

GATCheck: A Detailed Analysis of Graph Attention Networks

Lovish Madaan*

Indian Institute of Technology, Delhi
lovish97@gmail.com

Siddhant Arora*

Indian Institute of Technology, Delhi
siddhantarora1806@gmail.com

ABSTRACT

Graph Attention Networks (GATs) are widely used for Representation Learning in Graphs, but there is no proper study highlighting on what tasks GATs perform better than other models and why. In this appraisal paper, we aim to improve our understanding of GATs on a variety of tasks, including link prediction, multi-class node classification, and pairwise node classification on benchmark datasets. We also perform ablation studies on the various hyperparameters of GATs and try to reason about the importance of each of these in node classification and link prediction tasks. Our study offers insights into the effectiveness of GATs as compared to other techniques, and we make our code¹ public so as to facilitate future exploration.

ACM Reference Format:

Lovish Madaan and Siddhant Arora. 2020. GATCheck: A Detailed Analysis of Graph Attention Networks. In *Proceedings of MLG'20*. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Graph based data are encountered in many real-world applications across a number of domains. For instance biological networks, social networks, world-wide web, knowledge graphs, etc. can all be modeled using graphs in an intuitive way. These graphs can be very large and complex so as to capture the various relationships (edges) between the entities (nodes). Modern machine learning toolbox comprising of deep neural networks introduced originally for Vision and Natural Language Processing (NLP) has been adapted in the graphs domain with varying levels of success. The basic idea is to get good node embeddings which can be further used in a downstream task like node classification, link prediction, etc. Good node embeddings should be able to capture the structural and node level information present in a graph. Earlier methods to learn node embeddings aimed to extend standard word2vec approach over graphs by using random walks over node to generate sentences([1], [2]). Graph Convolutional Networks (GCNs) introduced in [3] gather information from a node's neighborhood and each neighbor node contributes equally (or weighted by it's degree) during message passing. [4] introduced various aggregation functions like mean, pool, LSTM along with GCN-like aggregation

*Equal Contribution

¹<https://github.com/lovishmadaan/gatcheck>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MLG'20, KDD, August 2020

© 2020 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM... \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

on a fixed-sized neighborhood sample of a node in the message passing step. Graph Attention Networks (GATs) [5] overcome the shortcomings of the previous works by *learning* to assign varying levels of importance to all the nodes in the neighborhood of the node under consideration, rather than treating each node as equally important or using a fixed weight. Moreover, GATs are generalizable to unseen nodes (inductive learning), thus simulating a real-world setting.

2 BACKGROUND

Here we will describe the working of a GAT layer in brief. The input to a layer is a set of node features, $\mathbf{x} = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N\}$, where N is the number of nodes. The layer transforms the input node features to $\mathbf{x}' = \{\vec{x}'_1, \vec{x}'_2, \dots, \vec{x}'_N\}$. The input features are first transformed to a different embedding space by multiplying with a transform matrix \mathbf{W} . The parameters of \mathbf{W} can be learned using a Multi-Layered Perceptron (MLP). After the transformation, an attention mechanism a is applied on every edge, indicating the importance of the features of source node to the target node. This attention is commonly referred to as self-attention or intra-attention.

$$e_{ij} = a(\mathbf{W}\vec{x}_i, \mathbf{W}\vec{x}_j) \quad (1)$$

where e_{ij} is the weight corresponding to the edge between nodes i and j . For each node i , the weights are then normalized for the neighborhood \mathcal{N}_i using the softmax function:

$$\alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \quad (2)$$

Various attention mechanisms have been used in the literature [6, 7], and the original GAT paper uses the additive attention introduced in [6], i.e.

$$e_{ij} = a(\mathbf{W}\vec{x}_i, \mathbf{W}\vec{x}_j) = \sigma(\vec{a}^T [\mathbf{W}\vec{x}_i \parallel \mathbf{W}\vec{x}_j]) \quad (3)$$

In equation 3, \vec{a} is a weight vector, σ is some non-linearity like LeakyReLU, \cdot^T represents transpose, and \parallel is the concatenation operator.

