Large-scale Data Classification based on the Integrated Fusion of Fuzzy Learning and Graph Neural Network

Václav Snášel[a,*], Martin Štěpnička[b,*], Varun Ojha[c,*], Ponnuthurai Nagaratnam Suganthan[d,*], Ruobin Gao[e,*] and Lingping Kong[a,**]

[a]Department of Computer Science, VSB-Technical University of Ostrava, Ostrava, Czech Republic
[b]CE IT4Innovations – IRAFM, University of Ostrava, Ostrava, Czech Republic, Ostrava, Czech Republic
[c]School of Computing, Newcastle University, Newcastle, United Kingdom
[d]KINDI Center for Computing Research, College of Engineering, Qatar University, Doha, Qatar
[e]School of Civil and Environmental Engineering, Nanyang Technological University, Singapore, Singapore

ARTICLE INFO

Keywords:
Graph neural networks
Fuzzy inference systems
Graph transformers
Sub-Space regression.

ABSTRACT

Deep learning and fuzzy models provide powerful and practical techniques for solving large-scale deep-learning tasks. The fusion technique on deep learning and fuzzy system are generally classified into ensemble and integrated modes and materializes in information fusion, model fusion, and feature fusion. In an ensemble-based fusion, the fuzzy model either acts as an activation function or is operated as a separate process aggregating/preprocessing the information. Some early attempts in the field have successfully fused deep neural networks and fuzzy modeling concepts in ensemble mode. However, no effective attempts were made to fuse fuzzy models as an *integrated feature-level fusion learning* with graph neural networks (GNNs). This is mainly due to two challenges related to this fusion: (1) the number of fuzzy rules grows exponentially with the number of features that causes computational inefficiency, and (2) the solution space created by this fusion of fuzzy rules becomes complex due to multiple regression relations between inputs and outputs. Additionally, a simple linear regression at the output space would not be sufficient to model deep learning tasks. Therefore, this paper addresses these challenges by proposing a feature-level fusion method to fuse deep learning and fuzzy modeling where the latter technique is for integrated feature learning, called *fuzzy forest graph neural network* (FuzzyGNN), which creates a fuzzy learning forest fusing the linear graph transformers for deep learning tasks. We conducted experiments on fourteen machine learning datasets to test and validate the efficiency of the proposed FuzzyGNN model. Compared to state-of-the-art methods, our algorithm achieves the best results on four out of five machine learning datasets. The source code will be available at `https://github.com/lingping-fuzzy/` and `https://github.com/P-N-Suganthan`.

1. Introduction

Deep neural networks (DNNs) have become a natural choice for solving large-scale data tasks. They learn from large amounts of data by uncovering intricate patterns in the data and permitting large-scale task-specific feature learning from data. However, most real-world raw data are often defined in terms of connections from one piece of information to another. Hence, data in the form of graphs benefit data analysis and modeling as a natural and flexible way to capture all the possible information and relation between data points encoded in graphs. A graph neural network (GNN) is a class of neural networks for processing data that can be represented as graphs, i.e., a GNN is a DNN architecture applied to graph data. Graph data and GNNs have become popular as a natural choice for representing and solving real-world problems in recent years; as a result, several GNN models are being designed [1]. They offer either state-of-the-art (SOTA) results on benchmark machine learning problems/datasets or new mechanisms embedded into the model [1].

*This document is the results of the research project funded by Czech Science Foundation through the grant 20-07851S; furthermore, we announce the support of DST/ INT/ Czech/ P-12/ 2019, by the Czech Republic Ministry of Education, Youth and Sports in the project META MO-COP.

*Corresponding author
**Principal corresponding author
`vaclav.snasel@vsb.cz` (. Václav Snášel); `martin.stepnicka@osu.cz` (M. Štěpnička); `varun.ojha@newcastle.ac.uk` (.V. Ojha); `p.n.suganthan@qu.edu.qa` (.P.N. Suganthan); `gaor0009@e.ntu.edu.sg` (.R. Gao); `lingping_kong@yahoo.com` (L. Kong)
ORCID(s): `0000-0002-9600-8319` (. Václav Snášel); `0000-0002-0285-075X` (M. Štěpnička); `0000-0002-9256-1192` (.V. Ojha); `0000-0003-0901-5105` (.P.N. Suganthan); `0000-0003-0781-1482` (.R. Gao); `0000-0002-6825-1469` (L. Kong)

Despite massive progress in Neural Networks (NNs) and DNNs, they have been found to be sensitive to training data; for instance, missing or noisy data influence the performance of NNs, which results in unreliable models [2]. In general, Fusion [3] is an approach that integrates data or features and enhances the forecast based on the hybridized system that can benefit each other. In order to improve the tolerance and robustness of NNs based models to missing and noisy data, researchers have attempted to fuse NNs with various fuzzy modeling techniques on raw or processed data, features, or methods, leading this fusion designs into information fusion, feature fusion, and model fusion under a framework known as neuro-fuzzy systems (NFS) [4, 5]. Due to the computational power of NNs and the robustness of fuzzy techniques, such fusion-based methods have provided potential improvement to model sensitivity in solving problems with instances of noisy and missing data [6, 7].

This improvement by NFS systems is also largely attributed to the fusion of human-like reasoning, robustness, and interpretability of fuzzy systems with the learning structure of neural networks [8]. Therefore, NFS benefits from the approximation ability of NNs and the interpretability ability of the fuzzy (if-then) rules. One of the most popular NFS models is the adaptive neuro-fuzzy inference system (ANFIS) [9, 10]. However, to the best of our knowledge, the successes of the model-based fusion method rely on fuzzy systems with NNs have not been replicated in the GNN field, i.e., applications of the efficient GNN model with fuzzy systems have not been thoroughly studied in the large-scale data domain.

There are exceptions to the above-mentioned lack of fusion; however, the approaches that model fusion of the two only do so from the perspective of transforming the representation to membership values by a fuzzy system, considered ensemble-based fusion. Examples of such approaches are as follows: Deng et al. [11] explore the image segmentation in an ensemble model with a fuzzy system that transforms input by a Gaussian-shaped fuzzy set; Tong et al. [12] presented a model incorporating a fuzzy system with a GNN on the so-called few-shot learning task; their model deduces the link relation degree between two nodes by using fuzzy systems.

The works in the literature on neuro-fuzzy hybrids are performed either in an ensemble mode fusion pattern [13, 14] or by applying the typical ANFIS model. Among the ensemble model fusion, some integrate fuzzy inference techniques as activation functions, and some adopt fuzzy inference for data-level fusion, such as pre-processing or post-processing [15, 16] data. Still, the processing results of these approaches are not integral to *feature-level fusion learning* procedure. Table 1 shows a brief research summary including the applications, data usage, and fusion types and if they adopt the ANFIS model. From the list in Table 1, we observe that most of the works apply fuzzy modules for ensemble purposes from a data-level fusion, and where there is an integrated model fusion, the model is applied to small data with non-deep neural networks. The main difference in applying the fuzzy system of the proposed model with the listed ones is shown in Fig. 1, where we integrate the fuzzy system with the neural learning instead of applying a membership function only. As mentioned, two challenges need to be solved for the integrated fusion techniques: (1) avoid the exponentially growing number of fuzzy rules to reduce the computational cost, and (2) create adaptive fuzzy division subspaces to fit complex regression relations between inputs and outputs. Hence, to the best of our knowledge, currently, no work can genuinely be classified as an integrated fusion of fuzzy-neural models for large dataset applications, i.e., for the DL application. In this paper, we present a novel integrated fusion model that utilizes a fuzzy system as an integral part of the representation vector learning mechanism assisting GNN learning to mitigate these two limitations, where the fuzzy-based learning mechanism helps improve the expressibility of vector representation on feature-level fusion. We adopt the graph transformers (GT) [17] with an attention mechanism for better communication between the nodes in a graph.

