Position-aware Graph Neural Networks: An experimental approach

Amokh Varma
Indian Institute of Technology
Delhi
2018MT60527
mt6180527@iitd.ac.in

Achint Kumar Aggarwal
Indian Institute of Technology
Delhi
2018EE10433
ee1180433@iitd.ac.in

ABSTRACT

This paper provides a brief report on Position-aware graph neural networks (P-GNNs) [3], benchmarking their performance at the same time. The datasets which have been used include Brightkite [2], PPI [4] and Protein [1]. The main aim throughout the paper remains to infer characteristics of P-GNNs using the results that we generate through the experiments. [3]. Section 1 is a very brief and high level introduction to PGNNs. In the following sections, task-specific results and the resulting conclusions have been discussed. We wrap up with section 5 which has some miscellaneous results and broader conclusions.

1. INTRODUCTION

The aim of position-aware graph neural networks is to learn low-dimensional position aware (Definition 1 in [3]) graph embeddings for the nodes of a graph. What makes them stand apart from other well-known GNN architectures is there ability capture the positional information whereas the latter build mainly upon the local spatial similarity of nodes. Moreover, like most modern architectures, PGNNs are able to capture node features (if any) in addition to the structural information and also scale well in inductive settings.

The working of PGNNs is based on the idea of having a certain number (chosen on the basis of Bourgain Theorem [3]) of anchor sets, randomly sampled in each forward pass, both while training and testing. The shortest distance of the target node from each node of these anchor sets is then used as a heuristic to generate position aware embeddings. PGNNs can have multiple layers in the sense that the latest set of node embeddings, along with the newly sampled set of anchor-sets can be used to make an update to the embeddings. Please refer [3] for a precise algorithmic framework.

Notations. F represents our message computing function. AGG_M is the message aggregator for all nodes in a given anchor-set while AGG_S represents the function which aggregates the (aggregated) messages of all anchor sets.

Section 2 benchmarks the performance of PGNNs in pairwise node classification using various experiments on protein dataset. In section 3, link prediction on PPI and Brightkite is explored and section 4 explores multi-class node classification using PPI. During these experiments, we also explore the effect of changes in certain hyperparameters including

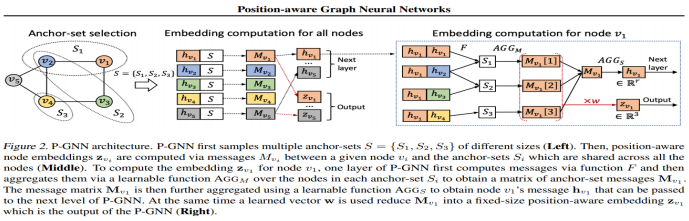


Figure 2. P-GNN architecture. P-GNN first samples multiple anchor-sets $S = \{S_1, S_2, S_3\}$ of different sizes (**Left**). Then, position-aware node embeddings $\mathbf{z}_{v_i}$ are computed via messages M_{v_i} between a given node v_i and the anchor-sets S_i which are shared across all the nodes (**Middle**). To compute the embedding $\mathbf{z}_{v_1}$ for node v_1, one layer of P-GNN first computes messages via function F and then aggregates them via a learnable function AGG_M over the nodes in each anchor-set S_i to obtain a matrix of anchor-set messages $\mathbf{M}_{v_1}$. The message matrix $\mathbf{M}_{v_1}$ is then further aggregated using a learnable function AGG_S to obtain node v_1's message $\mathbf{h}_{v_1}$ that can be passed to the next level of P-GNN. At the same time a learned vector $\mathbf{w}$ is used reduce $\mathbf{M}_{v_1}$ into a fixed-size position-aware embedding $\mathbf{z}_{v_1}$ which is the output of the P-GNN (**Right**).

Figure 1: A basic outline of PGNN architecture from [3]

number of layers, number of anchor sets. We also try varying AGG_S and AGG_M to note the corresponding effect. A point to be noted here is that the aggregator functions must always be permutation independent with respect to the input arguments (since (1) the anchor sets get re-sampled in each forward pass, (2) we want our model to be scalable to an inductive setting). Besides, we have also varied F; F computes the message to be passed by each node (which belongs to some anchor set) using the sender's embedding along with the shortest path distance between the target node and the sender itself. The commonly used idea is to simply scale the sender node's embedding based on the assumption "the nearer the node, higher the effect". Eg. The original paper suggests the use of $F = \frac{1}{1+d_{sp}(u,v)}$ where v and u denote the target node and the sender node respectively. Herein, we experiment by using a computationally faster 2-hop variant of d_{sp} where any distance larger than 2 is taken as ∞. A more general q-hop variant can be employed in a similar manner.