The coefficients e_{ij} obtained from equation 3 are put in equation 2 to get the normalized attention weights. Once we obtain these weights, the transformed feature vector \vec{x}'_i is obtained as a weighted sum of input feature vectors of the node's neighborhood, demonstrated as follows:

$$\vec{x}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}\vec{x}_j\right) \quad (4)$$

If we want the node to retain information from the input features, we add self-loops in the graph and consider node i as part of the neighborhood \mathcal{N}_i .

Table 1: Summary of the datasets used in our experiments. NC: Multi-Class Node Classification, PNC: Pairwise Node Classification, LP: Link Prediction

	PPI	Protein	Brightkite
Task	NC, LP	PNC	LP
# Graphs	24	1113	4
# Nodes	56944	43474	58222
# Edges	818716	162088	214078
# Features / Node	50	29	3
# Classes	121 (multilabel)	3	-

GAT employs multi-head attention mechanism to stabilize the learning process as credited to [7]. Moreover, each attention head can learn different features of the graph.

$$\vec{x}'_i = \prod_{k=1}^K \sigma \left(\sum_{j \in N_i} \alpha_{ij} \mathbf{W}^k \vec{x}_j \right) \quad (5)$$

where \mathbf{W}^k is the learnable transformation matrix of the k^{th} attention head and K is the total number of attention heads. In the output layer, *averaging* is used instead of concatenation to get the final output embedding as demonstrated below:

$$\vec{x}_i = \sigma \left(\frac{1}{K} \sum_{k=1}^K \sum_{j \in N_i} \alpha_{ij} \mathbf{W}^k \vec{x}_j \right) \quad (6)$$

3 EXPERIMENTS

In this section, we present our experiments and architectural settings in detail. All our code is in Python using the PyTorch [8] framework. We use Deep Graph Library (DGL) [9] and PyTorch Geometric [10] for GAT implementations. We use the evaluation format followed in [11] to load the datasets as well as evaluate our methods for pairwise node classification and link prediction tasks.

3.1 Datasets

We use the following datasets for our experiments:

- **PPI:** This dataset consists of graphs corresponding to different human tissues [12]. After removing the isolated² components, this datasets comprises of 24 graphs (20 train + 2 validation + 2 test). This is an inductive dataset - test graphs are completely unobserved during training. The average number of nodes per graph is 2372 and each node has 50 input features. There are 121 classes for the nodes and each node can belong to multiple classes.
- **Proteins:** This dataset consists of 1113 protein graphs (889 train + 112 validation + 112 test) where each node is labelled as function role for protein [13]. Each node has 29 input features and the number of nodes in each graph range from 4-620.
- **Brightkite:** This dataset comprises of a friendship network collected from a location-based social networking provider where users share their location by checking-in [14]. The

network has 58,222 nodes and 214,078 edges, along with 4,491,143 checkins of users over the period of Apr. 2008 - Oct. 2010.

A summary of the datasets used is given in Table 1.

3.2 Tasks

Given below is the description of the tasks under consideration.

Multi-Class Node Classification: We use the PPI dataset for this task. Each node can belong to zero or more of the 121 classes available. So, the output prediction is a 121-sized binary vector where elements corresponding to multiple indexes can be 1. In the test dataset, each node belongs to an average of 36.22 classes with minimum and maximum classes being 0 and 101, respectively.

Pairwise Node Classification: We use the Proteins dataset for this task. We first normalise all the features such that they have zero mean and unit variance. This is done to facilitate training. Then for each graph, we observe pair of nodes that have same label and add them to our *positive labeled* set for pairwise node classification. We follow the *transductive* setting for this task and keep 10% of positive set as separate for testing. Moreover for each of train, validation and test set splits, we add negative samples by sampling pair of nodes not in our *positive labeled* set.