The main aspects of our contribution are as follows.

- We present an integrated fusion model that fuses a fuzzy system as a part of the representation vector learning mechanism assisting GNN learning; This work is different from existing works that merge fuzzy systems with GNN in the sense that they take a fuzzy system as a training weight transformation tool. In contrast, our work facilitates neural network learning in feature-level fusion by a fuzzy system mechanism.

- We propose representation sampling methods to address fuzzy rules exponentially explosion problem for reducing trainable parameters and computational cost required to deal with large-scale machine learning datasets.

- We create a consequent layer in each subspace formed by the fuzzy system by generalizing a polynomial function that simulates the relationship between independent and dependent variables. This method is aimed at decreasing the fitting error between observation and fitting prediction.

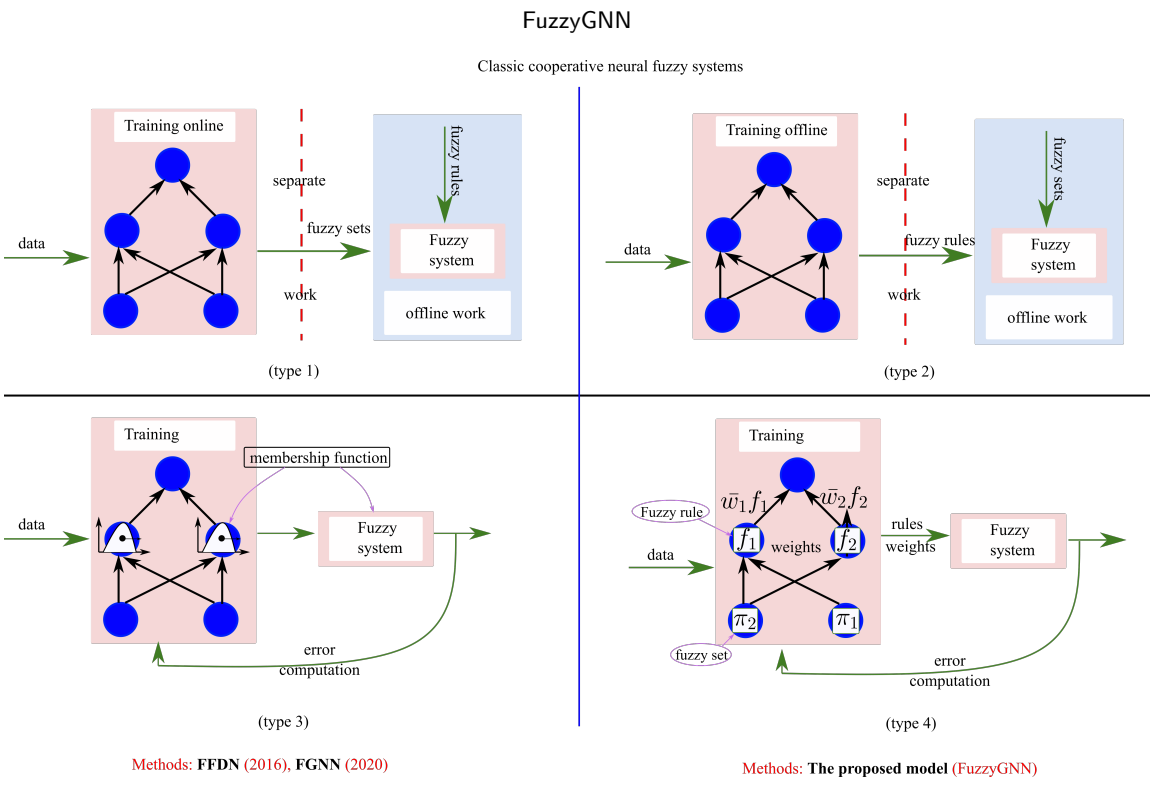


Figure 1: Four classic cooperative neural fuzzy systems, where ensemble model (SP) belongs to type-1 or type-2; FFDN and FGNN are in type-3, the proposed model follows type-4.

- We experimentally justify the performance of the proposed FuzzyGNN model by conducting experiments on five standard machine learning benchmark datasets, including CIFAR10, MNIST, PATTER, CLUSTER, and ZINC, and compare them with SOTA methods.

The structure of this paper is as follows. We review the preliminaries and related works in Section 2, and then we present the proposed model in Section 3. The experimental justification and discussions on the results are in Section 4. Lastly, we conclude the paper in Section 5.

2. Background and Related Work

In this section, we discuss the working mechanism of graph-neural networks (GNNs) and highlight the research works that attempt to fuse fuzzy systems and deep neural networks, especially works that integrate fuzzy systems with GNNs and their potential problems.

Generally, a GNN model consists of a sequence of layers transmitting a vector representation (embedding variables) through layers to update features of nodes (or edges). Each layer delivers updated representation vectors by message passing mechanism (MPM) to the next layer, which then processes these representations as inputs and produces outputs for the next layer [28]. In other words, in GNN, an MPM is a process of iteratively updating a node feature $\mathbf{x}_v^{(\ell)}$ in layer ℓ for a node $v \in \mathcal{V}$ in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ by aggregating localized information from their neighbors $w \in \mathcal{N}(v)$ using some user-defined aggregation function g specific to each GNN model. The node feature $\mathbf{x}_v^{(\ell)}$ is updated as:

$$\mathbf{x}_v^{(\ell+1)} = g\left(\mathbf{x}_v^{(\ell)}, \left\{\mathbf{x}_w^{(\ell)} : w \in \mathcal{N}(v)\right\}\right),$$

where the aggregation function g takes the node feature x_v^{ℓ} and a set of node features of neighbors $\{\mathbf{x}_w^{(\ell)} : w \in \mathcal{N}(v)\}$ as inputs and outputs an updated node feature $\mathbf{x}_v^{\ell+1}$ to be an input for layer $\ell + 1$. The number of intermediates (also hidden) layers placed between the input and output layer of a given network determines the depth of the NN architecture and directly impacts its complexity and power.

Table 1
Comparison of existing Fuzzy-based NN models. *Type*: *ensemble model* works as an activation function, *ensemble model (separate process: SP)* performs fuzzy inference, but the whole process is separate, working on data-level fusion/ output-level fusion, such as preprocess or post-process; *Integrated*: the fuzzy inference performs the learning process in a combined neural model on feature-level fusion. *MLP*: multi-layer perception.

