Beyond the Additive Nodes' Convolutions: a Study on High-Order Multiplicative Integration

Paolo Frazzetto
University of Padova
Padua, Italy
paolo.frazzetto@phd.unipd.it

Luca Pasa
University of Padova
Padua, Italy
luca.pasa@unipd.it

Nicolò Navarin
University of Padova
Padua, Italy
nicolo.navarin@unipd.it

Alessandro Sperduti
University of Padova
Padua, Italy
DISI, University of Trento
Trento, Italy
alessandro.sperduti@unipd.it

ABSTRACT

Graph Convolutional Neural Networks (GCNs) compute representations of graph nodes by exploiting convolution operators based on some neighborhood aggregation scheme. These operators are defined by using several stacked Graph Convolutional (GC) layers. They are usually defined as additive building blocks that fuse multiple information streams. However, when considering information integration in sequences, the flow of gradient has been shown to be more robust by adopting the Multiplicative Integration (MI) technique. Because of that, it is worth investigating the impact of MI in Graph Neural Networks. We propose three different GC layers that exploit MI to improve various aspects of the neighborhood aggregation scheme. We report both a theoretical and empirical comparison of our proposals with respect to the most common GC operators for the graph classification task.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

Graph Neural Network, Multiplicative Integration, Structured Data, Nodes Aggregation, Graph Classification

ACM Reference Format:
Paolo Frazzetto, Luca Pasa, Nicolò Navarin, and Alessandro Sperduti. 2024. Beyond the Additive Nodes' Convolutions: a Study on High-Order Multiplicative Integration. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3605098.3636016

1 INTRODUCTION

Recently, there has been an increased interest in machine learning models that deal with graph-structured data, such as kernel methods and neural networks. Graph Neural Networks (GNNs) define a neural architecture that follows the graph topology. From the neurons associated with a vertex and its neighbors, a hidden representation corresponds to the same vertex in another network layer. For every hidden layer of the GNN, a new transformation is performed. The transformations are determined by relying on the definition of a convolution operator in the graph domain. Graph Convolutions (GC) are generally based on a neighborhood aggregation scheme (*aggregate*) [8] considering, for each node, only its direct neighbors, and a *combine* operation that merges the representation of a node with the aggregation of its neighbors. The neighborhood aggregation schemes implemented by the various GCs proposed in literature usually exploit an *additive building block* [36].

In the structured domain, particularly in sequential learning, a different procedure of information integration has been studied: the Multiplicative Integration (MI) [36]. The idea is that, instead of utilizing the sum operation to join the information conveyed by the various elements that compose the recurrent model equation, MI exploits the Hadamard product. Without introducing any extra parameters, the authors leverage *second-order* interactions between features, i.e., relationships or dependencies that exist *between pairs of features* within the dataset. Unlike first-order interactions, which involve individual features in isolation, second-order interactions consider how two features jointly influence the output or prediction. One of the first applications of MI on sequential domains was proposed by Goudreau *et. al* [9] that introduced the Second-Order Single-Layer Recurrent Neural Networks (Second-order SLRNN).

The most common model for sequences that adopts the MI is the LSTM [14] (or variants of it, e.g. GRU [3]). This model employs the MI to implement a gating mechanism to manage long-term temporal dependencies. An enhanced version of the LSTM (and earlier, of the RNN) that exploits the MI also to define the recurrent mechanism is the multiplicative LSTM/RNN [23, 32]. These models use the Hadamard product to combine the projection of the current time step with the hidden states that come from the previous time step. The idea of using MI to manage a gating mechanism and to combine the information flow from different temporal domains is also used in many other models, like in Highway Network [39]. Another model that exploits a similar technique is the HyperNetwork [11]. The HyperNetwork dynamically generates the weights of a network

This work is licensed under a Creative Commons Attribution International 4.0 License.

SAC '24, April 8–12, 2024, Avila, Spain

© 2024 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0243-3/24/04.
https://doi.org/10.1145/3605098.3636016

using another (smaller) network. In particular, the recurrent version (the HyperLSTM) generates a multiplicative bias that drives the generation of dynamical weights. A similar approach that belongs to the Bayesian framework was proposed by Krugers *et al.* [24].

In Graph Neural Networks, GC's aggregation/combination operation shares some critical mechanisms with the time-based aggregation mechanism used by Recurrent Neural Networks. Indeed, in literature, the Multiplicative integration is shown to be particularly convenient to aggregate contextual information that comes from different sources[19]. Inspired by this similarity, in this paper, we explore how MI can be applied to define novel GCs.

Some recently proposed GNN approaches do exploit some form of multiplicative mechanisms, for instance implementing gating mechanisms [2, 29, 33, 34] or hypernetwork-like models [2, 12, 28]. However, they all do not adopt the MI paradigm concerning nodes' neighborhoods, and comparing such models with additive graph convolutions does not show a consistent performance improvement.

Inspired by the promising results obtained by Multiplicative Integration in sequential domains, we explore its application in GNNs. To the best of our knowledge, this is the first paper that explores the application of MI inside a GC operator. We propose three definitions of MI-based GC operators that stem from the commonly used and very effective *GraphConv* operator [27]. These three operators are defined with the aim of exploring how the MI can be embedded into a graph convolution to obtain a second-order GC operator. Such second-order interactions between features can capture more intricate patterns and relationships in the data, enabling us to go beyond traditional first-order feature analysis. We empirically evaluate the proposed MI-GNNs on eight commonly adopted graph classification benchmarks. The experiments show how MI, applied in the aggregation and/or combination step, allows us to uncover hidden dependencies contributing to improved model performances. We compare the proposed methods with the most common additive graph convolutional operators. In particular, we analyze the results regarding the accuracy and computational time required for training. The results highlight how the use of MI can help obtain improved performance in terms of accuracy and speed of convergence. We apply rigorous statistical hypothesis testing to assess the statistical significance of the observed improvements. Considering that the application of MI also influences the form and the flow of the gradient of the GNN, we analyze how gradient propagation differs among multiplicative and additive GCs.

