Probability Calibration for Graph Representation Learning

Tong Liu

München 2021

Probability Calibration for Graph Representation Learning

Tong Liu

Masterarbeit
an der Fakultät für Physik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Tong Liu
aus China

Erster Supervisor: Yushan Liu
Zweiter Supervisor: Prof. Dr. Armin Scrinzi

München, den 01. August 2021

Declaration

I hereby declare that this thesis is my own work, and that I have not used any sources and aids other than those stated in the thesis.

München,
Date of submission:
Author's signature:

Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to express my biggest gratitude to Yushan Liu! She has been very nice and supportive during the whole period of the thesis. She gave me an opportunity to write a thesis from a physics student and gave me valuable guidelines and feedback. The contract is really important for me to maintain my living in Munich. Thank Dr. Marcel Hildebrandt for many times' insightful and encouraging discussions. Thank Yushan, Marcel, Dr. Mitchell Joblin, and Serghei Mogoreanu for patiently introducing the use case and answer my questions, and make me think about research from a practical perspective. Chats with Hang over daily lunch since last month also gave me lots of inspiration and support. I could not thank all of you in our group enough due to a lack of words. Thanks for Prof. Dr. Armin Scrinzi, who agreed to be my internal supervisor. This is important. He also suggested me to do universal research and remember a physics perspective. Thank Prof. Dr. Volker Tresp for providing the research environment in LMU and Siemens. Thank my family to give me 100% freedom to have my own way abroad.

Chapter 1

Introduction

1.1 Physics and graphs

What is a graph? Graphs are a ubiquitous data structure and a universal language to describe complex systems. In most cases, a graph consists of objects (i.e., nodes), along with interactions (i.e., edges) between pairs of these objects, as shown in Fig. 1.1. Rather than only considering properties of individual points, graph formalism focuses more on their relationships, which leads to its generality. The same graph formalism can be used in various areas.

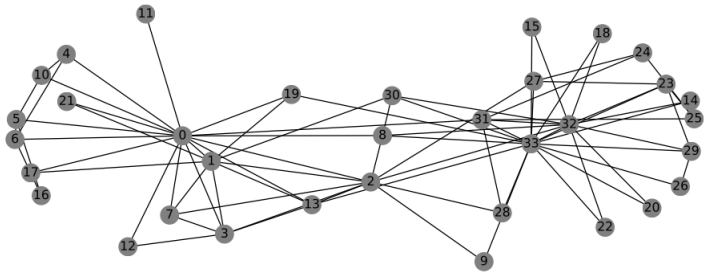


Figure 1.1: Zachary Karate Club Network represents the friendship relationships between members of a karate club. [1]

A heterogeneous graph is a graph in which could have multiple types of links. A knowledge graph is a special kind of heterogeneous graph that the links are directed.
Graphs are used in various physical applications, such as predicting the properties of molecules, drug repurposing, and processing data in high-

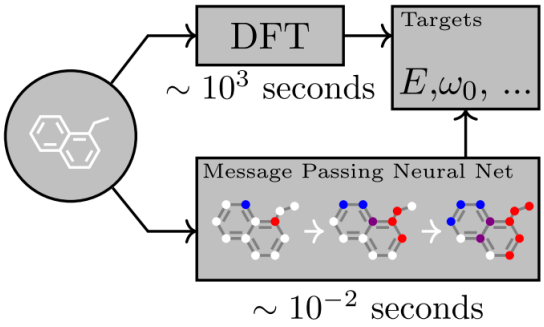


Figure 1.2: A Message Passing Neural Network predicts properties of a molecule, while DFT calculation is computationally expensive.[2]

energy physics experiments. In the following, three examples are described in more detail.

Properties of molecules Predicting the properties of molecules is a fundamental task in physics. Density functional theory (DFT) [9] is a commonly used computational method for molecular property prediction, which is directly derived or approximated by the Schrödinger equation. But it is time-consuming because it solves some big linear equations and the complexity of DFT is $O(N^3)$, where N is the number of electrons in the system.
In comparison, graph neural networks (GNN) have been greatly used to predict the quantum mechanical properties of molecules. Researchers treat the molecule as a graph, where the nodes denote atoms and edges denote the interactions between atoms. Neural layers project each node into latent space and pass its interaction message iteratively. Then the node messages can be aggregated to represent the molecule for property prediction, as shown in Fig. 1.2.
Several models are proposed in this question, including MPNN [2], SchNet [10] and DimeNet [11]. For example, in DimeNet, they proposed to utilize angles of different atom pairs and interatomic distance for the message embedding. In the experiment, they applied DimeNet to the dataset QM9 and gained a good performance. QM9 consists of 130 000 molecules in equilibrium with up to 9 heavy C, O, N, and F atoms. The task is to predict computed geometric, energetic, electronic, and thermodynamic properties for 134k stable small organic molecules.

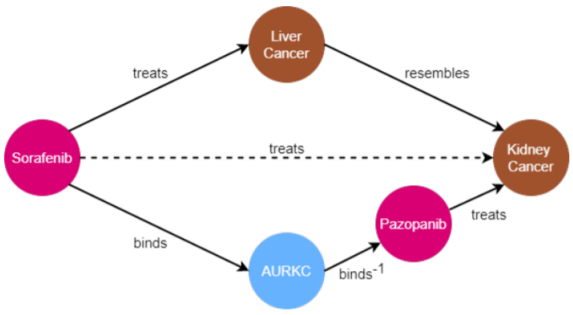


Figure 1.3: Illustration of the drug repurposing use case. [3]

Drug repurposing Biophysical researchers studying computational approaches rely on structured data, thus knowledge graphs are becoming a key instrument in biomedical knowledge discovery and modeling. Knowledge graphs are directed graph models that have different types of edges. Liu et al. [3] used knowledge graphs to do drug repurposing by formulating this task as a link prediction problem. It used head and tail to represent e.g., different compounds and diseases, used an edge to represent either treating between one compound and one disease, or resembling between two compounds or two diseases. For example in Fig. 1.3, the two paths that connect the chemical compound sorafenib and the disease kidney cancer can be used to predict a direct edge between the two entities.

Particle physics The application of graph networks into high-energy physics (HEP) experiments is evolving rapidly. Physicists have made an advance to embed graph neural networks into experimental particle physics to process detector data directly [12].
At many levels, the data of HEP are sets (unordered collection) of items. Physicists consider the geometrical or physical relation between items (i.e., different detectors of different sizes using different technology to measure the trace of particles) as transformation into a graph with an adjacency matrix, to apply deep learning to the intrinsic representation of data [4].
An example of HEP data shown as graphs is illustrated in Fig. 1.4. Applications in HEP include: (graph classification) jet classification, event classification; (node classification) pileup mitigation, calorimeter reconstruction; (edge classification) charged particle tracking, secondary vertex reconstruc-

tion, and so on.

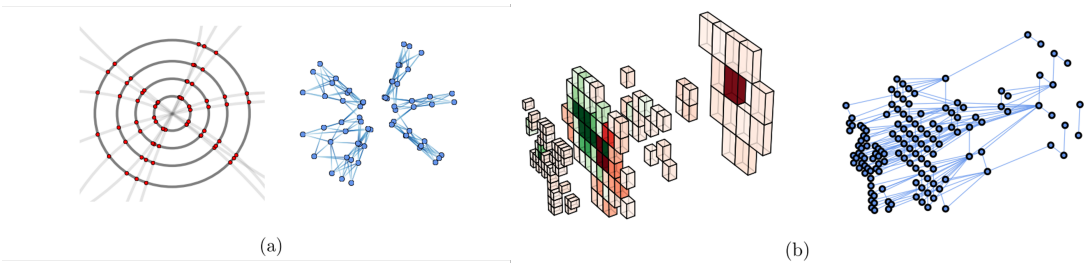


Figure 1.4: HEP data represented as a graph for many applications: (a) clustering tracking detector hits into tracks, (b) segmenting calorimeter cells. [4]

1.2 Motivation

In real-world decision-making systems, classification networks should not only be accurate but also should indicate when they are likely to be incorrect. As an example, consider a self-driving car that uses a deep learning model to detect obstructions, where the probability of being an obstruction is important. In other words, the probability associated with the predicted class label should reflect its ground truth correctness likelihood. We will describe it in detail in Chapter 2.2.1.

It has been shown that plain neural networks are mostly overconfident [13] and that knowledge graph embedding (KGE) models, as well as graph neural networks (GNNs), are both poorly calibrated. Existing calibration techniques seem to be able to improve the calibration of KGE models for link prediction tasks [14] but seem to be rather ineffective when used on graph neural networks for node classification [15]. In our thesis, we mainly focus on node classification in graph neural networks. The definition and details about node classification tasks can be found in section 2.1.1.

The goal of this thesis aims at gaining a better understanding of what affects the calibration of KGE models or GNNs by investigating existing calibration techniques and their relationship to specific properties of graph representation learning models. With a better understanding, further methods could be developed to improve the calibration for graph representation learning.

1.3 Outline

Chapter 2 gives an introduction to graph representation learning and probability calibration, including related state-of-the-art methods. Chapter 3 is an analysis work, doing an empirical analysis about calibration on benchmark datasets, model architectures, and dataset characteristics. Chapter 4 is a try to use knowledge distillation to help calibrate graph networks. Chapter 5 is from the perspective of graph signal processing. Chapter 6 is a short chapter that describes our calibration loss method. Chapter 7 is about our calibration on a real-world industrial use case. Chapter 8 is conclusion.

Chapter 2

Fundamentals

In this chapter, we give the basic knowledge about this thesis. It mainly covers two parts: graph representation learning and existing calibration work.

2.1 Graph representation learning

2.1.1 Machine learning on graphs

Machine learning tasks on graphs. Graphs are a ubiquitous data structure and a universal language to describe complex systems, like in Fig. 1.1. More formally, a graph is represented as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edges between the nodes. We denote an edge going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ as $(u, v) \in \mathcal{V}$. $A \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ is an adjacency matrix, where $A[u, v] = 1$ if $(u, v) \in \mathcal{V}$ and $A[u, v] = 0$ otherwise. The edge between node u and v is undirected if $(u, v) \in \mathcal{V} \leftrightarrow (v, u) \in \mathcal{V}$. If a graph contains only undirected edges (so called undirected graph), then A is symmetric. If a graph is directed graph (which means that the direction also matters), A is not necessarily symmetric. For attributed graphs, we denote with $X \in \mathbb{R}^{|\mathcal{V}| \times D}$ the matrix of D-dimensional features associated with each node.
A knowledge graph is a directed multi-relational graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, where $\mathcal{R}$ is the set of relation, with edges denoted as tuples $e = (u, \tau, v)$ indicating there is a link of type τ between node u and v.
The usual categories of supervised and unsupervised tasks are not the most

useful category when it comes to graphs. Usually, there are several following tasks on graphs. **Node classification** is to predict the label y_u for node u, when we are only given the true labels on the training set of the nodes $\mathcal{V}_{train} \subseteq \mathcal{V}$. **Link prediction** is to predict the relationship of edges $\mathcal{E} \setminus \mathcal{E}_{train}$, when we are given node set $\mathcal{V}$ and an incomplete set of edges $\mathcal{E}_{train} \subseteq \mathcal{E}$. **Community detection** is analogues of unsupervised clustering, where the task is to cluster the graph into different communities where nodes are more likely to form edges with nodes in the same community. **Graph classification and clustering** view each graph as an i.i.d. datapoint with a label, and the goal is to learn a mapping from datapoints to labels.

2.1.2 Node embeddings

The key process in machine learning on graphs is to find a way to incorporate information of graphs into a machine learning model. But the challenge is that there is no straightforward way to use the graph's information. Graph representation learning is proposed to solve this problem by encoding the graph's structural information. It aims to learn a mapping of embedded nodes as points in a low-dimensional vector space, so that geometric relationships in the embedding space reflect the structure of the original graph, as shown in Fig. 2.1. After finishing optimizing, the learned embedding functions can be utilized for downstream machine learning tasks.

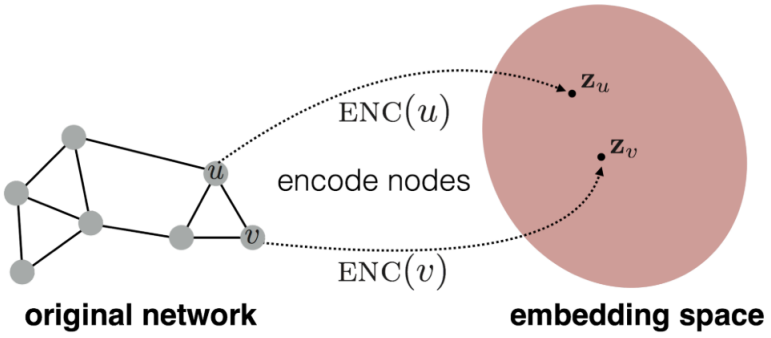


Figure 2.1: Illustration of node embedding problem. [1]

There are many ways to define the encoder and decoder. Some methods defines the encoder function as an embedding lookup based on the node

ID, or more generally, $ENC(v) = Z[v]$, where $Z \in \mathbb{R}^{|\mathcal{V}\times d|}$. The feature information of $v-th$ node encodes into $v-th$ row of Z. We could call this approach a shallow embedding.

2.1.3 Graph neural networks

Rather than a shallow embedding approach like node embedding, graph neural network (GNN) formalism is a more complicated encorder model. It generates representations of nodes that depend on both the graph structure and the feature information. Graph structure is different from the usual deep learning context, like CNNs for grid structure and RNNs for sequence structures. We need to define the learning framework, which includes neural message passing, neighborhood aggregation and updating.

2.1.3.1 Neural message passing

Framework of message passing. The framework of message passing can be represented as:

$$\begin{aligned} h_u^{(k+1)} &= UPDATE^{(k)}(h_u^{(k)}, AGGREGATE^{(k)}(h_v^{(k)}, \forall v \in \mathcal{N}(u))) \\ &= UPDATE^{(k)}(h_u^{(k)}, m_{\mathcal{N}(u)}^{(k)}), \end{aligned} \tag{2.1}$$

where UPDATE and AGGREGATE are two arbitrary differentiable functions, $h_u^{(k)}$ is a hidden embedding of node u for $k-th$ layer, which is updated according to the information aggregated from $u's$ neighborhood $\mathcal{N}(u)$, $m_{\mathcal{N}(u)}^{(k)}$ is that message aggregated and prepared to be updated to $(k+1)-th$ layer hidden embedding $h_u^{(k+1)}$. Usually the initial embedding $h_u^{(0)}$ is directly set to input features x_u, i.e., $h_u^{(0)} = x_u$. The output of final layer after K iterations defines the embedding for each node, i.e., $z_u = h_u^{(K)}$, and can be model output.

The basic GNN. The concrete formalism to UPDATE and AGGREGATE functions, or to say, the basic GNN message passing is defined as:

$$h_u^{(k+1)} = \sigma(W_{self}^{(k+1)} h_u^{(k)} + W_{neighbor}^{(k+1)} \sum_{v\in\mathcal{N}(u)} h_v^{(k)} + b^{(k+1)}), \tag{2.2}$$

where $W_{self}^{(k+1)}$ and $W_{neighbor}^{(k)}$ are parameters, σ is non-linearity and $b^{(k)}$ is a bias term. Here we use a SUM function as AGGREGATE, i.e. $m_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} h_v$, and a perceptron and a non-linearity as UPDATE, i.e. $h_u^{'} = \sigma(W_{self} h_u + W_{neighbor} \mathcal{N}(u))$.

Message passing with self-loops. As a simplification of neural message passing approach, it is common to write together the aggregation from neighbors and the node itself, or to say, message passing with self-loops. More precisely,

$$h^{(k+1)} = AGGREGATE(h^{(k)}, \forall v \in \mathcal{N}(u) \cup u), \tag{2.3}$$

or in the basic GNN form corresponding to Equation 2.2 and in the matrix form for all nodes,

$$\boldsymbol{H}^{(k+1)} = \sigma((\boldsymbol{A} + \boldsymbol{I})\boldsymbol{H}^{(k)}\boldsymbol{W}^{(k+1)}). \tag{2.4}$$

Actually this is derived from first-order expansion of N-polynomials of adjacency matrix due to translation equivariance (see equation 5.7 and 5.19).

2.1.3.2 Neighborhood aggregation

Neighborhood normalization. Aggregation plays the same important role as the updating. Equation 2.2 defines the most basic neighborhood aggregation operation, which takes the sum of all neighbor embeddings. But one issue is that it could be unstable and highly sensitive to node degrees. We cannot tolerate one node's embedding could be many times larger than the values in one another. Therefore we need to normalize aggregation. The simplest approach is to take an averaged:

$$m_{N(u)} = \frac{\sum_{v \in \mathcal{N}(u)} h_v}{|\mathcal{N}(u)|}. \tag{2.5}$$

Or use symmetric normalization like in Graph Convolutional Network (GCN) [16]:

$$m_{N(u)} = \sum_{v \in \mathcal{N}(u)} \frac{h_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}. \tag{2.6}$$

Actually, with the message passing with self-loops and symmetric normaliztion, we can write the message passing function, which is the same as GCN's:

$$h_u^{(k)} = \sigma(W^{(k)} \sum_{v \in \mathcal{N}(u) \cup u} \frac{h_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}). \tag{2.7}$$

GCN can be equivalently derived from a graph signal processing perspective.

Neighborhood attention. One of the most natural ideas to extend GCN is to consider weight on each edge, or to say, to apply attention. More important neighbors take higher attention weights. The first paper to introduce this idea is Graph Attention Network (GAT) [17], which uses attention weights to define a weighted sum of neighbors:

$$m_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} h_v, \tag{2.8}$$

where $\alpha_{u,v}$ defines the attention from v to u. In this way, we aggregate information from neighbors of node u to node u, with attention. The attention $\alpha_{u,v}$ is computed as:

$$\alpha_{u,v} = \frac{exp(a^T[Wh_u \bigoplus Wh_v])}{\sum_{m \in \mathcal{N}(u)} exp(a^T[Wh_u \bigoplus Wh_m])}, \tag{2.9}$$

where a is a trainable vector, W is the matrix in message passing, and $\bigoplus$ is the concatenation operation.
There are also other attention-style work, e.g., replacing the attention part in equation 2.9 with, $h_u^T W h_v$, or $MLP(h_u, h_v)$ [18].
Graph attention models have been shown to have close connections with transformer [19]. Graph attention models can be seen to first start with a transformer model on each vertex and then apply a mask to ensure information is only aggregated between nodes that are actually connected in the graph, and as a downside, increasing complexity from $O(|\mathcal{V}|^2)$ to $O(|\mathcal{V}||\mathcal{E}|)$, since generally the number of edges $|\mathcal{E}|$ is larger than the number of nodes $|\mathcal{V}|$ in a graph.

2.1.3.3 Updating in GNNs

Over-smoothing. The most common-used UPDATE operator is as stated in Eq. 2.2. However, one common issue with GNNs is over-smoothing. It means after several iterations of message passing, the representations for all nodes start to become similar and indistinguishable. This makes it hard to build deeper layer GNN models. Section 5.1.4 gives a more theoretical perspective to understand over smoothing from signal processing.
More precisely, the degree of over-smoothing can be formalized by defining the influence of node $u's$ input features on the final layer embedding of the other nodes as [20]:

$$I_K(u,v) = \mathbf{1}^T(\frac{\partial h_v^{(K)}}{\partial h_u^{(0)}})\mathbf{1}, \tag{2.10}$$

where $\mathbf{1}$ is a vector of all ones. This Jacobian matrix measures how much the initial embedding of node u influences the final embedding of v.
Theorem 2.1. [20] *For any GNN model using a self-loop update approach and an aggregation function of the form*

$$AGGREGATE(h_v, \forall v \in \mathcal{N}(u) \cup u) = \frac{1}{f(|\mathcal{N}(u) \cup u|)} \sum_{v \in \mathcal{N}(u) \cup u} h_v, \tag{2.11}$$

where f is a differentiable normalization function, we have that

$$I_K(u,v) \propto p_K(u|v), \tag{2.12}$$

where $p_K(u|v)$ denotes the probability of visiting node v on a length-K random walk starting from node u.
This theorem can also be extended to the basic GNN update without self-loop, like equation 2.2, as long as $||W_{self}^{(k)}|| < ||W_{neigh}^{(k)}||$ at each k. It states that for a K-layer GNN model, the influence of node u and node v is proportional to the probability of reaching node v on a K-step random walk starting from node u. Therefore, when K is large, the random walk range is large, the probability to reach one particular node is quite small, then the influence is small. In the limit situation $K \to \infty$, the probability to reach any node approaches to a constant, therefore the influence between any two nodes approaches the same, or to say, the local neighborhood information is lost.

Chen et al. (2019) [21] proposed a quantitative metric Mean Average Distance (MAD) to measure over-smoothing. Intuitively, it is a metric to compare the distance between nodes. More formally, given the graph hidden embedding matrix H, we can first compute the cosine distance $D_{i,j}$ between node i and j, then apply a mask matrix for nodes that exist in the real graph:

$$\begin{aligned} D_{i,j} &= 1 - \frac{H_{i,:} \cdot H_{j,:}}{|H_{i,:}| \cdot |H_{j,:}|} \\ D^{tgt} &= D \otimes M^{tgt} \end{aligned} \tag{2.13}$$

where $\otimes$ is an element-wise product, and $|H_{i,:}|$ denotes the Euclidean norm of i-th node embedding. Now the D^{tgt} is a matrix, we need to change it to a scalar, especially for the non-zero elements. Suppose there are n nodes:

$$\begin{aligned} \bar{D}_i^{tgt} &= \frac{\sum_{j=0}^{n} D_{i,j}^{tgt}}{\sum_{j=0}^{n} \mathbb{1}(D_{i,j}^{tgt})} \\ MAD^{tgt} &= \frac{\sum_{i=0}^{n} \bar{D}_i^{tgt}}{\sum_{i=0}^{n} \mathbb{1}(\bar{D}_i^{tgt})}, \end{aligned} \tag{2.14}$$

where $\mathbb{1}(x) = 1$ if $x > 0$, otherwise 0.
There are several updating methods to alleviate over-smoothing, such as concatenation [22], gated updates [23], and jumping knowledge connections [20]. There are also quite a lot of papers discussing how to overcome the over-smoothing problem from many different perspectives.

2.1.3.4 Methods

In this section, we will give a basic introduction to GNNs we use in our thesis. Several methods are described before and later.
Graph Convolutional Networks (GCN)[16]. The vanilla GCN is one of the most basic GNN method. Its spatial formalism is introducted in section 2.1.3. It is also a spectral method, which can be equivalently derived from the first-order approximation of $N-th$ polynomial function of Laplacian L of the graph, as introduced in section 5.1.5.
Graph Attention Network (GAT)[17]. Rather than only aggregating from equivalent edges from neighbors, GAT adopts self-attention, to give

different weights to neighbors' edges, as shown in section 2.1.3.

Simple Graph Convolution (SGC)[24]. SGC removes the non-linearity between each layer in GCN, resulting in not much performance decrease but huge running speed increase. SGC is introduced in section 5.1.5.

Graph filter neural network (gfNN)[25]. gfNN can be viewed as a simple perceptron after SGC, in order to improve the expression ability. It is introduced in section 5.1.5.

Approximate personalized propagation of neural predictions (APPNP) [26]. First, Klicpera et al. introduced Personalized Propagation of Neural Predictions (PPNP), an aggregation method based on Personalized PageRank (PPR), which is defined as $\pi_{PPR} = \alpha(I - (1-\alpha)\hat{\tilde{A}})^{-1}$, where $\alpha \in (0,1)$ is a hyperparameter, I being identity matrix, and $\hat{\tilde{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ being symmetric normalized adjacency matrix with self-loop as defined in Chapter 5. The influence of node i on node j is proportional to the element (x, y) in the PPR matrix π_{PPR}. Then the PPNP method can be written as:

$$\begin{aligned} Z^{(0)} &= MLP(X) \\ Z_{PPNP} &= softmax(\pi_{PPR} Z^{(0)}), \end{aligned} \tag{2.15}$$

where X is the node features. PPNP consists of a MLP for feature transformation $Z^{(0)}$ and PPR layer for feature aggregation Z_{PPNP}.

Suppose $\pi_{PPR} \in \mathbf{R}^{n \times n}$, the computational complexity and memory of π_{PPR} is $O(n^2)$, which is not effective in training and inference. Instead, Klicpera et al. introduced APPNP, an approximation of PPNP, which changes aggreagtion of Eq. 2.15 to an iterative way as:

$$\begin{aligned} Z^{(k+1)} &= (1-\alpha)\hat{\tilde{A}} Z^{(k)} + \alpha Z^{(0)} \\ Z_{APPNP} &= softmax(Z^{(K)}). \end{aligned} \tag{2.16}$$

Spectral Graph Convolution (Cheb) [27] is an early spectral method which uses Chebyshev polynomials to fit the function of eigenvalues of Laplacian of the graph. The details are introduced in section 5.1.5.

2.1.3.5 Tasks focused in the thesis In this thesis, node classification is the main task we focus on. Node classification is perhaps the most popular machine learning task on graphs. Definition is introduced in section

2.1.1. Examples of node classification include classification of the proteins [22] (where nodes represent proteins and edges represent interaction between proteins) , classification of the topic of a paper based on the citation graph [16] [28] (where nodes represent papers and edges represent citation between papers), classification of the types of goods co-purchased in the co-purchased network, in Amazon-computers and Amazon-photos [28] (where nodes represent goods and edges represent items are frequently co-purchased), classification of the users' information, e.g. religions, sex and home country, in Facebook's social networks [29] (where nodes represent users and edges represent their friendship or relationship), and so on.