Paper/Year	Type	Data	Application	ANFIS	NN
[11] **FFDN**	ensemble model	image, time-series	categorization, data prediction, MRI segmentation	No	DNN
[18] 2018	ensemble model (SP)	traffic data	incident detection	No	MLP
[12] **FGNN**	ensemble model	miniImageNet, tieredImageNet	Few-Shot Learning	No	GNN
[19] 2022	ensemble model (SP)	Small Movie and eBook review	recommendation	No	DNN
[20] 2022	ensemble model (SP)	image	tuberculosis detection	No	DNN
[21] 2019	ensemble model (SP)	Hypersonic Vehicles data	trajectory planning	No	DNN
[22] 2021	ensemble model (SP)	tweets	prediction	No	DNN
[23] 2020	ensemble model (SP)	image	classification	No	DNN
[24] 2020	ensemble model (SP)	time-series data	prediction	No	DNN
[25] 2020	ensemble model (SP)	time-series data, image	weather forecast	No	DNN
[26] 2017	ensemble model (SP)	industrial Accident data	Early warning	No	DNN
[27] 2017	ensemble model (SP)	Monthly Inflow	Prediction	Yes	
Our	Integrated	graph data, image	classification/regression	No	DNN

Neuro-Fuzzy is a fuzzy inference system implemented in the framework of NN that plays a vital role in the neuro-fuzzy controller field. For example, the adaptive node contains parameters associated with learning through links by the gradient-based or recursive least square-based procedure under given training data. Furthermore, ANFIS [9] adopts Takagi-Sugeno rules [29], which is a compact and computationally efficient representation. The inference of a single rule can be viewed as a linear combination of the firing degree with the consequent functional (constant) term, i.e., the relation/fitting curve between input and output is limited to linear mapping.

In Section 1, we discussed the fundamental problems with using DNNs, GNNs, and fuzzy systems alone and the need for alternative methods to resolve these fundamental limitations to improve the performance and robustness. These alternative methods are fusion-based models that integrate NN with an appropriately chosen fuzzy system. There are various ways to combine a fuzzy system with a NN system, including ensemble models in a sequential or parallel pattern [20, 27]; or integrated models [19]. The ensemble models either preprocess the representation vector using specific fuzzy systems or deep learning (DL), and the fuzzy system transforms the given representation vector in parallel. On the contrary, the integrated models fuze a fuzzy system as a part of the learning mechanism. Many investigations have proven that using a fuzzy system with DL can improve the robustness and efficiency of the models where data are biased, noisy, or vague [2, 30, 31].

In recent decades, diverse real-world applications have adopted data-driven fuzzy models identified by distinct learning algorithms [22, 24, 32]; we refer to, for example, applications in computer vision and image processing [23], device control [33], and data management [25]. Combining usage with fuzzy logic systems improves the situation, which allows for dealing with uncertainties and ambiguities of real-world data [4].

Furthermore, to address the drawbacks of DL, such as lack of interpretability and tedious learning for distinct tasks [2], many researchers proposed models that adapt the fuzzy systems for DL. For example, in [34], the authors suggested using fuzzy numbers to present the training weights of network nodes in so-called fuzzy restricted Boltzmann machines that were used in applications, such as airline passenger profiling [35] and early warning systems for industrial accidents [26]. A similar approach to NFS is based on implementing an integration model that replaces the perceptrons in the network with fuzzy logic units [36]. El and Boumhidi [18] implemented a fuzzy system to train part of the controllable parameters of a deep neural network. Apart from these examples, researchers attempted to use a fuzzy system to address large-scale dataset training problems. Muhammad et al. [37] worked on the information fusion by employing (fuzzy) Choquet integral, which performs a nonlinear aggregation function as a multi-layer NN. However, this Choquet integral

was not embedded in the DL architecture. It worked as an extra process aggregating the multiple outputs from several individual neural networks.

However, only a few works studied made fusion on a fuzzy model with a graph neural network. Tong [12] proposed a novel meta-learning combining a GNN and a fuzzy system in few-shot learning termed FGNN. In this work, DL acted as a feature extractor and a message-passing mechanism, and the collaborating fuzzy systems acted as the relational representation learner by performing the edge connection with node features. This representation learner was designed with the help of Gaussian-shaped fuzzy sets. Finally, Deng et al. [11] proposed a hierarchical DL network architecture, FFDN, that derives parallel representations from a fuzzy channel and a neuro-channel. The fuzzy channel transforms a D-dimensional input vector into a D-dimensional vector of membership values using the Gaussian-shaped fuzzy sets.

GT, which we will adopt in our model, is currently one of the best-performing NN architectures for handling long-term sequential datasets such as sentences in natural language processing [17]. Deep learning modules of GT adopt the self-attention mechanism, differentially weight the significance of individual input data, and relieve these limitations (e.g., over-smoothing [38], expressiveness bounds) by allowing nodes to observe all other nodes in a graph in a global attention way [8]. We, therefore, get a more expressive model by adding more layers, which may lead to a model that treats all nodes as indistinguishable vectors, and simultaneously, with suppressed adverse effects of the high number of layers. Although various GT models improve the performance of DL, the mentioned challenges in this field remain unresolved. For example, GT models depend on structured graph data for learning [39], yet the real-world data is far more complex than the simplified aligned graph [40], and benchmarks are not sufficiently representative [41]. Thus to deal with a slight variation in a problem, a GNN model would require retraining entirely.

Despite successful research, fuzzy systems have not been extensively studied and applied in up-to-date GNN architectures. The challenges are the growth of potentially required training parameters that may cause high computational costs and the difficulties of simulating nonlinear systems that build relations between independent and dependent variables. This paper proposes an integrated model incorporating GNN and fuzzy systems for large-scale datasets. The proposed feature-level fusion model assisting feature learning adopts the representation sampling method to reduce the number of training parameters and uses various regression functions to simulate the relations between the input and the output.

3. Integrated Fusion of Fuzzy Learning and Graph Neural Network

This section introduces our proposed feature-level fusion of fuzzy learning and graph neural network: *fuzzy forest-based graph neural network* (FuzzyGNN). This feature-level fusion in FuzzyGNN assists the feature learning. FuzzyGNN adopts a fuzzy forest-based mapping layer, which has an extension architecture of ANFIS and comprises two sub-parts: forest sampling representation and fuzzy-based input-output mapping, where learned features by the fuzzy system will be fused with a neural network architecture for task-specific layer.

The framework of the proposed FuzzyGNN is shown in Fig. 2. FuzzyGNN starts processing the input data (either graph or image data) by encoding the data to node representations. For example, an atom of molecular data or one row of pixels of image data is represented by a single node. The original node embedding is a $\hat{D}$-dimensional vector denoted by $\mathbf{a} = (a_1, a_2, \dots, a_{\hat{D}})$. The node embedding is updated with positional and structural encoding, improving the expressivity and becoming the node representation. The v-th node representation at the current (initial) stage is denoted by $\mathbf{h}_v^0 = (x_1, x_2, \dots, x_D)$, where $v = 1, 2, \dots, V$, and V denotes the number of nodes in the given graph NN. Then the GT is applied to update the node representation $\mathbf{h}_v^0$ in GNN layers. The output of GNN layers is denoted by $\mathbf{h}_v^\ell$ presenting the v-th node representation at ℓ-th layer. Next, the node representation is updated by the fuzzy forest-based mapping layer, which will be introduced in detail in the subsequent sub-section. Finally, the task-specific layer analyzes the node representation for the final output.