2 GRAPH NEURAL NETWORKS

Let $G = (V, E, \mathbf{X})$ be a graph, where $V = \{v_1, \dots, v_n\}$ denotes the set of nodes of the graph, $E \subseteq V \times V$ is the set of edges and $\mathbf{X} \in \mathbb{R}^{n \times s}$ is a multivariate signal on the graph nodes with the i-th row $\mathbf{x}_{v_i}$ representing the attributes of v_i. We define $\mathbf{A} \in \mathbb{R}^{n \times n}$ as the adjacency matrix of the graph, with elements $a_{ij} = 1 \iff (v_i, v_j) \in E$. With $\mathcal{N}(v)$, we denote the set of nodes adjacent to node v.

A Graph Neural Network (GNN) is a model that exploits the structure of the graph and the information embedded in feature vectors of each node in order to learn a representation $\mathbf{h}_v \in \mathbb{R}^m$ for each vertex $v \in V$. In modern GNN models, the computation of $\mathbf{h}_v$ can be divided into two main steps: *aggregate* and *combine*. We can define aggregation and combination by using two functions, $\mathcal{A}$ and C, respectively:

$$\mathbf{h}_v = C(\mathbf{x}_v, \mathcal{A}(\mathbf{x}_u : u \in \mathcal{N}(v))). \tag{1}$$

It is possible to extend the range of the considered neighborhood by iteratively performing aggregation and combination for k iterations. In this way, we obtain a hidden representation $\mathbf{h}_v^{(k)}$ of the node v that contains information about the structure and the neighbors that are at a distance k from v:

$$\mathbf{h}_v^{(i)} = C(\mathbf{h}_v^{(i-1)}, \mathcal{A}(\{\mathbf{h}_u^{(i-1)} : u \in \mathcal{N}(v)\})),\ i \in [1, \dots, k],$$
$$\mathbf{h}_v^{(0)} = \mathbf{x}_v, \quad \mathbf{h}_v^{(i)} \in \mathbb{R}^{m_i},$$

where m_i is the size of the convolutional layer i. We thus obtain a deep GNN of k-layers. The choice of aggregation function $\mathcal{A}$ and combination function C defines the type of Graph Convolution (GC) adopted by the GNN [21, 26].

In the last few years, several different GCs have been proposed, and most of them share the same computational building block that exploits a form of additive combination function. Generalizing, we can define this building block as:

$$\mathbf{h}_v^{(i)} = \phi\Big(\mathcal{F}(\mathbf{h}_v^{(i-1)}) + \mathcal{A}(\{\mathbf{h}_u^{(i-1)} : u \in \mathcal{N}(v)\})\Big), \tag{2}$$

where $\phi(\cdot)$ is a nonlinear activation function, and $\mathcal{F}$ is usually a linear projection. Moreover, it is important to notice that the aggregation function $\mathcal{A}$ is commonly based on the sum of the neighbor embeddings. This commutative operation allows it to be invariant with respect to the order of the neighbors.

A very common formulation is the *GraphConv* [27], that is based on the 1-dimensional Weisfeiler-Leman graph invariant (1-WL) [10]. The *GraphConv* is defined as follows:

$$\mathbf{h}_v^{(i)} = \phi\Big(\mathbf{W}\mathbf{h}_v^{(i-1)} + \mathbf{W}_\Sigma \sum_{u \in \mathcal{N}_v} \mathbf{h}_u^{(i-1)} + \mathbf{b}\Big) \tag{3}$$

where $\mathbf{W}, \mathbf{W}_\Sigma \in \mathbb{R}^{m_i \times m_{i-1}}$ (with $m_0 = s$) and $\mathbf{b} \in \mathbb{R}^{m_i}$ are the network learnable parameters. Here and in the following sections, we denote the parameters with subscripts referring to their related function.

3 MULTIPLICATIVE INTEGRATION AND GRAPH CONVOLUTION

The GC operators defined in literature generally follow the same general structure defined in Eq. (2), where $\mathcal{A}$ is defined as an *additive block*. This work explores how the Hadamard product can help perform aggregation and combination in GC. We decided to explore the application of Multiplicative Integration on GC because the aggregation and the combination blocks basically integrate contextual information from different sources, and MI is an effective methodology to perform this task [19]. The advantage of using Hadamard products to combine information flow in recurrent neural networks was discussed in [36]. Moreover, using MI allows the definition of second-order GCs, the resulting model will define a multiplicative interaction between all the neurons that compose two GC contiguous layers, consequently influencing the model's gradient. Using the second-order layer to learn about structured domains has already shown promising results, particularly considering sequential domains. In more detail, in [9], the authors study

the improvement in representational ability achieved by utilizing a second-order RNN where the recurrent neurons of a single layer were multiplied by each other. A similar approach is exploited in defining the mLSTM [23], where a multiplicative block is used to differentiate the recurrent transition function as a function of the input. Another advantage worth mentioning that arises from using the MI paradigm is that the model has better gradient properties, i.e., the gradient propagates in a stronger way from one layer to the other.

In the following part of this section, we present three different possibilities to extend the *GraphConv* operator by Multiplicative Integration.