2.2 Probability calibration

2.2.1 Model calibration

Only accuracy is not enough for many real-world decision-making systems. We need to know which predictions we should trust and which not. For models with softmax scores, we need to know if these scores can represent the real probabilities correctly. This leads to the concept of calibration. In a perfect calibrated model, *the softmax output should match the relative frequency that the prediction is correct.* More formally, for a given model f_θ (θ represents its parameters) with its input X and ground-truth label $Y \in \{1, ..., K\}$. For a given sample i, the predicted label $\hat{y}_i$:

$$\hat{y}_i = \hat{y}(X_i) = \underset{k \in \{1,...,K\}}{\arg\max} [f_\theta(X_i)]_k, \tag{2.17}$$

and its predicted probability:

$$\hat{p}_i = \hat{p}(X_i) = \max_{k \in \{1,...,K\}} [f_\theta(X_i)]_k. \tag{2.18}$$

The perfect calibration is defined as:

$$\mathbb{P}(Y = \hat{y}(X) | \hat{p}(X) = p) = p, \quad \forall p \in [0, 1] \tag{2.19}$$

2.2.2 Evaluation of calibration

2.2.2.1 Reliability diagrams

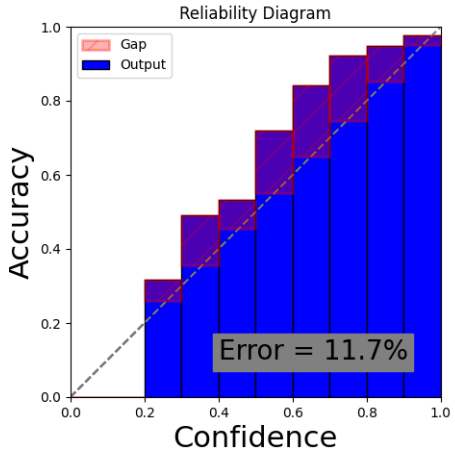


Figure 2.2: Reliability diagrams for GCN on Pubmed. Error here denotes ECE in Eq. 2.22.

Reliability diagrams [30, 31] is a visual representation of model calibration. It plots the accuracy as a function of confidence (the confidence is defined as the biggest class probability), as shown in Fig. 2.2. To estimate the expected accuracy, the predictions of the samples are grouped into M interval bins, according to their confidence value. For each bin B_m $(1 \leq m \leq M)$, the accuracy is:

$$acc(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{1}[Y_i = \hat{y}_i], \tag{2.20}$$

while the average confidence is:

$$conf(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i, \tag{2.21}$$

where $|B_m|$ is the number of samples in that bin, $\mathbb{1}$ is the indicator function, Y_i, y_i and $\hat{p}_i$ are the true label, predicted label and confidence for sample i in the bin. The perfect calibration is defined as $acc(B_m) = conf(B_m)$ for each $m \in \{1, ..., M\}$.

2.2.2.2 Calibration error

Expected Calibration Error (ECE) [32, 13] is a scalar which is the average of the gaps in the reliability diagram, weighted by the samples

number in each bin:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)|, \tag{2.22}$$

where M is the number of bins and n is the total number of samples.
$ECE_{\geq 50}$[15] is an ECE only computed for samples with confidence of higher than 50%. Since in some cases that models make mostly random predictions, ECE can also be small. In the cases of not many classes, decision-makers care more about the calibration of predictions with higher than 50% confidence, since otherwise, it is relatively unreliable.

Maximum Expected Calibration Error (Maximum ECE) [32], in comparison, summaries the worst-case deviation between confidence and accuracy:

$$MCE = \max_{k \in \{1,\ldots,M\}} |acc(B_m) - conf(B_m)|. \tag{2.23}$$

Kolmogorov-Smirnov calibration error (KS-error) [33]. Although ECE is a widely used metric, ECE relies on the number of the chosen histogram binning scheme. KS-error is introduced to overcome this shortcoming as KS uses a binning-free approach by the statistics of cumulative distributions.
KS-error computes the maximum difference between cumulative distributions. For $t \in [0, 1]$,

$$\begin{aligned} KS &= \max_t |acc(t) - conf(t)| \\ acc(t) = \frac{1}{n} \sum_{i:\ \hat{p}_i \leq t} \mathbb{1}[Y_i = \hat{y}_i], &\quad conf(t) = \frac{1}{n} \sum_{\hat{p}_i \leq t} \hat{p}_i, \end{aligned} \tag{2.24}$$

where acc and $conf$ are accuracy and confidence defined in KS-error, n being the number of samples.

Top-label calibration error (TCE) [34]. ECE can be seen as an approximation of TCE. TCE examines the difference between the model's probability for its top prediction and the true probability of that prediction

given the model's output:

$$TCE(f) = (\mathbb{E}[(\mathbb{P}(Y = \arg\max_{j\in\{1,\dots,M\}} f(X)_j | \max_{j\in\{1,\dots,M\}} f(X)_j) - \max_{j\in\{1,\dots,M\}} f(X)_j)^2])^{1/2} \tag{2.25}$$

Marginal expected calibration error (Marginal ECE)[34] compares averaged probability of each class in each bin with the averaged accuracy of samples in that class in that bin:

$$Marginal\text{-}ECE(f) = (\sum_{k=1}^{K} w_k \mathbb{E}[(\mathbb{P}(Y = k|f(X)_k) - f(X)_k)^2])^{1/2}, \tag{2.26}$$

where w_k is the weighted number for $k-th$ class, denoting how important class k is. If all classes are equally important, $w_k = 1/k$.

Classwise-ECE [35] is a metric very similar to Marginal-CE. It compares averaged probability of each class in each bin with *the proportion* of samples in that class in that bin:

$$Classwise\text{-}ECE = \sum_{k=1}^{K} w_k \sum_{i=1}^{M} \frac{B_{i,k}}{N} |y_k(B_{i,k}) - \hat{p_k}(B_{i,k})|, \tag{2.27}$$

where w_k is the weighted number for class k, K, M and N are the number of classes, bins and samples respectively, $|B_{i,j}|$ is the size of the bin, $\hat{p_k}(B_{i,k})$ and $y_k(B_{i,k})$ are the averaged prediction of class k in the bin and the actual proportion of class k in the bin.
Although classwise ECE is intuitively defined to make sense, it is also considered not to represent the classwise calibration very well and even collapse in some cases, e.g. in a balanced-class dataset, if we use a very high temperature to scale down all scores to about $1/k$, then almost all scores would fall into the same bin that contains $1/k$, which is equivalent to only one bin exists. In that bin, the balanced-class dataset has the same proportion of samples for each class, therefore $Classwise\text{-}ECE = 0$.

Debiased-ECE [34] is recently proposed to show that the original ECE has an inherent positive bias and can be subtracted as an approximated

correction term to reduce the biased estimate of ECE. It was shown to be a more accurate estimator of TCE than ECE, especially when TCE is small. It is defined using Gaussian bootstrapping:

$$Debiased\text{-}ECE = ECE - (\mathbb{E}_{R_{1:M}}[\sum_{m=1}^{M} \frac{|B_m|}{N} |conf(B_m) - R_m|] - ECE), \tag{2.28}$$

where $R_m \sim N(acc(B_m), \frac{acc(B_m)(1-acc(B_m))}{|B_m|})$.
Some metrics used in loss function can also be seen as calibration metrics.

Negative Log-Likelihood (NLL) [36], also known as the cross-entropy loss in multi-class classification setting of deep learning. Denote $p(\hat{y}_i|x_i, \theta)$ the predicted probability vector associated with the one-hot encoded ground truth label y_i for sample x_i, NLL is defined as:

$$L_{NLL} = -\sum_{i=1}^{N} y_i \log p(\hat{y}_i|x_i, \theta),$$

where N is the number of samples. NLL is minimal if and only if the prediction probability $p(\hat{y}_i|x_i, \theta)$ recovers true label y_i, therefore, it may cause over-confidence of the probability.

Brier scores [37], is defined as the squared error of the predicted probability $p(\hat{y}_i|x_i, \theta)$ and the ground truth label y_i. For classification problems, Brier score can be decomposed into a loss for accuracy and a loss for calibration. Brier scores is defined as:

$$Brier = -\frac{1}{K} \sum_{i=1}^{N} \sum_{k=1}^{K} (p(\hat{y}_i^k|x_i, \theta) - y_i^k)^2, \tag{2.29}$$

where N and K are the number of samples and the number of classes.

2.2.3 Theorems related to calibration

Rahimi et al. [38] gave a theoretical description about calibration. It is worthy of stating exactly.

Preliminaries. Let (X, K) be a pair of joint random variables, where X takes values in domain D_X and K takes values in the class set $\kappa = \{1, ..., n\}$. n is the number of labels. Let a function $f : D_X \to D_Z = \mathbb{R}^m$ denote our NN, where $Z = f(X)$ denotes the output. Let q denote the terminate function, or activation after the model f usually in the task of classification, e.g. q being the softmax function. $q : \mathbb{R}^m \to \mathbb{R}^n$, where $q(z) = (q_1(z), ...q_n(z))$. Here since $q_k(z)$ satisfies some condition, e.g. $\sum_{k=1}^{n} q_k(z) = 1$, we denote the set of $q_k(z)$ as the probability simplex Δ^{n-1}. Let $q \circ f$ denote the NN with activation, $q \circ f : D_X \to \Delta^{n-1}$.
Given a pair data $(x, k) \in D_X \times \kappa$, our expected negative log likelihood loss given the random variable (X, K) is: $L(q \circ f, X, K) = E_{(x,k)\ (X,K)} L(q \circ f, x, k) = -E_{(x,k)\ (X,K)} log(q_k(f(x)))$. But usually we cannot know the real distribution of random variables (X, K), therefore our expected negative log loss is approximated by empirical loss:

$$L(q \circ f, D) \approx -E_{(x,k)\ D} log(q_k(f(x))) = -\sum_{i=1}^{N} log(q_{k_i}(f(x_i))), \tag{2.30}$$

where we denote samples of data pairs sampled from the distribution of (X, K) as $D = \{(x_i, k_i)_{i=1}^N\}$.

Calibration. Here we state a theorem of calibration. Theorem 2.2 is a combined theorem of several theorems in [38]. It gives different condition to be calibrated for a model, a strong version, a weak version, and a third equivalent version. The proof of Theorem 2.2 can be seen in Appendix B.
Theorem 2.2. [38] *Consider joint random variables (X, Z, K), taking values from $D_X, \mathbb{R}^m$ and κ respectively. Let $f : D_X \to \Delta^{n-1}$ be our NN, $L_1(f, X, K) = -E_{(x,k)\sim(X,K)} log(f_k(x))$ be the first loss given our model f and input X, $L_2(q, Z, K) = -E_{(z,k)\sim(Z,K)} log(q_k(z)) = -E_{(x,k)\sim(X,K)} log(q_k(f(x)))$ be the second loss given our output Z, or equivalently given f and X.*
A strong version: if

$$f = \underset{f':D_X\to\Delta^{n-1}}{argmin}\ L_1(f', X, K), \tag{2.31}$$

then it is perfect calibrated with respect to f, i.e. $P(k|x) = f_k(x)$.

A weak version: if

$$\underset{g:\mathbb{R}^m\to\mathbb{R}^m}{argmin}\, L_2(q\circ g, Z, K) = id, \tag{2.32}$$

where id is the identify function, $g : \mathbb{R}^m \to \mathbb{R}^m$ is a modification, e.g. a post-processing calibration function, then it is perfect calibrated with respect to g, i.e. $P(k|z) = q_k(z)$.

An equivalently weak version: if

$$\underset{g:\mathbb{R}^m\to\mathbb{R}^m}{argmin}\, L_2(q\circ g\circ f, X, K) = id, \tag{2.33}$$

then it is perfect calibrated with respect to g, i.e. $P(k|z) = q_k(z)$, or to say, f is perfect calibrated.

The strong version of this theorem requires $f : D_X \to \Delta^{n-1}$ to be exactly minimized w.q.t. our final probability output $q(z) \in \Delta^{n-1}$, but it is usually impossible, even using any Bayesian method. Usually f is optimized w.q.t. logits $z \in \mathbb{R}^m$, therefore a weak version of this theorem appears. It does not require f to be optimal. It states that f is optimal to calibration if replacing f by $g \circ f$, i.e. using a post-processing calibration method g, can not decrease the L_2 loss.

2.3 Related work for calibration

2.3.1 Post-processing calibration methods

Post-processing calibration methods produce calibrated logits (or probabilities) with input model logits (or probabilities). Usually, post-processing methods get trained in a validation set and tested in a test set. Several early post-processing approaches were not originally formulated for deep learning, such as Platt Scaling[39], Histogram Binning[40], Isotonic Regression[41], and Bayesian Binning into Quantiles[32]. After the calibration problem got attention by deep learning researchers since [13], post-processing methods are rapidly developed in the context of deep learning.

2.3.1.1 Binary classification

In the binary setting, i.e., $Y = \{0, 1\}$, for a sample x_i, z_i is the model's non-probabilistic output, or logit, $\hat{p}_i = \sigma(z_i)$ be the model's predicted probability

of $y_i = 1$, where σ is the sigmoid function. The goal is to produce a calibrated probability $\hat{q}_i$ based on $y_i, \hat{p}_i$ and z_i.

Histogram binning [40] first divides predicted probability $\hat{p}_i$ into M bins, then learns a score in each bin to represent calibrated probabilities. More precisely, with bins $B_1, ..., B_M$, and bin boundaries $0 = a_1 \leq ... \leq a_{M+1} = 1$, each bin B_m belongs to the interval $(a_m, a_{m+1}]$. The calibrated score θ_m in B_m is learnt by:

$$\underset{\theta_1,...\theta_M}{min} \sum_{m=1}^{M} \sum_{i=1}^{n} \mathbb{1}(a_m \leq \hat{p}_i < a_{m+1})(\theta_m - y_i)^2, \tag{2.34}$$

This means that for predicted probabilities belong to $(a_m, a_{m+1}]$, histogram binning learns an averaged accuracy score θ_m, by a sum squared error to each sample's predicted labels in total.

Isotonic regression [41] learns a piece-wise constant function f to transform predicted probability, $\hat{q}_i = f(\hat{p}_i)$, where f is chosen by minimizing the sum squared loss $\sum_{i=1}^{n}(f(\hat{p}_i) - y_i)^2$, which is actually equivalent to a strict generalization to histogram binning with both calibrated scores and bin boundaries optimized, and restricts the calibrated scores be monotonically increasing:

$$\begin{aligned} &\underset{M,\theta_i,\alpha_i}{min} \sum_{m=1}^{M} \sum_{i=1}^{n} \mathbb{1}(a_m \leq \hat{p}_i < a_{m+1})(\theta_m - y_i)^2, \\ \textit{subject to} \quad &0 = a_1 \leq ... \leq a_{M+1} = 1, \\ &\theta_1 \leq ... \leq \theta_M = 1. \end{aligned} \tag{2.35}$$

Platt scaling [39] is a parametric approach. It learns scalar parameters $a, b \in \mathbb{R}$ in a logistic regression model to transform logit z_i into calibrated probability $\hat{q}_i = \sigma(az_i + b)$. a and b are optimized in validation setting with NLL loss. All samples share the same parameters a and b.

2.3.1.2 Multi-class classification

For multi-classification problems, usually we use softmax function to derive

$\hat{p}_i$ from z_i, $\hat{p}_i = \max_k \sigma(z_i)_k$. The goal is to produce a calibrated probability $\hat{q}_i$ and new predicted label y_i' based on y_i, $\hat{p}_i$ and z_i. In most of the cases, y_i' does not change or changes in a small range.

Extension of binning methods. [41] By treating multi classification problem as K one-versus-all problems, histogram binning and isotonic regression can be extended to multiclass setting. For $k = 1, ..., K$, we have binary classification problem of true label equal to 1 if $y_i = k$, 0 otherwise. Using K seperate models, we would get a unnormalized K-vector, $\hat{q}_i' = [\hat{q}_i^{(1)}, ..., \hat{q}_i^{(K)}]$, where $\hat{q}_i^{(k)}$ is calibrated probability for the class k. New predicted label $y_i = \arg\max \hat{q}_i'$, and the calibrated probability $\hat{q}_i$ is $\hat{q}_i' / \sum_{k=1}^{K} \hat{q}_i^k$.

Matrix and vector scaling. This is extended from Platt scaling in binary classification problem, by learning a matrix W and a vector b instead of two scalars:

$$\begin{aligned} \hat{q}_i &= \max_k \sigma_{SM}(\boldsymbol{W z_i + b})^{(k)} \\ \hat{y}_i' &= \arg\max_k \sigma_{SM}(\boldsymbol{W z_i + b})^{(k)}, \end{aligned} \tag{2.36}$$

where matrix caling becomes vector scaling when matrix W is restricted to be a diagonal matrix.

Temperature Scaling [13] was introduced to be a multi-class extension to Platt scaling for deep neural networks, using a single scalar $T > 0$ for all classes. For an output logit vector z of a neural network, the calibrated confidence is

$$v(z) = \sigma(\frac{z}{T}) = \frac{exp(\frac{z}{T})}{\sum_{i=1}^{K} exp(\frac{z_i}{T})}, \tag{2.37}$$

where σ is the softmax function. The parameter T is optimized with respect to NLL on the validation set. For $T > 1$, it "softens" the confidence, making the model change from overconfident to more calibrated state. Since T does not change the ranking of probabilities of different classes, it does not affect the model's accuracy. Results were shown that temperature scaling works better than the other several above post-processing methods in computer vision context. [13]

Non-parametric Gaussian process based calibration [42] proposed to use Gaussian process to help calibration. More formally, we assume the latent calibrated function g:

$$g \sim GP(\mu(.), k(., .|\theta)), \tag{2.38}$$

where μ, k, θ are mean function, kernel and kernel parameters respectively. The calibrated confidence is

$$v(z)_k = \sigma(g(z)_k = \frac{exp(g(z))}{\sum_{i=1}^{K} exp(g(z_i))}, \tag{2.39}$$

We want to use a variational approximation $q(u) = \mathbb{N}(u|m, S)$ to approximate posterior $p(u|y)$, where $u \in \mathbb{R}^M$ is defined inducing variables, such that $p(g, u|y) \approx p(g|u)q(u)$, therefore calibrated probabilities at new input $(z_1^*, ..., z_L^*) \in \mathbb{R}^{LN}$ can be written as:

$$p(g_*|y) = \int p(g_*|g, u)p(g, u|y)dgdu \approx \int p(g_*|u)q(u)du. \tag{2.40}$$

After using maximizing a lower bound, we can find the variational parameters m and S for $q(u)$, and therefore can compute $p(g_*|y)$ here.
The predicted label y_* can be obtained by marginalization via Monte-Carlo integration:

$$p(g_*|y) = \int p(y_*|g_*)p(g_*|y)dg_* \tag{2.41}$$

Beta calibration [43] is a method that is based on the likelihood ratio between two Beta distributions. The calibration method is given by:

$$v(z) = (1 + exp(-c)(1 - z)^b z^{-a})^{-1}, \tag{2.42}$$

where z is the model output, $v(z)$ is the calibrated probability, a, b and c are fit on the validation data.

2.3.2 Built-in calibration methods

Except for post-processing methods, there are also built-in calibration methods that require modifying the classifier learning algorithm or training procedure. Some methods were originally designed for regularization and then

found to be helpful in calibration. Some methods directly modify the loss function.

Focal loss [44] was used as an alternative to the cross-entropy loss to improve calibration performance while preserving accuracy. First, cross entropy loss is defined as follows and can be shown to be an upper bound on the KL-divergence between the target distribution q and model's predicted distribution $\hat{p}$, therefore minimizing cross entropy loss results in minimizing $KL(q||\hat{p})$:

$$\mathcal{L}_c = -(1-\hat{p}_{i,y_i})log\hat{p}_{i,y_i}, \quad \mathcal{L}_c \geq KL(q||\hat{p}) \tag{2.43}$$

In comparison, focal loss is defined via a user-defined hyperparameter γ, and forms a upper bound on the regularised KL-divergence $KL(q||\hat{p}) - \gamma\mathbb{H}[\hat{p}]$, where $\mathbb{H} = \sum_{i=1}^{K} \hat{p}_i log\hat{p}_i$ is the negative entropy of the predicted distribution $\hat{p}$:

$$\mathcal{L}_f = -(1-\hat{p}_{i,y_i})^{\gamma}log\hat{p}_{i,y_i}, \quad \mathcal{L}_f \geq KL(q||\hat{p}) - \gamma\mathbb{H}[\hat{p}] \tag{2.44}$$

Therefore using focal loss results in adding a maximum-entropy regularizer to the original cross-entropy loss. On the other hand, encouraging the predicted distribution to have higher entropy can help overcome the over-confident problem by modern neural networks [45].

Label smoothing [46] was discussed to help calibration, which has similar effects with using temperatures. Rather than minimizing cross-entropy between the true label y_k and network output p_k as $H(y,p) = -\sum_{k=1}^{K} y_k log(p_k)$, label smoothing uses a parameter α to minimize the cross-entropy between modified label y_k^{LS} and networks' output p_k as $H(y,p) = -\sum_{k=1}^{K} y_k^{LS} log(p_k)$, where $y_k^{LS} = y_k(1-\alpha) + \alpha/K$, where $\alpha \in (0,1)$. If $\alpha \to 0$, $y_k^{LS} \to y_k$. If $\alpha \to 1$, the distribution of y^{LS} approximates a uniform distribution. In their paper, their experiments showed that label smoothing with $\alpha = 0.1$ has a similar effect with temperature scaling $T = 1.13$. Both of them improved calibration performance independently compared to the vanilla model. This result can be understood as label smoothing can improve the entropy of training signals, therefore provide better calibration.

Mix-up training [47] was defined as convexly combining random pairs of

images and their associated labels. More precisely,

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1-\lambda)x_j \\ \tilde{y} &= \lambda y_i + (1-\lambda)y_j, \end{aligned} \tag{2.45}$$

where x_i and x_j are two randomly sampled inputs, y_i and y_j are their associated labels. Results show that by simple mixup of data and labels, model accuracy and calibration performance can be improved. This behavior is thought not to be surprising given that mixup is a form of data augmentation, since by random sampling x_i and x_j, the probability that the model sees the same data twice is smaller than before.
However, it cannot be only explained by data augmentation, since the author did a series of experiments to explore if simply combining the features without combining the labels (which is only a data augmentation) can provide the same calibration benefit. Results show that merely mixing features does not provide the calibration benefit seen in the full-mixup case, which suggests that mixing the labels also contributes to calibration. By mixing training labels, the entropies of labels increase. As the strength of interpolation between the pairs of labels increases (which can be understood as λ increases), the entropy distribution of labels becomes from nearly a point-mass at 0 to nearly a point-mass towards 1. [47] further explains mixup by relating mixup loss function to the standard loss with additional adaptive regularization terms:

$$\hat{L}_n^{mix}(\theta, S) := L_n^{std}(\theta, S) + \sum_{i=1}^{3} R_i(\theta, S), \tag{2.46}$$

where $\hat{L}_n^{mix}(\theta, S)$ and $L_n^{std}(\theta, S)$ are loss induced by mixup and the standard empirical loss respectively. θ and S are model parameters and training sets respectively. $R_i(\theta, S)(i = 1, 2, 3)$ are penalty terms. They are functions of the first and second directional derivatives with respect to x_i: $\bigtriangledown f_\theta(x_i)$ and $\bigtriangledown^2 f_\theta(x_i)$. Therefore by minimizing mixup loss function, it has an effect of regularization and therefore helps improve generalization and calibration performance.

Fast Adversarial Calibration [48] is a more straightforward way to help calibration under domain shift. They add two loss functions to the original

cross-entropy loss: the entropy loss L_S and adversarial calibration loss L_{adv}. The entropy loss is defined as:

$$L_S = \sum_{i=1}^{n} \sum_{j=1}^{C} -\frac{1}{C} log(p_{ij}(1 - y_{ij}) + y_{ij}), \tag{2.47}$$

where p_{ij} denotes the confidence associated to the j-th class of sample i. y_{ij} is its one-hot encoded label. n and C are sample number and class number respectively. This loss comes from that they remove the confidence score for the correct class and compute the cross-entropy between a uniform distribution and the remaining false probabilities. By adding this loss function, the network has an effect of softening the probabilities and encouraging an increased entropy, and therefore, has an effect of preventing overfitting.
The adversarial calibration loss is defined as:

$$L_{adv} = \sqrt{\sum_{i=1}^{n} (acc(B_{m_i}) - conf(i))^2}, \tag{2.48}$$

which is written by summing over all samples in all bins and changing the L1 norm to L2 norm in ECE. In this way, they penalize the confidence scores. The total loss function can be written as $L = L_{CCE} + \alpha L_S + \beta L_{adv}$, with two hyperparameters α and β.

Regularizing Class-wise Predictions [49] is a method proposed to regularize class-wise predictions using self knowledge distillation. The class-wise regularization loss is KL divergence between different samples in the same class:

$$L_{clc}(x, x'; \theta, T) := KL(P(y|x'; \tilde{\theta}, T)||P(y|x; \theta, T)) \tag{2.49}$$

where x and x' are an input and another randomly sampled input having the same label y. $\tilde{\theta}$ is a fixed copy of the parameters θ. The gradient is not propagated through $\tilde{\theta}$.Therefore this loss enforces consistent predictive distributions in the same class. And the total loss is:

$$L(x, x', y; \theta, T) = L_{CE}(x, y; \theta) + \lambda_{cls} \cdot T^2 \cdot L_{clc}(x, x'; \theta, T) \tag{2.50}$$

Other work. Beside of our introduced calibration work, there are still some methods that are quite interesting and very worthy of reading, but due to

that they are not very related to this thesis, we will not give a detailed introduction. Rahimi et al. [38] proposed to use another neural network to calibrate the model and did good theoretical calibration proofs. This could be very simple and effective. Gupta et al. [33] proposed to use splines to help calibrate. It is also the paper proposing KS-error. Kumar et al. [34] proposed to combine scaling and binning method and verify it from both theoretical and empirical perspectives. Xing et al. [50] proposed a very novel calibration method, by first using prototypical learning to produce a representation space. Their belief is that a test sample's location in the representation space and its distance to training samples should contain important information about the model's decision-making process. Rahimi et al. [51] saw a connection between calibration and intra order-preserving functions, and proposed a novel method to help calibrate. Zhang et al. [52] used an ensemble-based calibration method that satisfies all the following three requirements: accuracy-preserving, data-efficient, and high expressive power. Since Guo et.al. [13] brought probability calibration into our sights in 2017, more and more calibration methods were proposed, but we believe attention on calibration is still not enough and there still exists a huge space for possible future calibration work in the next few years.