The fundamental idea of the GNN is to learn the node representation that conveys the expressive and instinctive information in structured graph data, also known as representation learning [42]. The node representation is also the node feature. Two key issues related to integrating a fuzzy system into a GNN are (1) the curse of dimensionality leading to the exponential growth of fuzzy rules and (2) the potential inaccuracy in capturing the functional relationship between inputs and output. The first one is addressed by decomposing representation vectors of a long length into sub-vectors, dramatically reducing the number of the needed fuzzy rules. The latter issue involves various regression functions to model the input-output relationship. Such a fuzzy system integration pattern is applied to the node representation learning and the edge representation as well. The following sections address the hybrid learning rule, the solutions adopted to address the two problems mentioned above, and elaborate on the process.

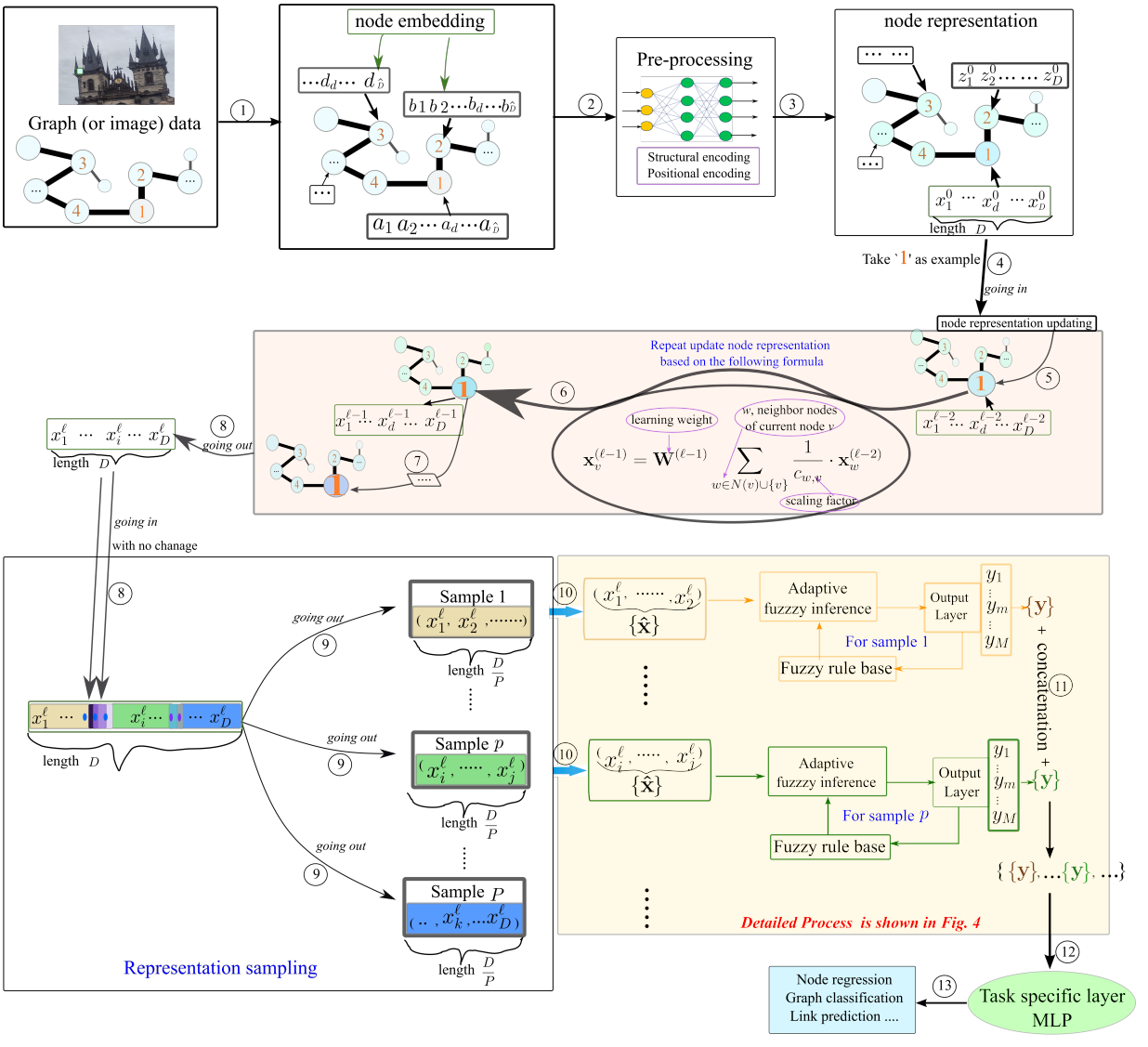


Figure 2: The proposed FuzzyGNN framework with its main component and NFS module. Vector $(x_1^\ell, \ldots, x_D^\ell)$ is the node representation in a hidden layer, $\{\hat{x}\}$ denotes the sampled node features (sub-set of node representation), $\mathbf{y} = (y_1, \ldots, y_M)$ is the output vector produced by the fuzzy output layer for a single sample.

3.1. Hybrid Learning Rule

The principle of ANFIS is not to simulate the dependent variables function. Instead, it learns the input space by dividing it into subspaces to make the output function linear. For example, Fig. 3 (left) presents the data points that are challenging to create a function that relates $\mathbf{X}$ and Y. In contrast, each subspace of a circle shows linear, parabola, or exponential relations on the right side of Fig. 3, which decreases the complexity of the problem.

The concept of fuzzy membership function from ANFIS is similar to a kernel function that maps the dependent variables to a new coordinate system facilitated for analysis. The critical part of this mapping is to choose a relation $\mathbf{Y} = F(\mathbf{X})$ for the dataset $(\mathbf{x}_k, \mathbf{y}_k)$ denoting attributes and labels, where $k = 1, \ldots, n$. And this relation may not be linear, as shown in Fig. 3 (right). The hybrid learning rule for ANFIS adopts least squares estimate to identify parameters that optimize/minimize the fitting error ϵ, which is defined as:

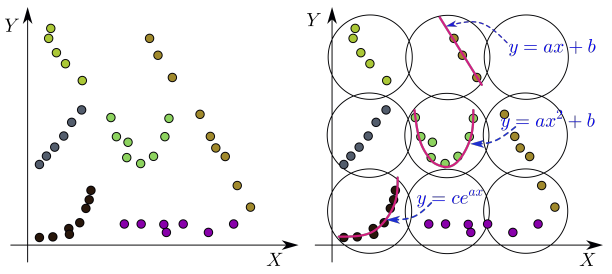


Figure 3: Sample data (left) and Corresponding fuzzy subspaces division (right) indicated by circles and linear, parabola, or exponential functions.

$$\varepsilon(A, B, \mathbf{x}_k, \mathbf{y}_k) = \sqrt{\frac{1}{n}\sum_{k=1}^{n} \|f(\mathbf{x}_k) - \mathbf{y}_k\|^2}, \quad f(\mathbf{x}_k) = A\mathbf{x}_k + B \tag{1}$$

or alternatively, we could re-write it as follows:

$$\varepsilon(A, B, \mathbf{x}_k, \mathbf{y}_k) = \sqrt{\frac{1}{n}\sum_{k=1}^{n} ((A\mathbf{x}_k + B) - \mathbf{y}_k)^2}\,, \tag{2}$$

where $\mathbf{x}_k$ denotes the k-th point in the space and $f(\mathbf{x}_k)$ is the fitted corresponding output, $\mathbf{y}_k$ is the observation output. Here A and B are the parameters/weights/solutions to the optimal fitting of the data.

