dependent (Veličković et al., 2018).

GNN layers are typically stacked sequentially, but can be optionally enhanced by skip connections, *e.g.*, $\vec{h}_v^{(t)} \leftarrow \vec{h}_v^{(t)} + \boldsymbol{\Theta}_s^{(t)} \vec{h}_v^{(t-1)}$ (Cangea et al., 2018), or updated using Gated Recurrent Units via $\vec{h}_v^{(t)} \leftarrow \text{GRU}(\vec{h}_v^{(t-1)}, \vec{h}_v^{(t)})$ (Cho et al., 2014; Li et al., 2016). After T layers, $\vec{h}_v^{(T)}$ holds the T-hop subgraph representation centered around node v.

Jumping Knowledge (JK) networks enable deeper GNNs by introducing layer-wise jump connections and selective aggregations to leverage node-adaptive neighborhood ranges (Xu et al., 2018). Given layer-wise representations $\vec{h}_v^{(1)}, \ldots, \vec{h}_v^{(T)}$ of node v, its final output representation is obtained by either

$$
\begin{array}{ccc}
(1)\ \textbf{concatenation}, & (2)\ \textbf{pooling} \quad \text{or} & (3)\ \textbf{summation} \\[4pt]
\vec{h}_v^{(1)} \,\|\, \ldots \,\|\, \vec{h}_v^{(T)} & \max\left(\vec{h}_v^{(1)}, \ldots, \vec{h}_v^{(T)}\right) & \sum_{t=1}^{T} \alpha_v^{(t)} \vec{h}_v^{(t)}
\end{array}
\tag{2}
$$

where scorings $\alpha_v^{(t)}$ are obtained from a bi-directional LSTM (Hochreiter & Schmidhuber, 1997).

Attention modules weight the values of a set of key-value pairs $\boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n \times d}$ according to a given query $\vec{q} \in \mathbb{R}^d$ by computing scaled dot-products between key-query pairs and using the softmax-normalized results as weighting coefficients (Vaswani et al., 2017):

$$
\text{Attention}(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\vec{q}^{\top} \boldsymbol{K}^{\top}}{\sqrt{d}}\right) \boldsymbol{V}
\tag{3}
$$

In practice, the attention function is usually performed h times (with each head learning separate attention weights and attending to different positions) and the results are concatenated.

Grouped operations control the channel-wise connections between an input $\boldsymbol{X} \in \mathbb{R}^{n \times c}$ and an output $\boldsymbol{Y} \in \mathbb{R}^{n \times d}$ to reduce the number of parameters by g, the number of groups (Krizhevsky et al., 2012). If $c = g$, the operation is performed independently over every channel (Chollet, 2017).

2 METHOD

Closely related to the JK networks (Xu et al., 2018), we are seeking for a way to node-adaptively craft receptive-fields for a specific task at hand. JK nets achieve this by dynamically jumping to the most representative layer-wise embedding *after* a fixed range of node representations were obtained. Hence, Jumping Knowledge can not guarantee that higher-order features will not become "washed out" in later layers, but instead will just fall back to more localized information preserved from earlier representations. In addition, fine-grained details may still get lost very early on in expander-like subgraph structures (Xu et al., 2018).

In contrast, we propose to allow jumps to earlier knowledge immediately *while* aggregating information from neighboring nodes. This results in a highly-dynamic receptive-field in which neighborhood information is potentially gathered from representations of differing locality. Each node's representation controls its own spread-out, possibly aggregating more global information in one branch, and falling back to more local information in others.

Formally, we allow each node-neighborhood pair $(v, w) \in \mathcal{E}$ to attend to *all* its former representations $\vec{h}_w^{(1)}, \ldots, \vec{h}_w^{(t-1)}$ while using its output $\vec{h}_{v \leftarrow w}^{(t)}$ for aggregation:

$$
\vec{h}_v^{(t)} = f_{\boldsymbol{\Theta}}^{(t)}\left(\vec{h}_{v \leftarrow v}^{(t)}, \left\{\vec{h}_{v \leftarrow w}^{(t)}\right\}_{w \in \mathcal{N}(v)}\right) \quad \text{where}
$$