3.1 MI-GNN

The first proposed version of MI-GNN (named **MI-GNN-v1**) exploits MI to integrate the information from the current node with the one from its neighborhood. This is obtained by replacing the sum of the additive version of the *GraphConv* with the Hadamard product between the current node projection and the aggregation of its neighbors:

$$\mathbf{h}_v^{(i)} = \phi\Big(\Big(\mathbf{W}\mathbf{h}_v^{(i-1)} + \mathbf{b}\Big) \odot \sum_{u \in \mathcal{N}(v)} \Big(\mathbf{W}_\Sigma \mathbf{h}_u^{(i-1)} + \mathbf{b}_\Sigma\Big)\Big). \quad (4)$$

The two bias terms $\mathbf{b}$ and $\mathbf{b}_\Sigma$ are inserted to obtain a more expressive formulation. In fact, by distributing the product over the sum and rearranging the terms, we get the equivalent equation

$$\mathbf{h}_v^{(i)} = \phi\Big(|\mathcal{N}(v)|\Big(\mathbf{W}\mathbf{h}_v^{(i-1)} + \mathbf{b}\Big) \odot \mathbf{b}_\Sigma + \mathbf{b} \odot \mathbf{W}_\Sigma \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(i-1)} + \\ + \mathbf{W}\mathbf{h}_v^{(i-1)} \odot \Big(\mathbf{W}_\Sigma \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(i-1)}\Big)\Big). \quad (5)$$

Note that the first two terms of the above equation constitute a scaled version of the regular additive interaction between $\mathbf{h}_v^{(i-1)}$ and $\sum_u \mathbf{h}_u^{(i-1)}$. The scaling is dynamic since, in addition to the constant bias terms, the scaling factor also depends on the cardinality of the neighborhood. Moreover, the weights regulating the multiplicative integration between these two components, defined by the third term, are obtained by combining the two weight matrices $\mathbf{W}$ and $\mathbf{W}_\Sigma$ of the convolution, with no increase in the number of parameters with respect to the additive *GraphConv*, except for the additional bias. To further enhance the expressiveness of the MI-GNN, at the cost of increasing the number of convolution parameters, the additive and multiplicative building blocks can be combined explicitly, each with its own weights. We named this variation **MI-GNN-v2**:

$$\mathbf{h}_v^{(i)} = \phi\Big(\Big(\mathbf{W}\mathbf{h}_v^{(i-1)} + \mathbf{b}\Big) + \mathbf{W}_\Sigma \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(i-1)} + \\ + \mathbf{W}_\odot\Big(\mathbf{h}_v^{(i-1)} \odot \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(i-1)}\Big)\Big) \quad (6)$$

The third MI variation of the *GraphConv* that we propose, **MI-GNN-v3**, exploits the Hadamard product to define both the combination and aggregation steps of the GC operator:

$$\mathbf{h}_v^{(i)} = \phi\Big(\Big(\mathbf{W}\mathbf{h}_v^{(i-1)} + \mathbf{b}\Big) \odot \Big(\mathbf{W}_\Pi \Big[\prod_{u \in \mathcal{N}(v)}^{\odot} \mathbf{h}_u^{(i-1)}\Big] + \mathbf{b}_\Pi\Big)\Big) \quad (7)$$

This formulation makes the interaction among all nodes involved in the convolution uniform, implementing a global gating mechanism while preserving the sharing scheme of the parameters used in *GraphConv*.

The use of MI as a combination mechanism, however, involves multiplying several node embeddings (projected using the same shared weights) that can lead to numerical stability issues in case of extremely small (close to 0) or large (significantly higher than 1) values. This can make the training phase unstable. To solve this issue, by maintaining a multiplicative integration approach as an aggregation mechanism, we propose to transform the product among the neighbors into a sum by exploiting the logarithm function jointly with the *ReLU* function to ensure that the co-domain of $\mathcal{A}(\cdot)$ is limited to a set of values that ensure a more stable training phase:

$$\mathbf{h}_v^{(i)} = ReLU\Big(\mathbf{W}\mathbf{h}_v^{(i-1)} + \mathbf{b}\Big) \odot \\ \odot \Big(\mathbf{W}_{log}\Big[\sum_{u \in \mathcal{N}(v)} \log\Big[ReLU(\mathbf{h}_u^{(i-1)}) + \epsilon\Big]\Big] + \mathbf{b}_{log}\Big), \quad (8)$$

where ϵ is a small positive constant which prevents the input to log to be 0. Notice that when $\epsilon = 1$, we get as output only positive values, avoiding negative values with high modules. We applied the *ReLU* function also on the current node embedding ($\mathbf{W}\mathbf{h}_v^{(i-1)} + \mathbf{b}$) projection because we want to ensure that the new embedding will be a positive tensor. This is critical since it will be used as input to a logarithm in the subsequent GC layer (if it exists, i.e., if $i < k$).

4 RELATED WORKS

Several models inspired by the graph convolution idea have been proposed recently. Some of them exploit the multiplicative operator to implement a gating mechanism. In particular, the Graph Attention Networks (GAT) [34] uses multiplication in defining a convolution operator based on masked self-attention. The idea is to replace the adjacency matrix in the convolution with a matrix of attention weights for multiple heads. While it may be more complex to train, GAT allows assigning a different weight to each neighbor of a node; thus, it is a very expressive graph convolution.

A similar gating approach is proposed in [25], where the authors propose an extension of Graph Neural Networks to produce sequences in output. They define a propagation model reminiscent of the GRU cell definition. In the resulting convolutional operator, the aggregation is still defined using an addictive block. In contrast, a projection of the node embeddings is used to perform GRU-like updates of the embedding that mixes the information from the other nodes and the previous timestep.

The recent work of Koishekenov and Bekkers [22] explores how to combine features best to condition GNNs on additional information. Their "strong conditioning", is the Hadamard product between the weighted adjacency and feature matrices that replace the layers in an MLP. Differently from MI-GNNs, these three discussed gating convolutions [22, 25, 33] do not exploit multiplicative integration to

aggregate the embeddings of the various nodes in message-passing graph neural network.

A different way to exploit the multiplicative operator in defining the GC is proposed in [2], where the author proposes the GNN-FiLM. This model exploits a hyper-network that, given the node embedding, outputs two coefficients. One of the resulting coefficients is multiplied by the weights of the GCN, while the other is summed to the affine transformation of the node embeddings. Similar to the previously discussed gated GCs, in GNN-FiLM, the aggregation and the combination steps are based on a summation between node representations.