The goal of the fitting learning curve is to minimize ε, with respect to the attributes A and B.

$$\min_{A,B} \varepsilon(A, B, \mathbf{x}_k, \mathbf{y}_k) = \min_{A,B} \sum_{k=1}^{n} \|A\mathbf{x}_k + B - \mathbf{y}_k\|^2\,. \tag{3}$$

The optimal solution (coefficient, element of a fitting curve function) is obtained by setting the partial derivative of the above function to zero, as follows:

$$\frac{\partial \varepsilon}{\partial A} = 0 \text{ and } \frac{\partial \varepsilon}{\partial B} = 0\,,$$

then we obtain the equations as follows:

$$\sum_{k=1}^{n} 2(A\mathbf{x}_k + B - \mathbf{y}_k)\mathbf{x}_k = 0 \text{ and } \sum_{k=1}^{n} 2(A\mathbf{x}_k + B - \mathbf{y}_k) = 0\,.$$

By simplifying the above equations, we get the following linear equation form:

$$\begin{pmatrix} \sum \mathbf{x}_k^2 & \sum \mathbf{x}_k \\ \sum \mathbf{x}_k & n \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} \sum \mathbf{y}_k \mathbf{x}_k \\ \sum \mathbf{y}_k \end{pmatrix}\,, \tag{4}$$

which is a classical problem of solving linear equations where A, B are two unknowns that produce the optimal weights A, B to fit the equations (fitting curve) in order to minimize ε of Eq. (1).

If we apply the same principle and fit a parabola verse to a line where $f(\mathbf{x}_k) = A\mathbf{x}_k^2 + Bx_k + C$ then the problem turns into the following minimization problem:

$$\min_{A,B,C} \varepsilon(A, B, C, \mathbf{x}_k, \mathbf{y}_k) = \min_{A,B,C} \sum_{k=1}^{n} \|A\mathbf{x}_k^2 + Bx_k + C - \mathbf{y}_k\|^2$$

The optimal solution to this minimization problem can be written as follows:

$$\begin{pmatrix} \sum \mathbf{x}_k^4 & \sum \mathbf{x}_k^3 & \sum \mathbf{x}_k^2 \\ \sum \mathbf{x}_k^3 & \sum \mathbf{x}_k^2 & \sum \mathbf{x}_k \\ \sum \mathbf{x}_k^2 & \sum \mathbf{x}_k & n \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} \sum \mathbf{y}_k \mathbf{x}_k^2 \\ \sum \mathbf{y}_k \mathbf{x}_k \\ \sum \mathbf{y}_k \end{pmatrix}, \tag{5}$$

which is similar to Eq. (4) is a classical problem of solving linear equations with three unknowns A, B, C, and which produces the optimal weights to the parabola curve fitting. This polynomial curve fitting can be generalized for polynomial functions of any degree (power) m: $f_{C_1,\dots,C_m}(\mathbf{x}_k)$. For this polynomial function of degree m, the minimization of ε with respect to the attributes $C_1, \dots, C_m$ take the the follows form:

$$\min_{C_1,\dots,C_m} \varepsilon(C_1, \dots, C_m, \mathbf{x}_k, \mathbf{y}_k) = \min \sqrt{\sum_{k=1}^{n} \| f(\mathbf{x}_k, C_1, \dots, C_m) - \mathbf{y}_k \|^2} \, .$$

The optimal solution of this minimization problem is obtained as follows:

$$\frac{\partial \varepsilon}{\partial C_j} = 0, \quad j = 1, 2, \dots, m \, ,$$
$$\sum 2(f(\mathbf{x}_k, C_1, C_2, \dots, C_m) - \mathbf{y}_k) \frac{\partial \varepsilon}{\partial C_j} = 0 \, .$$

However, taking higher degrees of polynomials does not always improve the curve-fitting accuracy. This is typically known as the polynomial wiggle problem [43]. Therefore, we only test the power of 2 polynomial functions as the division subspace is set to be a smaller value in the experiment.

3.2. Representation Sampling

The reason for proposing representation sampling is to decrease the number of fuzzy rules determining the number of trainable parameters in the NFS module. The representation sampling operates after the GNN layers on the GNN output node representation $\mathbf{h}_v^\ell$. And then, $\mathbf{h}_v^\ell$ serves as the input to the representation sampling. The representation sampling comprises two steps. In particular, it starts with a representation division that divides a long vector into multiple sub-vectors; then, each sub-vector is mapped into a *representation sample*.

First, let us consider a situation where the node representation is being updated without being divided into several sub-sectors. For the sake of simplicity, let us consider only a two-dimensional representation vector and let two fuzzy sets of two sub-areas cover each axis (dimension). Then the given NFS model creates four fuzzy rules dividing the two-dimensional input space into a "grid" (in a broader sense) of four subareas with unsharp (fuzzy) boundaries [44]. Generally, if the input is a D-dimensional vector $\mathbf{h}_v^\ell = (x_1^\ell, x_2^\ell, \dots, x_D^\ell)$, and the number of fuzzy sets on each axis is N (**user-defined parameter**) then there will be N^D of fuzzy rules for dividing the D-dimensional input space into N^D subareas. As a single rule covers each subarea, we talk about the so-called *curse of dimensionality* or *exponential explosion of rules*, and it is the primary source of the computational complexity burden. Indeed, relevant works focus on this issue, e.g., [45]. However, paper [45] adopts an adaptive number of rules during processing, which is quite challenging to be used in NN.

As stated above, the inference of a single rule can be viewed as a linear combination of the firing degree with the consequent functional term, which is usually a linear function, so we need, in general, $D + 1$ parameters for each consequent, respectively. However, as we consider the output to be a more-dimensional vector and not a single value, the overall output for a single input sample can be denoted by $\mathbf{y} = (y_1, \dots, y_m, \dots, y_M)$ where M is the fixed parameter denoting the required output dimensionality chosen by the user based on the complexity of the problem, see Fig. 2. As a result, the number of parameters in the consequent layer is $N^D \times (D + 1) \times M$, which determines the computational burden for this FuzzyGNN and demonstrates that the exponential explosion makes the system most sensitive to the parameters N and D.