Link Prediction: We use PPI and Brightkite datasets for this task. We use the transductive setting exactly similar to the one considered in Pairwise Node Classification. We logarithmically scale all the features in the PPI dataset so as to avoid being thrown off by big counts. This methodology was used in [11] and we found it helpful in our training framework.

For Brightkite, we found that the entire dataset consists of 4 connected parts where one of the connected parts is a very large graph consisting around 212,000 edges and the other graphs are relatively small. For each node, we have 3 features - average check in time, average latitude and average longitude. We logarithmically scale all the features as done in PPI and follow the transductive setting. Scalability was a very important component in our experiments for this dataset. In P-GNN, the entire graph is loaded in GPU which is not feasible in this setting. Also, the entire connected component is passed to the model in first iteration. We make our model scalable by passing all the node features but only passing the edge index in batches of 512. However, since the parameters to compute hidden feature of the nodes should consider all the edges in the connected component, we perform gradient accumulation and take a step of optimiser only after making a pass over the whole graph. Finally, to make an inference over the validation and test set, we bring our model to CPU and pass the entire connected component to the model for computing the prediction scores. We observe that by using these simple heuristics, we were able to achieve scalability of Graph Attention Networks (GATs) on large graphs.

3.3 Experimental Setup

For all the tasks in this section, we use 5-fold cross validation for training and evaluation. We report Precision/Recall/F1 scores for

²Connected components with less than 10 nodes

Table 2: Multi-class Node Classification on PPI. $L = 4, h_{dim} = 256, K_{hid} = 4, K_{out} = 6, act = ReLU, concat = True$

Fold Number	Valid Precision	Valid Recall	Valid F1 Score	Test Precision	Test Recall	Test F1 Score
1	0.947	0.913	0.930	0.986	0.979	0.983
2	0.959	0.908	0.932	0.987	0.973	0.980
3	0.942	0.893	0.917	0.984	0.970	0.977
4	0.971	0.958	0.964	0.985	0.975	0.980
5	0.990	0.980	0.985	0.986	0.972	0.979
Cumulative	0.962 ± 0.019	0.930 ± 0.037	0.946 ± 0.028	0.986 ± 0.001	0.972 ± 0.004	0.979 ± 0.002
Node2Vec [1]	-	-	-	-	-	0.479
Deepwalk [2]	-	-	-	-	-	0.431
GCN [3]	-	-	-	-	-	0.768
GraphSAGE [5]	-	-	-	-	-	0.768

Table 3: Link Prediction on PPI and Brightkite. P-GNN (best) refers to the best P-GNN variant reported in [11]. Bold value is the best in terms of performance on the test set. $L = 3, h_{dim} = 128, K_{hid} = 1, K_{out} = 6, act = ReLU, concat = True$

Fold Number	PPI		Brightkite	
	Valid ROC AUC	Test ROC AUC	Valid ROC AUC	Test ROC AUC
1	0.803	0.803	0.754	0.856
2	0.815	0.812	0.738	0.780
3	0.806	0.806	0.685	0.784
4	0.799	0.799	0.737	0.714
5	0.778	0.776	0.721	0.701
Cumulative	0.8 ± 0.013	0.812 ± 0.013	0.727 ± 0.023	0.856 ± 0.055
Node2Vec [1]	-	0.557	-	0.813
Deepwalk [2]	-	0.553	-	0.742
GCN [11]	-	0.794	-	0.804
GraphSAGE [11]	-	0.712	-	0.501
P-GNN (best) [11]	-	0.808 ± 0.003	-	-

Multi-Class Node Classification and ROC AUC scores for Pairwise Node Classification and Link Prediction. For cumulative scores, we report the mean cross-validation scores and test scores corresponding to the best cross-validation score. We consider a range of hyperparameters for tuning in our experiments as described below:

- **Layers (L)** - The number of Graph Attention layers used in our methodology.
- **Hidden Dimension (h_{dim})** - The dimension of the node embeddings in the hidden layers.
- **Hidden GAT Layer Attention Heads (K_{hid})** - The number of attention heads in the hidden layers of GAT network.
- **Output GAT Layer Attention Heads (K_{out})** - The number of attention heads in the output layer of GAT network.
- **Activation (act)** - The non linearity activation function used in the GAT network.
- **Aggregation ($aggr$)** - The aggregation function used in the message passing step. It can be *add*, *max*, *mean*.
- **Concatenation for Attention Heads ($concat$)**: A boolean value indicating whether to concatenate or average the outputs obtained from the different attention heads in the hidden layer.