Hua *et al.* has proposed node pooling based on invariant multiplication [16], where they define a GNN layer that can be seen as the composition of a linear layer with a weight matrix, a multiplicative pooling layer, and another linear map. This layer is used as both the aggregation and update steps of a GNN, thus it does not explicitly leverage the graph structure, but it relies purely on higher-order feature interactions.

Finally, in [28], the authors propose two novel convolutions based on the hyper-networks framework. Both operators dynamically adjust the GC parameters based on the input node, and to do it, the model uses a hyper-network generating a vector having a different parameter for each node. This vector is then multiplied by the GC's weights to adjust them. Similar to the GNN-FiLM, the multiplicative blocks are used to adapt the weights for the current input without explicitly defining a multiplicative interaction among the embeddings of the nodes and their neighborhood.

5 EXPERIMENTAL SETUP AND RESULTS

Our experimental assessment aimed to empirically verify whether multiplication can be used to define graph convolutions with competitive performance. Specifically, our experimental results show that representative methods in literature exploiting multiplicative operators, i.e., GAT and GNN-FiLM, do not show a performance advantage over addition-based GCNs. We then explore the use of MI in graph neural networks, evaluating the proposals presented in Section 3, along with four widely adopted baselines, on seven graph classification benchmark datasets. We start discussing the considered GNN architectures and the adopted model selection procedure in Section 5.1. Then, we describe the adopted datasets in Section 5.2. We then present the comparison in terms of classification accuracy in Section 5.3, where we also empirically study the impact of adopting multiplicative integration in terms of training time and speed of convergence. Finally, we validate the statistical significance of the difference between the results obtained by the various considered models in Section 5.4.

5.1 GNN Architecture and Model Selection

We considered as baselines two commonly adopted GC operators, GCN [21] and *GraphConv*. Moreover, we also experiment with two powerful convolutions for graphs that exploit the multiplicative operation differently than the MI-GNN: GAT and GNN-FiLM.

We handled the experiments and validated all the models' hyper-parameters adopting the GraphGym [38] framework. Specifically, we started off from their findings on the GNN architecture design space by setting a common baseline configuration for all the experiments based on the work of [38]. We set the PReLU as activation function ϕ [13], batch normalization [18] for each layer, and among layers, we adopted the SKIP-CAT scheme [17]. The training is carried out with the ADAM optimizer [20], cosine learning rate schedule (starting from 0.01 and annealed to 0, no restarting), 5×10^{-4} L2 weight decay for regularization. The batch size is set to 32 for all the datasets and we let every experiment run for 400 epochs. We used the libraries `PyTorch=2.0.0`, `PyTorch Geometric=2.3.0`, and our experiments have been carried out in a computing cluster equipped with GPUs Nvidia RTX A5000. Each tested network consists of Multilayer Perceptron (MLP) layers before and after the GC operator layers. This particular architectural setting is the one suggested in GraphGym. The amount of these layers and their hidden units are hyperparameters. In our evaluation, we consider graph classification tasks; therefore, all the considered models have a global pooling layer to compute a graph-level representation given the node embeddings. The pooling layer is defined by concatenating the global mean, max, and sum aggregations [5]:

$$\mathbf{h}^s = [mean_{v \in V}(\mathbf{h}_v^{(k)}), max_{v \in V}(\mathbf{h}_v^{(k)}), sum_{v \in V}(\mathbf{h}_v^{(k)})], \quad (9)$$

where k is the number of GC layers.

Each dataset is split in train/validation/test sets according to a [80%, 10%, 10%] random split. Every configuration is run 3 times, and we take the average of all the evaluation metrics (accuracy, time, etc.) taken on the test set at the best epoch in validation. The random generator seed is set likewise at the beginning of each run, thus ensuring that the dataset splits are equal for each model and making their comparison more robust and fair. We performed a full grid search over all the hyper-parameters combinations reported in Table 1, resulting in 96 configurations tested for each of the 7 datasets and of the 7 GNN layer types. The layers for *GraphConv*, GCNConv, GATConv, and FiLMConv are taken from the `PyTorch` library.

Table 1: Hyper-parameters Grid

Hyper-parameters	Values
Pre-MLP layers	1, 2
GC layers	2, 4, 6, 8
Post-MLP layers	2, 3
Hidden units	64, 128, 256
Activation ϕ	PReLU, Tanh

5.2 Datasets

All the considered methods were empirically validated on seven commonly adopted graph classification benchmarks. Namely, we used four datasets modeling bio-informatics problems: NCI1 [35], PROTEINS, [1], D&D [7] and ENZYMES [1]. NCI1 involves chemical compounds represented by their molecular graph, where node labels represent the atom type, and bonds correspond to edges. In NCI1, the graphs represent anti-cancer screens for cell lung cancer. The remaining datasets, PROTEINS, D&D and ENZYMES involve graphs that represent proteins. Amino acids are represented by nodes and edges that connect amino acids that in the protein are

less than 6Å apart. All the prediction tasks are binary classification tasks, except for the ENZYMES dataset, where a multi-class classification of chemical compounds (six classes) is represented. We further considered three large social graph datasets: COLLAB, IMDB-B, IMDB-M [37]. In COLLAB, each graph represents a collaboration network of a corresponding researcher with other researchers from three fields of physics. The task consists of predicting which one of the researcher's three physics fields belongs to. IMDB-B and IMDB-M are composed of graphs derived from actors/actresses who played in different movies collected on IMDB, together with the movie genre information. Each graph has a target that represents the movie genre. IMDB-B models a binary classification task, while IMDB-M contains graphs belonging to three classes. In contrast to the bio-informatics datasets, the nodes contained in the social datasets do not have any associated label, and therefore, only the graph topology is regarded.