Notations.

- "PGNN-F" will be used to refer to the faster variant talked above. Thus, "PGNN" implictly refers to the exact version.
- "PGNN-x-y" denotes a model with 'x' layers and 'y' anchor sets.

2. PAIRWISE NODE CLASSIFICATION

The task of pairwise node classification (PNC) is a binary hypothesis testing problem. Given a pair of nodes, the task at hand is to predict if the nodes belong to the same class (i.e. have the same labels) or not, irrespective of the exact class that they belong to. In the following subsections, we describe our dataset and the experimental setup, report our results and try reaching certain implications based on any patterns observed in the findings.

2.1 The dataset

Protein[1]. The dataset has 1113 protein graphs. Each node is labeled with a functional role of the protein and has a 29 dimensional feature vector.

2.2 Problem Setting

890 of the 1113 graphs ($\approx 80\%$) have been used for training. Remaining graphs work as our test data. Since we test our model on graphs unseen at the time of training, this is an inductive setting.

2.3 Results and conclusions

- The following table summarizes the results obtained by first varying the number of layers and then varying the nummber of anchor sets.

Table 1: PNC results for PGNN-layer_num-anchor_num

Model	ROC-AUC	Precision	Recall	F1-score
PGNN-3-64	0.746	0.74	0.4746	0.556
PGNN-2-64	0.756	0.748	0.497	0.573
PGNN-1-64	0.782	0.732	0.634	0.658
PGNN-2-32	0.768	0.765	0.538	0.61
PGNN-2-16	0.811	0.803	0.576	0.647

Given a certain number of anchor sets (64 in our case), upon **reducing the number of layers we see an improved performance of our model**. While this may seem counter-intuitive at first, this can have a simple explanation. One must understand that as we make the model deeper (increase the number of layers), we encode an increasing amount of positional information into the resultant embeddings. However, this comes at a cost. Since in each layer we use the latest set of embeddings, we can at best capture the same amount of textual information (given to us in the form of node feature vectors) as we did in the previous layer. Thus, our findings suggest that for PNC (at least for this particular dataset), **the significance of textual information supersedes the relevance of positional knowledge**. The results upon varying the number of anchor sets are comparable and do not reveal much about either the dataset or the model.

- Varying the aggregator functions, we get:

Model	ROC-AUC	Precision	Recall	F1-score
PGNN_Mean	0.756	0.748	0.497	0.573
PGNN_Add	0.783	0.785	0.489	0.576
PGNN_Max	0.785	0.76	0.569	0.625

While the comparative performance of aggregators can vary a lot with the dataset one is observing, the aggregators used here give similar results which at the very least testifies to the compatibility of PGNNs with various aggregation functions.

- We now run the model with and without dropout. An improvement in all the metrics was reported when the model was run without dropout. This suggests that the complexity of our model wasn't enough to capture the information we needed and removing dropout helped us improve this complexity. Another possible way to change the complexity can be to change the dimension of the output computed by the message computation function, F or change the number of anchor sets. However, changing the number of anchor sets is generally not a good way out since it may affect position awareness of our embeddings and causes a change in dimensionality of our output. (Kindly refer Table 2 for the results.)

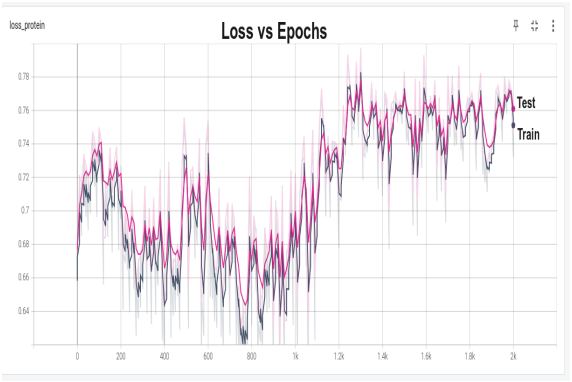


Figure 2: Loss vs Epoch for Pairwise Classification in Protein Dataset

Table 2: Effect of dropout - Protein

Model	ROC-AUC	Precision	Recall	F1
PGNN-dropout	0.756	0.748	0.497	0.573
PGNN-dropout_no	0.851	0.810	0.661	0.702

Table 3: Using the PGNN-F-2-64 variant - Protein

Model	ROC-AUC	Precision	Recall	F1
PGNN	0.756	0.748	0.497	0.573
PGNN-F	0.784	0.700	0.572	0.632

- We also experiment by varying the message computation function, F to analyze a faster variant of PGNNs, as mentioned in Section 1. One must note that the computation which one saves in this manner isn't the training time but the pre-processing time since the shortest distances between nodes are computed only once and then used in a "look-up table" fashion.