In order to deal with this large number of parameters issue in the NFS, the fuzzy forest-based mapping layer adopts the representation sampling, which divides a node representation into multiple sub-set representations that are called *samples*. This sampling is similar to the multi-head scheme in the graph transformer attention mechanism. The difference between our sampling and multiple heads lies in the division pattern, in which we provide uniform sampling and random sampling options.

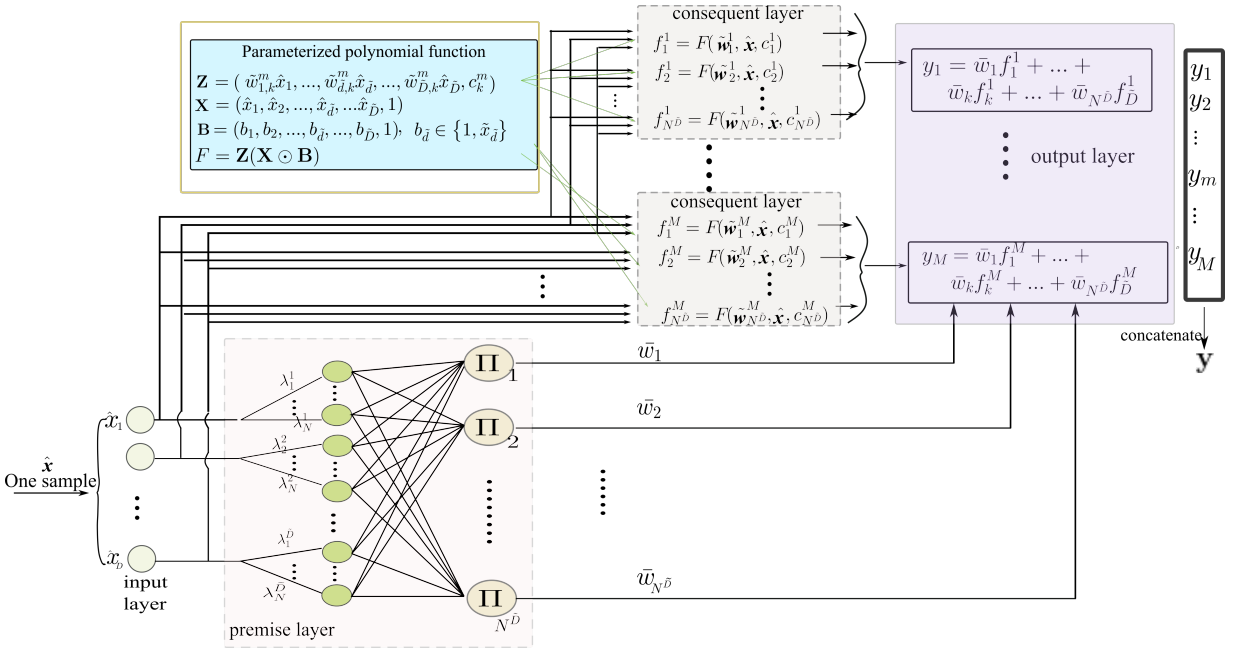


Figure 4: Proposed neuro-fuzzy fusion. In this NFS module, the blue window introduces a parameterized polynomial function that models the relationship between independent and dependent variables in the consequent layer.

Suppose we are given a D-dimensional vector node representation $\mathbf{h}_v^\ell = (x_1^\ell, x_2^\ell, \dots, x_D^\ell)$ and the division number is set (user's choice) to $P \in \mathbb{N}$. We define the representation division in the *uniform sampling* as follows:

Division 1 contains $\{x_1^\ell, x_2^\ell, \dots, x_{\frac{D}{P}}^\ell\}$.

Division 2 contains $\{x_{\frac{D}{P}+1}^\ell, x_{\frac{D}{P}+2}^\ell, \dots, x_{2\frac{D}{P}}^\ell\}$.

$\dots, \dots,$

Division $\frac{D}{P}$ contains $\{x_{D-\frac{D}{P}+1}^\ell, \dots, x_{D-1}^\ell, x_D^\ell\}$;

where division number P has been chosen in such a way that fraction $\frac{D}{P}$ is an integer.

When the *random sampling* is considered, we randomly divide the node representation into P subsets. The representation division may contain a discontinued feature sequence in such a case. For example, such a sample may be a vector $(x_2^\ell, x_5^\ell, \dots, x_D^\ell)$ or any other assuming it is of the length $\frac{D}{P}$.

After the division, the second step of representation sampling consists of passing each division into a dimension-transforming mapping to obtain the required *sample*. This mapping function serves as the dimensional conversion; it maps a $\frac{D}{P}$ dimensional division vector to the input vector of the subsequent NFS with a required dimensional. In particular, if the NFS input dimension $\tilde{D}$ (**user-specified parameter**) equals $\frac{D}{P}$, the mapping becomes an identity mapping. Otherwise, there will be a mapping that will remove a number of $(\frac{D}{P} - \tilde{D})$ randomly chosen elements from the given division. This mapping requires $\tilde{D} \leq \frac{D}{P}$. The output of the sampling is called a *sample* and it is a vector $\hat{\mathbf{x}}$ consisting of $\tilde{D}$ components $\hat{x}_{\tilde{d}}$, where $\tilde{d} = 1, \dots, \tilde{D}$. The sample passes to the neuro-fuzzy system module that is described in detail in the subsequent subsection and depicted in Fig. 4.

It is worth mentioning that there can be an optional linear mapping layer between the input $\mathbf{h}_v^\ell$ and the representation sampling. When $\frac{D}{P}$ is not an integer, this optional layer is used to convert the D-dimensional vector to a vector whose dimension can be divided by the sampling size P.

3.3. Proposed Neuro-fuzzy System Module (FuzzyGNN Architecture)

This module assists feature learning by fusing the fuzzy system learned features with features in neural architectures. The *neuro-fuzzy system module* (i.e., NFS module) transforms each of provided P samples into a vector $\mathbf{y} = (y_1, y_2, \dots, y_M)$, where M is a **user-defined neuro-fuzzy module output parameter**. Finally, all results samples are concatenated into a single representation that is a $(P \times M)$-dimensional vector denoted by $\mathbf{h}_v^{\ell+1}$, see Fig. 4.

Firstly, the input $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{\tilde{D}})$ from the sampling is passed to the premise layer. Then each element $\hat{x}_{\tilde{d}}$ is transformed into a membership value by a fuzzy set $\lambda_n^{\tilde{d}}$. There are $N \cdot \tilde{D}$ antecedent fuzzy sets denoted by $\lambda_n^{\tilde{d}}$, where N and $\tilde{D}$ are user-defined parameters, fuzzy sets in each axis, and input data dimension to NFS, respectively. The fuzzy sets $\lambda_n^{\tilde{d}}$ is defined as follows:

$$\lambda_n^{\tilde{d}}(\hat{x}_{\tilde{d}}) = exp\left(-\frac{(\hat{x}_{\tilde{d}} - \mu_n^{\tilde{d}})^2}{2(\sigma_n^{\tilde{d}})^2}\right), \tag{6}$$

where $\tilde{d} = 1, 2, \dots, \tilde{D}$ and $n = 1, \dots, N$.