5.3 Results and Discussion

The results of our experiments are presented in Table 2. We can start noticing that the two existing multiplication-based methods (GNN-FiLM and GAT) do not result in the better-performing methods on any of the considered datasets. This observation enforces our intuition that research is still required in multiplication-based graph convolutions to achieve competitive performance. As for the proposed methods, at least one version of the proposed MI-GNNs is the best-performing method in six of the eight datasets. The GCN performs well only in the COLLAB dataset, which is the dataset that has the highest edges/graph ratio. We argue that the normalized adjacency matrix of the GCN is more robust for such cases, whereas the other multiplicative or additive operators are penalized by the higher average degree. The *GraphConv* has the highest accuracy only for the NCI1 dataset; however, MI-GNN-v1 and -v2 perform similarly. MI-GNN-v1 and MI-GNN-v2 are the best-performing methods on two datasets each. MI-GNN-v3 performs on par with MI-GNN-v1 on IMDB-B (but with a higher variance) and is the best-performing method on IMDB-M.

Additionally, we analyzed whether the improved performances of our implementation of the MI-GNNs also translate into briefer training times. In Figure 1, we display each dataset's accuracy and total training time until the best validation epoch is reached. In the plot, we also report the Pareto frontier—the set of all Pareto-efficient points. In our case, those points refer to the methods for which no improvement in one dimension (either accuracy or training efficiency) is possible without losing performance on the other dimension. Such points can be informally interpreted as the best time/accuracy trade-off methods. We can see that the MI-GNNs are often present in the Pareto-front, meaning that not only do they achieve the best performances most of the time, but they also require less or comparable time than the baselines. Notice that on many datasets, some methods are extremely efficient (e.g., GCN on COLLAB, IMDB-M, and PROTEINS) but perform very poorly. Even though those points are part of the Pareto-front because of the low training times, they are not interesting solutions for their degraded predictive performance. We want to mention that GCN results are the slowest method because, by default, it does not store the normalized adjacency matrix. For this reason, its training could be tweaked and sped up; however, its predictive performances would not be altered.

5.4 Statistical Significance of the Results

Inspired by the analysis of [6], we investigated the performances of our proposed models beyond a simple but naive maximum-accuracy benchmark. Indeed, when comparing multiple classifiers on multiple datasets, one should apply rigorous statistical hypothesis testing before assessing whether the improvement is statistically significant. Additionally, when testing multiple hypotheses simultaneously, multiplicity issues arise and one should adopt the proper corrections.

The Friedman test [30] is a non-parametric test that does not assume the distribution and variances of the samples. For each dataset and for each configuration, it ranks the accuracies of all the models. Then, it computes a statistic χ_F^2 under the null hypothesis, which states that all the models are equivalent and their ranks should be random. In our case, this test gives a p-value< 0.01, so the null hypothesis is rejected, and we can proceed with a post-hoc test to tell which algorithms perform the best. To calculate the statistical significance of the pairwise comparisons between the models, we used the Conover post-hoc test for unreplicated blocked data [4] where the p-values are adjusted with the step-down method using the Sidak corrections [31]. Other common p-values adjustments yielded equivalent outcomes. The outcome of this analysis is neatly presented with the critical difference (CD) diagrams in Fig. 2. This plot displays the averages of the normalized ranks of the models among all the configurations. On the x-axis, 1 would stand for a model that always scores better; on the contrary, a model at 0 would always be the last one in the rankings. The groups that could not be statistically deemed different by the Conover test are linked by a horizontal crossbar. We can see that *GraphConv*, MI-GNN-v1, and -v2 are significantly better ranked than all the other models. While the MI-GNN-v1 has the highest average rank of 0.71, indicating that it tends to be the best model more frequently for a given configuration and dataset, there is insufficient statistical evidence to confirm this conclusion. Therefore, conducting experiments on additional datasets would be necessary to endorse this assessment. Nevertheless, this shows that MI-GNN-v1 is a valid alternative to GraphConv, and all the parameters and settings being equal, merely replacing the additive term with the Hadamard product can lead to improved performances. Moreover, the CD diagram shows us that MI-GNN-v3 is, on average, in the middle of the rankings despite achieving the best accuracy in PROTEINS and ENZYMES. This advises us that when evaluating new machine learning models, when it is possible, it is crucial to go beyond the maximum accuracy rationale—only by testing whether the new model performs statistically better than the baselines for multiple configurations, one can ensure that such improvements are significant and applicable across different scenarios.

5.5 Open Graph Benchmark

To prove the validity of our approach, we additionally evaluated the performances of the MI-GNNs on the *molhiv* dataset belonging to the Open Graph Benchmark (OGB) [15]. This dataset is made

Table 2: Accuracy and standard deviation, in percentages, on the test set for the best-validated models on all the datasets. The best performances are highlighted in boldface.

Dataset \ GNN	GNN-FiLM	GAT	GCN	GraphConv	MI-GNN-v1	MI-GNN-v2	MI-GNN-v3
COLLAB	68,9±9,2	55,7±1,4	**79,4±1,6**	75,9±1,7	76,9±2,7	75,9±0,7	74,5±0,9
DD	77,9±2,1	77,6±2,0	77,6±3,4	77,6±2,4	**79,6±2,4**	77,3±0,7	77,0±5,3
ENZYMES	58,9±0,8	58,3±3,6	62,2±4,2	62,2±0,8	62,8±0,8	**63,3±2,4**	60,6±3,9
IMDB-B	53,7±4,8	55,7±2,5	71,7±5,3	73,0±3,3	**74,0±2,2**	73,7±1,7	**74,0±4,9**
IMDB-M	41,1±7,5	40,0±2,2	50,5±2,7	50,9±2,5	49,6±3,8	49,8±3,8	**51,3±3,3**
NCI1	79,0±2,1	80,2±2,0	80,6±1,4	**81,7±1,7**	81,6±1,0	81,6±2,0	78,3±1,7
PROTEINS	73,8±3,0	74,1±2,6	73,8±2,3	75,0±4,1	75,3±3,4	**75,9±2,6**	74,4±2,8