- The following graphs show the variation of Loss and ROC-AUC with number of epochs. We can clearly see the problem of **overfitting** when trained for **2000** epochs. This motivates us to use lesser number of epochs.

3. LINK PREDICTION

The task of Link Prediction (LP) is another binary hypothesis testing problem. Given a pair of nodes, the task at hand is to predict if the nodes are connected by an edge or not. In the following subsections, we describe our dataset(s) and the experimental setup, report our results and try reaching certain implications based on any patterns observed in the findings.

3.1 The datasets

PPI[4]. The dataset has 24 protein-protein interaction network graphs. Each graph has 3000 nodes with an average degree of 28.8. Further, each node has 50 dimensional feature vector.

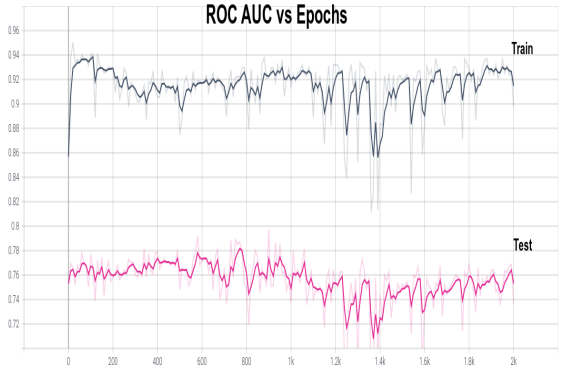


Figure 3: ROC AUC vs Epoch for Pairwise Classification in Protein Dataset. We can clearly see overfitting.

Brightkite[2]. The dataset has a friendship network and consists of 58,228 nodes and 214,078 edges. The network has been used after some modifications to the original version, as mentioned in [2].

3.2 Problem Setting

PPI[4]. Here as well, we train on $\approx 80\%$ of the graphs and test and validate on the remainder of the dataset, thus enabling us to work in an inductive setting.
Brightkite[2]. Since the dataset only has a single graph, we train only on 80 % of the nodes and utilize others as test nodes. The setting here is thus transductive.

3.3 Results and Conclusions

- Varying the number of layers and the number of anchor sets for both the datasets, we get:

Table 4: PPI Dataset

Model	ROC-AUC	Precision	Recall	F1
PGNN-3-64	0.747	0.723	0.018536	0.667
PGNN-2-64	0.784	0.730	0.701	0.715
PGNN-1-64	0.733	0.689	0.660	0.674
PGNN-2-32	0.687	0.667	0.600	0.631
PGNN-2-16	0.707	0.690	0.602	0.642

Table 5: Brightkite Dataset

Model	ROC-AUC	Precision	Recall	F1
PGNN-3-64	0.804	0.765	0.699	0.731
PGNN-2-64	0.795	0.751	0.694	0.721
PGNN-1-64	0.756	0.703	0.718	0.711
PGNN-2-32	0.689	0.636	0.686	0.659
PGNN-2-16	0.766	0.710	0.696	0.703

  Upon varying the number of layers for a given number of anchor sets, we see that for most of the metrics, **2/3 - layer models outperform the model with a single layer** (unlike section 3.3) for both the datasets. These results suggest that positional information has an increased relevance for Link Prediction as compared to Pairwise Node Classification. This implicitly means a decreased significance of textual information when compared to PNC.

  Secondly, an improvement in the metrics is observed with an increase in the number of anchor sets. The above reasoning also suggests that the model with 64 anchor sets captures positional information better than other variants, which is inline with our intuition and hence boosts our confidence in the understanding of the model.

- Results obtained by varying the aggregator functions were also captured but didn't differ significantly in either case and thus didn't bring out an implications regarding the model or the dataset. They have thus not been reported in detail for the sake of brevity. An **ROC-AUC of around 0.76** with an **F1 score of 0.70** was reported on average. Although not very different, the best results were found to be **best using mean as the aggregator**, in agreement to what is mentioned in [3].