2.4 Our work

We focus on graph representation learning which has been gaining increasing attention in the machine learning community, but the calibration of graph learning models have not been sufficiently explored yet. In addition, we focus on node classification tasks, which are one of the most important tasks in real-world graph datasets. Teixeira et al. [15] perform an empirical evaluation of GNNs on multiple datasets and conclude that the calibration performance varies on datasets and models. GNNs can be calibrated in some datasets but also badly miscalibrated in other datasets. Also, existing calibration methods cannot help fix the miscalibration. But Teixeira et al. only did an end-to-end analysis and did not propose new methods.
Our work did a more detailed investigation about the relationship between calibration and properties of graph models, e.g. model capacity, aggre-

gation, and regularization, and the relationship between calibration and dataset characteristics. Especially, we propose our defined calibration metric to evaluate the balancing-class result. Next with the obtained understanding, we dive into graph neural models from the perspective of knowledge distillation and signal processing, aiming at finding some methods to improve calibration. Then we propose a straightforward method to modify the loss function to help calibrate. Our work is the first work to analyse KD's effect on calibration. Finally, we apply obtained knowledge to a real-world use case.

Chapter 3

Analysis

In this chapter, we first investigate the calibration performance on benchmark datasets with benchmark GNN models, and existing state-of-the-art calibration methods' effects. Then we explore the influence from model properties and dataset characteristics since the two core parts of machine learning are models and datasets. The conclusion can be made when comparing state-of-the-art models and calibration methods, also about which kind of models properties affects the calibration result the most.

3.1 Analysis on benchmark datasets

3.1.1 Datasets

Cora, Citeseer and Pubmed are the most usual benchmark classification datasets in recent published graph neural networks papers. They are three citation networks, where nodes represent papers and the edges represent a citation between papers, which are undirected edges. Node features are textual features, or bag-of-words. The predicted label of nodes is the topic of that paper. The statistics of the dataset can be viewed in Table 3.1. Since most of the models have already the best hyperparameters among these three citation networks, we follow their hyperparameters in these datasets in semi-supervised setting.

Table 3.1: Dataset statistics for citation datasets.

Dataset	Classes	Features num.	Nodes num.	Edges num.	Label rate (training node num. / total num.)	Avg. shortest path
Citeseer	6	3703	3327	4732	0.036	9.31
Cora	7	1433	2708	5429	0.052	5.27
Pubmed	3	500	19717	44338	0.003	6.34

3.1.2 GNN models, metrics and methods

In our experiments, we employed the Graph Convolutional Networks (GCN) [16], Graph Attention Networks (GAT) [17], Simple Graph Convolution (SGC) [24], GFNN [25], APPNP [11]. All models are implemented using the Pytorch-Geometric library [53].
Calibration metrics used include ECE, classwise-ECE, and marginal-ECE. We choose these three metrics since ECE is quite widely used and convenient to compare among different methods, and classwise-ECE and marginal-ECE are suitable metrics for all classes' comparison.

3.1.3 Experiment setup and hyperparameters

For citation datasets, hyperparameters of GCN, GAT, SGC, APPNP and Cheb are taken from their best values in Pytorch geometric library. For SGC, we set lr = 0.2, training for 100 epochs and tune weight decay for 60 iterations using hyperopt, like their original paper. We train without early-stopping. For gfNN, we take lr = 0.2, nhid = 32, epochs = 50, and layer = 2 from their original paper. Weight decay is tuned for 60 iterations using hyperopt. Each model is run in averaged 100 times.

3.1.4 Results

In this subsection, we discuss first uncalibrated performances of models on citation datasets. This could be a comparison of models. Then we compare the performance of different calibration methods.

Table 3.2: (Uncalibrated performance) Accuracy, uncalibrated ECE, Marginal ECE, class-wise ECE (w/ std. deviation) for each GNN family model.

Dataset	Model	Acc	ECE	Marg. ECE	Clw. ECE
Cora	GCN	81.34±0.47	21.17±0.73	6.50±0.16	6.74±0.17
Cora	GAT	83.26±0.41	15.19±0.54	5.08±0.20	5.45±0.22
Cora	SGC	80.98±0.05	24.95±0.19	7.62±0.06	7.78±0.07
Cora	gfNN	80.69±1.05	12.28±2.49	4.71±0.80	5.08±0.72
Cora	APPNP	**83.44±0.44**	16.96±0.56	5.61±0.18	5.89±0.21
Cora	Cheb	81.26±0.71	**11.01±0.92**	**4.24±0.33**	**4.56±0.38**
CiteSeer	GCN	71.24±0.65	21.04±0.89	8.49±0.25	8.47±0.24
CiteSeer	GAT	70.9±0.56	17.00±0.55	7.61±0.27	7.58±0.24
CiteSeer	SGC	71.91±0.05	42.89±0.08	15.54±0.01	14.91±0.01
CiteSeer	gfNN	70.28±0.98	13.68±4.34	8.04±1.72	7.96±1.70
CiteSeer	APPNP	**72.04 ± 0.46**	13.24±0.57	6.24±0.32	6.26±0.27
CiteSeer	Cheb	70.03±0.81	**7.75±0.93**	**4.87±0.35**	**4.75±0.36**
Pubmed	GCN	79.17±0.46	6.59±0.65	5.30±0.51	5.49±0.47
Pubmed	GAT	78.22±0.41	**4.46±0.76**	**4.70±0.75**	**4.88±0.71**
Pubmed	SGC	78.32±0.16	20.99±0.02	13.84±0.01	14.07±0.02
Pubmed	gfNN	78.84±0.95	6.94±8.30	5.70±5.83	5.94±5.78
Pubmed	APPNP	**80.13±0.29**	5.14±0.83	5.27±0.92	5.33±0.87
Pubmed	Cheb	73.43±2.03	5.32±1.07	5.15±1.3	5.31±1.29

3.1.4.1. Uncalibrated performance

In Table 3.2 we present the results for the GNNs applied to benchmark citation datasets.

APPNP, Cheb, and GAT are the best-performed models on citation datasets. APPNP has the best performance in terms of accuracy. Cheb and GAT have the best performance in terms of calibration metrics.

GNNs exhibit under-confident phenomenon. Reliability diagrams and confidence histograms are shown in Fig. 3.1. Here we only present figures of GCN and GAT. Figures of the other models are exhibited in Appendix A. Maybe it is because citation datasets are too small. This under-confidence phenomenon alleviates on Pubmed.

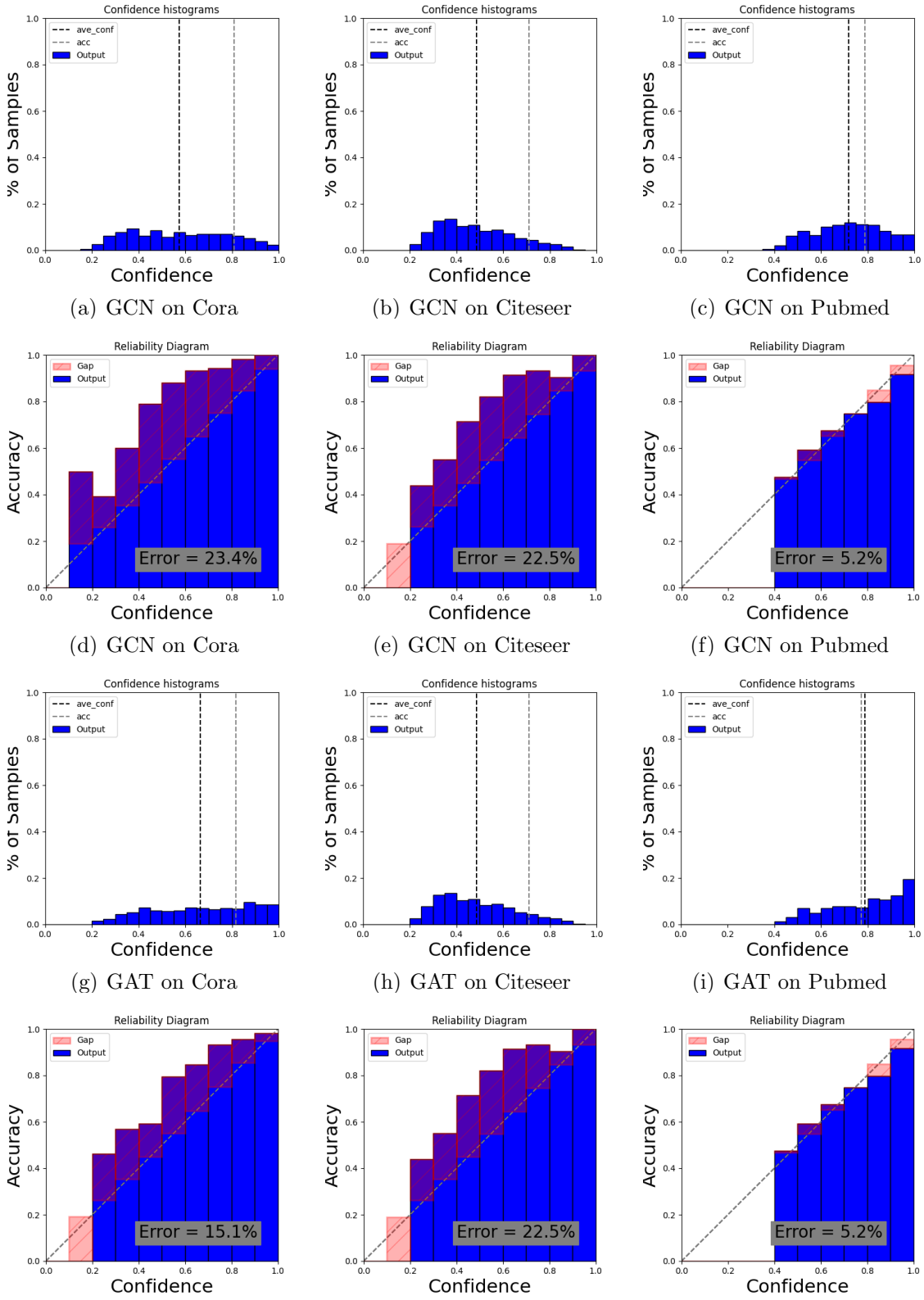
GAT is better calibrated than GCN on citation datasets. The difference between these two models is only the attention of each edge, therefore it is fair to compare them. Like we discussed in Chapter 2, GAT has some connections to transformers. And some papers [54] shows that pre-trained transformer (Bert and Robert) are generally better calibrated than non-pre-trained models (ESIM and DA) in both in-domain and out-of-domain setting. Our result shows some consistency that by learning attention, models could be more calibrated.

Models tend to be better calibrated on larger datasets. As the size of the dataset increases, e.g., GCN becomes from around 21% ECE on Cora and Citeseer to 6.6% ECE on Pubmed, GAT becomes from around $15-17\%$ on Cora and Citeseer to 4.5% on Pubmed.

3.1.4.2. Calibration performance.

Results are shown in Table 3.3. We can see that all post-processing methods could effectively calibrate GNN models. Among them, Histogram Binning, Isotonic Regression, and Beta Calibration perform the best. On Cora and CiteSeer, Histogram Binning and Isotonic Regression calibrate the best. On Pubmed, Beta Calibration is the best. Temperature Scaling, as expected to work better than Binning and Regression, but not in our case.

For a calibration method, accuracy-preservation or accuracy-improvement would also be good, therefore we present results of calibrated accuracy for



(a) GCN on Cora (b) GCN on Citeseer (c) GCN on Pubmed

(d) GCN on Cora (e) GCN on Citeseer (f) GCN on Pubmed

(g) GAT on Cora (h) GAT on Citeseer (i) GAT on Pubmed

(j) GAT on Cora (k) GAT on Citeseer (l) GAT on Pubmed

Figure 3.1: Confidence histogram and Reliability Diagram in GCN and GAT.

Table 3.3: (Calibrated performance) ECE (w/ std. deviation) for each calibration method and each GNN family model.

Dataset	Model	Uncal.	His	Iso	BBQ	TS	Beta	Meta
dataset	model	Uncal.	His	Iso	BBQ	TS	Beta	Meta
Cora	GCN	21.17±0.73	4.42±0.77	3.92±0.71	4.5±0.73	4.76±1.39	4.6±0.6	**3.88±0.6**
Cora	GAT	15.19±0.54	4.83±0.56	4.07±0.57	4.12±0.55	8.13±8.85	4.23±0.39	**3.37±0.65**
Cora	SGC	24.95±0.19	4.89±0.3	**3.63±0.27**	4.25±0.43	4.84±0.33	5.18±0.14	4.02±0.4
Cora	gfNN	12.28±2.49	3.93±0.85	**3.53±0.91**	4.02±0.87	5.33±1.78	3.88±0.88	4.06±1.33
Cora	APPNP	16.96±0.56	4.37±0.67	**3.78±0.59**	3.99±0.69	4.03±0.74	3.94±0.39	3.67±0.66
Cora	Cheb	11.01±0.92	4.32±0.76	**3.41±0.7**	3.99±0.73	3.53±1.66	4.56±0.66	5.59±0.83
CiteSeer	GCN	21.04±0.89	**4.58±0.76**	4.62±0.92	5.44±1.0	9.08±7.01	5.41±0.68	4.93±0.74
CiteSeer	GAT	17.00±0.55	5.32±0.75	**5.19±0.68**	5.27±0.73	7.86±6.85	6.38±0.68	6.27±0.65
CiteSeer	SGC	42.89±0.08	4.63±0.21	**3.94±0.15**	5.94±0.31	12.49±0.37	5.17±0.06	4.5±0.23
CiteSeer	gfNN	13.68±4.34	**4.84±1.37**	4.9±1.12	5.16±1.54	6.97±3.21	5.49±1.76	5.73±1.67
CiteSeer	APPNP	13.24±0.57	4.96±0.92	**4.85±0.84**	5.47±0.87	10.19±8.46	5.18±0.64	5.4±0.73
CiteSeer	Cheb	7.75±0.93	**5.2±0.84**	5.34±0.86	5.54±1.0	7.06±3.47	6.19±0.81	6.08±0.84
Pubmed	GCN	6.59±0.65	5.06±0.8	4.85±0.74	4.98±0.83	9.86±9.13	**4.1±0.55**	5.03±0.79
Pubmed	GAT	4.46±0.76	4.77±0.86	4.73±0.77	4.97±0.86	6.57±5.31	**3.59±0.68**	4.56±1.09
Pubmed	SGC	20.99±0.02	4.43±0.2	3.74±0.29	4.0±0.21	4.94±0.09	**3.67±0.11**	4.41±0.69
Pubmed	gfNN	6.94±8.30	4.89±0.98	5.02±1.03	5.02±1.3	8.69±8.21	**4.64±1.21**	6.04±2.69
Pubmed	APPNP	5.14±0.83	4.9±0.77	4.47±0.6	4.74±0.78	6.49±5.8	**4.04±0.58**	5.11±0.95
Pubmed	Cheb	5.32±1.07	4.98±0.77	5.01±0.8	4.9±0.87	8.39±6.25	**4.64±0.72**	5.22±1.02

Table 3.4: (Calibrated performance) Accuracy (w/ std. deviation) for each calibration method and each GNN family model.

Dataset	Model	Uncal.	TS	Meta	Iso	His	Beta	BBQ
Cora	GCN	81.34±0.47	∼	78.46±1.87	81.55±0.53	80.22±0.7	81.78±0.48	81.17±0.71
Cora	GAT	83.26±0.41	∼	79.13±1.56	83.88±0.41	81.43±0.46	83.81±0.33	83.43±0.51
Cora	SGC	80.98±0.05	∼	78.7±1.73	81.57±0.18	79.65±0.09	81.2±0.02	80.92±0.07
Cora	gfNN	80.69±1.05	∼	76.74±3.98	80.51±0.84	79.34±0.76	80.54±0.81	80.08±0.94
Cora	APPNP	83.44±0.44	∼	80.73±1.67	83.7±0.54	82.75±0.58	83.77±0.38	83.19±0.67
Cora	Cheb	81.26±0.71	∼	78.64±2.33	81.42±0.7	80.85±0.74	81.8±0.63	81.06±0.75
CiteSeer	GCN	71.24±0.65	∼	68.12±1.95	72.5±0.75	72.01±0.77	72.75±0.56	71.98±0.99
CiteSeer	GAT	70.9±0.56	∼	67.17±2.02	71.97±0.53	71.59±0.61	72.11±0.35	71.51±0.7
CiteSeer	SGC	71.91±0.05	∼	69.42±1.76	74.04±0.08	73.26±0.14	74.07±0.06	72.49±0.15
CiteSeer	gfNN	70.28±0.98	∼	63.72±7.73	71.63±1.03	71.22±1.13	72.33±0.76	71.02±1.31
CiteSeer	APPNP	72.04 ± 0.46	∼	68.98±1.92	72.68±0.48	72.95±0.51	72.92±0.3	72.54±0.79
CiteSeer	Cheb	70.03±0.81	∼	67.3±1.76	71.09±0.59	70.76±0.68	71.04±0.5	70.81±0.87
Pubmed	GCN	79.17±0.46	∼	77.05±1.7	78.81±0.54	78.42±0.71	78.95±0.38	78.64±0.85
Pubmed	GAT	78.22±0.41	∼	75.27±2.06	77.62±0.39	77.09±0.45	77.5±0.37	76.84±0.44
Pubmed	SGC	78.32±0.16	∼	77.86±1.64	78.98±0.12	78.99±0.05	78.81±0.04	79.09±0.12
Pubmed	gfNN	78.84±0.95	∼	77.05±1.7	78.81±0.54	78.42±0.71	78.95±0.38	78.64±0.85
Pubmed	APPNP	80.13±0.29	∼	78.09±1.88	80.15±0.37	79.93±0.45	79.85±0.29	79.8±0.62
Pubmed	Cheb	73.43±2.03	∼	71.55±2.44	73.21±1.74	72.55±1.8	73.39±1.76	72.91±1.73

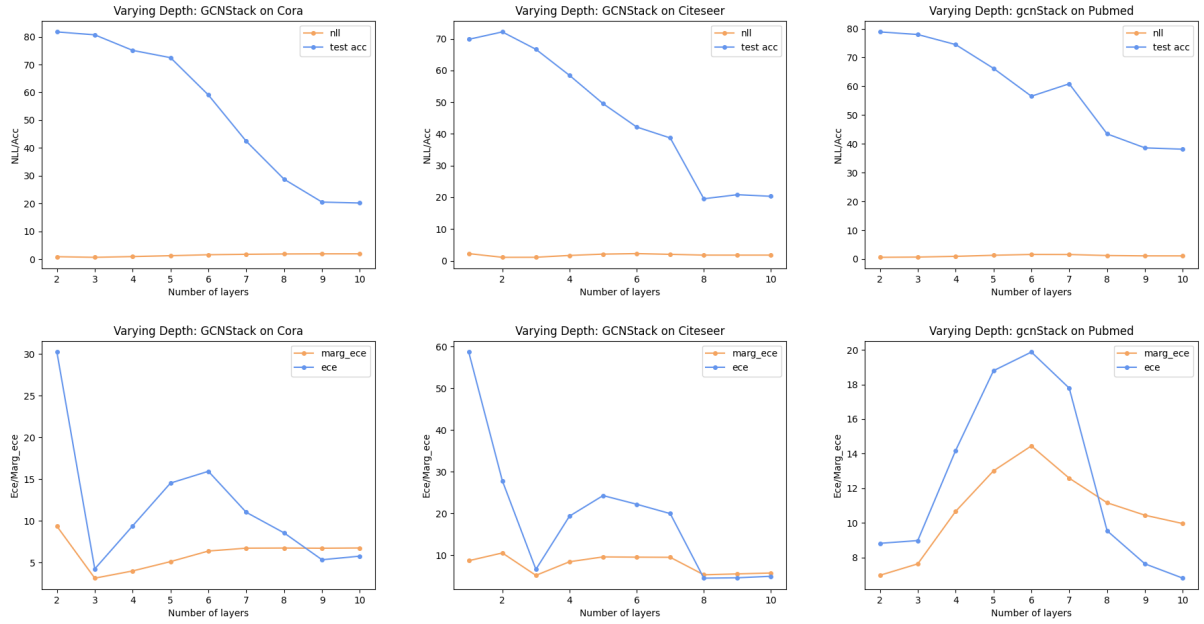


Figure 3.2: Varying depth for GCN on citation datasets.

each method, as shown in Table 3.4. Results show that all methods we choose can preserve accuracy well.

3.2 Analysis of graph model architecture components

3.2.1 Model depth

In these experiments, we investigate the influence of model depth (number of layers) on calibration performance.

Setting. Experiment setting follows by Appendix B in GCN [16]. We set the number of layers in GCNStack and GATStack (#attention heads is set to 1) from 1 to 10; the number of units for each hidden layer as 16; training for 400 epochs (without early stopping) using Adam (lr = 0.01, weight decay = 5e-4); dropout rate = 0.5, in the first and last layer. Each result is 10 time-averaged. For SGC, we vary the number of layers in 1,2,...,10,15,...,60.

Results. As in Fig. 3.2, Fig. 3.3, and Fig. 3.4 shown, first, over-smoothing is obvious. Accuracy starts to decrease after 2 layers' iterations for GCN and GAT. For SGC, it becomes clear after 10 layers. The best results for both classification and calibration are obtained with 2-3 layers. GNNs

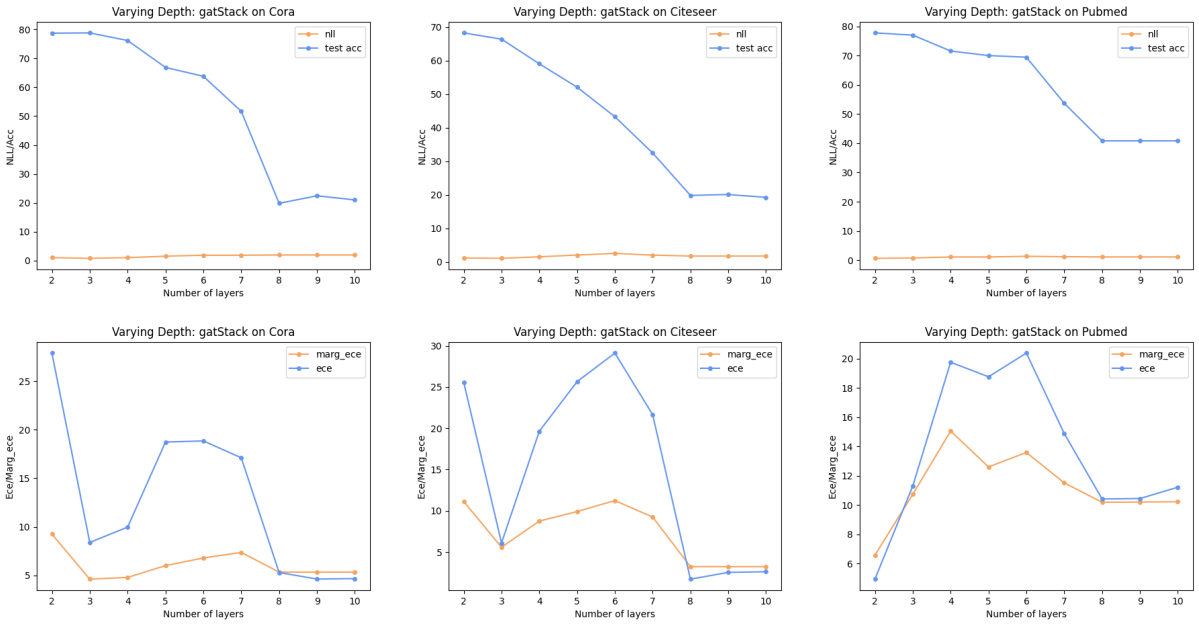


Figure 3.3: Varying depth for GAT on citation datasets.

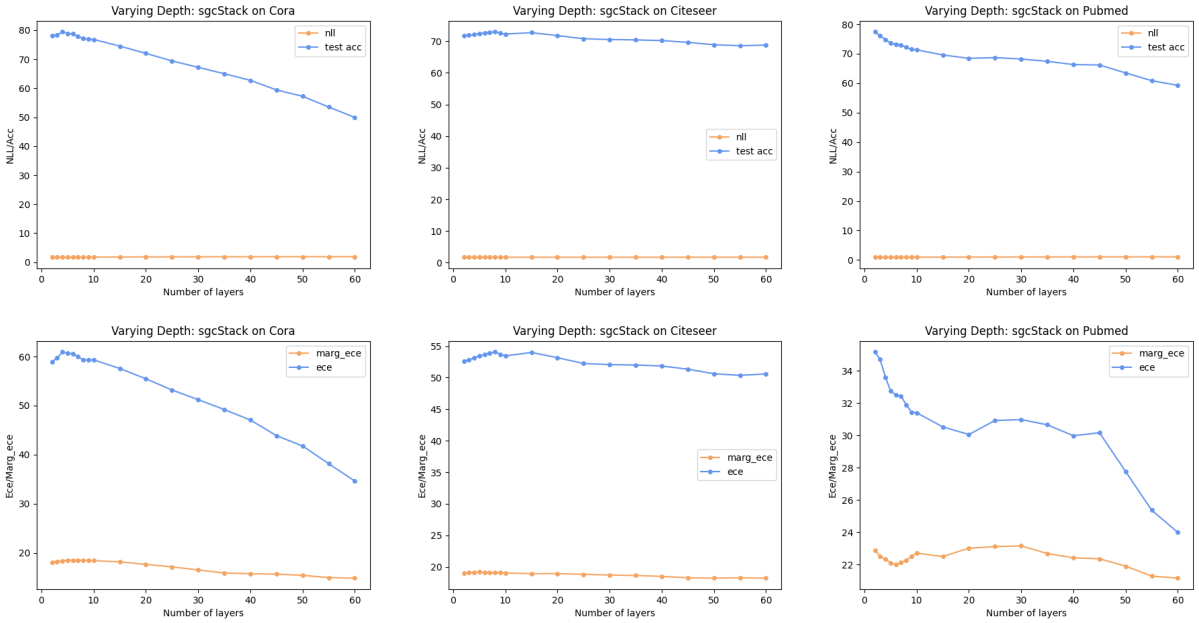


Figure 3.4: Varying depth for SGC on citation datasets.

are usually set to 2 layers for node classification tasks on various datasets. For models deeper than 3 layers, over-smoothing starts to become obvious, with accuracy decreasing drastically. ECE first increases resulting from meaningful information not as much as before, then decreases due to the low accuracy. The layer range of ECE-increasing is $3 - 6$ for GCN and GAT, and less than 10 for SGC. Therefore, we can say, *over-smoothing hurts calibration.*

3.2.2 Model width

Compared with model depth whose increasing causes over-smoothing, model width is more like a factor of model capacity. We define model width as the number of neurons per layer in GNN. Experiment setting is like before, we keep the other condition the same and vary the number per layer in $8, 16, 32, ..., 2048$ and observe accuracy, ECE and marginal-ECE. We do 10 times run.
Results are in Fig. 3.5. Wider nets can help calibrate.