$$
\vec{h}_{v \leftarrow w}^{(t)} = \text{Attention}\left(\left(\vec{h}_v^{(t-1)}\right)^{\top} \boldsymbol{\Theta}_Q^{(t)},\ \left[\vec{h}_w^{(1)}, \ldots, \vec{h}_w^{(t-1)}\right]^{\top} \boldsymbol{\Theta}_K^{(t)},\ \left[\vec{h}_w^{(1)}, \ldots, \vec{h}_w^{(t-1)}\right]^{\top} \boldsymbol{\Theta}_V^{(t)}\right)
\tag{4}
$$

with $\boldsymbol{\Theta}_Q^{(t)}, \boldsymbol{\Theta}_K^{(t)}, \boldsymbol{\Theta}_V^{(t)} \in \mathbb{R}^{d \times d}$ denoting trainable symmetric projection matrices. A scheme of this layer is depicted in Figure 1. By ensuring that former information is preserved, our operator can be stacked deep by design, in particular without the need of JK nets.

In practice, we replace the single attention module by multi-head attention with a user-defined number of heads h while maintaining the same number of parameters. We implemented $f_{\boldsymbol{\Theta}}^{(t)}$ as the
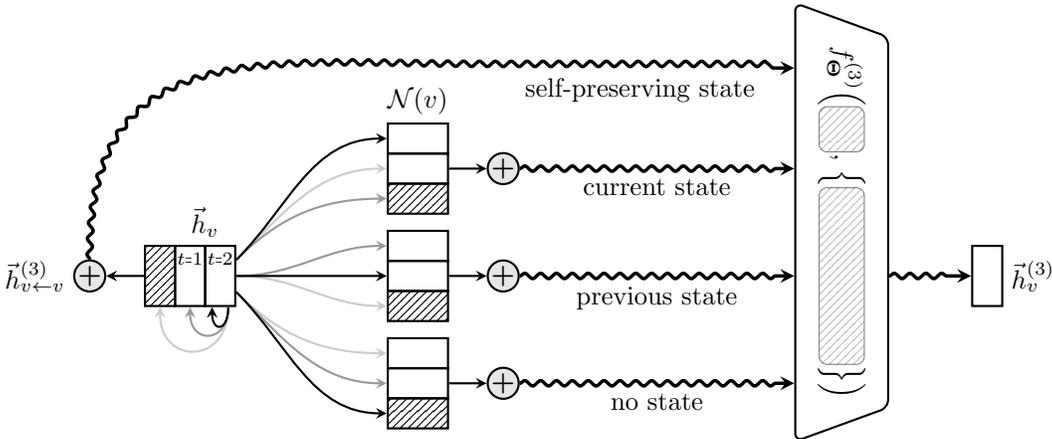
Figure 1: Given current node representation $\vec{h}_v^{(2)}$ as query, a node-adaptive embedding $\vec{h}_{v \leftarrow w}^{(3)}$ gets computed for all neighbors $w \in \mathcal{N}(v)$ based on their former representations $\vec{h}_w^{(1)}$ and $\vec{h}_w^{(2)}$, either preserving current state, previous state, or no state at all. In addition, self-attention is applied to retain central node information.

graph convolutional operator from Kipf & Welling (2017), although any other GNN layer may be applicable. Due to being already projected, we do not transform incoming node embeddings in $f_{\Theta}^{(t)}$.

Furthermore, we incorporate an additional parameter to the softmax distribution of the attention module to allow the model to refuse the aggregation of individual neighboring embeddings in order to preserve fine-grained details (*cf.* Figure 1). Instead of actually overparametrizing the resulting distribution, we restrict this parameter to be fixed (Goodfellow et al., 2016). This results in a softmax function of the form

$$\text{softmax}(\vec{x})_i = \frac{\exp(x_i)}{1 + \sum_j \exp(x_j)} \in \mathbb{R}^n, \quad \text{where } \vec{x} \in \mathbb{R}^n. \tag{5}$$

Feature dimensionality. In order to leverage the attention module, input and output feature dimensionality are forced to remain equal across *all* layers. We found this to be only a weak constraint since this is already common practice (Gilmer et al., 2017; Xu et al., 2018).

Regularization. We apply dropout (Srivastava et al., 2014) to the softmax-normalized attention weights and use grouped linear projections with g groups to reduce the number of parameters from d^2 to d^2/g, where g must be chosen so that $\max(g, h)$ is divisible by $\min(g, h)$. The grouped projections regulate the attention heads by forcing them to only have a local influence on other attention heads (or even restricting them to have no influence at all). We observed that these adjustments greatly help the model to avoid overfitting while still maintaining large effective hidden sizes.