- The next experiment we ran involved tweaking with the dropout feature. An enhancement in the performance in almost all metrics for PPI was observed once dropout wasn't taken into account. A possible explanation for the same can be the use of a model which had lower complexity than what would have been optimal. However, dropout helps the model perform better for Brightkite which suggests it may have a good regularization effect on the dataset. This reasoning is also supported by the fact that overfitting (a very high value for ROC-AUC for the training data) was observed after only a few epochs in the training phase for Brightkite.

  Note that, to increase the model complexity, we can also change the output dimensions of the message computation function, F or change the number of anchor sets. The results with the change in latter have already been reported above.

Table 6: Effect of dropout - PPI

Model	ROC-AUC	Precision	Recall	F1
PGNN-dropout	0.78	0.730	0.701	0.715
PGNN-dropout_no	0.791	0.749	0.697	0.722

Table 7: Effect of dropout - Brightkite

Model	ROC-AUC	Precision	Recall	F1
PGNN-dropout	0.795	0.751	0.694	0.721
PGNN-dropout_no	0.786	0.737	0.691	0.710

- Using the PGNN-F variant for link prediction, we get:

Table 8: Dataset used: PPI

Model	ROC-AUC	Precision	Recall	F1
PGNN	0.78	0.730	0.701	0.715
PGNN-F	0.694	0.653	0.624	0.638

  The table shows a decline in performance for the fast (approximate) variant which further cements the stronger role structural information plays in link prediction as compared to pairwise node classification.

- The following graphs show quality (measured in terms of ROC-AUC) versus the number of epochs for the PPI dataset.

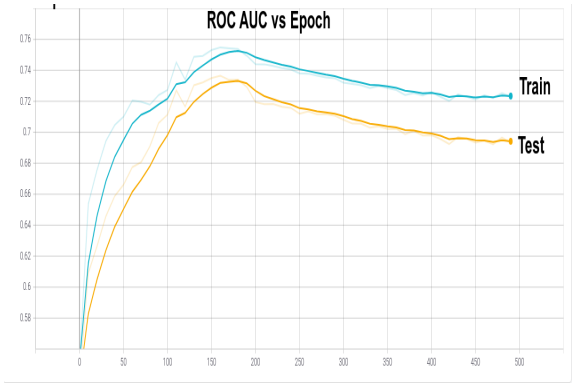


Figure 4: ROC AUC vs Epoch for Link Prediction in PPI Dataset

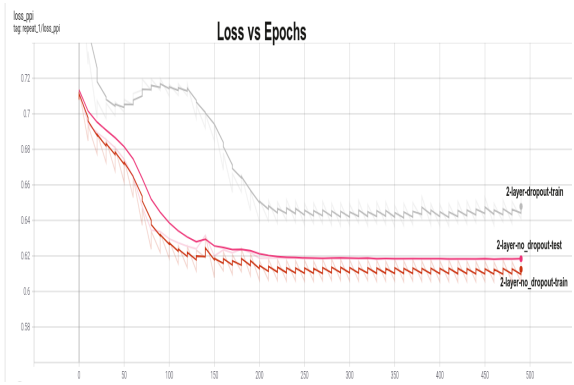


Figure 5: Loss vs Epoch for Pairwise Classification in Protein Dataset

The graph in Fig 5 shows the training loss with progression of time. As one must expect, the same decreases monotonically.

4. MULTICLASS NODE CLASSIFICATION

The task of multiclass node classification (MNC) is a M-ary hypothesis testing problem if M different labels are provided. Given a node, the task at hand is to predict the class/ label, the node belongs to. In the following subsections, we describe our dataset(s) and the experimental setup, report our results and try reaching certain implications based on any patterns observed in the findings. **PGNN are not the ideal architecture for this task.** This is because of the variation caused in anchor selection due to permutation of nodes, as mentioned in [3]. We also verify this empirically.

4.1 The dataset

PPI [4] We use ppi as the dataset. It contains 24 graphs. The graphs have an average degree of 28.8. Each node has a 50 dimensional vector and belongs to multiple classes , out of a total of 121 classes.

4.2 Problem Setting

The data is divided into 20 train and 2 validation and test sets respectively. Our aim is to predict the classes corresponding to each of the nodes.