3.2.3 Aggregation

In this experiment, we investigate the influence of aggregation of neighbors on calibration performance. We remove the edge list in models randomly from 0% to 100%, and observe their accuracy and ECE performance. The averaged number of edges per node ranges from 0, which is equivalent to a MLP + self-loops, to their original values. Metrics include accuracy, ECE, marginal-ECE and classwise-ECE. Results are repeated for 10 times.
Results are shown in Fig. 3.6, GCN is relatively stable but GAT's neighborhood aggregation can help calibrate. Attention of edges to neighbors brings in helpful information to calibration.

3.2.4 Regularization

Regularization methods to mitigate overfitting are related to calibration. Methods like Label smoothing and Mix-up training are shown to help calibration. But the exact relationship and the best approach to prevent overfitting with respect to calibration remain an open question. Relationship

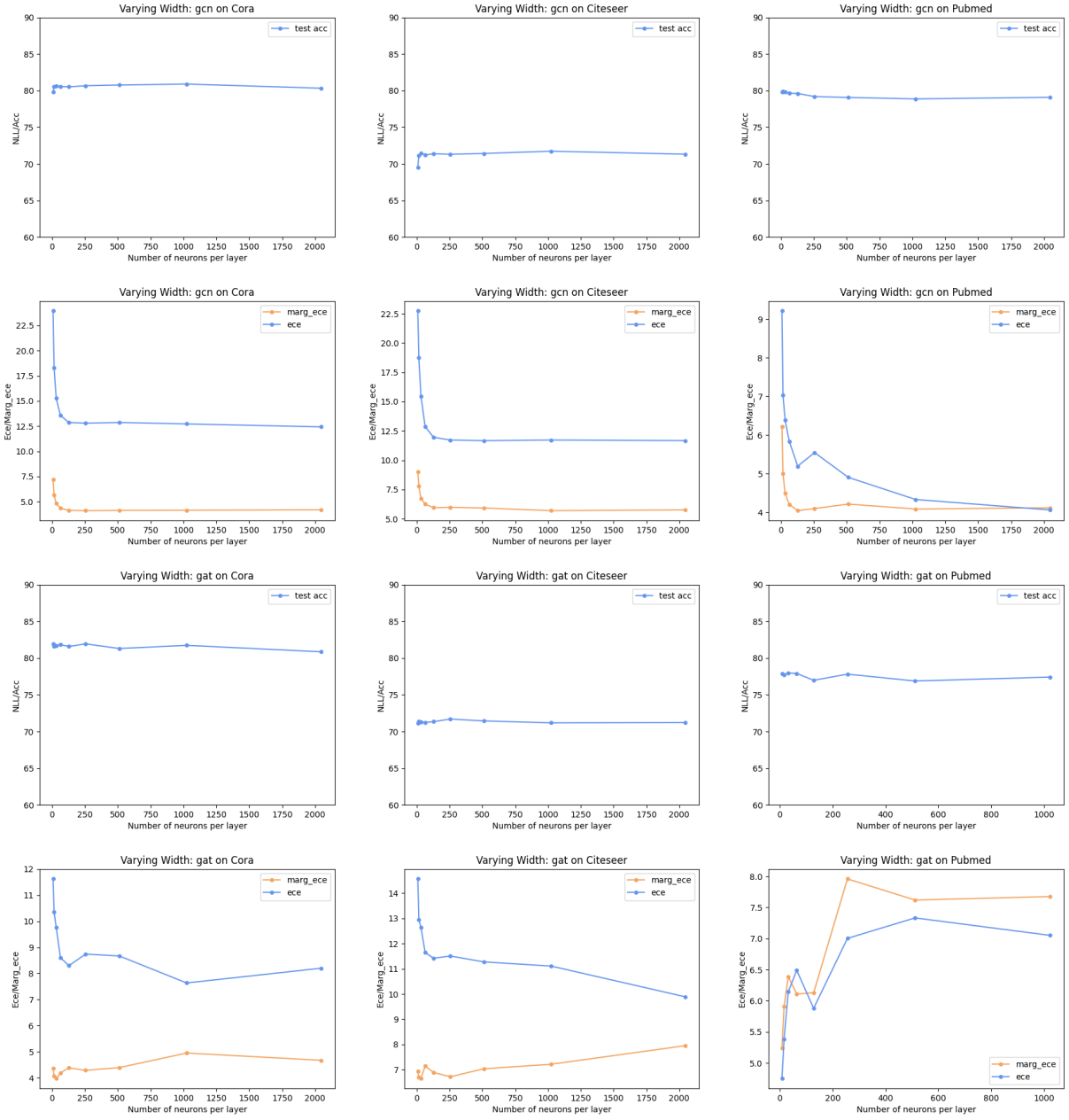


Figure 3.5: Varying width for GCN and GAT on citation datasets.

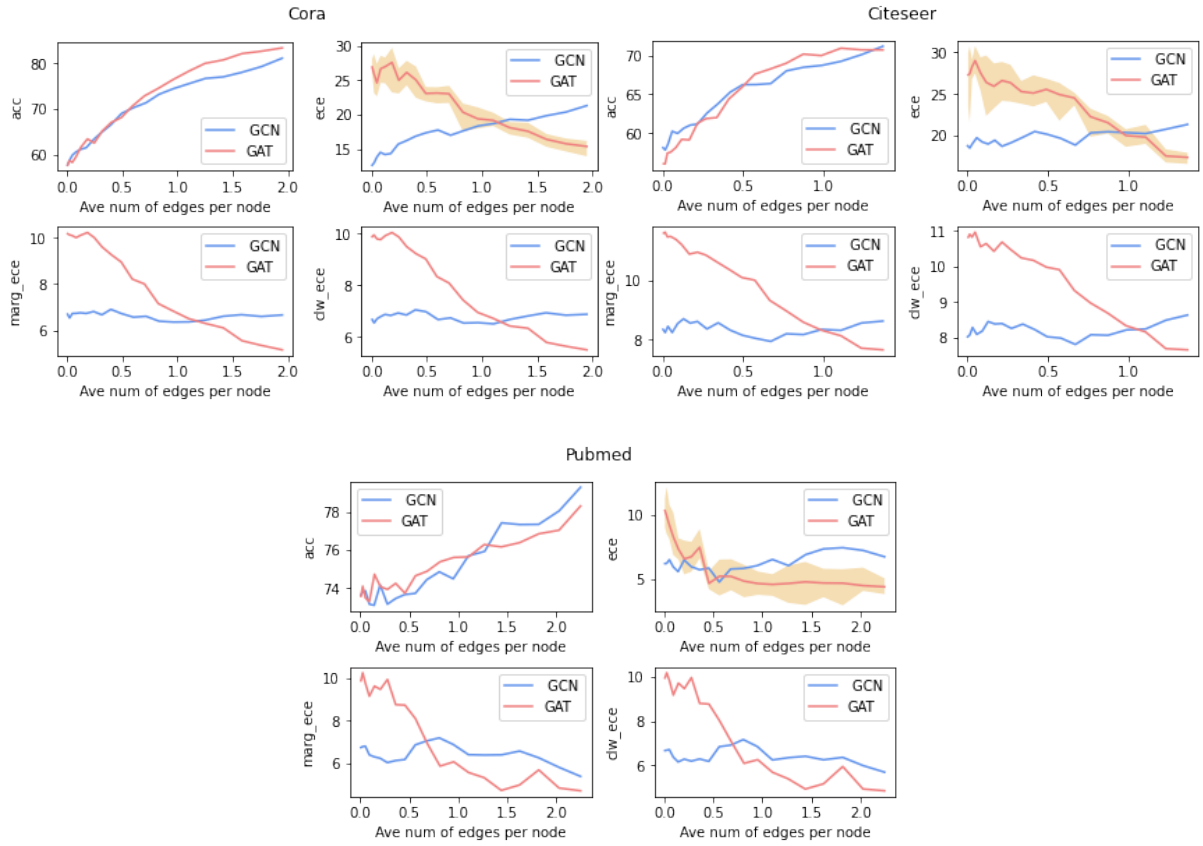


Figure 3.6: Aggregation's influence in GCN and GAT. Here we only show uncertainty shading for ECE of GAT.

between Monte Carlo dropout [55] and calibration performance was explored in computer vision context [56]. Their conclusion is that "increasing the drop probability improves the ECE at the cost of hurting the accuracy, which is as expected since randomized predictions can naturally get more calibrated but less accurate as we increase the level of randomization". Here we explore the simple dropout on layers in graph models and datasets.

Setting. 2-layer GCN, GAT and SGC models on Citation datasets, with 4 conditions of dropout from lelf to right in the figure: without dropout, with dropout before the layer, after each layer, and on all places, with dropout rate = 0.5. Each data is 10-time experiments averaged. Hyperparameters of models are taken by their original papers.

Results. Adding dropout to the model could improve accuracy at times but hurt ECE in almost all cases, in Fig. 3.7. This conclusion is universal with GNN models and datasets, but contrary to the conclusion there [56]. Our results are as [13] said, "Modern neural networks exhibit a strange phenomenon: probabilistic error and miscalibration worsen even as classification error is reduced."

3.3 Analysis of dataset characteristics

In this part, we explore the influence of dataset characteristics (imbalanced or balanced classes). Balancing the classes is important for imbalanced tasks where the model cannot learn something meaningful. A common balancing method is to simply weight the loss function, such that each example contributes the same to the loss. And it is also important to balance the test data (otherwise we will have a different test data distribution from training.), and balance the metrics.

[15] shows that by balancing the classes for a difficult and severe class imbalance task (Friendster), it could lead to a more calibrated result. They even show that for a model training under balanced distribution, evaluating also under balanced distribution has a greater impact on calibration results than using normal calibration methods, such as Temperature scaling, but evaluating on the unbalanced metrics.

But that observation only appears in their hardest dataset (Friendster). Our

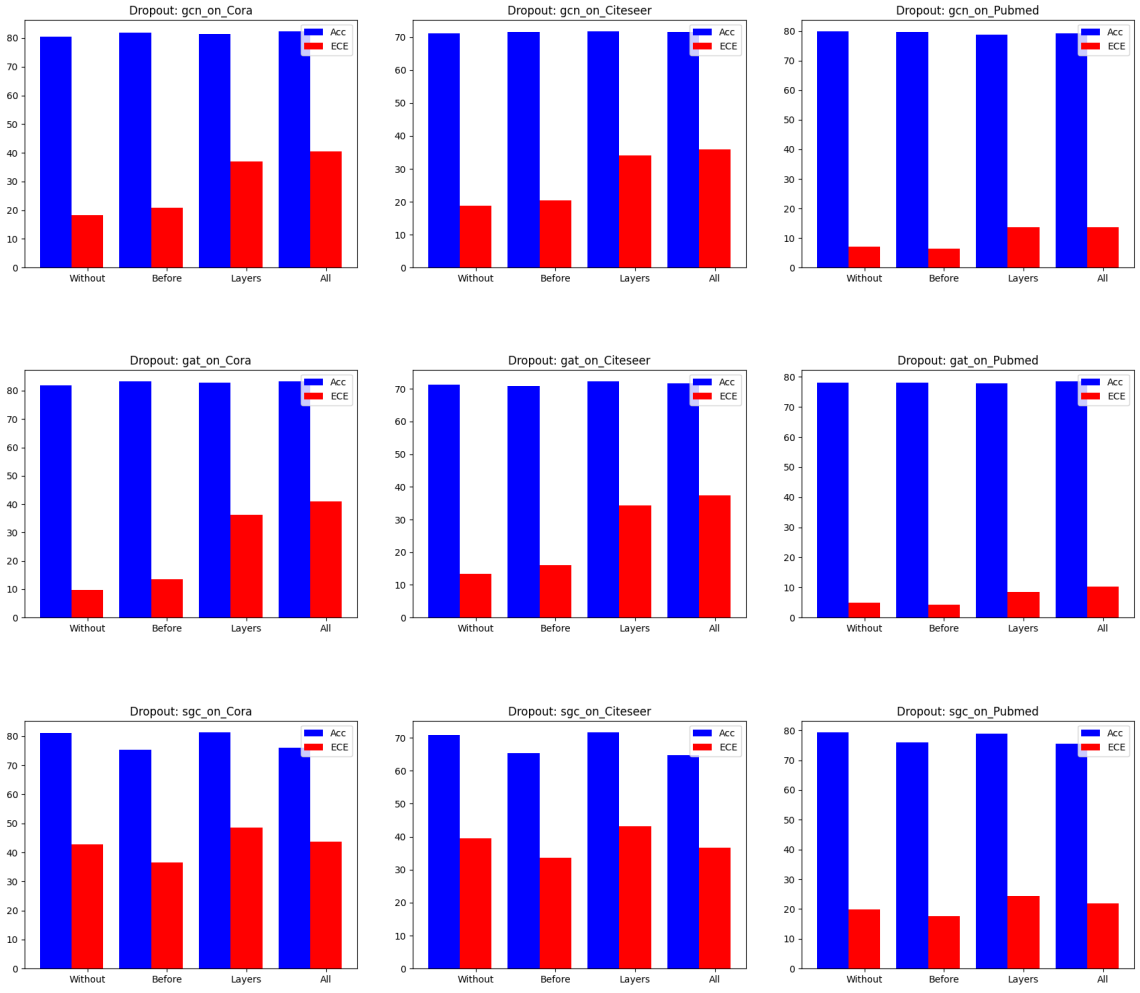


Figure 3.7: Dropout in different places in GNN models.

Table 3.5: Dataset imbalance statistics for citation datasets.

Dataset	Classes	Imbalance Ratio (Largest/Smallest classes)
Citeseer	6	2.82
Cora	7	4.54
Pubmed	3	1.92

experiments here explore for the influence of balancing the distribution in benchmark citation datasets with semi-supervised setting. Table 3.3 shows the imbalance result for citation datasets. Distribution is balanced for both training and test data, and evaluating metrics. We define a new metric called balanced ECE.

Balanced ECE is defined as:

$$Balanced - ECE = \sum_{m=1}^{M} \sum_{k=1}^{K} \frac{|B_m|}{N} w_k \cdot |acc(B_{mk}) - conf(B_{mk})|,$$

where M is the number of bins, N is the number of samples, $|B_m|$ is the number of samples in that bin, B_{mk} is the number of samples in m-th bin in k-th class, w_k is a weighted matrix for the class k, denoting how important the class k is. If all classes are equally important, $w_k = 1/k$. $acc(B_{mk})$ and $conf(B_{mk})$ are accuracy of samples in m-th bin with k-th class true label, and averaged confidence scores of those samples respectively.

Note that this metric is *different* defined from classwise-ECE and marginal-ECE. Classwise-ECE's accuracy term is defined as the ratio of the number of samples of k-th class in m-th bin to the number of samples in m-th bin. Marginal-ECE's accuracy term is defined as accuracy of *k-th class scores in m-th bin.*

Setting. The experiment setting is the same as section 3.1.3, with all parameters taken from the original papers. Data distribution is balanced in both training and test losses. Both accuracy and calibration metrics are balanced. Here we show results of uncalibrated balanced ECE and balanced classwise-ECE. Experiments are repeated 10 times averaged.

Results. Results can be seen in Table 3.6. Compared with Table 3.2, it shows that for not very imbalanced datasets, balancing data distribution and evaluating metrics cannot provide obvious improvement to calibration

Table 3.6: (Balanced uncalibrated performance) Accuracy, uncalibrated ECE, Marginal ECE, class-wise ECE (w/ std. deviation) for each GNN family model.

dataset	model	acc	Uncal. ece	Uncal. Marg. ece	Uncal. Clw. ece
Cora	GCN	81.29±0.59	21.76±0.77	6.52±0.18	7.59±0.2
Cora	GAT	83.17±0.34	16.24±0.67	5.07±0.2	5.85±0.22
Cora	SGC	80.75±0.07	39.85±0.15	11.8±0.03	13.7±0.06
Cora	gfNN	80.01±0.95	19.73±1.54	6.22±0.62	7.43±0.76
Cora	APPNP	**83.66±0.54**	17.97±0.71	5.61±0.18	6.5±0.22
Cora	Cheb	82.12±0.57	**13.26±0.78**	**4.24±0.33**	**5.1±0.33**
CiteSeer	GCN	68.52±0.78	22.3±0.96	8.49±0.25	9.14±0.35
CiteSeer	GAT	67.83±0.48	20.49±0.89	7.61±0.26	8.35±0.47
CiteSeer	SGC	68.48±0.01	41.52±0.02	16.09±0.02	15.79±0.01
CiteSeer	gfNN	67.5±1.01	18.97±2.23	7.97±1.33	8.61±2.09
CiteSeer	APPNP	**68.85±0.53**	17.59±0.69	6.24±0.33	7.04±0.52
CiteSeer	Cheb	67.1±0.77	**14.22±0.94**	**4.86±0.34**	**5.46±0.55**
Pubmed	GCN	79.15±0.47	7.73±0.66	5.3±0.52	6.39±0.68
Pubmed	GAT	78.4±0.41	**7.66±0.74**	**4.69±0.74**	**5.67±0.91**
Pubmed	SGC	78.05±0.17	21.14±0.07	14.13±0.02	15.63±0.04
Pubmed	gfNN	78.75±0.8	10.42±8.06	6.2±5.57	6.8±5.32
Pubmed	APPNP	**80.25±0.38**	8.01±0.68	5.27±0.91	6.32±1.11
Pubmed	Cheb	73.45±1.62	7.79±1.24	5.19±1.28	**5.67±1.38**

performance. Balancing all places data plays a part (train and test loss, test accuracy and calibration metrics) is equivalent to balancing dataset, therefore it could be a fair comparison before and after balancing. Graph neural networks have the ability to automatically balance classes. But claims can also be made that our citation datasets are not very imbalanced (since the imbalanced ratio is not very high in Table 3.5), future work could be done when considering more imbalanced datasets (e.g. Amazon-computer [28] with imbalanced ratio of 18 and Cora-Full [28] with imbalanced ratio of 62).

3.4 Conclusion

We provide a comprehensive analysis of GNNs and node classification datasets in this chapter.

1. We first show that benchmark GNNs on citation datasets are underconfident. Existing calibration methods lead to well-calibrated results.

2. Further, we show that model depth, width, aggregation and regularization could all affect models' calibration.
2.1. When increasing depth from the most proper depth value (usually set to 2 or 3), models become first less calibrated, but deeper models leading to over smoothing improve calibration.
2.2. Wider networks can help calibrate.
2.3. Neighborhood aggregation with attention could help calibrate.
2.4. Miscalibration worsens when regularization is used to reduce classification errors.
3. GNNs perform stably as datasets are balanced or not.

Chapter 4

Knowledge distillation for calibration

In this chapter, first, we give an introduction about existing knowledge distillation methods (response-based and feature-based), and knowledge distillation methods on graph representation learning. Then we investigate whether knowledge distillation methods could help calibrate.

4.1 Knowledge distillation

4.1.1 The basic knowledge distillation

Knowledge Distillation (KD) is first proposed in [57]. Basically, a knowledge distillation system consists of three key components: knowledge, distillation algorithm, and teacher-student architecture, which is as shown in Fig. 4.1. The goal is to distill the knowledge from a usually large teacher model into a smaller model so that the student model can hold a similar performance to the big one.
Knowledge distillation is inspired by the intuition that the relative probabil-

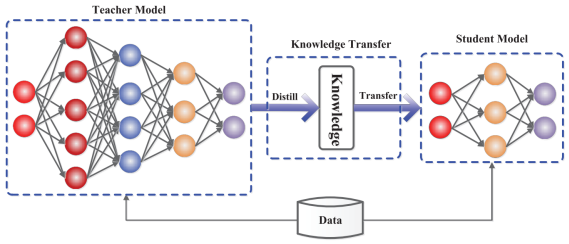


Figure 4.1: An example schematic for knowdge distillation. [5]

ities of incorrect answers also tell us a lot. Usually, the training objective of a benchmark model is to maximize the average log probability of the correct answer, but a side-effect is that the trained model assigns probabilities to all the incorrect answers to make them smaller to make them less different from each other. However, these smaller probabilities also contribute to the final performance, since some of these small values are still higher than the others. For example, a Shiba Inu image may have a small wrong probability of being a dog of another category, which is still many times more probable than being a car.

A way to transfer the generalization ability of the big model to a small one is to use the probabilities of the big model as soft targets for training the small model. The teacher's output is smoothed by setting a temperature in the softmax function:

$$q(z) = \sigma(\frac{z}{T}) = \frac{exp(\frac{z}{T})}{\sum_{i=1}^{K} exp(\frac{z_i}{T})}, \tag{4.1}$$

where $q(z)$ is the distilled knowledge from the teacher. The temperature T is usually a value higher than 1 in order to soften the logits. A higher value for T produces a softer probability distribution over classes. Then for the distilled model, the loss function is a weighted average of two different loss functions. The first one is the cross-entropy with the soft targets from the cumbersome model. This cross-entropy uses the same temperature as was used for generating the soft targets from the cumbersome model. The second loss function is simply the cross-entropy loss for the distilled model with its labels. The best results usually come from a lower weight on the second loss. At the same time, it is important to multiply the first loss with T^2, since the magnitudes of the gradients produced by the scale of the soft targets as $1/T^2$[57]. In this way, it ensures that when the temperature changes, the relative contributions of these two losses remain roughly unchanged. More precisely the total loss can be written as:

$$L = (1 - \lambda)CE(p, y) + \lambda \cdot T^2 \cdot KL(p, q), \tag{4.2}$$

where p, q and y are probabilities from distilled model, distilled probabilities from the cumbersome model, and labels.

4.1.2 Response-based KD

Based on the different objective of distilling, variants of KD can be divided into the following categories: response-based (logits-based), feature-based, and relation-based [5]. The above KD is one kind of response-based method, while there are some others methods. We will give a short introduction to two response-based KD methods as an example.

Deep mutual learning [6] use a pool of untrained student models which learn simultaneously to solve the task together, rather than using a pre-trained cumbersome model. Each student is trained with two losses: a conventional supervised learning loss, and a mimicry loss.

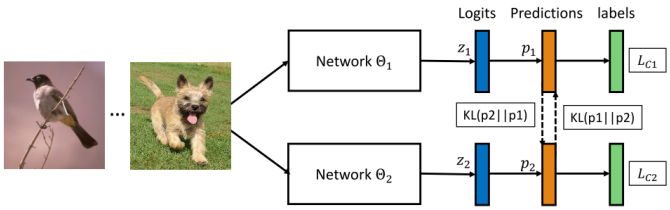


Figure 4.2: An example schematic for Deep mutual learning. [6]

Suppose there are N given samples $X = x_{i}{}_{i=1}^{N}$ from M classes. Labels are denoted by $Y = y_{i}{}_{i=1}^{N}$ with $y_i \in 1, 2, ..., M$. $p^m_{1/2}(x_i) = \sigma_{SM}(z^m_{1/2})$ is the probability of class m of output z_i from model 1 or 2. σ_{SM} is the softmax function. As shown in Fig. 4.2, take a cohort of two networks as an example. Θ_1 and Θ_2 are two peer networks, which work together to get better performance. A KL divergence is used to measure the match of the two network's predictions p_1 and p_2,

$$KL(p_2||p_1) = \sum_{i=1}^{N}\sum_{m=1}^{M} p_2^m(x_i) log\frac{p_2^m(x_i)}{p_1^m(x_i)}. \tag{4.3}$$

The overall loss function L_1 and L_2 for model 1 and 2 are defined as:

$$\begin{aligned} L_1 &= L_{CE1} + KL(p_2||p_1) \\ L_2 &= L_{CE2} + KL(p_1||p_2), \end{aligned} \tag{4.4}$$

where $L_{CE1/2}$ are their own cross entropy losses. In the optimization step, this mutual learning strategy is performed in each mini-batch step throughout the whole training process. The optimization of model Θ_1 and Θ_2 is done iteratively until convergence. Results show that each student model in such a peer-teaching way significantly performs better than when learning alone. Moreover in this way student model can perform better than the conventional distillation from a larger cumbersome model.

Label refinery [7] used KD as a label refinery process. They input the refined label into the sample model several times. The refined label in each iteration is the last model's output. The first refined label is the ground-truth label. As shown in Fig. 4.3, after a few iterations of label refining, the labels of the final model are informative. Results show that this method improves the model accuracy as well as model generalization.