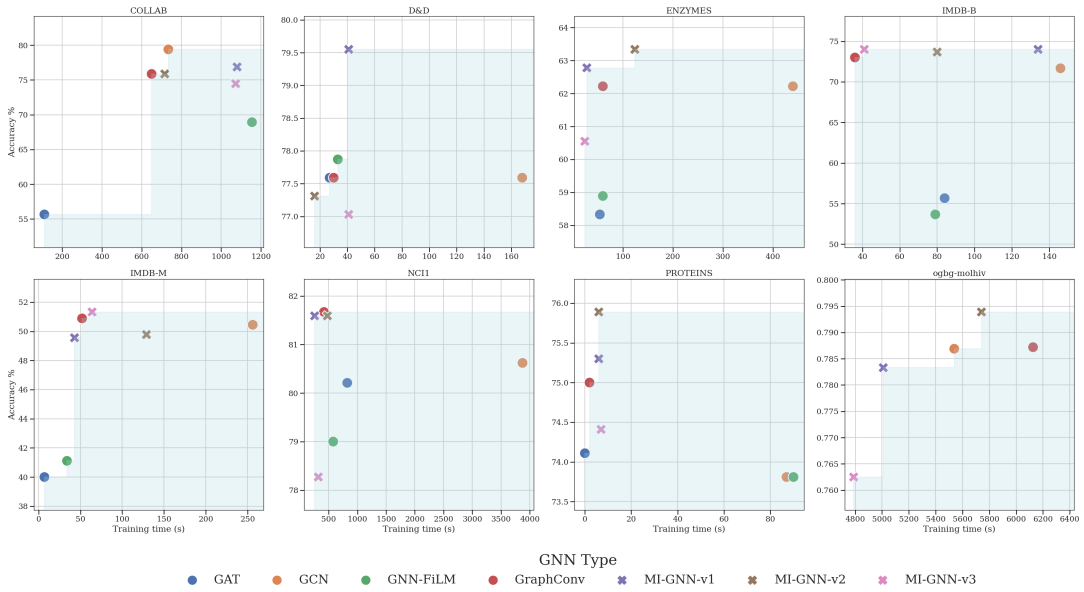

Figure 1: Distribution of the training times of the best performing models w.r.t their accuracy. The MI-GNNs are marked with a cross, and the other baselines are marked with a rounded point. The light-blue area (left side) helps to identify the Pareto-front. For ogbg-molhiv, the AUC is reported on the vertical axis.

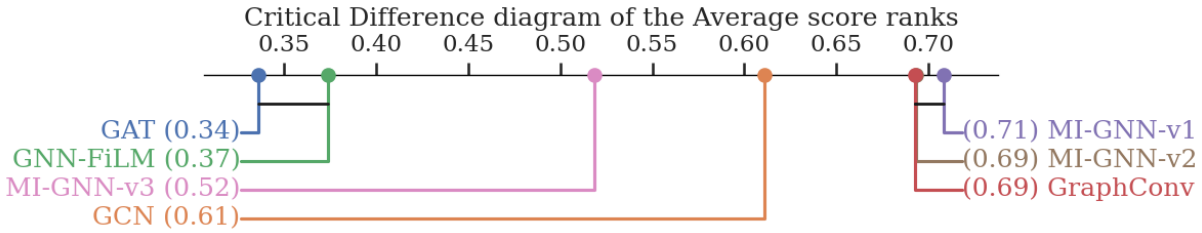

Figure 2: Critical Difference diagram of the Average score ranks.

of 41 127 graphs, an order of magnitude more than the ones previously considered, and it follows its pre-defined pre-processing and evaluation pipeline. Due to its complexity and time constraints, we restricted our experimental setup to a smaller hyper-parameters grid and fewer models. For these reasons, it cannot be analyzed along with the other datasets and the procedure described in the previous section. Table 3 reports the best AUC on the test set, training time, and architecture for the model with the highest AUC score on the validation set. MI-GNN-v2 has the best performance and it requires fewer parameters, thus, this supports our intuition that MI is able to grasp relevant complex interactions among nodes. It is

worth noticing that such an AUC score would place our model in the top 20 leaderboard for this dataset.

6 GRADIENT ANALYSIS

Pursuing the goal of characterizing the strengths and the weakness of the application of the multiplicative integration in GC operators, we theoretically analyze the gradient of the various proposed versions of MI-GNN and we compare them with the gradient of the most similar additive model, the *GraphConv*. We denote as $\mathbf{H} \in \mathbb{R}^{m_i \times n}$ the matrix of all the nodes' embeddings at layer i, $\bar{\mathbf{H}} \in \mathbb{R}^{m_{i+1} \times n}$ is the same matrix at the following layer $i+1$, and δ_{ij} the Kronecker delta function. In the following, we use Einstein's notation of summation over repeated indexes: $a_{ij} = \mathbf{A}^i{}_j = \sum_k b_{ik} c_{kj} = \mathbf{B}^i{}_k \mathbf{C}^k{}_j$. For the *GraphConv*, the derivatives w.r.t. the weights are (omitting the Jacobian of ϕ and the bias term):

$$\frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{W}^a{}_b} = \delta_{ax} \mathbf{H}^b{}_v \text{ and, } \frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{W}_\Sigma{}^c{}_d} = \delta_{cx} \mathbf{H}^d{}_u \mathbf{A}^u{}_v, \tag{10}$$

while the derivative of the MI-GNN-v1 w.r.t the weights are the following:

$$\frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{W}^a{}_b} = \delta_{ax} \mathbf{H}^b{}_v \cdot \left(\mathbf{W}_\Sigma{}^x{}_\rho \mathbf{H}^\rho{}_u \mathbf{A}^u{}_v \right), \tag{11}$$

$$\frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{W}_\Sigma{}^c{}_d} = \delta_{cx} \mathbf{H}^d{}_u \mathbf{A}^u{}_v \cdot \left(\mathbf{W}^x{}_\sigma \mathbf{H}^\sigma{}_v \right). \tag{12}$$