4.3 Results and conclusion

We do this by fixing the number of anchors and then having an MLP layer over the embeddings. The results are as follows :

Table 9: PPI Dataset

Model	ROC-AUC	Precision	Recall	F1
PGNN-3-64	0.5426	0.5138	0.504	0.512
PGNN-2-64	0.6556	0.6212	0.4807	0.5426
PGNN-1-64	0.6236	0.6312	0.5463	0.5872
PGNN-2-32	0.6146	0.6032	0.6245	0.6111
PGNN-2-16	0.6064	0.5897	0.5574	0.5701

It is clearly visible that the results are **very poor**. We reason that in the following way, quoting [3] , "Inductive position-aware node classification is not well defined due to permutation of labels in different graphs. However pairwise node classification, which only decides if nodes are of the same class, is well defined in the inductive setting". However, fixing the anchors makes the results more sensible and that is what we have done. As with the previous cases, we still notice that the 2 layer case gives a good result. Further, increasing anchors gives only limited change in the metrics. In accordance to the previous results, the "mean" gives the strongest results.

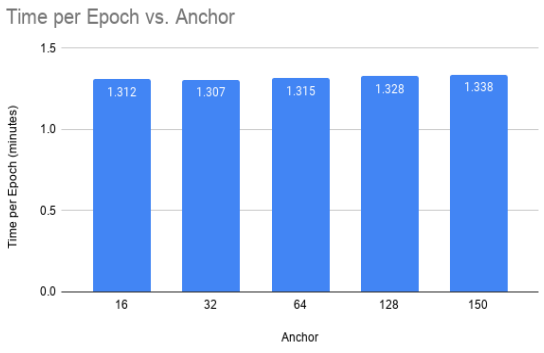
Table 10: PPI Dataset

Model	ROC-AUC	Precision	Recall	F1
PGNN_{Mean}	0.6556	0.6212	0.4807	0.5426
PGNN_{Add}	0.6356	0.6101	0.4422	0.5527
PGNN_{Max}	0.6136	0.6119	0.5471	0.5872
$\text{PGNN-no}_{dropout}$	0.5426	0.5897	0.5574	0.5701
PGNN-F	0.6464	0.6161	0.4907	0.5601

5. MISCELLANEOUS RESULTS

5.1 Computation time Analysis

* The following bar graph shows Time taken per epoch for a model to run upon variation of the number of anchor sets. As can be seen, the different is not very significant and the number of anchor sets (and hence the dimensionality of the output embedding) can be varied without much computational expense.



* The following chart shows a significant difference in the computation time of various models. PGNNs happen to be computationally heavier when compared to other models. The time taken includes the forward pass followed by the backpropagation. This time of PGNN will get even worse

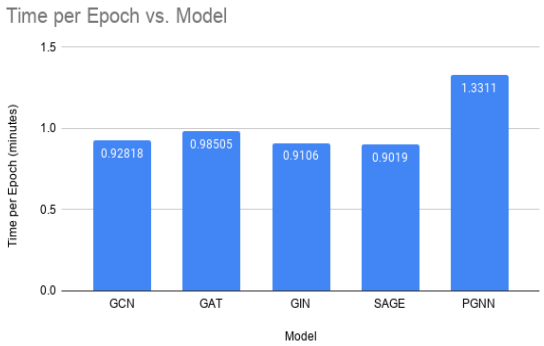


Figure 6: Time taken by different architectures per epoch.

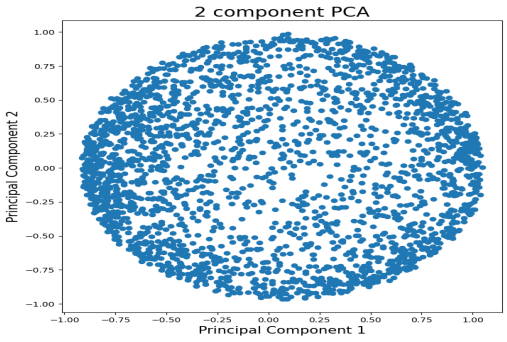


Figure 7: *2D PCA for GCN embeddings on grid dataset*

if we account for the time to load the data, because we are actually pre-computing the distances.