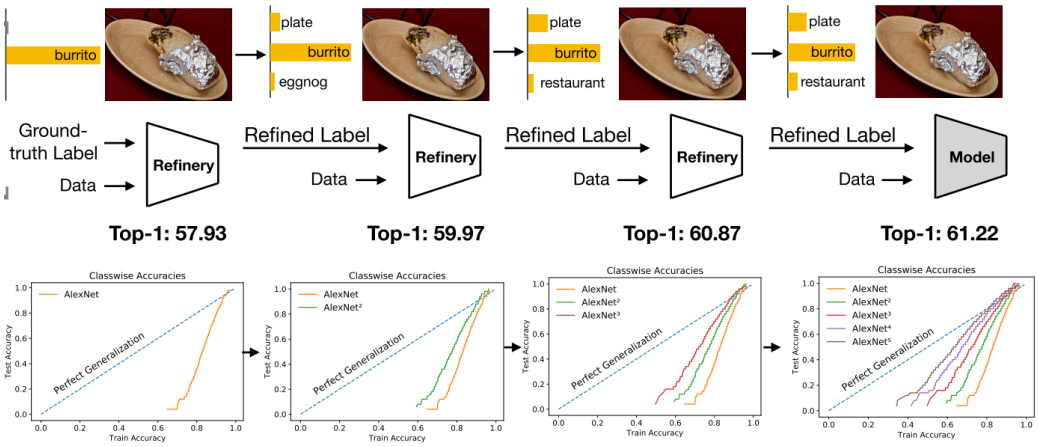


Figure 4.3: An example schematic for Label refinery. [7]

Response-based KD has an advantage of being straightforward and easy to implemented. The effectiveness of teacher's soft labels is analogous to label smoothing or regularizers. However, it relies on the output of the last layer and fails to utilize teacher's intermediate-level information, therefore it may fail when the gap of capacity between the student and teacher is too much.

4.2 KD for graph representations

Although KD is widely used in the computer vision area for model compression and improving generalization ability, most of the papers focus on CNNs. In comparison, less attention was devoted to combining knowledge distillation with GNNs. [58] proposed a method to compress a deep GCN with large feature maps into a shallow one with fewer parameters in the computer vision area, by transferring the local structure information from the teacher model. [59] proposed a KD framework to preserve prior knowledge, which results in $1.4\% \sim 4.7\%$ accuracy increases in student models.

Local structure-preserving (LSP) [58] is the first paper to introduce knowledge distillation into GNNs framework. For a graph $G = (\mathcal{V}, \mathcal{E})$ and a set of node features $\mathcal{Z} = \{z_1, ..., z_n\} \in \mathbb{R}^F$, where n is the number of nodes, F is the dimension of the feature maps. The local structure of graph G can be formulated as a set of local structure LS_i of each node i :

$$\mathcal{LS} = \{LS_1, ..., LS_n\},$$

where $LS_i \in \mathbb{R}^d$ is a vector, d is the dimension of node i. For different nodes, their dimension could be different, which depends on their local graph structure. Each element of local structure LS_i can be calculated as:

$$\begin{aligned} LS_{ij} &= \frac{e^{SIM(z_i, z_j)}}{\sum_{j:(j,i)\in\epsilon}(e^{SIM(z_i, z_j)})} \\ SIM(z_i, z_j) &= ||z_i - z_j||_2^2, \end{aligned} \tag{4.5}$$

where SIM is a similarity function between the given node features of node i and j. Here it is defined as the euclidean distance. Then LS_{ij} is defined as a softmax function of the similarity function, where ϵ is the neighbor set of node i.

After computing local structure LS_i for each node i, given teacher and student models in knowledge distillation, the local structure for node i between teacher and student is computed as a KL divergence between their corresponding LS_i^s and LS_i^t:

$$S_i = KL(LS_i^s || LS_i^t) = \sum_{j:(j,i)\in\epsilon} LS_{ij}^s log(\frac{LS_{ij}^s}{LS_{ij}^t}). \tag{4.6}$$

Finally, the local structure preserving loss L_{LSP} and the total loss L are defined as:

$$\begin{aligned} L_{LSP} &= \frac{1}{n}\sum_{i=1}^{n} S_i \\ L &= L_{CE}(p_s, y_s) + \lambda L_{LSP}, \end{aligned} \tag{4.7}$$

where p_s and y_s are the prediction and labels of student model, L_{CE} sis the cross entropy, λ is a hyperparameter.

CPF [59] is motivated by the issue that GNNs could not make full use of structure knowledge lying in the data. The solution to address it is to design a KD framework where the student model includes a mechanism to preserve structure-based prior knowledge, which is called *label propagation* (LP). Formally, we denote the prediction of LP after K iterations as f_{LP}^k and set the initialization as:

$$f_{LP}^0(v) = \begin{cases} (0, ..., 1, ...0) \in \mathcal{R}^{|Y|}, & for\ v \in V_L \\ (\frac{1}{|Y|}, ...\frac{1}{|Y|}, ...\frac{1}{|Y|}) \in \mathcal{R}^{|Y|}, & for\ v \in V_U \end{cases} \tag{4.8}$$

where V_L and V_U are labeled node set and unlabeled set. The update function for $k+1$ layer can be written as:

$$f_{LP}^{k+1}(v) = \sum_{u \in N_v \cup \{v\}} w_{uv} f_{LP}^k(u), \tag{4.9}$$

where w_{uv} is the edge weight between node u and node v computed by:

$$w_{uv} = \frac{exp(c_u)}{\sum_{u' \in N_v \cup \{v'\}}}. \tag{4.10}$$

where c_v is called confidence score. For inductive setting, the confidence score c_v can be computed by:

$$c_v = z^T X_v, \tag{4.11}$$

where $z \in \mathcal{R}^d$ is a learnable parameter to project node features to the confidence score.

Besides the above label propagation module, there is also a feature transformation module (FT), where the node features are emphasized. A final

FT is simply a two-layer MLP of the feature matrix:

$$f_{FT}(v) = softmax(MLP(X_v)). \tag{4.12}$$

A final score is the combination of PLP and FT with $\alpha \in [0, 1]$:

$$f_{CPF}^{k+1}(v) = \alpha_v \sum_{u \in N_v \cup \{v\}} w_{uv} f_{CPF}^{k}(u) + (1 - \alpha_v) f_{FT}(v), \tag{4.13}$$

where f_{CPF} is the same as f_{CP} in Eq. 4.9.

4.3 KD for calibration

Although a lot of KD-related work has been proposed, none of them discussed the calibration performance of KDs, or whether KD can be used to improve calibration. Our work here first aims to evaluate the calibration performance of KDs on graph neural networks. The intuition to use KD for calibration is that by learning from the teacher's "softer" logits, students could learn more information about the model's decision-making and confidence estimation, which should be useful for calibration. Besides, for those overconfident models, KD can be interpreted as a regularization term, like Li et al. [60], which should help prevent overfitting and help calibrate.

4.3.1 Empirical studies of calibration performance of KDs on graphs.

A first try to evaluate the influence of hyperparameters KD to calibration could be done easily. We distill the information from a 4-layer GCN teacher model to a student that has the same architecture as the teacher. We evaluate the influence of α and T as KD hyperparameters (like 4.2 denoted λ as α), with α chosen in $\{0.99, 0.95, 0.5, 0.1, 0.05\}$ and T chosen in $\{1, 2, 5, 10\}$, like in [61].
Results can be viewed in Fig. 4.4, where the left figure is the influence on student accuracy, the right figure is for ECE. Since it is only a one-time experiment and only on one dataset, the result might be not accurate enough, but still we can see that by choosing optimal α and T, the student can obtain a higher accuracy than the teacher. Like the original paper suggested,

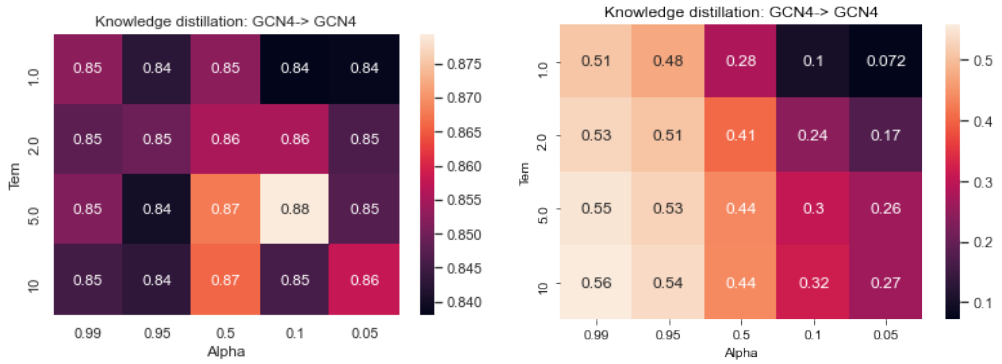


Figure 4.4: Evaluating KD hyperparameters on Cora. Here the teacher has accuracy of 0.84 and ECE of 0.05. Left: Accuracy. Right: ECE. Note the train-val-test split is not the same as experiments before.

a small α can lead to better performance. While for ECE, the student consistently performs worse than the teacher with different parameters.

More KD benchmark methods were utilized to explore whether KD can help calibrate. The baseline we choose includes the baseline KD and CPF. For KD, hyperparameter tuning could be an important factor in the final result. Here we state our choice of parameters in each method. For the baseline KD (the most normal KD), we take $\alpha = 0.1$, the same as the original paper. T is tuned in validation set in $\{1, 5, 10, 50, 100\}$. We use Adam optimizer like the original GNNs papers with $lr = 0.01$ and weight decay tuned in the range of $[1e-9, 1e-3]$. Teacher and student models are repeated 10 times and averaged.

For CPF, we trained the teacher model one time and trained student models with 10 time-averaged learning from the same teacher. Training epochs of the teacher and students are 500, with early-stopping of 50 patience. Hyperparameters are taken from the original paper [59].

For CPF, we also utilize two more datasets, Amazon-computer and Amazon-photos [28]. They are items co-puchased graph in Amazon, like mentioned in Section 2.1.3.5. These two datasets' nodes represent goods and edges represent they are frequently co-purchased. Features are information about the goods and labels denote their categories.

Results are shown in Table. 4.3.1 (CPF) and Table. 4.3.1 (the basic KD). The basic KD and CPF methods could improve accuracy a little for most

Table 4.1: CPF of GCN and GAT on Cora, Citeseer, Pubmed, Amazon-computers and photos datasets.

Datasets	Teacher	Teacher acc	CPF	Acc ↑ (%)	Teacher ECE	CPF	ECE ↑ (%)
Cora	GCN	82.6	83.63±0.48	1.2	25.4	38.27±1.79	50.7
Citeseer	GCN	69.2	72.33±0.56	4.5	6.63	33.09±7.60	399.1
Pubmed	GCN	79.8	75.93±7.58	-4.8	9.02	27.33±1.95	203.0
A-computers	GCN	83.95	84.16±0.87	0.3	4.98	5.62±0.65	12.9
A-photos	GCN	90.88	93.26±0.34	2.6	6.46	14.86±2.91	130.0
Cora	GAT	81.7	83.18±0.44	1.8	17.77	31.80±2.12	79.0
Citeseer	GAT	70.7	72.68±0.75	2.8	35.13	44.30±2.88	26.1
Pubmed	GAT	76.6	72.21±6.60	-5.7	32.46	31.24±3.50	-3.8
A-computers	GAT	79.86	80.66±0.39	1.0	25.85	27.97±0.61	8.2
A-photos	GAT	89.53	90.59±0.60	1.2	21.53	21.96±1.14	2.0

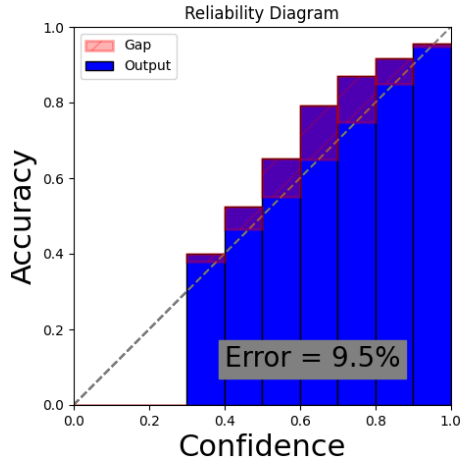
Table 4.2: KD of GCN and GAT on Cora, Citeseer, and Pubmed datasets.

Datasets	Teacher	Teacher acc	KD	Acc ↑ (%)	Teacher ECE	KD	ECE ↑ (%)
Cora	GCN	81.07±0.31	81.45±0.26	0.5	21.32±0.94	19.64±0.94	-7.9
Citeseer	GCN	71.17±0.6	71.33±0.45	0.2	21.32±1.16	21.11±1.03	-1.0
Pubmed	GCN	79.32±0.32	79.07±0.37	-0.3	6.73±0.71	8.38±0.75	24.5
Cora	GAT	82.99±0.45	83.2±0.39	0.3	15.11±0.72	19.27±0.49	27.5
Citeseer	GAT	70.7±0.28	71.55±0.38	1.2	17.36±0.39	26.22±0.7	51.0
Pubmed	GAT	78.2±0.39	78.14±0.15	-0.1	4.42±0.67	4.34±0.58	-1.8

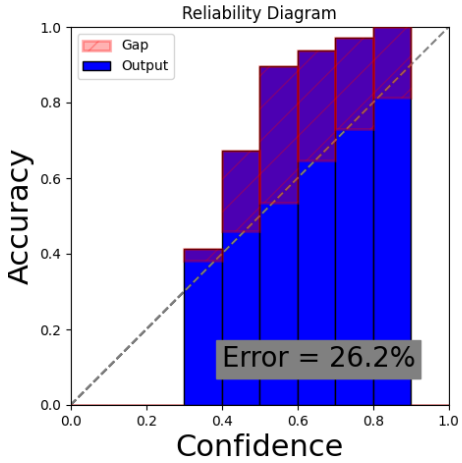
cases but at the cost of hurting ECE. The accuracy decrease of CPF on Pubmed is due to the hyperparameter is not stable enough. For other cases, CPF can obtain a clear increase of accuracy and ECE. If we further observe the confidence histograms, we will see that the knowledge distillation leads to a more underconfident student model, as shown in Fig. 4.5.

4.3.2 A close look at KD and student calibration performance

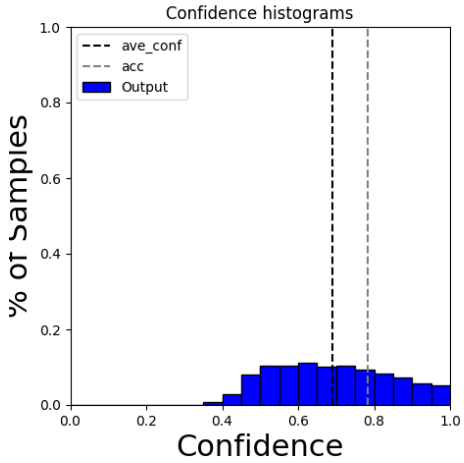
Yuan et al. [60] explores the relationship between KD and label smoothing and found that label smoothing (LS) [46] (LS was introduced in Section 2.3.2.) can be interpreted as ad-hoc KD, with teacher's probability distribution is uniform and $T = 1$. Therefore, KD should inherit the benefits of label smoothing, i.e. helping prevent models from over-confident, for nor-



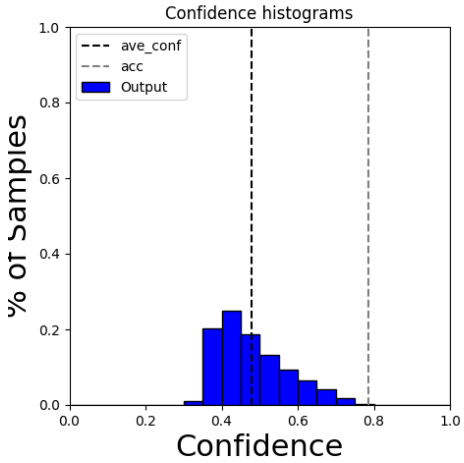
(a) CPF teacher, GCN on Pubmed



(b) CPF student, GCN on Pubmed



(c) CPF teacher, GCN on Pubmed



(d) CPF student, GCN on Pubmed

Figure 4.5: Confidence histograms and reliability diagrams for CPF. Take an example of GCN on Pubmed. Left: teacher. Right: student.

mal modern over-confident neural networks. But if the teacher model is under-confident, KD, or label smoothing, leads to a more under-confident result, i.e. our GNNs case.

Property 1. *Suppose the student model has the same architecture as the teacher, KD helps calibrate an original over-confident model, and miscalibrate an original under-confident model. In the original under-confident case, a minus teacher student loss ($\lambda < 0$ in Eq. 4.2) helps calibrate.*

This property shows that KD pushes down top-class scores, compared to the original teacher model. It could be understood by that students trained by LS-produced soft labels ($y_k^{LS} = y_k(1-\alpha) + \alpha/K$, where $\alpha \in (0,1)$) has an equivalent effect of assigning a smoothing score α/K to incorrect classes, and therefore push down the top-class score. KD's effects should be similar to and smaller than LS's, since KD produces "not that soft" labels. If the hard labels already leads the model to be under-confident, the after KD, it becomes more under-confident and more miscalibrated. Besides, KD also brings the domain knowledge, or to say, the incorrect class relationship, which should also helps student learn to calibrate. The above two reasons also explain the right figure in Fig. 4.4, that as λ increases, ECE goes up.

A simple example experiments can be viewed in the following. We choose an over-confident model (one-layer NN on Fashion-MNIST dataset) and a under-confident model (GCN on Pubmed). Here our KD experiments are conducted without hard labels, solely with teacher-produced soft labels, i.e. $\lambda = 1$ in Eq. 4.2. Results verify Property 1, in Table 4.3 and Fig. 4.6. After students' distillations, it becomes more miscalibrated for underconfident models and more calibrated for overconfident models.

Table 4.3: Two examples of KD's effects on over/under-confident models.

	Dataset	Model	Teacher ece	Student ece	Student's student ece
Overconfident	Fashion-MNIST	1layerNN	5.4	2.8	2.3
Underconfident	Pubmed	GCN	7.8	19.3	24.8

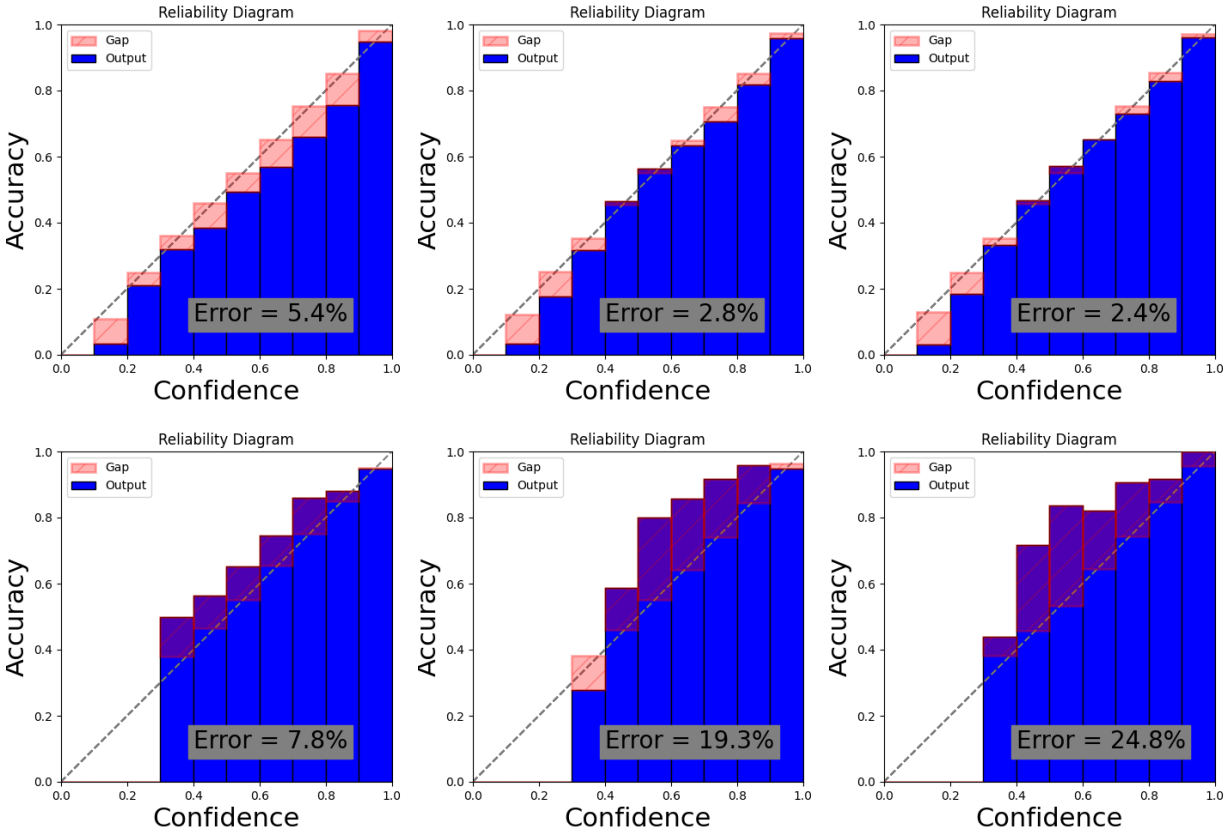


Figure 4.6: Two examples of KD's effects on over/under-confident models. Top: One-layer NN on Fashion-MNIST. Down: GCN on Pubmed. From left to right: teacher, student, student's student.

4.4 Conclusion

As stated in Property 1, KD helps calibrate in cases where the model exhibits overconfident and miscalibrate in underconfident cases.

Chapter 5

Graph signal processing for calibration

In Chapter 4, we tried to help calibrate from a common machine learning technique, KD. In this chapter, we would dive deeper into the theory of graph networks, from the perspective of graph signal processing. We first introduce graph signal processing and spectral graph models, then investigate whether these methods could be used to calibrate models.

5.1 Introduction

There have been close connections between graph signal processing and graph neural networks. From the first benchmark graph network work, GCN, people started to understand and improve graph networks from the perspective of graph signal processing. In this section, we will give a **detailed** introduction little by little. It covers the part of understanding graph signals, message passing, graph convolutions, over-smoothing, benchmark spectral GNNs, and low-pass filtering.

5.1.1 From time signals to graph signals

There is a connection between discrete time-varying signals with signals on a graph. Suppose the time-varying signals are represented as $f(t_0), f(t_1), ...f(t_{N-1})$, we view them on a cyclic chain graph, like in Fig. 5.1. Therefore $f[(t+1)_{mod\ N}]$ represent the next time signal of $f[t]$. The adjacency matrix and

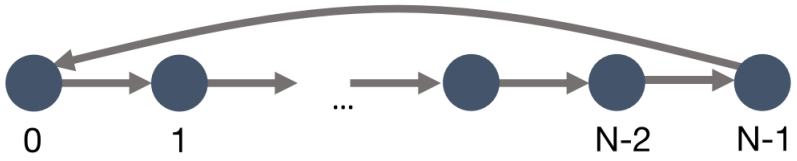


Figure 5.1: Representation of a (cyclic) time-series as a chain graph [1]

Laplacian of the one-dimension graph correspond to:

$$\begin{aligned} A_c[i,j] &= \begin{cases} 1 \; if \; i \; and \; j \; are \; neighbors \\ 0 \; otherwise, \end{cases} \\ L_c &= I - A_c. \end{aligned} \tag{5.1}$$

Therefore we can represent time shifts as multiplications by the adjacency matrix and represent the difference operation by the multiplication by the Laplacian:

$$\begin{aligned} (A_c f)[t] &= f[(t+1)_{mod \; N}] \\ (L_c f)[t] &= f[t] - f[(t+1)_{mod \; N}]. \end{aligned} \tag{5.2}$$

5.1.2 From convolution filter to message passing

We can see the connection between the adjacency and Laplacian of a graph and the notions of shifts and differences of a signal. Similarly, we can also represent convolution of a graph (represented by a convolution filter h) by matrix multiplication on the signal vector f:

$$\begin{aligned} (f \star h)(t) &= \sum_{\tau=1}^{N-1} f(\tau) h(\tau - t) \\ &= \mathbf{Q}_h \mathbf{f}, \end{aligned} \tag{5.3}$$

where $\mathbf{Q}_h \in \mathbb{R}^{N \times N}$ is the matrix version of convolution filter, and $\mathbf{f} = [f(t_0), f(t_1), ... \; f(t_{N-1})]^T \in \mathbb{R}^N$ is the vector representation of signal f. We can see that in order to satisfy the second equality of the above fomular, $\mathbf{Q}_h$ needs to satisfy translation equivariance, or to say, commute with adjacency matrix A_c:

$$A_c \mathbf{Q}_h = \mathbf{Q}_h A_c, \tag{5.4}$$

which is equivalent to:

$$\mathbf{Q}_h = \sum_{i=0}^{N-1} \alpha_i A_c^i, \tag{5.5}$$

where α_i is arbitary constant. Or for a more general graph we multiply the above equation by a node feature matrix $\mathbf{X}$, it becomes:

$$\mathbf{Q}_h\mathbf{X} = \sum_{i=0}^{N-1} \alpha_i A_c^i\mathbf{X}. \tag{5.6}$$

Obviously, this corresponds to the message passing formular in the basic GNNs, with one-order expansion:

$$\mathbf{Q}_h = \mathbf{I} + \mathbf{A}. \tag{5.7}$$

For a general graph, Laplacian $\mathbf{L = D - A}$. But in practice, people usually use symmetric normalized Laplacian $\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$ and symmetric normalized adjacency matrix $\mathbf{A}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ for two reasons. First, symmetric matrices have bounded eigenvalues. Secondly, these two symmetric matrices share the same eigenvectors. Since $\mathbf{L}_{sym} = \mathbf{I} - \mathbf{A}_{sym}$, we have $\mathbf{L}_{sym} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ and $\mathbf{L}_{sym} = \mathbf{U}(\mathbf{I} - \mathbf{\Lambda})\mathbf{U}^T$. In this regard, it makes (5.4), when we consider to let $\boldsymbol{Q}_h$ commute with whether adjacency matrix or Laplacian, no longer to be a trade-off question.