3.4 Results

We experimented over the following hyperparameters for each dataset - $L \in \{2, 3, 4\}$, $h_{dim} \in \{128, 256, 512\}$ and $K_{hid} \in \{4, 6\}$. A summary of the results along with their best hyperparameters is presented in Tables 2, 3, and 4.

Multi-class Node Classification (Table 2): The final GAT layer with 6 attention heads is used for the (multi-label) classification. Similar to [5], we use skip connections across the 2^{nd} attention layer. We use cross entropy with logits as the loss function because it is more stable while training. The default epochs used is 400, but we use early stopping with a patience of 20 epochs. Training takes around 30 minutes to complete.

Link Prediction (Table 3): We observe that we get better results than the ones reported in [11] and are marginally better than the state of the art P-GNN. This shows that P-GNNs are not much better than GATs for link prediction. We ran the training loop in PPI dataset for about 400 epochs and the running time is around 1 hour. Similarly for Brightkite, we trained for 400 epochs which took around 10 hours for each fold. We further observe the variation of

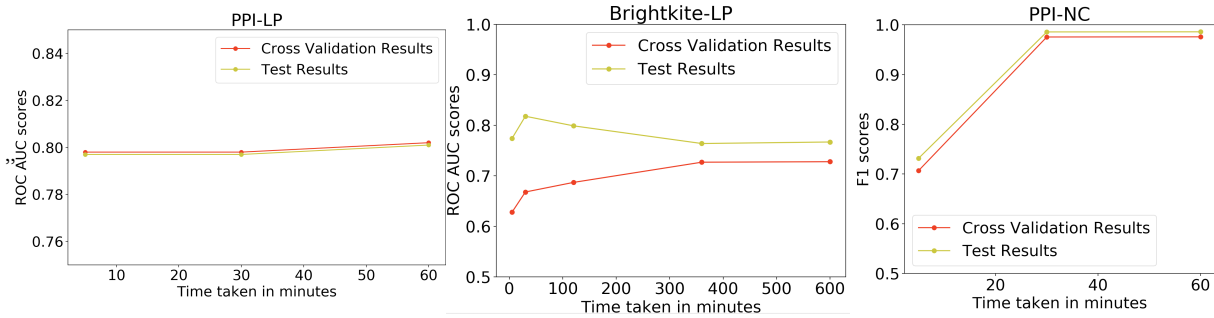


Figure 1: Graph showing the accuracy score [ROC AUC for left, center graphs and F1 score for right graph] (y-axis) obtained at the training time (x-axis). The left graph shows the variation for PPI dataset and center graph shows the variation for Brightkite dataset on Link Prediction. The right graph shows the variation of Multi Label Node Classification on PPI dataset.

Table 4: Pairwise Node Classification on Proteins. P-GNN (best) refers to the best P-GNN variant reported in [11]. $L = 3$, $h_{dim} = 32$, $K_{hid} = 1$, $K_{out} = 1$, $act = ReLU$, $concat = True$

Fold Number	Valid ROC AUC	Test ROC AUC
1	0.516	0.514
2	0.511	0.513
3	0.518	0.505
4	0.524	0.512
5	0.515	0.514
Cumulative	0.517 ± 0.004	0.512 ± 0.003
Node2Vec [1]	-	0.604
Deepwalk [2]	-	0.557
GCN [3]	-	0.63
GraphSAGE [4]	-	0.65
P-GNN (best) [11]	-	0.729 ± 0.176

quality of our GAT network with training time in Figure 1. The ROC-AUC scores do not vary a lot with training time for PPI and we observe very good performance even for small training time in our framework, indicating that the training reaches the local minima very quickly. However, for a dataset with large graphs like BrightKite, we observe gains in Cross Validation Performance with increasing training time whereas our test performance seems to plateau after some time. Eventually both our Cross Validation and test Performances plateau after a training time of 6 hours in each fold.