Runtime. Our proposed operator does scale linearly in the number of previously seen node representations for each edge, *i.e.* $\mathcal{O}(|\mathcal{E}|T)$. To account for large T, we suggest to restrict the inputs of the attention module to a fixed-sized subset of former representations.

3 EXPERIMENTS

We evaluate our approach on 8 transductive benchmark datasets: the tasks of classifying academic papers (Cora, CiteSeer, PubMed, Cora Full) (Sen et al., 2008; Bojchevski & Günnemann, 2018), active research fields of authors (Coauthor CS, Coauthor Physics) (Shchur et al., 2018) and product categories (Amazon Computers, Amazon Photo) (Shchur et al., 2018). We randomly split nodes into 20%, 20% and 60% for training, validation and testing. Descriptions and statistics of all datasets can be found in Appendix A. The code with all its evaluation examples is integrated into the *PyTorch Geometric*[1] library (Fey & Lenssen, 2019).

---
[1] https://github.com/rusty1s/pytorch_geometric

Table 1: Results of our DNA approach, in comparison to GCN and GAT with and without Jumping Knowledge. Accuracy and standard deviations are computed from 10 random data splits.

	Model	Cora	CiteSeer	PubMed	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
GCN	JK-None	83.20 ± 0.98	73.87 ± 0.81	86.93 ± 0.25	62.55 ± 0.60	92.90 ± 0.14	95.90 ± 0.16	89.32 ± 0.20	93.11 ± 0.27
	JK-Concat	83.99 ± 0.72	73.77 ± 0.89	87.52 ± 0.25	65.62 ± 0.49	95.44 ± 0.32	96.71 ± 0.15	90.27 ± 0.28	94.74 ± 0.29
	JK-Pool	84.36 ± 0.62	73.86 ± 0.97	87.61 ± 0.27	65.14 ± 0.81	$\mathbf{95.47 \pm 0.21}$	$\mathbf{96.74 \pm 0.17}$	90.30 ± 0.37	94.64 ± 0.24
	JK-LSTM	80.46 ± 0.88	72.92 ± 0.69	87.38 ± 0.29	55.39 ± 0.40	94.40 ± 0.28	96.55 ± 0.08	90.06 ± 0.23	94.54 ± 0.30
GAT	JK-None	$\mathbf{86.35 \pm 0.74}$	73.70 ± 0.53	86.76 ± 0.25	65.70 ± 0.32	93.54 ± 0.17	96.21 ± 0.08	88.02 ± 1.39	93.00 ± 0.42
	JK-Concat	84.70 ± 0.57	73.97 ± 0.46	$\mathbf{88.73 \pm 0.30}$	66.18 ± 0.47	95.12 ± 0.18	96.66 ± 0.09	89.67 ± 0.59	94.93 ± 0.31
	JK-Pool	83.91 ± 0.87	73.42 ± 0.71	88.44 ± 0.33	61.52 ± 1.17	94.84 ± 0.16	96.62 ± 0.06	89.42 ± 0.47	94.80 ± 0.24
	JK-LSTM	78.08 ± 1.53	71.84 ± 1.20	87.85 ± 0.26	55.41 ± 0.35	94.09 ± 0.23	96.45 ± 0.05	87.26 ± 1.82	94.47 ± 0.33
DNA	$g = 1$	83.88 ± 0.50	73.37 ± 0.83	87.80 ± 0.25	63.72 ± 0.44	94.02 ± 0.17	96.49 ± 0.10	90.52 ± 0.40	94.89 ± 0.26
	$g = 8$	85.86 ± 0.45	74.19 ± 0.66	88.04 ± 0.17	66.50 ± 0.42	94.46 ± 0.15	96.58 ± 0.09	$\mathbf{90.99 \pm 0.40}$	94.96 ± 0.24
	$g = 16$	86.15 ± 0.57	$\mathbf{74.50 \pm 0.62}$	88.04 ± 0.22	$\mathbf{66.64 \pm 0.47}$	94.64 ± 0.15	96.53 ± 0.10	90.81 ± 0.38	$\mathbf{95.00 \pm 0.19}$

Setup. We compare our DNA approach to GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018) with and without Jumping Knowledge, closely following the network architectures of Xu et al. (2018): We first project node features separately into a lower-dimensional space, apply a number of GNN layers $\in \{1, 2, 3, 4, 5\}$ with effective hidden size $\in \{16, 32, 64, 128\}$ and ReLU non-linearity, and perform the final prediction via a fully-connected layer. All models were implemented using grouped linear projections and evaluated with the number of groups $\in \{1, 8, 16\}$.