This layer encodes a number of $N^{\tilde{D}}$ antecedents of fuzzy rules that are given by the Cartesian product of $\tilde{D}$ fuzzy sets, one on each axis. For instance, $\Pi_1 = \lambda_1^1 \times \lambda_1^2 \times \cdots \times \lambda_1^{\tilde{D}}$, $\Pi_2 = \lambda_2^1 \times \lambda_1^2 \times \cdots \times \lambda_1^{\tilde{D}}$, and so on until the last $\Pi_{N^{\tilde{D}}} = \lambda_N^1 \times \lambda_N^2 \times \cdots \times \lambda_N^{\tilde{D}}$. The *firing degree* of a particular rule for a given input is then obtained as the membership degree of this input in the antecedent of rule, for example, $\Pi_1(\hat{\mathbf{x}}) = \lambda_1^1(\hat{x}_1) \times \lambda_1^2(\hat{x}_2) \times \cdots \times \lambda_1^{\tilde{D}}(\hat{x}_{\tilde{D}})$ which can be easily computed using formula (6).

Secondly, a consequent layer models the relation between the input and the output in each "subarea" that is determined by the particular $\tilde{D}$-tuple of antecedent fuzzy sets from their Cartesian product.

The task for the consequent layer is to capture or approximate a given relationship between the input variables $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{\tilde{D}})$ and the consequent functional expressions $f_k^m = F(\tilde{\mathbf{w}}_k^m, \hat{\mathbf{x}}, c_k^m)$ for $k = 1, \dots, N^{\tilde{D}}; m = 1, \dots, M$, and so, each f_k^m corresponds to a particular rule, where $\tilde{\mathbf{w}}_k^m = (\tilde{w}_{1,k}^m, \dots, \tilde{w}_{\tilde{D},k}^m)$. Then the function F is defined as a parameterized polynomial function defined as follows:

$$\begin{aligned} F &= \mathbf{Z}(\mathbf{X} \odot \mathbf{B}) \,, \\ \mathbf{Z} &= (\tilde{w}_{1,k}^m \hat{x}_1, ..., \tilde{w}_{\tilde{d},k}^m \hat{x}_{\tilde{d}}, ..., \tilde{w}_{\tilde{D},k}^m \hat{x}_{\tilde{D}}, c_k^m) \end{aligned}$$

where $\odot$ is element-wise multiplication of vectors, $\mathbf{Z}$ is a vector derived from trainable weight $\tilde{\mathbf{w}}_k^m$ and $\hat{\mathbf{x}}$, $\mathbf{X}$ is an extension of $\hat{\mathbf{x}}$ given by $\mathbf{X} = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{\tilde{D}}, 1)$, and $\mathbf{B} = (b_1, b_2, \dots, b_{\tilde{d}}, \dots, b_{\tilde{D}}, 1)$ with its components $b_{\tilde{d}} \in \{\frac{1}{\hat{x}_{\tilde{d}}}, 1\}$. The user specifies function F by changing the vector $\mathbf{B}$, which is determined by elements $b_{\tilde{d}}$ that are equal to '1' or to $\frac{1}{\hat{x}_{\tilde{d}}}$. For example, if $b_{\tilde{d}} = \frac{1}{\hat{x}_{\tilde{d}}}$ for any $\tilde{d} = 1, \dots, \tilde{D}$, i.e., we get $\mathbf{B} = (\frac{1}{\hat{x}_1}, \dots, \frac{1}{\hat{x}_{\tilde{d}}}, \dots, \frac{1}{\hat{x}_{\tilde{D}}}, 1)$, then $\mathbf{X} \odot \mathbf{B} = \mathbb{1}$ (a vector with all elements of '1'), and consequently, we get $F = \mathbf{Z}\mathbb{1} = \tilde{w}_{1,k}^m \hat{x}_1 + ... + \tilde{w}_{\tilde{d},k}^m \hat{x}_{\tilde{d}} + ... + \tilde{w}_{\tilde{D},k}^m \hat{x}_{\tilde{D}} + c_k^m$. On the other hand, if $b_{\tilde{d}} = 1$ for any $\tilde{d} = 1, \dots, \tilde{D}$ then we get $F = \mathbf{ZX}$ which produced the following function $F = \tilde{w}_{1,k}^m \hat{x}_k^2 + ... + \tilde{w}_{\tilde{d},k}^m \hat{x}_{\tilde{d}}^2 + ... + \tilde{w}_{\tilde{D},k}^m \hat{x}_{\tilde{D}}^2 + c_k^m$.

Note that the linear consequent function, i.e., the polynomial, is given $\mathbf{X} \odot \mathbf{B} = \mathbb{1} = (1, \dots, 1)$ for $F = \mathbf{Z}\mathbb{1}$, that is, the traditional ANFIS method, may not be sufficiently powerful to capture highly non-linear relationships between inputs and outputs. Indeed, though as proclaimed in [45] that linear regressions hold on very small subareas determined by the antecedents of the rules and the overall output is non-linear, this view requires the sufficient size reduction of the subareas and consequently the increase of the total number of rules, which is exactly in the opposite direction of our intention to reduce the already high complexity. We have to take into account that whenever ANFIS is used to deal with graph NN, its dimensionality is much higher than in standard regression problems; therefore, we apply the sampling beforehand. Generally, we write the output of the k-th rule as given by $f_k^m = F(\tilde{\mathbf{w}}_k^m, \hat{\mathbf{x}}, c_k^m)$ and the total number of trainable parameters is $N^{\tilde{D}} \times (\tilde{D} + 1)$.

As soon as the individual outputs of rules f_k^m are determined, the NFS module in its output layer generates the final output vector $(y_1, \dots, y_M)$ with its components given by the following formula:

$$y_m = \sum_{k=1}^{N^{\tilde{D}}} \bar{w}_k f_k^m \,, \quad \text{for } m = 1, 2, \dots, M, \tag{7}$$

Table 2
The machine learning dataset description, abbreviations: class. – classification; regr. – regression; MAE – mean absolute error.

Dataset	Task – Prediction	class#	Evaluation	Description
MNIST	class.– graph	10	Accuracy	70K graph
CIFAR10	class. – graph	10	Accuracy	60K graph
ZINC	regr. – graph	-	MAE	12K graph
PATTERN	class. – node	2	Accuracy	14K graph
CLUSTER	class. – node	6	Accuracy	12K graph

where the weight values are determined as follows:

$$\bar{w}_k = \frac{\Pi_k}{\sum_{k=1}^{N^{\tilde{D}}} \Pi_k}$$

in order to ensure their normalization $\bar{w}_1 + \bar{w}_2 + ... + \bar{w}_{N^{\tilde{D}}} = 1$.