Pairwise Node Classification (Table 4): We follow the default setting used in [11] of using 3 layers with 1 attention head and hidden dimension of 32 in our GAT network. We observe that our scores are similar to the ones reported in paper. We even increased the number of input and output attention heads but did not observe any clear increase in performance. We can clearly observe that Position Aware Graph Neural Networks [11] seem better suited for this task.

Table 5: Hyperparameters considered for analysis. Bracketed value is the default.

Hyperparameter	Range of values considered
L	2,3,4 (3)
h_{dim}	32,128,256 (32)
K_{hid}	1,4,6 (1)
K_{out}	1,4,6 (1)
act	ReLU, LeakyReLU, tanh (ReLU)
$aggr$	add, mean, max (add)
$concat$	True, False (True)

4 ANALYSIS

We analyse our approach by using GAT networks for link prediction on the Grid dataset [11] over a wide array of hyperparameters described in Section 3.3. A summary of the parameters used are listed in Table 5. The grid dataset consist of a 2D 20*20 grid graph. Since it is a relatively smaller dataset, it is ideal for running our grid search experiments consisting of 243 runs. We also highlight out the top 5 results and compare it with the default setting and reported numbers of [11] in Table 6. We observe that since the GAT baselines are not tuned very well, the apparent difference between GAT and P-GNN is often exaggerated in literature. We observe that the effect of various parameters can often be difficult to study independently since the parameter values affect each other. However, to get some insights into the importance of each parameter, we average the ROC AUC scores obtained over the entire grid of other hyperparameters for each value of the parameter under consideration and use it to compare the variations. We explain our results in subsequent sections.

4.1 Layers (L)

As observed in Figure 2, we see that increasing the number of GAT network seems to increase the performance slightly. However, as the layer number increases, the variation of results also increases because of increasing number of parameters, because of overfitting on the small grid dataset. To handle this, L_2 regularization or dropout is necessary.