It is worth noticing how the use of MI instead of an additive block leads the gradient with respect to $\mathbf{W}$ to be influenced by $\mathbf{W}_\Sigma$ and vice-versa. Moreover, in the MI-GNN, the gradient of the weights that multiply the current node v also depends on the adjacency matrix $\mathbf{A}$, making it possible to carry information about the neighbors when learning $\mathbf{W}$. This enriched gradient for the first weights matrix of Eq. 4 implies that the neighborhood will directly influence the projection of the node v. We can notice a similar effect comparing the gradient of the *GraphConv* and the MI-GNN-v1 w.r.t. the node embeddings of the previous layer. Indeed for the *GraphConv*, the gradient is the following:

$$\frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{H}^z{}_t} = \delta_{tv} \mathbf{W}^x{}_z + \mathbf{W}_\Sigma{}^x{}_z \mathbf{A}^t{}_v, \tag{13}$$

while the one of the MI-GNN-v1 is:

$$\frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{H}^z{}_t} = \delta_{tv} \mathbf{W}^x{}_z \cdot \left(\mathbf{W}_\Sigma{}^x{}_\rho \mathbf{H}^\rho{}_u \mathbf{A}^u{}_v \right) + \left(\mathbf{W}^x{}_\sigma \mathbf{H}^\sigma{}_v \right) \cdot \left(\mathbf{W}_\Sigma{}^x{}_z \mathbf{A}^t{}_v \right). \tag{14}$$

For what concerns the MI-GNN-v2, we have three weight matrices (see Eq. 6). The gradient of the hidden representations w.r.t $\mathbf{W}$ and $\mathbf{W}_\Sigma$ are the same as reported in Eq. 12, so the consideration made for the MI-GNN-v1 holds also for these second version. Interestingly, the gradient w.r.t. $\mathbf{W}_\odot$ keeps the same capability of conveying information about the current node and the neighborhood as in the case of the other two weights matrices:

$$\frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{W}_\odot{}^e{}_f} = \delta_{ex} \left(\mathbf{H}^f{}_v \cdot \mathbf{A}^f{}_l \mathbf{H}^l{}_v \right). \tag{15}$$

Differently from the gradient of the hidden representation with respect to the previously considered $\mathbf{W}$ and $\mathbf{W}_\Sigma$ parameter matrices, in this case, the gradient is not influenced by the other weights of the model.

Considering the MI-GNN-v3, the derivative w.r.t. the previous layer is highly influenced by the ReLU function applied to the projection of the node v and the projection of the neighborhood. Recall that the ReLU function is used to avoid instability during the training. Let us start considering the gradient of the hidden representation of a layer w.r.t. the representation of the layer before:

$$\frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{H}^z{}_t} = \delta_{tv} \Theta\left(\mathbf{W}^x{}_\sigma \mathbf{H}^\sigma{}_v \right) \mathbf{W}^x{}_z \cdot \left(\mathbf{W}_{\log}{}^x{}_\rho \log\left[\mathrm{ReLU}\left(\mathbf{H}^\rho{}_u \right) \right] \mathbf{A}^u{}_v \right) + + \mathrm{ReLU}\left(\mathbf{W}^x{}_\sigma \mathbf{H}^\sigma{}_v \right) \cdot \left(\mathbf{W}_{\log}{}^x{}_z \mathbf{A}^t{}_v \right), \tag{16}$$

where $\Theta(\cdot)$ is the derivative of the ReLU. If we consider the gradient w.r.t. the weights $\mathbf{W}$, $\mathbf{W}_{\log}$ we can notice that the gradient takes into account the interaction between them, as well as the adjacency matrix:

$$\frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{W}^a{}_b} = \delta_{ax} \Theta\left(\mathbf{W}^x{}_\sigma \mathbf{H}^\sigma{}_v \right) \mathbf{H}^b{}_v \cdot \left(\mathbf{W}_{\log}{}^x{}_\rho \log\left[\mathrm{ReLU}\left(\mathbf{H}^\rho{}_u \right) \right] \mathbf{A}^u{}_v \right) \tag{17}$$

$$\frac{\partial \bar{\mathbf{H}}^x{}_v}{\partial \mathbf{W}_{\log}{}^c{}_d} = \mathrm{ReLU}\left(\mathbf{W}^x{}_\sigma \mathbf{H}^\sigma{}_v \right) \cdot \delta_{cx} \log\left[\mathrm{ReLU}\left(\mathbf{H}^d{}_u \right) \right] \mathbf{A}^u{}_v. \tag{18}$$

Unlike the other version of MI-GNN, this one also considers the log of the ReLU activation of the neighbors' embeddings and the ReLU derivative applied to the embedding of the current node v. In particular, for $\mathbf{W}_{\log}$, this behavior ensures to have always a positive value for the gradient.

We are committed to ensuring transparency and reproducibility in our research. To facilitate this, all the detailed step-by-step computations, along with the relevant code and experimental analysis, are made openly accessible and available online[1].

7 CONCLUSIONS AND FUTURE WORKS

This paper analyzed how Multiplicative Integration (MI) can be exploited in defining graph convolution operators. We proposed three variants that explore different ways to apply MI on graph-structured data, which we dub MI-GNN. We empirically evaluated the three versions of MI-GNN on eight benchmark classification datasets, adopting a fair experimental setting and analyzing the results with a solid statistical setting. The experimental assessment showed competitive results of MI-GNNs in terms of accuracy, while MI-GNNs tend to show a reduced training time compared to competing models.