5.1.3 Representing graph convolutions by polynomials of Laplacian

Consider the eigendecomposition of the general graph Laplacian:

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \tag{5.8}$$

where the eigenvector $\boldsymbol{U}$ is defined as *graph Fourier modes*, and eigenvalues of the diagonal matrix $\mathbf{\Lambda}$ is defined as frequencies of the graph. Thus, for a signal $\mathbf{f} \in \mathbb{R}^{|\mathcal{V}|}$ defined on vertex of the graph, we have the Fourier transform and its inverse transform:

$$\begin{aligned} \mathbf{f}^{'} &= \mathbf{U}^T\mathbf{f} \\ \mathbf{f} &= \mathbf{U}\mathbf{f}^{'} \end{aligned} \tag{5.9}$$

With the defined Fourier modes, we can rewrite a graph convolution as:

$$\boldsymbol{f} \star \boldsymbol{h} = \boldsymbol{U}(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{f} \circ \boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{h}) = (\boldsymbol{U}\boldsymbol{diag}(\boldsymbol{\theta_h})\boldsymbol{U}^{\boldsymbol{T}})\boldsymbol{f}, \tag{5.10}$$

where $\mathbf{U}$ is the matrix of eigenvectors of Laplacian and $\circ$ denotes element-wise products. $diag(\theta_h)$ is a matrix with value θ_h on the diagonal, with $\theta_h = \mathbf{U}^T\mathbf{h} \in \mathbb{R}^{|\mathcal{V}|}$ called graph Fourier coefficient. θ_h is the only parameter here. To ensure that it corresponds to a meaningful convolution on the graph, a natural solution is to parameterize θ_h based on the eigenvalues Λ of the graph, such as approximating it as a degree N polynomial $p_N(\Lambda)$, therefore:

$$\begin{aligned} f \star h \approx (U p_N(\Lambda) U^T) f &= U(\sum_{i=0}^{N} \theta_i \Lambda^i) U^T f \\ &= p_N(L) f, \end{aligned} \tag{5.11}$$

where if we use a degree N, we ensure that the filtered signal at each node depends on its N-hop neighbors. Till now this is the key idea that graph convolutions are represented by polynomials of Laplacians. It reveals more general strategies for defining convolutions on graphs.

5.1.4 Understanding over-smoothing from graph signal processing

In section 2.1.3, we talk about the over-smoothing idea, a theorem to represent the influence of over-smoothing, and how to measure over-smoothing. In section 3.2.1, we empirically investigate the influence of over-smoothing to calibration when increasing GNNs' depth. Here we give some theoretical groundings from the eigenvalues and eigenvectors.

First, let's recall that the intuitive idea of over smoothing is that after too many rounds of message passing, the embedding of nodes becomes similar and uninformative.

Suppose we have a simply basic GNN like before:

$$\mathbf{H}^{(k)} = \mathbf{A}_{sym}\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}, \tag{5.12}$$

where we remove the non-linearity and self loops for simplicity. Then after K rounds updatings, we will end up with a $k-th$ power of adjacency matrix:

$$\mathbf{H}^{(K)} = \mathbf{A}_{sym}^{K}\mathbf{X}\mathbf{W}, \tag{5.13}$$

where $\mathbf{X}$ is the node feature matrix.
What does this mean? A higher order of K in $\mathbf{A}_{sym}^{K}$ emphasizes the largest eigenvalue of $\mathbf{A}_{sym}$. And since $L_{sym} + A_{sym} = I$, the largest eigenvalue of $\mathbf{A}_{sym}$ correspond to the smallest eigenvalues of $\mathbf{L}_{sym}$. Thus, multiplying a signal by high powers of A_{sym}, $\mathbf{A}_{sym}^{K}\mathbf{X}$, corresponds to a convolutional filter based on its lowest eigenvalues (or frequencies) of $\mathbf{L}_{sym}$. Therefore, we can interpret $\mathbf{A}_{sym}^{K}\mathbf{X}$ as a convolutional filter based on the lowest-frequency signals of graph Laplacian. A result of stacking many rounds of message passing leads to signals with a low-pass filter for many rounds, and in the worst case, these node representations *converge to constant values*. If we use adjacency matrix with self-loops, this issue can be alleviated but cannot be solved.

5.1.5 Understanding several benchmark graph networks from graph signal processing

We now analyze several benchmark graph networks and show that they are closely related to graph signal processing.

Graph Convolutional Networks (GCN) [16] employs an affine approximation (K = 1) of Eq. (5.11) with coefficients $\theta_0 = 2\theta$ and $\theta_1 = -\theta$ and attain the basic GCN convolution formular:

$$\begin{aligned} f \star h &\approx \theta(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{f} \\ &= \theta(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{f}, \end{aligned} \tag{5.14}$$

where the second step of the above equation is *renormalization trick*: $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \rightarrow \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$, with $\tilde{A} = A + I$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Or in a spatial perspective, a GCN layer is defined as:

$$\boldsymbol{H^k} = \boldsymbol{\sigma}(\boldsymbol{\tilde{A}H^{k-1}W^k}), \tag{5.15}$$

where $\boldsymbol{W^k}$ is a learnable parametric matrix.

Simple Graph Convolution (SGC) [24] hypothesize that the non-linearity between GCN layers is not critical - but that the majority of the benefit arises from the local averaging. Therefore, they remove the non-linearity function between each layer, with the resulting SGC layer represented as:

$$\boldsymbol{H^k = \sigma(\tilde{A}^k XW)}, \tag{5.16}$$

where $W = W^1W^2...W^k$ is a reparameterized weight matrix.

Graph filter neural network (gfNN) [25] takes advantage of the benefits of GCN and SGC, simplifying the non-linearity of GCN and at the same time adding a perceptron after SGC, in order to solve that SGC cannot tackle the issue if approximated signal $\tilde{A}^k X$ is non-separable. gfNN can be represented as:

$$\boldsymbol{H^k = \sigma(\sigma(\tilde{A}^k XW_1)W_2)}. \tag{5.17}$$

Spectral Graph Convolution (Cheb) [27] consider N polynomial of Λ, $p_N(\Lambda)$ in Eq. 5.11 as Chebyshev polynomials $T_k(\Lambda)$ up to the $K-th$ order:

$$\begin{aligned} p_N(\Lambda) &= \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) \\ T_0(\tilde{\Lambda}) &= 1 \\ T_1(\tilde{\Lambda}) &= \tilde{\Lambda} \\ T_k(\tilde{\Lambda}) &= 2\tilde{\Lambda}T_{k-1}(\tilde{\Lambda}) - T_{k-1}(\tilde{\Lambda}) \end{aligned} \tag{5.18}$$

where $\tilde{\Lambda} = \frac{1}{2\lambda_{max}}\Lambda - I_n$ is the scaled version of Λ and λ_{max} is the maximum eigenvalue of L.
The first order filter in (5.14) correspond to the propagation matrix:

$$\boldsymbol{S_{1-order} = I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}}. \tag{5.19}$$

Since the normalized Laplacian is $L_{sym} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, then

$$\boldsymbol{S_{1-order} = 2I - L_{sym}}. \tag{5.20}$$

Therefore, $N-th$ feature propagation $S^N_{1-order}$ implies the corresponding filter coefficients θ_{hi}, $i \in (0, |\mathcal{V}|)$ (see (5.10)):

$$\theta_{hi} = \theta_h(\lambda_i) = (2 - \lambda_i)^N, \tag{5.21}$$

where λ_i denotes the eigenvalues of L_{sym}, and $\lambda \in [0, 2]$. Obviously, high powers of $\boldsymbol{S_{1-order}}$ lead to exploding filter coefficients and over-amplify signals at frequencies $\lambda_i < 1$.
The above mentioned *renormalization trick* tackles this issue by using augmented normalized adjacency matrix $\tilde{S}_{adj}$ to replace $S_{1-order}$ (5.14):

$$\begin{aligned} \tilde{S}_{adj} &= (\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}) \\ &= I - \tilde{L}_{sym}, \end{aligned} \tag{5.22}$$

where $\tilde{L}_{sym} = I - \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the augmented normalized Laplacian. $N-th$ power filter coefficients θ_{hi} becomes:

$$\tilde{\theta}_{hi} = \tilde{\theta}_h(\tilde{\lambda}_i) = (1 - \tilde{\lambda}_i)^N. \tag{5.23}$$

Theorem 5.1. *[24] Given adjacency matrix A, degree matrix D, augmented adjacency matrix $\tilde{A} = A + \gamma I$, where $\gamma > 0$ and augmented degree matrix $\tilde{D}$. Let λ_1, λ_n and $\tilde{\lambda}_1$, $\tilde{\lambda}_n$ denote the smallest and largest eigenvalues of L_{sym} and $\tilde{L}_{sym}$. We have:*

$$0 = \lambda_1 = \tilde{\lambda}_1 < \tilde{\lambda}_n < \lambda_n, \tag{5.24}$$

This theorem means that by augmentation to graphs, it shrinks the spectrum (eigenvalues) of Laplacian.
Theorem 5.1.1 *[25] With the condition of Theorem 5.1, and let $\lambda_i(\gamma)$ be the i-th smallest generalized eigenvalue, we have $\lambda_i(\gamma)$ is a non-negative number and monotonically non-increasing in $\gamma \geq 0$. Moreover, $\lambda_i(\gamma)$ is strictly monotonically decreasing if $\lambda_i(0) \neq 0$.*
Theorem 5.1.1 is a stricter version of **Theorem 5.1**.

5.1.6 Theoretical analysis about low-pass filtering

First we show that graph networks' filtering has a *bias-variance trade-off* under the Assumption 1.
Assumption 5.1. *[25] In graph networks, observed features X consist of low-frequency true features $\hat{X}$ and high-frequency noise Z. $\hat{X}$ has a frequency at most $0 \leq \epsilon \ll 1$ and noise follows a Gaussian distribution. True features are informative enough for the machine learning task.*

Theorem 5.2. *[25] Under the Assumption 1, for any $0 < \delta < 1/2$, with probability at least $1 - \delta$, we have*

$$||\bar{X} - \hat{A}^k X||_D \leq \sqrt{k\epsilon}||\bar{X}||_D + O(\sqrt{log(1/\delta)R(2k)})\mathbb{E}[||Z||_D], \quad (5.25)$$

where $||X||_D$ is the norm induced by D and has a relationship with X's Fourier transform $\hat{X}$ by: $||X||_D = ||\hat{X}||_2$. $\hat{A}^k_{rw}$ is augmented random walk matrix, R(2k) is a probability that a random walk with a random initial vertex returns to the initial vertex after 2k steps.

Theorem 5.2 can be understood that the left-hand term is the difference between observed and true features. The first term in the right hand is the bias term that is proportional to the level of the length of random walk (which is a similar matrix to adjacency matrix) and the frequency of the true signal. The second term is the variance term, which decreases in $O(1/deg^{k/2})$ since for a small k, $R(2k)$ behaves like $O(1/deg^k)$, where deg is the typical degree of the graph. This theorem shows that by multiplying $k - th$ layer random walk matrix (or adjacency matrix), we can obtain a more accurate estimation of true features. By minimizing the right-hand side, we can also obtain the best choice of k.

5.1.7 Performance of filtering

From Theorem 5.2, we see that the low-pass filtered signal $\hat{A}^k_{rw}X$ could be seen as a relatively accurate estimate of true signal $\bar{X}$ in a high possibility. Then what is the performance of this operation?

More precisely, let $h_{MLP}(X|W_1, W_2)$ be a two-layer NN. As stated in Assumption 1, true features are informative enough for machine learning task. Therefore, we want to compare the difference between our trained $h_{MLP}(\hat{A}^k_{rw}X|W_1, W_2)$ and the true signal's perceptron $h_{MLP}(\bar{X}|W_1, W_2)$.

Theorem 5.3. *[25] Under Assumption 5.1, the outcomes of SGC, GCN, and gfNN are similar to those of the corresponding NNs using true features. More precisely, under the Assumption 5.1, by choosing a optimal k^*, with probability at least $1 - \delta$ for $\delta < 1/2$,*

$$||h_{MLP}(\bar{X}|W_1, W_2) - h_{MLP}(\hat{A}^k_{rw}X|W_1, W_2)||_D \leq \tilde{O}(\sqrt{\epsilon})\mathbb{E}[||Z||_D]\rho(W_1)\rho(W_2)$$
$$||h_{MLP}(\bar{X}|W_1, W_2) - h_{GCN}(X|W_1, W_2)||_D \leq \tilde{O}(\sqrt[4]{\epsilon})\mathbb{E}[||Z||_D]\rho(W_1)\rho(W_2), \quad (5.26)$$

where $\rho(W)$ is the maximum singular value of W. The first equation of (5.26) *tells us that if the maximum frequency ϵ is small enough, we can obtain an accurate enough performance of filtering. The second equation of* (5.26) *tells us that if ϵ is small enough, GCN could have a similar performance to MLP on the true signal. Similarly, SGC and gfNN both could have the same performance at that condition.*
Since spectral-based graph networks can be seen as a combination of filtering step and using a MLP step, till now, we analyze the performance of filtering and conclude that using a MLP to the filtered signal, machine learning task could gain high enough accuracy, and all spectral methods could gain high enough accuracy like using a MLP to the true signal.

5.2 Analysis on calibration

If we want to analyze calibration performance from the perspective of graph signal processing, a first question to ask is what is the influence of our eigenvalues λ_i of Laplacian here? Can we reconstruct a signal with low enough calibration performance with part of small eigenvalues? Will higher-than-one λ_i hurt the calibration performance? Our belief is that *by optically distilling eigenvalues, graph networks can obtain better accuracy but almost the same calibration performance as the original models.*

Lemma 5.1. *[38] $f : D_X \to \mathbb{R}^m$ is a function. q is an activation like softmax function. g is a post-processing calibration function. f is said to optimal with respect to calibration for a loss-function $L(\cdot, X, K)$ and activation q if*

$$\underset{g:\mathbb{R}^m \to \mathbb{R}^m}{\arg\min} L(q \circ g \circ f, X, K) = id, \tag{5.27}$$

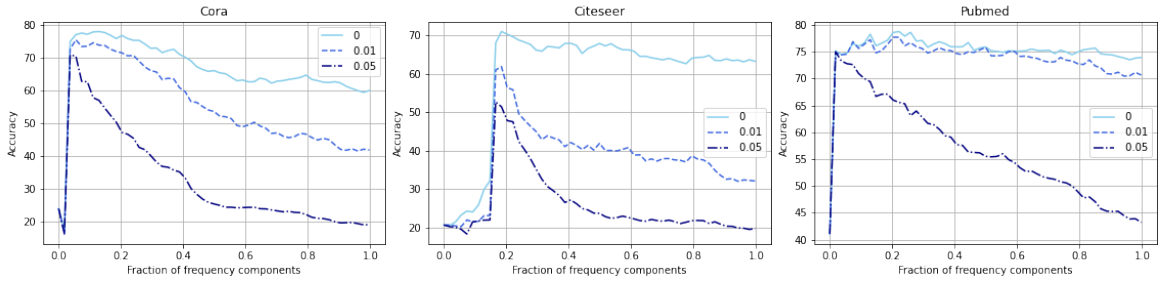
where $id : \mathbb{R}^m \to \mathbb{R}^m$ is the identity function.
Lemma 5.1 tells us that a model is perfectly calibrated if no post-processing methods could help calibrate anymore. From this perspective, we can think that the calibration information is hidden in the model. Post-processing methods' effect is to distill this information.

Claim 5.1. *Under Assumption 5.1, true signals cannot help obtain better calibration performance by improving hidden calibration information in graph networks.*

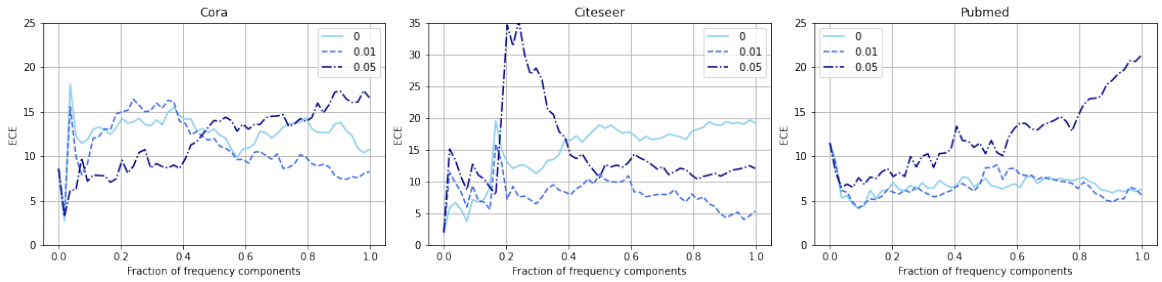
5.2.1 Empirical evidence of Claim 5.1

Experiment process. We first verify Claim 1 in citation datasets: Cora, Citeseer, and Pubmed. We follow the process of gfNN paper to compute k-frequency component [25]: 1. Compute the graph Fourier modes U from $\tilde{L}$; 2. Add Gaussian noise to the input features; 3. When we compute the first k-frequency components, there are two options to consider (this is different with gfNN's setting): $\hat{X}_{:k} = U[:k]^T \tilde{D}^{1/2} X$, or $\hat{X}_{:k} = U[:k]^T X$, where $U[:k]$ represents the first k rows of U; 4. Reconstruct features: $\tilde{X}_k = \tilde{D}^{-1/2} U[:k] \hat{X}_{:k}$, or $\tilde{X}_k = U[:k] \hat{X}_{:k}$; 5. Train with a simple two-layer MLP; 6. Distill the calibration information from the model using a post-processing method, we use non-parametric Histogram Binning and Temperature Scaling.

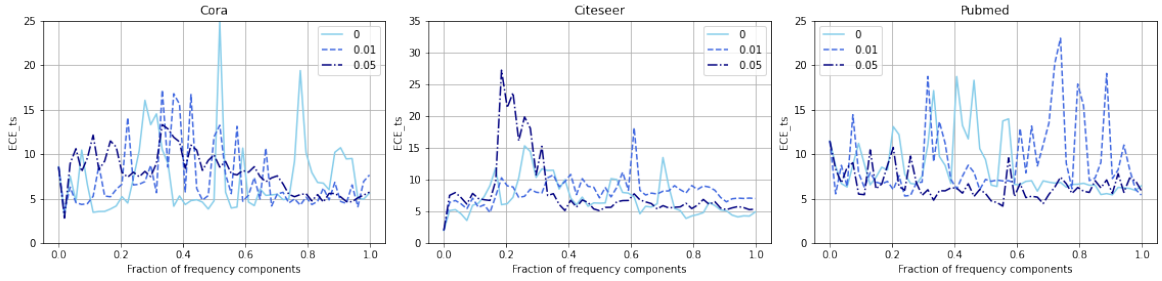
Experiment setting. The hidden layer of MLP is set to 32, with a RELU activation function. Optimizer is Adam with lr of 0.01 and weight decay of 5e-4. EPOCHS are set to 200 with early-stopping of patience 10. For Temperature Scaling, its learning rate is set to be 0.01. Each point in the plot is an average run of 10 times.



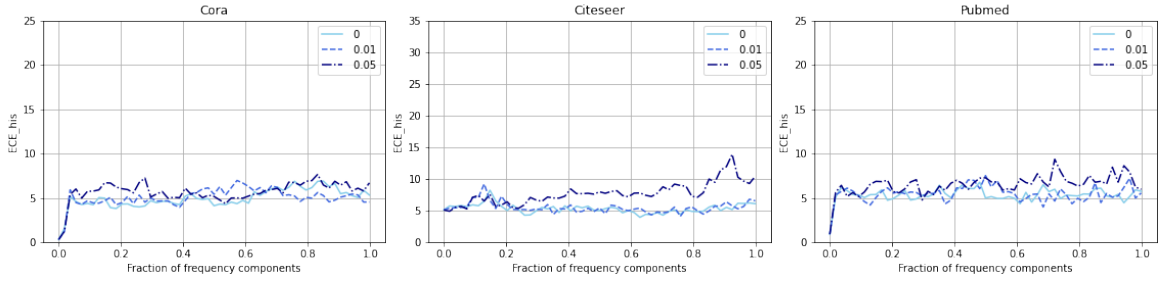
(b) Accuracy



(e) Uncal. ECE



(h) ECE after Temperature Scaling



(k) ECE after Histogram Binning

Figure 5.2: Model performance on citation datasets. From left to right is Cora, Citeseer and Pubmed respectively. From top to down is accuracy, uncalibrated ECE, ECE after Temperature Scaling and after Histogram Binning respectively. Gaussian noises are added in the level of $\{0, 0.01, 0.05\}$.

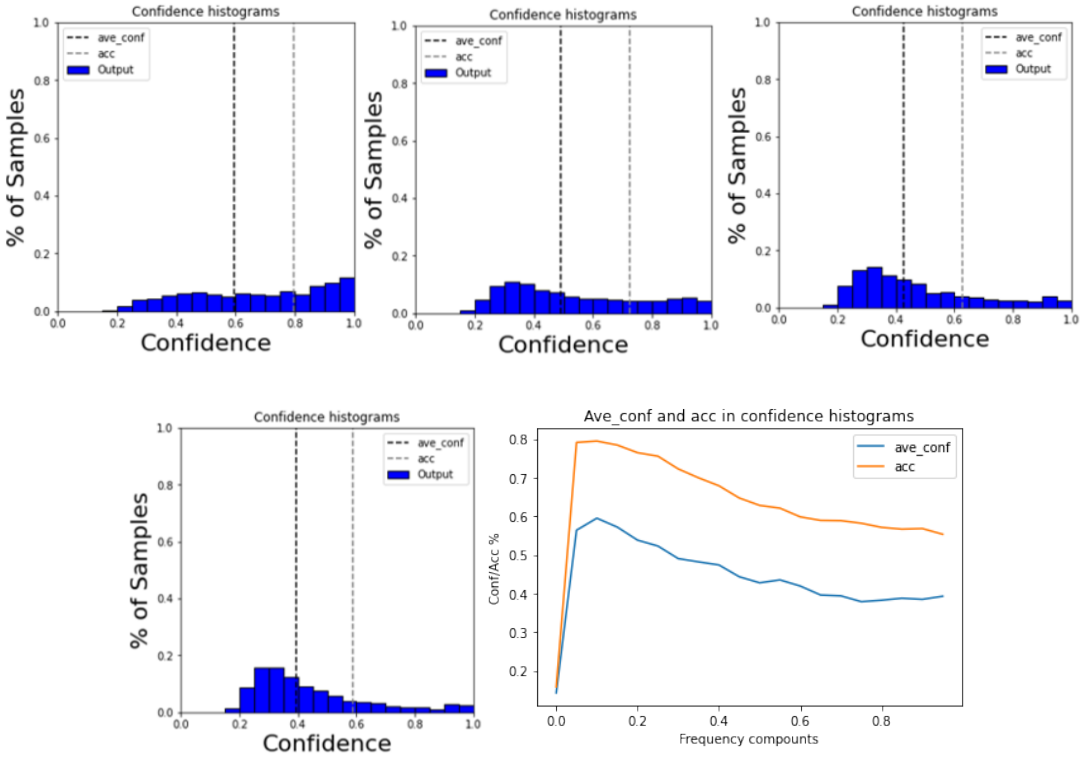


Figure 5.3: A little example of averaged confidence and accuracy on frequency components with a MLP on Cora dataset. The first four sub-figures correspond to fraction of frequency compounts = 0.1, 0.3, 0.5 and 0.7 respectively. The last sub-figure is the line of averaged confidence and accuracy. Both confidence and accuracy decrease with accuracy. This may explain why ECE becomes unstable and even increasing as the fraction of frequency components increases.

Results. We can see the following results from Fig. 5.2 and Fig. 5.3.
1. Inputting low-frequency signals could help improve model accuracy, especially in Cora and Citeseer datasets, as shown in Fig. 5.2. Fraction of high-frequency components signals performs like adding noise, in terms of accuracy. This is consistent with what we've analyzed in Section 5.1.
2. Low-frequency signals cannot help calibration. Uncalibrated calibration performance is not like accuracy, as shown in Fig. 5.2, which decreases with the fraction of frequency. There is no obvious empirical influence between them. From confidence histograms in Fig. 5.3 we can see that on a higher frequency, scores tend to shift to the left and assemble in small confidences, while this process does not alleviate the under-confidence.
3. Adding noise to signals makes uncalibrated ECE more unstable, in some

cases, i.e. 0.01-level noises on Citeseer dataset compared with no noise on Citeseer dataset, adding noises could even decrease ECE, as shown in Fig. 5.2. We interpret this as a result of the decline of accuracy and averaged confidence, as shown in Fig. 5.3. If the according accuracy decrease on frequency components, resulting averaged confidence would also decrease at the same pace. Therefore it is hard to determine ECE in that case. Adding noises to Citeseer dataset, accuracy decreases from about 65% (at 0.6 frequency) to 40% (0.01) and less than 30% (0.05), which dominates the influence to resulting ECE.

4. Post-processing calibration methods have denoising abilities. Non-parametric methods like Histogram Binning are more robust than parametric methods like Temperature Scaling. After Histogram Binning, almost all frequency components have a similar performance.

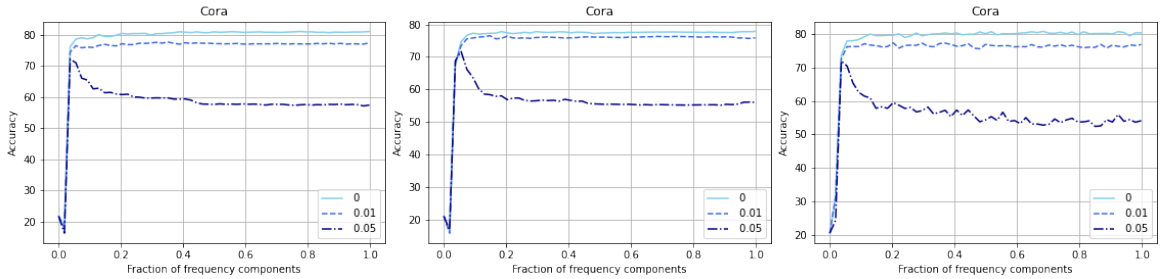
5. Main calibration information is not hidden in the datasets. Post-processing methods are simple but effective.