5.2 PCA on Node Embeddings:

Dataset. The dataset we use for this purpose is a simple one. It is a 2D grid graph representing a 50×50 grid with $\|V\| = 2500$ and no node features. The dataset has been chosen because (1) It is easy to visualise, hence helps in interpreting the results. (2) The dataset does not contain textual information, hence helps us understand the structure capturing ability of the models under consideration.
In the following scatter plots, embeddings generated by PGNNs and GCNs have been reported. The embeddings have been reduced to two dimensions using PCA. Also, 2 layer model was used for GCN while a 3 layer model worked well for PGNN. 128 anchor sets were used in case of PGNNs.
The choice of hyperparameters was largely empirical but driven by the intuition of being able capture the structural information's spatial component for PGNNs and positional component for GCNs. The findings of this experiment are rather refreshing and are explained in some detail in what follows.

GCNs. One observes a circular pattern in the emebeddings with an increasing density as we move towards the edges. Since the embeddings generated by GCN are expected to have local spatial awareness, the result can be explained using the same reasoning. We have used a 2 layer model, hence it captures the spatial information for each node upto 2 hops. This implies a large majority of the nodes must have similar embeddings. This is clearly visible in our scatter plot since most of the node embeddings lie towards the periphery of the circular region and denote this similarity. However, though the GCNs understand that these embeddings must differ in some sense (as they are towards different directions from the centre, though on the periphery) due to different embeddings for the neighbors, they do not capture the positional information.

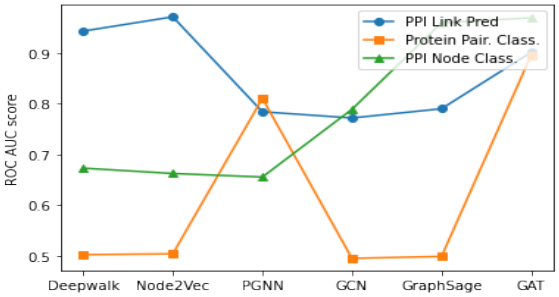
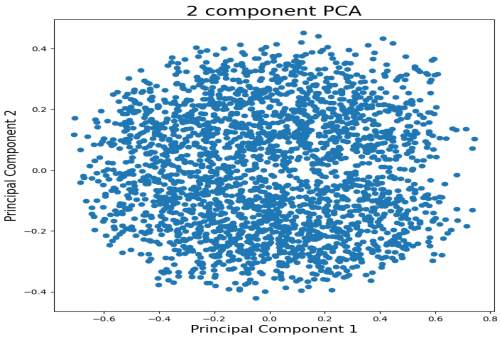


Figure 8: ROC- AUC line plot for all architectures



: Figure 9 : *2D PCA for PGNN embeddings on grid dataset*

PGNNs. The scatter plot for PGNNs seems positionally more aware as compared to that obtained using GCNs as shortest distances are more accurately captured by the embeddings here. This is because it possesses a more or less constant density over the entire region except the periphery. This is indeed close to what one would expect from an ideal position aware set of embeddings. Though position aware, these embeddings do not very successfully capture local spatial information unlike GCNs.

5.3 All Architectures

We conclude with the above plot (8) shows varying performance for different architectures under different tasks. Thus none of the architectures is optimal for all tasks and performance varies with tasks and the datasets under consideration.

6. CONCLUSION

In this report, we have empirically analysed the different variants of PGNN and tried to establish a comparative study. We briefly discuss the different characteristics of PGNNs and also use various datasets to form our conclusions. We finally summarize all the results through graphs and a visualisation of the embeddings and the time requirement.

References

[1] Karsten M. Borgwardt et al. "Protein Function Prediction via Graph Kernels". In: *Bioinformatics* 21.1 (Jan. 2005), pp. 47–56. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bti1007. URL: https://doi.org/10.1093/bioinformatics/bti1007.

[2] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. "Friendship and Mobility: User Movement in Location-Based Social Networks". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. San Diego, California, USA: Association for Computing Machinery, 2011, pp. 1082–1090. ISBN: 9781450308137. DOI: 10.1145/2020408.2020579. URL: https://doi.org/10.1145/2020408.2020579.

[3] Jiaxuan You, Rex Ying, and Jure Leskovec. "Position-aware Graph Neural Networks". In: *CoRR* abs/1906.04817 (2019). arXiv: 1906.04817. URL: http://arxiv.org/abs/1906.04817.

[4] Marinka Zitnik and Jure Leskovec. "Predicting multicellular function through multi-layer tissue networks". In: *Bioinformatics* 33.14 (July 2017), pp. i190–i198. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx252. eprint: https://academic.oup.com/bioinformatics/article-pdf/33/14/i190/25157097/btx252.pdf. URL: https://doi.org/10.1093/bioinformatics/btx252.