We use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 0.005 and stop training early with a patience value of 10. We apply a fixed dropout rate of 0.5 before and after GNN layers and add a ℓ_2 regularization of 0.0005 to all model parameters. For our proposed model and GAT, we additionaly tune the number of heads $\in \{8, 16\}$ and set the dropout rate of attention weights to 0.8. Hyperparameter configurations of the best performing models with respect to the validation set are reported in Appendix B.

Results. Table 1 shows the average classification accuracy over 10 random data splits and initializations. Our DNA approach outperforms traditional stacking of GNN layers (JK-None) and even exceeds the performance of Jumping Knowledge in most cases. Noticeably, the use of grouped linear projections greatly improves attention-based approaches, especially when combined with a large effective hidden size. We noticed gains in accuracy up to 3 percentage points when comparing the best results of $g = 1$ to $g > 1$, both for GAT and DNA, especially when combined with a large effective hidden size. Best hyperparameter configurations (*cf.* Appendix B) show advantages in using increased feature dimensionalities across all datasets. For GCN, we found those gains to be negligible. Similar to JK nets, our approach benefits from an increased amount of stacked layers.

4 QUALITATIVE ANALYSIS ON CORA

We use the (normalized) influence score $I_v(w) = \mathbb{1}^\top \left| \frac{\partial \vec{h}_v^{(T)}}{\partial \vec{h}_w^{(0)}} \right| \mathbb{1}$ (Xu et al., 2018) to visualize the differences in aggregation starting at a node which is correctly classified by DNA, but is incor-

Figure 2: Influence distributions of different 5-layer GNNs starting at the squared node. Due to visibility, we visualize only its 2-hop neighborhood.

(a) GCN JK-Pool   (b) DNA
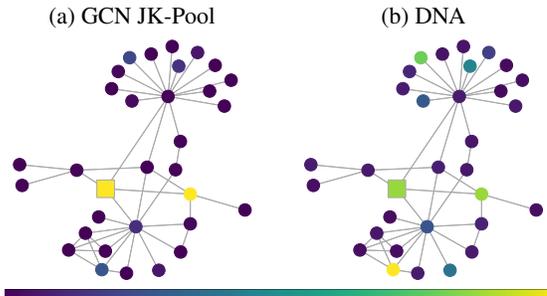
Figure 3: Final attention weight distribution of a 5-layer DNA-GNN.
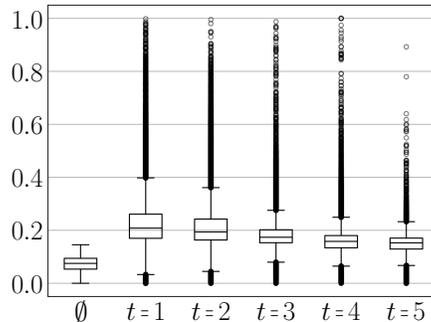
rectly classified by GCN JK-Pool (*cf.* Figure 2). While the node embedding of GCN JK-Pool is nearly exclusively influenced by its central node and a node nearby, DNA aggregates *localized* information even from nodes far away. Figure 3 signals that aggregations typically attend to earlier representations. This verifies that nearby information is indeed often sufficient to classify most nodes. However, there are some nodes that do make heavy usage of information retrieved from latter representations, indicating the merits of a dynamic neighborhood aggregation procedure.

5   CONCLUSION

We introduced a dynamic neighborhood aggregation (DNA) scheme which computes new embeddings for a node by attending to all previous embeddings of its neighbors. This dynamic aggregation allows the model to learn to use specific receptive fields and depths for a given task and naturally solves the problems of exponential spread-outs and "washed out" representations when naively stacking GNN layers. In contrast to JK nets, our DNA scheme enables fine-grained node representations in which both local and global information can effectively be combined across different neighborhood branches. Finally, we showed empirically that grouped operations can be an effective regularizer for attention heads which can additionally enable the usage of larger feature dimensionalities in GNNs.