5.2.2 Spectral low-pass filtering calibration performance

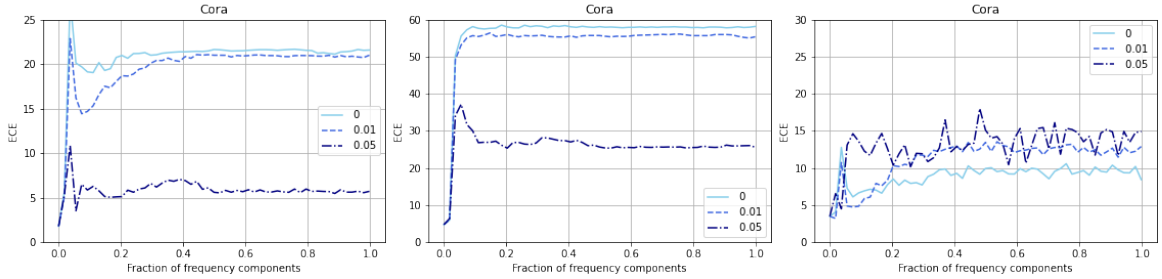
Above we see the influence of frequency components on a MLP model. How about spectral graph networks like SGC, GCN, and gfNN? We choose the most frequency-influential dataset Cora to run the experiment. Results are in Fig. 5.4. As we expected, spectral graph networks have an effect of low-pass filtering for both accuracy and calibration. But Temperature Scaling does not perform stably as expectedly.

5.3 Conclusion

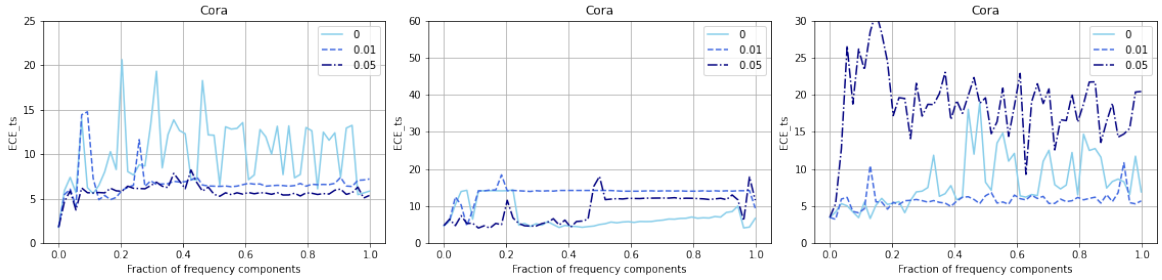
Although graph signal processing plays an important role in gaining meaningful information to classification tasks, it is not a dominative factor to model calibration performance. The nature of GNNs, i.e. low-pass filtering, could not increase the hidden calibration information in models. Viewing high-frequency components of signals as noises, is not suitable for calibration. They are not "real" noises for calibration. Besides, post-processing calibration methods have the ability to "denoise". Besides, spectral GNNs,



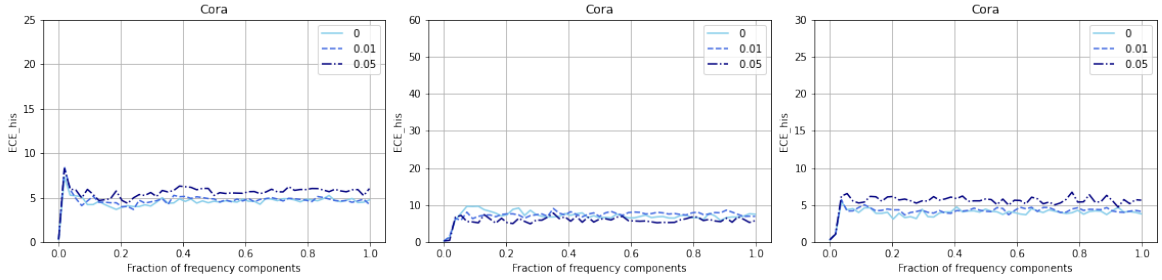
(b) Accuracy



(e) Uncal. ECE



(h) ECE after Temperature Scaling



(k) ECE after Histogram Binning

Figure 5.4: Model performance on Cora. From left to right is GCN, SGC and gfNN respectively. From top to down is accuracy, uncalibrated ECE, ECE after Temperature Scaling and after Histogram Binning respectively.

like GCN, have the effects of low-pass filtering for both accuracy and calibration.

Chapter 6

Adversarial calibration loss

Since the requirement of a calibrated network is different from the requirement to find the minimum of the cross-entropy loss function, it is hard to come up with a built-in method to satisfy the minima of two different loss functions at the same time, unless we replace the cross-entropy with a loss function that has both cross-entropy and calibration loss effects. Adding adversarial calibration error to the cross-entropy loss could be one possible solution.

6.1 Expected calibration loss

A first idea to come up with is to add ECE to the original cross entropy L_{CE}, with a parameter α:

$$\begin{aligned} L2_{ECE} &= \sqrt{\sum_{i=1}^{n}(acc(B_{m_i}) - conf(i))^2} \\ L &= \alpha L_{CE} + (1-\alpha)L_{ECE} = \alpha L_{CE} + (1-\alpha)L2_{ECE}, \end{aligned} \tag{6.1}$$

where $L2_{ECE}$ is our $L2$ norm ECE error.

More detailedly, we can set L_{ECE} to anneal with epochs, since the early epochs are usually used for reaching the cross entropy minimum. More

Table 6.1: (Uncalibrated performance) Accuracy, uncalibrated ECE, Marginal ECE, class-wise ECE (w/ std. deviation) for GCN and GAT on NLL loss and adding-calibration-loss versions.

		Acc	ECE	Marginal-ECE	clw-ECE
Cora	GCN	81.34±0.47	21.17±0.73	6.50±0.16	6.74±0.17
Cora	GCN+ece	**81.49±0.48**	**19.27±0.72**	**5.97±0.19**	**6.21±0.19**
Cora	GAT	83.26±0.41	15.19±0.54	5.08±0.20	5.45±0.22
Cora	GAT+ece	83.24±0.4	**13.15±0.6**	**4.53±0.2**	**4.9±0.2**
Citeseer	GCN	**71.24±0.65**	21.04±0.89	8.49±0.25	8.47±0.24
Citeseer	GCN+ece	70.95±1.15	**12.63±1.64**	**6.05±0.45**	**6.07±0.45**
Citeseer	GAT	70.9±0.56	17.00±0.55	7.61±0.27	7.58±0.24
Citeseer	GAT+ece	**71.0±0.66**	**6.68±0.94**	**4.45±0.51**	**4.56±0.47**
Pubmed	GCN	**79.17±0.46**	6.59±0.65	5.30±0.51	5.49±0.47
Pubmed	GCN+ece	78.68±0.44	**4.47±0.62**	**4.67±0.75**	**4.92±0.73**
Pubmed	GAT	78.22±0.41	4.46±0.76	4.70±0.75	4.88±0.71
Pubmed	GAT+ece	78.23±0.38	4.3±0.9	4.63±0.66	4.86±0.7

precisely,

$$\begin{aligned} anneal\_coef &= \min\left(1, \frac{epoch}{EPOCHS \cdot anneal\_max}\right) \cdot \lambda \\ L_{ECE} &= anneal\_coef \cdot L2_{ECE} \\ L &= \alpha L_{CE} + (1-\alpha) L_{ECE}, \end{aligned} \tag{6.2}$$

where *epoch* and $EPOCHS$ are the current training epoch and the total epochs number, $anneal\_max$ and λ being two parameters.

In experiment, we tune α, $anneal\_max$ and λ on validation set from $\{0.9, 0.95\}$, $\{5e-3, 1e-2, 1e-1, 1, 2, 5\}$ and $\{1e-2, 5, 10, 80, 160\}$. In Table 6.1 we list the results of GCN and GAT on citation datasets. In Table 6.2 we list the chosen hyperparameters. We fix two options, $\alpha = 0.95, anneal_max = 1e-2, \lambda = 5$ or $\alpha = 0.95, anneal_max = 2, \lambda = 160$ for cases except GAT on Pubmed. Experiments are run 10 times. We can see our adversarial calibration loss could improve model calibration in all cases. In some cases, it could also improve accuracy a bit, while in other cases, it is at the cost of accuracy.

There is also other options to add other calibration loss to the original loss, e.g. marginal calibration error, or KS-error, described in section 2.2.2. We believe these losses also could help calibrate.

Table 6.2: Hyperparameter choice.

Dataset	Model	alpha	max	lambda
Cora	GCN	0.95	1e-2	5
Cora	GAT	0.95	1e-2	5
Citeseer	GCN	0.95	2	160
Citeseer	GAT	0.95	2	160
Pubmed	GCN	0.95	2	160
Pubmed	GAT	0.95	1e-2	1e-2

6.2 Conclusion

By adding calibration loss to loss function, GNNs tend to be more calibrated and can improve accuracy in some cases, but also hurt accuracy a bit in others. Adding other calibration loss, e.g. marginal ece, KS-error, class-wise ece and so on, could also be options, which we believe would all help calibrate.

Chapter 7

Use case

This chapter is for our calibration's application on a use case, which is mainly a node classification task with GNNs. With short introduced background, we apply our calibration on use case. Then we set a higher demand suiting for the use case: to learn hidden compatibility information and analyze our results.

7.1 Introduction and motivation

Model calibration is an area relevant in various industrial applications. We apply a series of calibration state-of-the-art works on a Siemens Amesim project. Amesim is an extensive hydraulic and mechanical simulation platform.

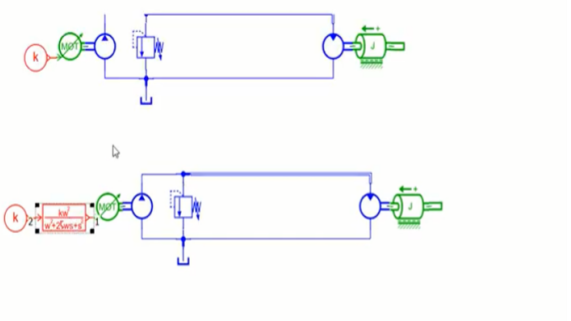


Figure 7.1: A screen shot from Amesim demo as a simulation example [8]. A pump connects to a motor and a relief valve and so on.

Our dataset is based on a project which uses GNNs for recommendation systems. Each mechanical device can be seen as a node in a graph, with

the relationship between every two devices seen as an edge between these two nodes. The aim of the project is to do a node classification, with scores in each class indicated the probability of being that kind of icon. Scores are used to recommend to customers to determine which device to connect between the target node and the source node according to the rankings.
Our aim is to help their project to provide a reasonable indicator to customers that indicates how confident our recommendation system model is.

7.2 Calibration evaluation

7.2.1 Methods and metrics

Graph models we used include GraphSage, GCN, and GAT.
Calibration methods include Isotonic Regression (IR), Histogram Binning (HB), Temperature Scaling (TS), Bayesian Binning into Quantiles (BBQ), Beta calibration (Beta), Meta calibration under the miscoverage rate constraint (Meta_mis), and under the coverage accuracy constraint (Meta_acc).
Calibration metrics include ECE, Brier scores, NLL, Marginal-ECE, Classwise-ECE, and Debiased-ECE, in order to provide a comprehensive comparison.
Models, methods, and metrics were introduced in previous chapters.

7.2.2 Experiment, results and discussion

Experiment setting. The total train data has about 55k small graphs, each with its own source node. The total test data has about 13k small graphs, with half-half split into real validation and test data, either with about 6.5k small graphs. All models are one layer. Hyperparameters are tuned on each models with learning rate = 0.01, 0.001, hidden number = 16, 32, 64, weight decay = 2e-7. Epochs are set to be 61, with early-stopping of patience of 30. Results are performed with 10 times averaged run. For calibration methods, ECE is estimated with 15 equal-width bins. Debiased ECE is estimated in an average of 30 runs. For meta calibration, miscoverage rate is set to be 0.05, the coverage accuracy is set to be 0.35. Run time evaluation is performed with one-time run.

Table 7.1: Calibration results for GraphSage

		Calibration						
Metrics	Uncal	IR	HB	TS	BBQ	Beta	Meta_mis	Meta_acc
ECE	12.76	2.31	**1.89**	2.31	2.54	2.35	4.26	3.68
CW-ece	0.0414	**0.013**	0.015	**0.013**	0.014	0.016	0.046	0.051
Max-ece	28.40	34.47	**24.70**	34.48	27.26	31.23	26.53	25.16
Brier	0.064	0.064	0.064	0.064	0.0645	0.064	0.064	0.064
NLL	3.77	6.76	3.42	6.76	3.00	**2.96**	3.82	4.12
Deb-ece	12.75	1.84	**1.37**	1.84	2.08	1.90	4.13	3.52
Marg-ece	0.036	0.014	**0.013**	0.014	0.02	0.02	0.048	0.053

Table 7.2: Calibration results for GCN

		Calibration						
Metrics	Uncal	IR	HB	TS	BBQ	Beta	Meta_mis	Meta_acc
ECE	7.43	1.94	**1.65**	2.52	2.71	2.2	3.87	2.04
CW-ece	0.036	**0.010**	0.011	0.035	**0.011**	0.014	0.040	0.067
Max-ece	26.63	**24.80**	79.16	42.00	48.80	36.18	56.67	56.55
Brier	0.0570	**0.0536**	0.0585	0.057	0.0572	0.0557	0.0574	0.0594
NLL	3.75	6.93	3.51	3.648	3.085	**3.155**	3.885	5.445
Deb-ece	7.38	1.37	**1.07**	2.11	2.29	1.73	3.64	1.94
Marg-ece	0.034	0.013	**0.011**	0.034	0.0180	0.019	0.044	0.069

Table 7.3: Calibration results for GAT

		Calibration						
Metrics	Uncal	IR	HB	TS	BBQ	Beta	Meta_mis	Meta_acc
ECE	6.53	**1.79**	2.37	2.22	2.02	2.00	4.18	2.49
CW-ece	0.0350	**0.0102**	0.0118	0.0344	**0.0102**	0.0134	0.0403	0.0641
Max-ece	48.43	**19.30**	63.00	59.05	85.21	34.10	56.72	57.89
Brier	0.0573	**0.0536**	0.0584	0.0569	0.05760	0.0555	0.0574	0.0590
NLL	3.70	8.07	3.89	3.60	3.16	**3.111**	3.87	5.26
Deb-ece	6.46	**1.19**	1.98	1.75	1.46	1.45	4.01	2.42
Marg-ece	0.033	0.013	**0.011**	0.034	0.018	0.018	0.045	0.067

Results and discussion. The results are shown in Table 7.1, 7.2, 7.3, 7.4 and 7.5. First, GNNs on Amesim dataset tend to be overconfident, like

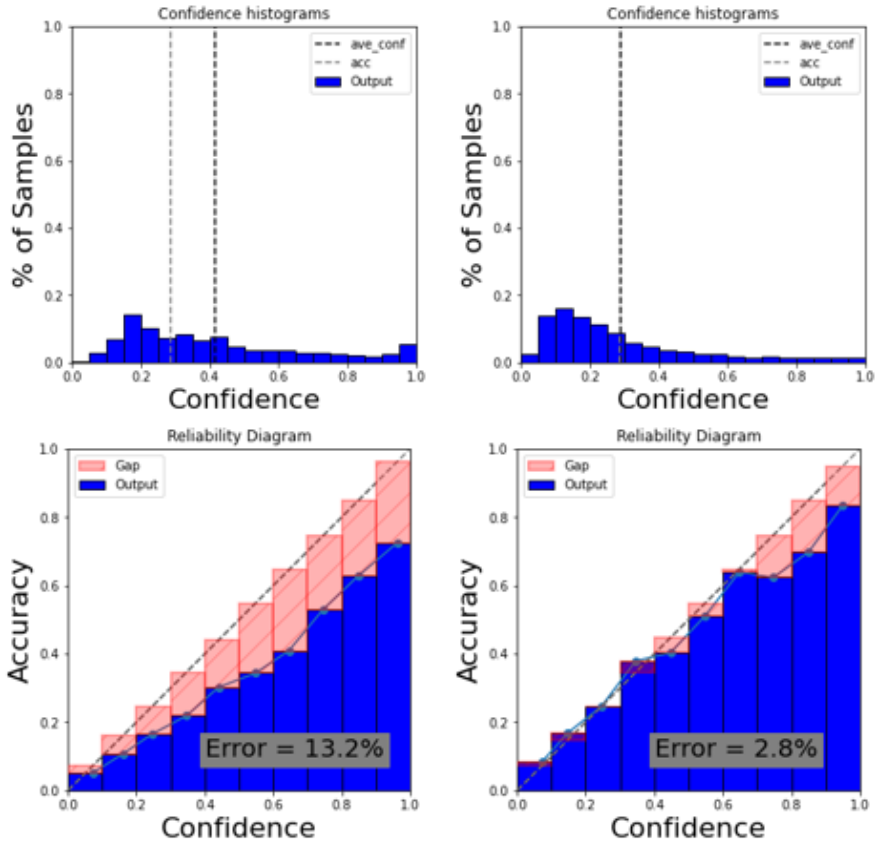


Figure 7.2: An example of Confidence Histograms and Reliability Diagram for Sage on Amesim. Left top: Confidence histograms for uncalibrated scores. Left down: Reliability Diagram for uncalibrated scores. Right top: Confidence histograms after beta calibration. Right down: Reliability Diagram after beta calibration

shown in Fig. 7.2. After calibration, this overconfidence gets effectively alleviated, with ECE decreasing from 13.2% to 2.8%. Secondly, Isotonic regression and Histogram binning perform the best for most of the metrics. Thirdly, when comparing among models, GraphSage has the best accuracy but the worst uncalibrated performance, while GAT is almost the inverse. This is consistent with observations in section 3.1.4 and [54] that attention could help calibrate. After calibration, all these three models can have almost the same calibration results with their best chosen calibration method. Finally, evaluating the running time of calibration methods, Meta calibration is the fastest one, while BBQ needs the most time. Besides, we believe that KD could help calibrate due to the overconfidence of Amesim.

Table 7.4: Model comparison in accuracy and ECE

Model	Acc	Uncal. ECE	Best cal. ECE
SAGE	**28.5 ± 0.26**	12.76±0.48	1.89±0.26
GCN	24.35 ± 0.32	7.43±0.47	1.65±0.37
GAT	24.42 ± 0.48	**6.53±0.73**	1.79±0.28

Table 7.5: Run time evaluation for calibration

	Calibration						
Metrics	IR	HB	TS	BBQ	Beta	Meta_mis	Meta_acc
time/s	41.3	41.1	46.6	660.7	70.3	33	24.6

7.3 Compatibility evaluation

Although calibration results are important, it is still not very straightforward to see what calibration corresponds to in reality. We choose another series of metrics concerning compatible items.

For each source icon, it has 1550 target icons to classify. But not all icons are possible to connect to all source nodes. For most cases, only about a half of icons can physically connect to the source icon. It can be evaluated in terms of port types. Each source or target node has its corresponding port types. Only when the port type of source node and target node are the

same, they can be connected. These physically possible connected target icons are called compatible icons. Therefore, for each test sample, we can have its corresponding compatible items, which is the true compatibility information. If we compare it to the rankings in our models, we will find that not all the top K (i.e. $K = 10$ or 100) rankings of the classes are compatible.
Therefore, we could (a). evaluate probability scores produced by our models using our defined compatibility metrics; (b). see if there are some components that should have been compatible but due to some reasons, they are wrongly classified as incompatible in the list.
In our case, we define a threshold for probability scores, with classes above the threshold classified as compatible and below as incompatible, and then compared to the true information to evaluate our scores.

7.3.1 Compatibility metrics

Mean ranks of all compatible items is defined as the averaged rank (probability scores ranked in descending) over all compatible items on test set:

$$r_{mean} = \frac{1}{MN} \sum_{j}^{M} \sum_{i}^{N} r_{ij}, \tag{7.1}$$

where r_{ij} is the rank for i-th sample and j-th class, N and M being the number of samples and classes.

Averaged compatible number in top K ranks is defined the number of compatible items in top K ranks:

$$\overline{N}_{comp}^{K} = \frac{1}{N} \sum_{i}^{N} n_{comp}^{i,K}, \tag{7.2}$$

where n_{comp}^{i} is the number of compatible items above the K-th rank for sample i.

Precision, Recall are widely used metrics in statistical learning. Precision is a measure of result relevancy, while recall is a measure of how many truly

relevant results are returned. In our case,

$$\begin{aligned} precision &= \frac{1}{N}\sum_{i}^{N}\frac{n_{comp}^{i,above}}{n^{i,above}} \\ recall &= \frac{1}{N}\sum_{i}^{N}\frac{n_{comp}^{i,above}}{n_{comp}^{i}}, \end{aligned} \tag{7.3}$$

where $n_{comp}^{i,above}$, $n^{i,above}$ and n_{comp}^{i} are respectively the number of compatible items above the threshold, the total number of items above the threshold and the number of compatible items for sample i.

7.3.2 Experiment and results

Like the experiment before, validation and test are split half and half, with about 6.5k test samples. Note, in the filtering step, data with the link type of "inConnectedWith" between source and target node are removed. Therefore, about 4.5k test samples are left. In these samples, there are about 1.8k samples with all target items are compatible, either due to that the source icon's port type is "signal", or all the target icon's port types are "signal". For effective comparison, *test samples are only chosen in the left about 2.8k samples since they are samples that would actually be affected by filtering.* For the calibration part, only methods that can change the order of probability scores are chosen. Each experiment is done one time. K is chosen to be 10 and 100.

Table 7.6: Compatible metrics for calibration

	Calibration					
Metrics	Uncal.	IR	HB	BBQ	Beta	Meta_mis
Comp ranks↓	774.3	764	**763.2**	765.7	765	773.1
Comp # in top 10↑	**9.3**	8.6	8.1	8.1	8.7	9.1
Comp # in top 100↑	**84.4**	74.7	71.9	79.6	77.4	84.1

Results and discussions. Table 7.6 shows the results of mean ranks and compatible number metrics. Compared to uncalibrated results, calibration

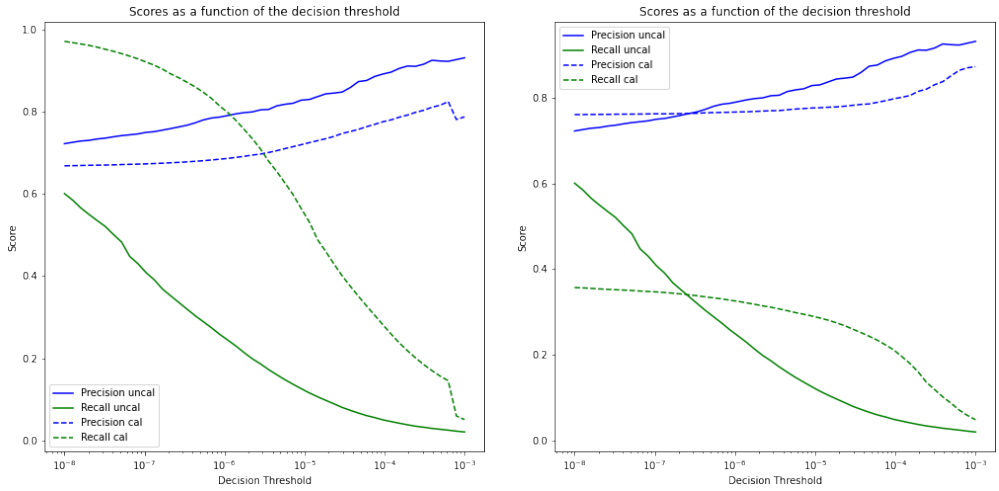


Figure 7.3: Precision and recall before and after calibration. Calibration method is chosen as Meta calibration (Left figure) and Beta calibration (Right figure) as an example. Solid line represents uncalibrated scores. Dashed line represents calibrated scores.

can push compatible items to rank forward on the whole, but in the sense of top 10 or 100 rankings, the compatible item number decreases.
Fig. 7.3 shows the scores of Precision and Recall under the different decision thresholds. First, we can learn a threshold with respect to the metric we are interested in. As the threshold increases, there are fewer items above the threshold, resulting in lower recall and higher precision. Secondly, different thresholds also work on different ranges.

7.4 Multi-class classification vs. binary setting

In a multi-class classification setting, the scores are equivalently input to a softmax function, where each class score interacts with the others. Besides, due to the variety of source nodes, it is hard to determine a threshold that is suitable for all samples. Alternatively, we choose to use the binary setting, where the scores are equivalently input to a sigmoid function and then compared with each binary label. This leads to a fair choice of threshold among different classes and samples.
The experiment is run for one time, with all parameters the same as multi-class classification setting. We choose one layer GraphSage model, with learning rate = 0.001, hidden number = 64, weight decay = 2e-7. We

trained our model for 61 epochs, with early-stopping of patience of 30.

Metrics	Multi-class	Binary
Acc↑	0.359	0.358
Mrr↑	0.460	0.460
Comp ranks↑	774.3	782.1
Comp # in top 10↑	9.3	9.3
Comp # in top 100↑	84.4	83.9

Table 7.7: Multi-class setting vs. binary setting.

Table 7.7 shows the results of binary setting, compared with multi-class classification setting. We can see that the binary setting can reach almost the same accuracy and mrr performance as the multi-class setting. For compatibility metrics, binary setting also performs very similarly to multi-class setting.
Fig. 7.4 shows the diagrams of scores distribution, confidence histograms, and reliability diagrams for these two setting. Binary setting has an effect of pushing the scores down a bit, resulting in the alleviation of overconfidence. Scores of different classes in binary setting do not have as strong interactions as multi-class setting, therefore tend to be closer to their ground truth probabilities.