We also analyzed how the use of MI influences the training with a theoretical study of the gradients, which showed the capability of the MI-GNN models to convey structural information when computing the gradients to update the weights of the model.

Moreover, the gradient flow of the MI-GNN was influenced by the interaction between the different weights of the GC, showing the capability to convey more information compared to the widely adopted *GraphConv*.

In the future, we plan to study how MI can be exploited to design novel architectures for learning in graph domains. In particular, we will explore the application of MI in defining pooling layers to compute a graph-level representation of the input. Moreover, we also plan to explore if the richer gradient flow of the MI-GNN can be helpful in graph continual learning settings where it is particularly useful to consider the dynamic information of the neighbors (that changes over time) during training. Finally, we intend to extend

[1] https://github.com/paolofraz/MI-GNN

Table 3: Experimental results and architecture for the best-validated model on the ogbg-molhiv dataset.

GNN Type	Pre-MLP layers	GC-layers	Post-MLP layers	Hidden units	# Param.	AUC	Training Time (s)
GCN	1	2	3	384	24 517 826	78,69±0,86	5538
GraphConv	1	4	3	384	69 506 498	78,72±2,15	6126
MI-GNN-v1	2	6	2	256	31 872 322	78,33±0,90	5009
MI-GNN-v2	2	4	3	128	7 983 554	**79,39±0,15**	5741
MI-GNN-v3	2	2	3	384	25 109 954	76,25±0,03	4788

the applicability of our approach beyond graph classification tasks and explore its effectiveness in the domain of node classification.

REFERENCES

[1] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.

[2] Marc Brockschmidt. 2020. GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 1144–1152. http://proceedings.mlr.press/v119/brockschmidt20a.html

[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[4] William Jay Conover. 1999. *Practical nonparametric statistics*. Vol. 350. john wiley & sons.

[5] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems* 33 (2020), 13260–13271.

[6] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research* 7 (2006), 1–30.

[7] Paul D Dobson and Andrew J Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology* 330, 4 (2003), 771–783.

[8] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning*. 1263–1272. arXiv:1704.01212

[9] Mark W Goudreau, C Lee Giles, Srimat T Chakradhar, and Dong Chen. 1994. First-order versus second-order single-layer recurrent neural networks. *IEEE Transactions on Neural Networks* 5, 3 (1994), 511–513.

[10] Martin Grohe. 2017. *Descriptive complexity, canonisation, and definable graph structure theory*. Vol. 47. Cambridge University Press.

[11] David Ha, Andrew M. Dai, and Quoc V. Le. 2017. HyperNetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rkpACe1lx

[12] David Ha, Andrew M. Dai, and Quoc V. Le. 2017. HyperNetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rkpACe1lx

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. arXiv:1502.01852 [cs.CV]

[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[15] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687* (2020).

[16] Chenqing Hua, Guillaume Rabusseau, and Jian Tang. 2022. High-Order Pooling for Graph Neural Networks with Tensor Decomposition. arXiv:2205.11691 [cs.LG]

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[18] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. pmlr, 448–456.

[19] Siddhant M Jayakumar, Jacob Menick, Wojciech M Czarnecki, Jonathan Schwarz, Jack Rae, Simon Osindero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. 2020. Multiplicative interactions and where to find them. In *International Conference on Learning Representations*.

[20] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[21] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*. 1–14. https://doi.org/10.1051/0004-6361/201527329 arXiv:1609.02907

[22] Yeskendir Koishekenov and Erik J. Bekkers. 2023. An Exploration of Conditioning Methods in Graph Neural Networks. *ArXiv* abs/2305.01933 (2023). https://api.semanticscholar.org/CorpusID:258461000

[23] Ben Krause, Liang Lu, Iain Murray, and Steve Renals. 2016. Multiplicative LSTM for sequence modelling. *arXiv preprint arXiv:1609.07959* (2016).

[24] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. 2017. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759* (2017).

[25] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks. In *ICLR*. https://doi.org/10.1103/PhysRevLett.116.082003 arXiv:1511.05493

[26] Alessio Micheli. 2009. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks* 20, 3 (2009), 498–511.

[27] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609. https://doi.org/10.1609/aaai.v33i01.33014602 arXiv:1810.02244

[28] Luca Pasa, Nicolò Navarin, Wolfgang Erb, and Alessandro Sperduti. 2023. Empowering Simple Graph Convolutional Networks. *IEEE Transactions on Neural Networks and Learning Systems* (2023), 1–15. https://doi.org/10.1109/TNNLS.2022.3232291

[29] Luca Pasa, Nicolo Navarin, and Alessandro Sperduti. 2021. Simple Multi-resolution Gated GNN. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. 01–07. https://doi.org/10.1109/SSCI50451.2021.9660046

[30] Dulce G Pereira, Anabela Afonso, and Fátima Melo Medeiros. 2015. Overview of Friedman's test and post-hoc analysis. *Communications in Statistics-Simulation and Computation* 44, 10 (2015), 2636–2653.

[31] Zbyněk Šidák. 1967. Rectangular confidence regions for the means of multivariate normal distributions. *J. Amer. Statist. Assoc.* 62, 318 (1967), 626–633.

[32] Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*. 1017–1024.

[33] Shyam A Tailor, Felix Opolka, Pietro Lio, and Nicholas Donald Lane. 2021. Do we need anisotropic graph neural networks?. In *International Conference on Learning Representations*.

[34] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[35] Nikil Wale, Ian A Watson, and George Karypis. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems* 14, 3 (2008), 347–375.

[36] Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Russ R Salakhutdinov. 2016. On multiplicative integration with recurrent neural networks. In *Advances in neural information processing systems*. 2856–2864.

[37] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1365–1374.

[38] Jiaxuan You, Rex Ying, and Jure Leskovec. 2020. Design Space for Graph Neural Networks. In *NeurIPS*.

[39] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2017. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 4189–4198.