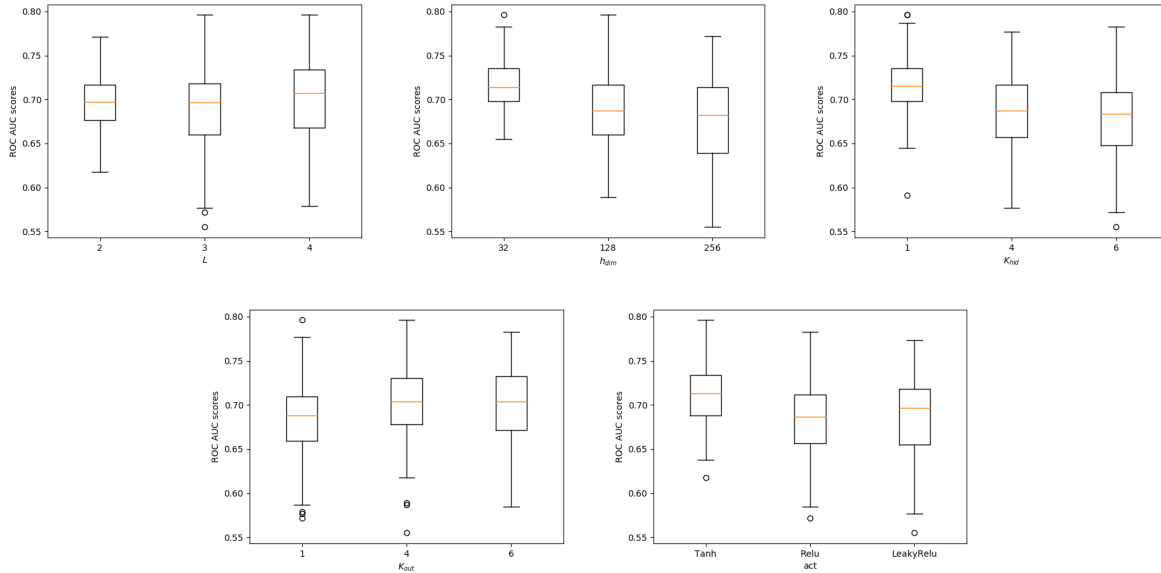


Figure 2: Plots showing the ROC-AUC scores (y-axis) obtained with different values of hyperparameters (x-axis) on Grid.

Table 6: Link Prediction on the Grid Dataset. P-GNN corresponds to the best P-GNN variant reported in [11].

Method	L	h_{dim}	K_{hid}	K_{out}	act	Test ROC AUC
Tuned	3	32	1	1	tanh	0.796 ± 0.037
Tuned	4	128	1	4	tanh	0.796 ± 0.04
Tuned	3	128	1	4	tanh	0.787 ± 0.025
Tuned	3	32	6	6	ReLU	0.783 ± 0.062
Tuned	3	32	6	6	tanh	0.779 ± 0.03
Default	3	32	1	1	ReLU	0.693 ± 0.028
P-GNN	-	-	-	-	-	0.834 ± 0.099

4.2 Hidden Dimension (h_{dim})

For hidden dimension, we observe a more clear trend, with performance seem to be decreasing with increasing number of hidden dimensions. Although, we must realise that this variation can be different for different datasets, and since the grid dataset is a comparatively small dataset, it requires less parameters to model the hidden dimension. Moreover, it gives the intuition that it is generally better practice to increase the number of GAT layers instead of increasing the hidden dimension to improve performance.

4.3 Hidden Layer Attention Heads (K_{hid})

Similar to as seen above, increasing number of attention heads in the hidden layer seem to decrease performance. But, more attention heads give better performance in the PPI dataset. So, we can infer that we should use attention heads proportional to the size of the dataset. If you want to use more attention heads for smaller datasets, instead of concatenation, use averaging for aggregation of outputs across different attention heads.

4.4 Output Layer Attention Heads (K_{out})

The variation of quality with the number of attention heads in the output GAT layer follows a different trend. We see that by increasing number of output attention heads actually help in achieving better performance. This concludes that we should by default start with a GAT network with less attention heads in the hidden layers and more heads in the output layer.

4.5 Activation (act)

We observe that the tanh activation seems to be much better than ReLU (default) and LeakyReLU activations seeing an average of 3% gain on ROC AUC scores over the default values.

5 INTERPRETING ATTENTION WEIGHTS

In this section, we focus on the distribution of attention weights learned by our model in the Multi-class Node Classification task on the PPI dataset. We first pass the test set graphs through the trained model for inference, and store the graphs at each layer. We plot the sampled subgraph in Figure 3 at each layer with edge colors corresponding to the learned attention weights. Darker edges mean that the attention weight for that edge is higher. We can clearly observe that in the inner layers, attention weights are more uniformly distributed as compared to the outer layers. In the outer layers, only a few nodes contribute in the message passing step for the final embedding of a node.

To understand the attention weight distribution more clearly as motivated by [9], we also plot the entropy of attention weight distribution in Figure 4. For a given node i , α_{ij} , $j \in \mathcal{N}_i$ defines the probability distribution over all the neighbors of node i , with entropy given by:

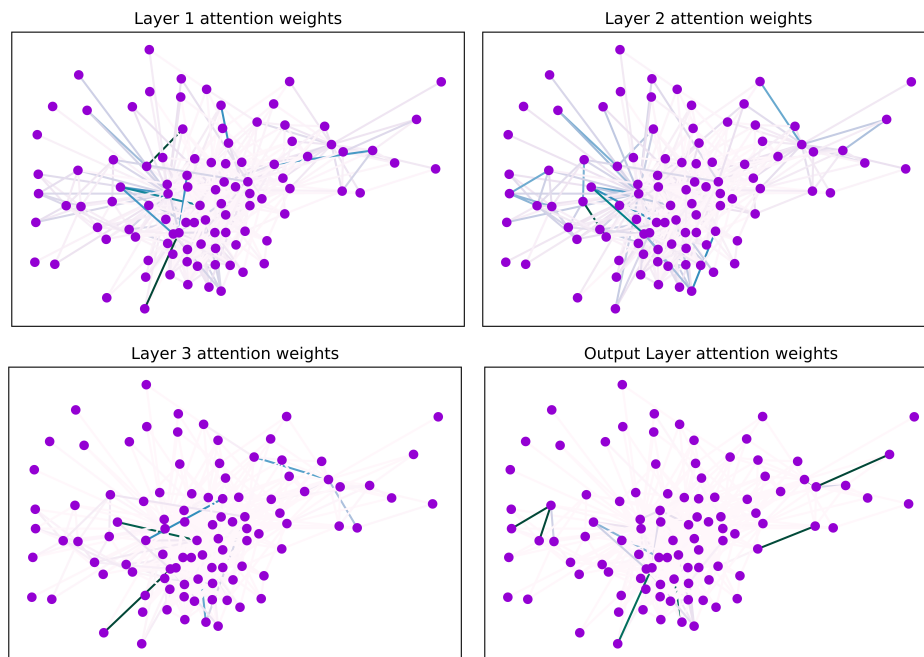


Figure 3: Attention Weight Distribution learned at each layer (for the K^{th} head) on a sampled test subgraph in the PPI Dataset. Darker edges represent higher attention weights.

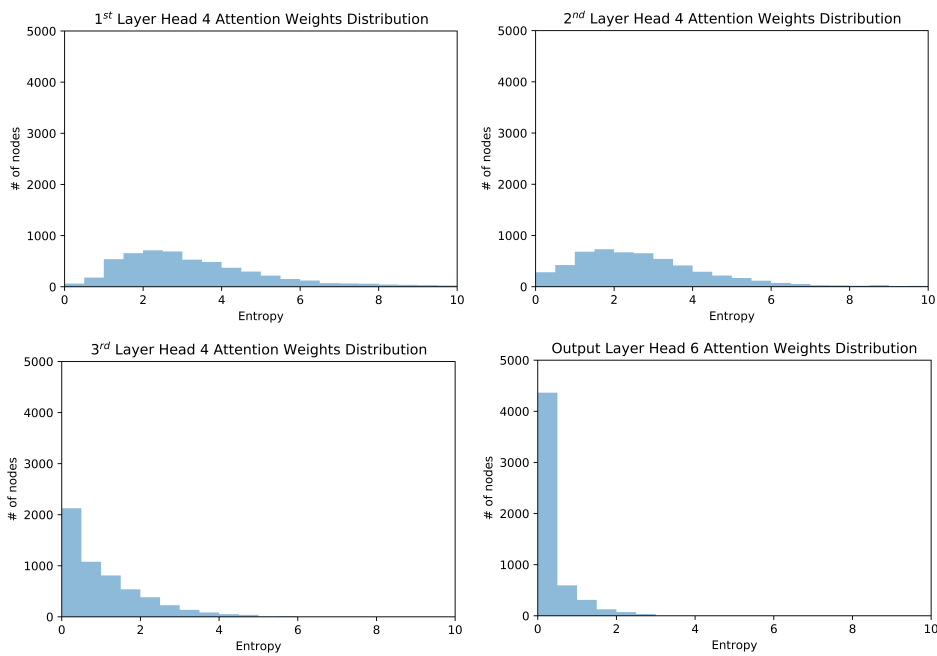


Figure 4: Entropy of Attention Distribution learned at each layer (for the K^{th} head) on a sampled test subgraph in the PPI Dataset.