7.5 Compatibility loss setting

Due to the motivation that we want our model to learn compatibility information directly from samples, we set the following two setting: setting masks on loss function and setting two objective loss functions.
The intuition of the first setting is that we want to push the incompatible scores down as much as possible. Therefore, we set the original positive as the only positive like before, and the original incompatible negative target items now are set as negatives, while the original compatible negative target items are set neither positive nor negative. Implementation is done by a mask on loss function. Note it is a different mask on each sample. This setting is implemented in both multi-class classification and binary setting.
The intuition of the second setting is that rather than push the incompatible

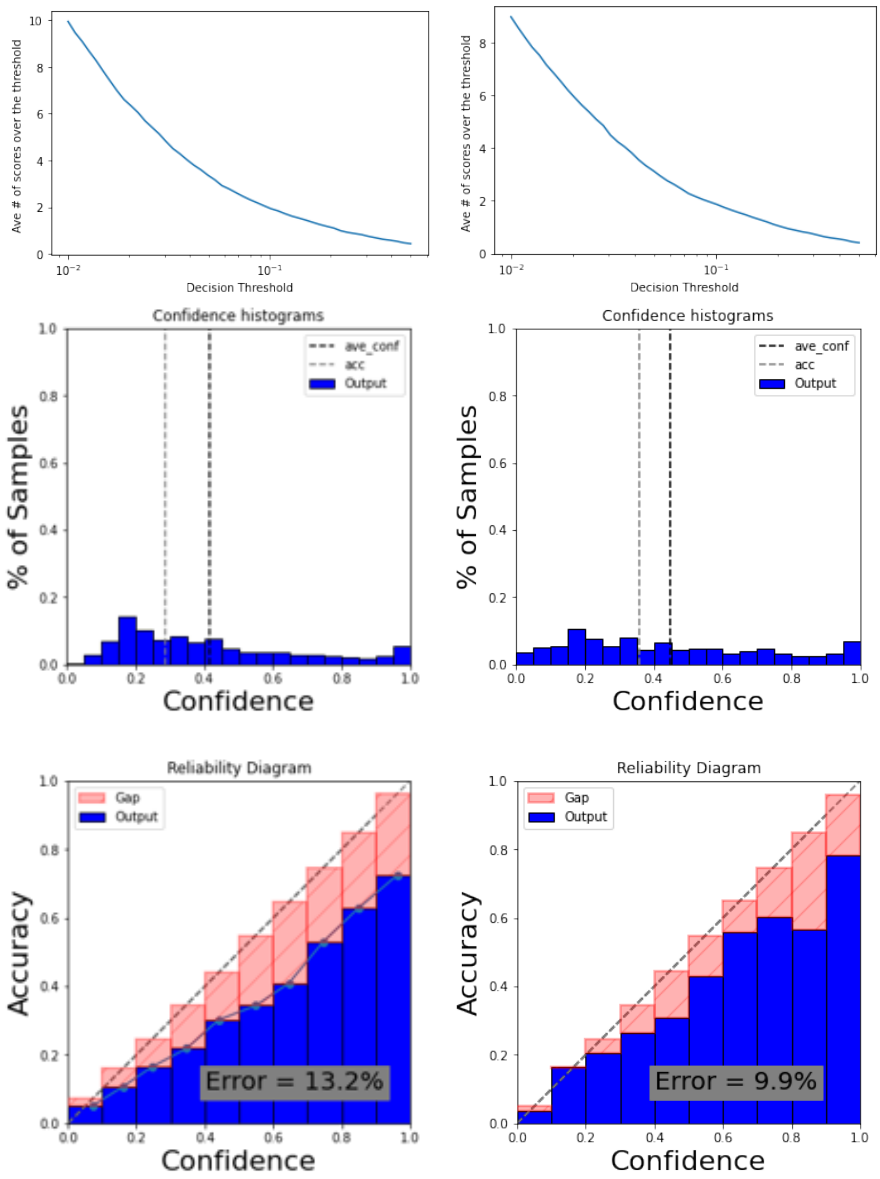


Figure 7.4: An example of all-class scores distribution (Top), Confidence Histograms (Middle) and Reliability Diagram (Down) for Sage on Amesim, for multi-class classification setting (Left) and binary setting (Right).

scores down, we choose to push the compatible scores up. Therefore, we set the original positive as 1, and the original incompatible negative as α, where α is a parameter, $\alpha \in (0, 1)$. This setting gives us more freedom by tuning a parameter. Note that this setting is only for binary setting.
Although we did not try this setting, this could also be a constructive proposition.

7.6 Conclusion

GNNs on Amesim datasets exhibit the overconfidence phenomenon. After using existing post processing calibration methods, the overconfidence could be effectively alleviated. Besides, we show that in terms of compatibility metrics, post processing calibration methods are helpful in some metrics but do not fix the problem. Further, the binary setting has the same accuracy performance as the multi-class classification setting, but could lead to a more calibrated model.

Chapter 8

Summary and Conclusion

In this work, we investigate the model calibration on Graph Neural Networks (GNNs). We focus our tasks on node classification.

First we empirically investigate the calibration performance of benchmark GNNs on benchmark datasets. We show that GNNs tend to be underconfident. Existing benchmark post-processing calibration methods could lead to well-calibrated results. Model properties, i.e. depth, width, aggregation and regularization, could lead to different calibration performances respectively. Besides, dataset balancing hardly affects calibration performance. We also give our contribution to a new-defined metric called balanced ECE to evaluate under the balance of class.

Next we evaluate the influence of Knowledge Distillation (KD) on GNNs to calibration. Although learning from more informative scores, KD aggravates the underconfidence, and therefore helps calibrate overconfident models and miscalibrate underconfident models. Our work is the first paper to state KD does not always help calibrate.

Then we focus on the influence of to calibration from the perspective of graph signal processing. The nature of GNNs, i.e. low-pass filtering, could not increase the hidden calibration information in models. Spectral methods like GCN, outperforms MLPs also because they have the low-pass filtering ability for both accuracy and calibration.

Then by adding calibration loss to loss function of GNNs, we gain better calibrated GNN models, with higher or lower accuracy at cost.

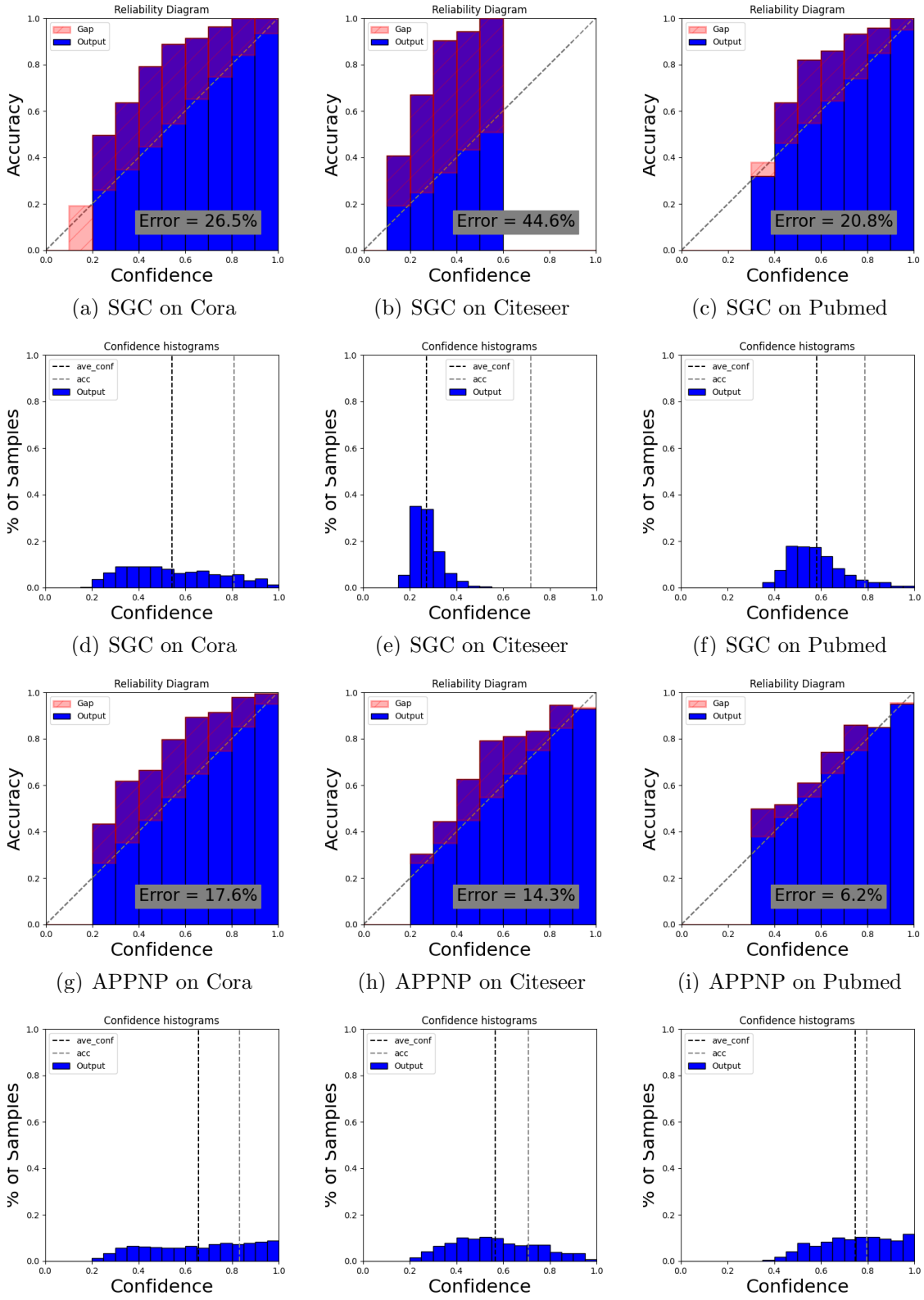
Finally, we apply our calibration on Amesim use case, where we show that GNNs on Amesim datasets express overconfident. After using existing post

processing calibration methods, the overconfidence could be effectively alleviated. With our defined compatibility metrics, post processing calibration methods are helpful in some metrics but do not fix the problem. Further, the binary setting could lead to a more calibrated model.
Future work could be done in Section 3.3 when considering more imbalanced datasets and evaluating whether such a case could help model calibration, and in Section 7.5 when utilizing compatibility loss setting and evaluating further results on Amesim. This could be an interesting direction to explore for an industrial purpose. In Section 6.1, there are also options to add other calibration errors, e.g. KS-error, marginal ece and so on.

Appendix A

Uncalibrated diagrams of the other models

Here we exhibit confidence histograms and reliability diagrams of SGC, APPNP, Cheb, and gfNN not shown in section 3.1.4. Figures are shown in Fig. A.1 and Fig. A.2.



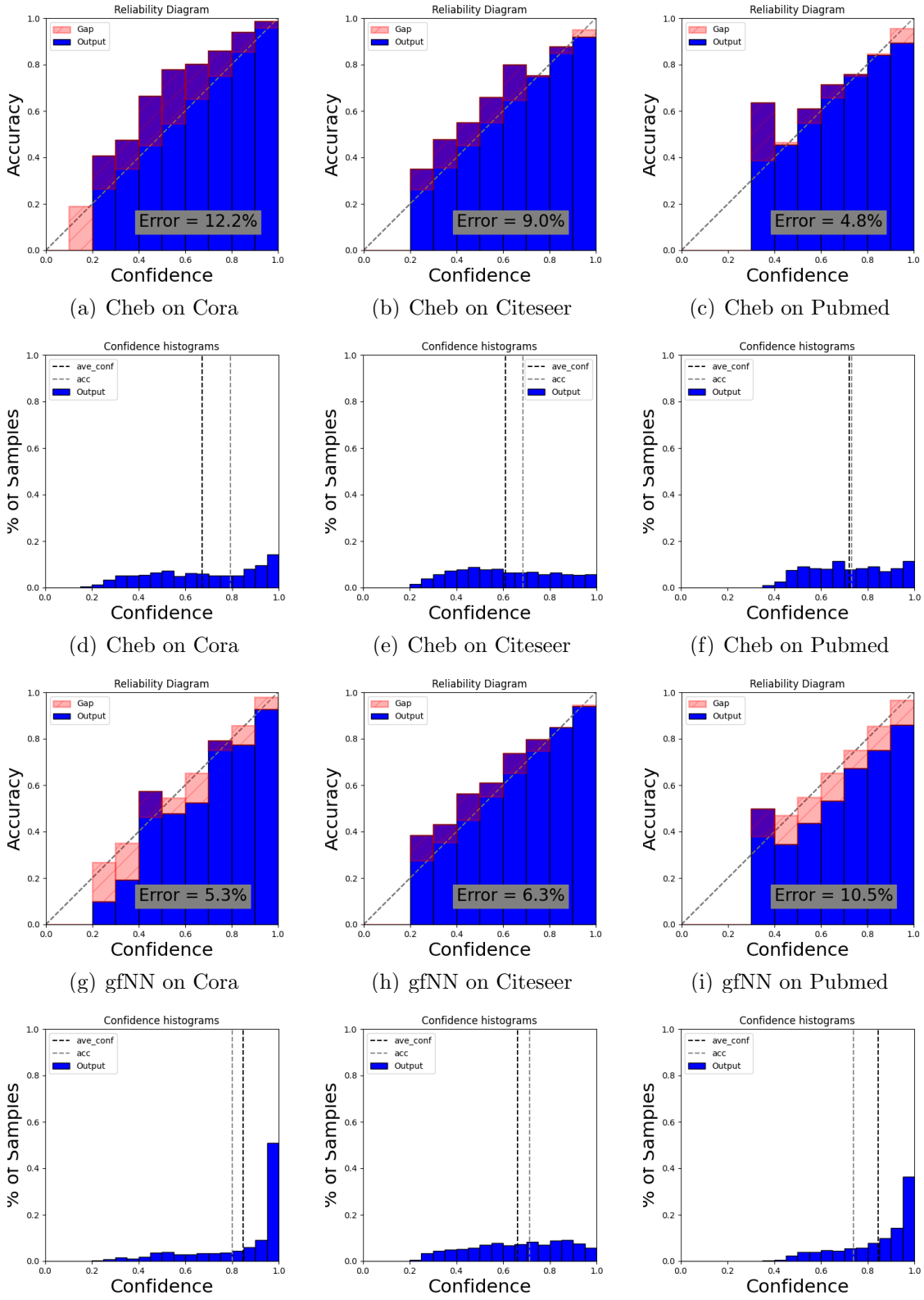
(a) SGC on Cora (b) SGC on Citeseer (c) SGC on Pubmed

(d) SGC on Cora (e) SGC on Citeseer (f) SGC on Pubmed

(g) APPNP on Cora (h) APPNP on Citeseer (i) APPNP on Pubmed

(j) APPNP on Cora (k) APPNP on Citeseer (l) APPNP on Pubmed

Figure A.1: Confidence histogram and Reliability Diagram of SGC and APPNP.



(a) Cheb on Cora (b) Cheb on Citeseer (c) Cheb on Pubmed

(d) Cheb on Cora (e) Cheb on Citeseer (f) Cheb on Pubmed

(g) gfNN on Cora (h) gfNN on Citeseer (i) gfNN on Pubmed

(j) gfNN on Cora (k) gfNN on Citeseer (l) gfNN on Pubmed

Figure A.2: Confidence histogram and Reliability Diagram of Cheb and gfNN.

Appendix B

Proof of Theorem 2.2

Theorem 2.2. [38] *Consider joint random variables (X, Z, K), taking values from $D_X, \mathbb{R}^m$ and κ respectively. Let $f : D_X \to \Delta^{n-1}$ be our NN, $L_1(f, X, K) = -E_{(x,k)\sim(X,K)} log(f_k(x))$ be the first loss given our model f and input X, $L_2(q, Z, K) = -E_{(z,k)\sim(Z,K)} log(q_k(z)) = -E_{(x,k)\sim(X,K)} log(q_k(f(x)))$ be the second loss given our output Z, or equivalently given f and X.*
A strong version: if

$$f = \underset{f':D_X\to\Delta^{n-1}}{argmin}\ L_1(f', X, K), \tag{B.1}$$

then it is perfect calibrated with respect to f, i.e. $P(k|x) = f_k(x)$.
A weak version: if

$$\underset{g:\mathbb{R}^m\to\mathbb{R}^m}{argmin}\, L_2(q \circ g, Z, K) = id, \tag{B.2}$$

where id is the identify function, $g : \mathbb{R}^m \to \mathbb{R}^m$ is a modification, e.g. a post-processing calibration function, then it is perfect calibrated with respect to g, i.e. $P(k|z) = q_k(z)$.
An equivalently weak version: if

$$\underset{g:\mathbb{R}^m\to\mathbb{R}^m}{argmin}\, L_2(q \circ g \circ f, X, K) = id, \tag{B.3}$$

then it is perfect calibrated with respect to g, i.e. $P(k|z) = q_k(z)$, or to say, f is perfect calibrated.
Proof. This theorem defines the condition of calibration for a model. It gives different conditions to be calibrated for a model: a strong version, a

weak version, and a third equivalent version. The strong version is the ideal calibration condition. It is more like a definition. Here we proof the weak version. Then the equivalent version can be simply derived from the weak version.

The weak version states that f is optimal to calibration if replacing f by $g \circ f$, i.e. using a post-processing calibration method g, can not decrease the L_2 loss. Therefore, we want to optimize $L_2(q \circ g, Z, K)$ over g.

The Euler–Lagrange equations in physics states that given the Lagrangian $L = L(t, q, \dot{q})$ of a system, where q is a generalized coordinate, $\dot{q} = \frac{\partial q}{\partial t}$ is a generalized momentum, the following equation holds:

$$\frac{\partial L}{\partial q^i}(t, \boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}^i}(t, \boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) = 0, \quad i = 1, \ldots, n. \tag{B.4}$$

In our case,

$$\begin{aligned} -L_2(q \circ g, Z, K) &= \underset{(z,k)\sim(Z,K)}{E} log(q_k(g(z))) \\ &= \int \sum_{k=1}^{n} p(z,k) \log\left(q_k(g(z))\right) dz \\ &= \int L(z, g, \dot{g}) dz, \end{aligned} \tag{B.5}$$

where $L = L(z, g, \dot{g}) = \sum_{k=1}^{n} p(z,k) \log\left(q_k(g(z))\right)$.

The Euler–Lagrange equations become:

$$\frac{\partial L}{\partial g} = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{g}}. \tag{B.6}$$

$RHS = 0$ since L has no relation to $\dot{g}$. Then since L is optimized when $g = id$, LHS becomes:

$$\begin{aligned} LHS &= \frac{\partial}{\partial g} \sum_{k=1}^{n} p(z,k) \log\left(q_k(g(z))\right) \\ &= \sum_{k=1}^{n} p(z,k) \frac{q_k^{'}(\cdot)}{q_k(g(z))} \\ &\overset{g=id}{=} \sum_{k=1}^{n} p(z,k) \frac{q_k^{'}(z)}{q_k(z)} = 0 \end{aligned} \tag{B.7}$$

Since the derivative $q_k^{'}(z)$ is over any class j, we can rewrite it as:

$$\sum_{k=1}^{n} \frac{p(z,k)}{q_k(z)} \frac{\partial q_k}{\partial z_j} = 0 \tag{B.8}$$

Since for $q : \mathbb{R}^m \to \Delta^{n-1}$, we have $\sum_{k=1}^{n} q_k(z) = 1$. Taking partial derivative we have $\sum_{k=1}^{n} \frac{\partial q_k}{\partial z_j} = 0$. Compared with the above equation, we imply:

$$\frac{p(z,k)}{q_k(z)} = constant, \quad for\ any\ k. \tag{B.9}$$

Given $\sum_{k=1}^{n} p(z,k) = c \cdot \sum_{k=1}^{n} q_k(z) = c$, and $\sum_{k=1}^{n} p(z,k) = p(z)$, we have $p(z) = c$. Therefore, $p(k|z) = p(z,k)/p(z) = q_k(z)$. This is the statement of the weak version.

For the third equivalent version, the only difference is that in the loss L_2, the $q \circ g$ becomes $q \circ g \circ f$, and given Z becomes given X. It can be simply derived in the following lemma.

Lemma B.1. [38] Consider joint random variables (X, Z), taking values from D_X and $\mathbb{R}^m$ respectively. Let $f : D_X \to D_Z$ be our model, satisfying $Z = f(X)$. Let $L = L(f(x))$ be any function. We have:

$$\underset{x \sim X}{E} L(f(x)) = \underset{z \sim Z}{E} L(z). \tag{B.10}$$

Bibliography

[1] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[2] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[3] Y. Liu, M. Hildebrandt, M. Joblin, M. Ringsquandl, R. Raissouni, and V. Tresp, "Neural multi-hop reasoning with logical rules on biomedical knowledge graphs," in *European Semantic Web Conference*. Springer, 2021, pp. 375–391.

[4] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics," *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 021001, 2020.

[5] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[6] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4320–4328.

[7] H. Bagherinezhad, M. Horton, M. Rastegari, and A. Farhadi, "Label refinery: Improving imagenet classification through label progression," *arXiv preprint arXiv:1805.02641*, 2018.

[8] Simcenter amesim demo. [Online]. Available: https://www.youtube.com/watch?v=4ja0HkBTxUc&t=94s

[9] A. D. Becke, "Perspective: Fifty years of density-functional theory in chemical physics," *The Journal of chemical physics*, vol. 140, no. 18, p. 18A301, 2014.

[10] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions," *arXiv preprint arXiv:1706.08566*, 2017.

[11] J. Klicpera, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," *arXiv preprint arXiv:2003.03123*, 2020.

[12] Z. Savitsky. The next big thing: the use of graph neural networks to discover particles. [Online]. Available: https://news.fnal.gov/2020/09/the-next-big-thing-the-use-of-graph-neural-networks-to-discover-particles/

[13] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1321–1330.

[14] P. Tabacof and L. Costabello, "Probability calibration for knowledge graph embedding models," *arXiv preprint arXiv:1912.10000*, 2019.

[15] L. Teixeira, B. Jalaian, and B. Ribeiro, "Are graph neural networks miscalibrated?" *arXiv preprint arXiv:1905.02296*, 2019.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[18] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in

Advances in neural information processing systems, 2017, pp. 5998–6008.

[20] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5453–5462.

[21] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3438–3445.

[22] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[23] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[24] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[25] H. Nt and T. Maehara, "Revisiting graph neural networks: All we have is low-pass filters," *arXiv preprint arXiv:1905.09550*, 2019.

[26] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[27] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in neural information processing systems*, vol. 29, pp. 3844–3852, 2016.

[28] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[29] J. Yang, B. Ribeiro, and J. Neville, "Stochastic gradient descent for relational logistic regression via partial network crawls," *arXiv preprint arXiv:1707.07716*, 2017.

[30] M. H. DeGroot and S. E. Fienberg, "The comparison and evaluation of forecasters," *Journal of the Royal Statistical Society: Series D (The Statistician)*, vol. 32, no. 1-2, pp. 12–22, 1983.

[31] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 625–632.

[32] M. P. Naeini, G. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using bayesian binning," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[33] K. Gupta, A. Rahimi, T. Ajanthan, T. Mensink, C. Sminchisescu, and R. Hartley, "Calibration of neural networks using splines," *arXiv preprint arXiv:2006.12800*, 2020.

[34] A. Kumar, P. Liang, and T. Ma, "Verified uncertainty calibration," *arXiv preprint arXiv:1909.10155*, 2019.

[35] M. Kull, M. Perello-Nieto, M. Kängsepp, H. Song, P. Flach *et al.*, "Beyond temperature scaling: Obtaining well-calibrated multiclass probabilities with dirichlet calibration," *arXiv preprint arXiv:1910.12656*, 2019.

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[37] G. W. Brier *et al.*, "Verification of forecasts expressed in terms of probability," *Monthly weather review*, vol. 78, no. 1, pp. 1–3, 1950.

[38] A. Rahimi, K. Gupta, T. Ajanthan, T. Mensink, C. Sminchisescu, and R. Hartley, "Post-hoc calibration of neural networks," *arXiv preprint arXiv:2006.12807*, 2020.

[39] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[40] B. Zadrozny and C. Elkan, "Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers," in *Icml*, vol. 1. Citeseer, 2001, pp. 609–616.

[41] ——, "Transforming classifier scores into accurate multiclass probability estimates," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 694–699.

[42] J. Wenger, H. Kjellström, and R. Triebel, "Non-parametric calibration for classification," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 178–190.

[43] M. Kull, T. Silva Filho, and P. Flach, "Beta calibration: a well-founded and easily implemented improvement on logistic calibration for binary classifiers," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 623–631.

[44] J. Mukhoti, V. Kulharia, A. Sanyal, S. Golodetz, P. H. Torr, and P. K. Dokania, "Calibrating deep neural networks using focal loss," *arXiv preprint arXiv:2002.09437*, 2020.

[45] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, and G. Hinton, "Regularizing neural networks by penalizing confident output distributions," *arXiv preprint arXiv:1701.06548*, 2017.

[46] R. Müller, S. Kornblith, and G. Hinton, "When does label smoothing help?" *arXiv preprint arXiv:1906.02629*, 2019.

[47] S. Thulasidasan, G. Chennupati, J. Bilmes, T. Bhattacharya, and S. Michalak, "On mixup training: Improved calibration and predictive uncertainty for deep neural networks," *arXiv preprint arXiv:1905.11001*, 2019.

[48] C. Tomani and F. Buettner, "Towards trustworthy predictions from deep neural networks with fast adversarial calibration," *arXiv preprint arXiv:2012.10923*, 2020.

[49] S. Yun, J. Park, K. Lee, and J. Shin, "Regularizing class-wise predictions via self-knowledge distillation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 13 876–13 885.

[50] C. Xing, S. Arik, Z. Zhang, and T. Pfister, "Distance-based learning from errors for confidence calibration," *arXiv preprint arXiv:1912.01730*, 2019.

[51] A. Rahimi, A. Shaban, C.-A. Cheng, R. Hartley, and B. Boots, "Intra order-preserving functions for calibration of multi-class neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 456–13 467, 2020.

[52] J. Zhang, B. Kailkhura, and T. Y.-J. Han, "Mix-n-match: Ensemble and compositional methods for uncertainty calibration in deep learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 11 117–11 128.

[53] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[54] S. Desai and G. Durrett, "Calibration of pre-trained transformers," *arXiv preprint arXiv:2003.07892*, 2020.

[55] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[56] Y. Bai, T. Ma, H. Wang, and C. Xiong, "Improved uncertainty post-calibration via rank preserving transforms," 2020.

[57] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[58] Y. Yang, J. Qiu, M. Song, D. Tao, and X. Wang, "Distilling knowledge from graph convolutional networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7074–7083.

[59] C. Yang, J. Liu, and C. Shi, "Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework," in *Proceedings of the Web Conference 2021*, 2021, pp. 1227–1237.

[60] L. Yuan, F. E. Tay, G. Li, T. Wang, and J. Feng, "Revisiting knowledge distillation via label smoothing regularization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3903–3911.

[61] H. Li, "Exploring knowledge distillation of deep neural."