REFERENCES

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

A. Bojchevski and S. Günnemann. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. In *ICLR*, 2018.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. In *Signal Processing Magazine*, 2017.

C. Cangea, P. Veličković, N. Jovanović, T. N. Kipf, and P. Liò. Towards sparse hierarchical graph classifiers. In *NeurIPS-W*, 2018.

K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *CVPR*, 2018.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), 1997.

D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.

F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*, 2017.

C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

G. Sen, G. Namata, M. Bilgic, and L. Getoor. Collective classification in network data. *AI Magazine*, 29(3), 2008.

O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. In *NeurIPS-W*, 2018.

D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3), 2013.

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), 2014.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomes, and Ł. Kaiser. Attention is all you need. In *NIPS*, 2017.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9), 1968.

K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

A   DATASETS

Table 2: Dataset statistics of the transductive node-classification experiments.

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,278	1,433	7
CiteSeer	3,327	4,552	3,703	6
PubMed	19,717	44,324	500	3
Cora Full	19,793	63,421	8,710	70
Coauthor CS	18,333	81,894	6,805	15
Coauthor Physics	34,493	247,962	8,415	5
Amazon Computers	13,752	245,861	767	10
Amazon Photo	7,650	119,081	745	8

Cora, CiteSeer, PubMed and Cora Full (Sen et al., 2008; Bojchevski & Günnemann, 2018) are citation network datasets where nodes represent documents, and edges represent (undirected) citation links. The networks contain bag-of-words feature vectors for each document.

Coauthor CS and Coauthor Physics (Shchur et al., 2018) are co-authorship graphs where nodes are authors which are connected by an edge if they co-authored a paper. Given paper keywords for each author's paper as node features, the task is to map each author to its most active field of study.

Amazon Computers and Amazon Photo (Shchur et al., 2018) are segments of the Amazon co-purchase graph where nodes represent goods which are linked by an edge if these goods are frequently bought together. Node feature encode product reviews as bag-of-word feature vectors, and class labels are given by product category.

B   HYPERPARAMETER CONFIGURATIONS

Table 3: Hyperparameter configuration (number of layers / effective hidden size / number of groups) of the best GCN models with respect to the validation set.

Model	Cora	CiteSeer	PubMed	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
JK-None	1/128/16	1/128/8	1/16/1	1/128/16	1/128/16	1/32/16	1/128/16	1/64/16
JK-Concat	2/128/8	2/64/8	2/16/16	2/128/8	2/128/1	3/64/1	1/128/1	3/128/1
JK-Pool	2/128/1	2/128/1	2/16/16	5/128/16	5/128/16	5/64/1	1/128/8	3/128/16
JK-LSTM	1/128/8	1/128/1	2/16/8	1/128/8	1/64/1	1/64/8	1/128/16	1/64/1

Table 4: Hyperparameter configuration (number of layers / effective hidden size / number of groups / number of heads) of the best GAT models with respect to the validation set.

Model	Cora	CiteSeer	PubMed	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
JK-None	3/128/16/8	1/128/16/8	1/64/8/8	1/128/16/8	1/128/8/8	1/128/1/8	1/128/1/16	1/128/8/16
JK-Concat	2/128/1/8	2/128/1/8	5/128/16/8	5/128/8/16	3/128/1/8	2/128/1/8	2/128/8/16	2/128/8/16
JK-Pool	5/128/1/16	4/128/1/16	3/128/16/16	2/128/1/16	2/128/1/16	1/128/1/8	2/128/1/16	2/128/8/16
JK-LSTM	1/128/1/16	1/128/1/16	2/16/1/8	1/128/1/8	1/64/1/16	1/64/1/8	1/64/1/16	1/64/1/8

Table 5: Hyperparameter configuration (number of layers / effective hidden size / number of heads) of the best DNA models with respect to the validation set.

Model	Cora	CiteSeer	PubMed	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
$g = 1$	1/128/16	2/128/8	2/16/8	2/128/8	1/128/16	1/32/16	2/128/8	1/64/8
$g = 8$	4/64/8	3/128/16	2/64/8	3/128/8	1/64/8	1/64/8	2/128/16	1/128/8
$g = 16$	4/128/8	4/128/8	2/64/16	2/128/8	1/128/16	1/128/16	1/128/16	1/128/16