The impact on the computational complexity is as follows. The function F determines the output f_k^m using the linear combination of the components of the input vector $\hat{\mathbf{x}}$ and of the weight vector $\tilde{\mathbf{w}}_k^m$. Consider a hypothetical yet real example with a 52-dimensional node representation and three fuzzy sets on each axis that would lead to 3^{52} rules. The curse of dimensionality would lead to $3^{52} \times 53 = 3.4 \times 10^{25}$ parameters for the consequent layer of the NFS module without apriori applied to split the vector representations. If we use splitting of the vectors to 13 samples of the length 4, the total number of the parameters of the consequent layer of the NFS module would be given by $N^{\tilde{D}} \times (\tilde{D} + 1) \times P$ that is $3^4 \times (4 + 1) \times 13 = 5265$.

4. Experiments and Results

The section conducts experiments on five diverse machine learning datasets to evaluate the proposed FuzzyGNN model's effectiveness and compares the SOTA results.

4.1. Training related setting

We take three graph datasets, ZINC, PATTERN, CLUSTER, and two image datasets, CIFAR10 and MNIST, from Benchmarking GNNs [46][47] by `torch_geometric` [48]. The details of the machine learning datasets are shown in Table 2. The dataset splits follow the standard provision defined for train/valid/test data splits. The experiments execute ten runs with the same parameter setting and random seed. Afterward, we calculate the average performance (Avg.) and its standard deviation (Sd.) from these 10 runs for the global performance evaluation. The parameter size follows a 500K budget for ZINC, PATTER, and CLUSTER; and a 100K budget for image datasets MNIST and CIFAR10. The AdamW optimizer [49] with the 'warm-up' strategy for learning rate in default parameters are used in all experiments, i.e., $\beta_1, \beta_2, \epsilon$ are 0.9, 0.999, and 1e-8, respectively. The learning rate is set to the initial value of 0.001 and the 'Weight decay' value of 1e-5. The learning rate starts with an initial value at the early stage; then, it is modified in a cosine trend. The maximal number of epochs is set to 2000 for ZINC and 100 for the other dataset. The batch size is set to 32 for ZINC and PATTERN and 16 for the rest of the datasets. The feature dimension hyperparameters follow the adjusted suggestions from GPS [47] and GraphTransformer [50]. The warm-up setting follows the hyperparameters choices in [47]. FuzzyGNN is implemented under the framework of GPS with PyG, and its GraphGym [51] module supports Python under an MIT license. All experiments were conducted in a mixed computing environment, a cluster of 4 NVIDIA GeForce GPU 8GB Memory, 2 Tesla V100-PCIE 32GB GPU with intel Xeon GOLD-6154 CPU.

FuzzyGNN provides two representation samplings, Random and Uniform. For each dataset, We experiment with three different F consequent functions with first-order, mixed-order, and second-order equations, which leads to six combinations as Random- first-order (R(0)), Random-second-order(R(1)), Random-mixed-order (R(0.5)), Uniform-first-order(U(0)), Uniform-second-order(U(1)), and Uniform-mixed-order(U(0.5)). Finally, we summarize all of the experiment results and present them in the next subsection.

Table 3
FuzzyGNN hyperparameters in five machine learning datasets. GTF-Graph transformer

Parameter	PATTER	CLUSTER	ZINC	MNIST	CIFAR10
Frame	GTF	GTF	GTF	GTF	GTF
Hidden dim	64	48	64	52	52
# Sample (P)	10	12	16	13	13
$\tilde{D}$	4	4	4	4	4
M	2	2	2	2	2
N	3	3	3	3	3

Table 4
The parameters and GNN-related setting, GlobAttn-global attention technique, GateGCN [52], GINE [53], Transformer [50, 54, 55].

Model	Parameter	PATTERN	CLUSTER	ZINC	MNIST	CIFAR10
GPS	# Epochs	100	100	2000	100	100
FuzzyGNN		100	100	2000	100	100
GPS	MPNN	GatedGCN	GatedGCN	GINE	GatedGCN	GatedGCN
FuzzyGNN		GatedGCN	GatedGCN	GINE	GatedGCN	GatedGCN
GPS	GlobAttn	Transformer	Transformer	Transformer	Transformer	Transformer
FuzzyGNN		Transformer	Transformer	Transformer	Transformer	Transformer
GPS	# Parameters	337,201	502,054	423,717	115,394	112,726
FuzzyGNN		347971	515350	438,813	127352	124,756

4.2. Technical details of the experimental justification

We provide the GNN training-related setting in Table 4. We only count the number of trainable parameters and ignore the non-trainable parameters. For example, during the representation sampling process, the parameter on the selection of random variables for creating samples is set to be fixed when initializing the model, which means it is a non-trainable parameter. Likewise, the parameter required in the selection of variables for the mixed-order regression function is also non-trainable. Table 4 shows that FuzzyGNN uses slightly more parameters than the baseline GPS model. We choose graph isomorphism network with edges (GINE) [53] as the message-passing updating technique on the ZINC molecular dataset, and we apply gated graph convolutional network [52] as the message-passing updating technique to the other dataset. Those MPNN settings follow GPS. The global attention technique adopts graph transformer [50, 54, 55]. The parameters required for the NFS module of the FuzzyGNN are presented in Table 3. The parameter of vector dimension in node representation follows the suggestions from [50, 47]. The number of samples P varies in different datasets, and we set the output vector dimension M to be equal to 2; the number of fuzzy sets used on each axis is set to 3 for all datasets. Other activation function-related parameters and the parameter dropout are provided in Appendix.

4.3. Results and Analysis on large dataset

We compare FuzzyGNN with a set of benchmark GNNs, namely with GCN [56], GAT [57], GatedGCN [52], GT [50], and with several SOTA methods, namely with DGN [58], EGT [60], ARGNP [61], and GPS [47]. The results with the best performance out of the six settings (R(0), R(0.5), R(1), U(0), U(0.5), U(1)) of FuzzyGNN is provided in Table 5, where each result runs ten times independently (Avg.$\pm$ Sd. are provided).

FuzzyGNN performs the best on the data CLUSTER and CIFAR10, producing 78.25%, and 74.79% accuracy, respectively. There is an obvious accuracy outperformance of the baseline model GPS of 0.3%, 2.49% on CLUSTER and CIFAR10. The performance of FuzzyGNN on ZINC data does not show a significant privilege. As a result, it is equal to the one provided by GPS. On the PATTERN and MNIST datasets, there is a slight accuracy increase in favor of the FuzzyGNN. Though this tiny accuracy increase can be neglected, the very low standard deviation provides a strong argument in favor of the FuzzyGNN justified by very high robustness and, consequently, the reliability/stability of the FuzzyGNN approach.

Table 5
Performance on Test data in five datasets from [46] Shown as Avg./Sd. of 10 runs. The results (compared) are taken from [46, 47]. The **best** results are highlighted in boldface numbers, and the *second-best results are highlighted in italics. The results by FuzzyGNN are the best-performed ones by using different consequent functions* F.

Model	PATTERN	CLUSTER	ZINC	MNIST	CIFAR10
	Accuracy ↑	Accuracy ↑	MAE ↓	Accuracy ↑	Accuracy ↑
GCN [56]	71.89 ± 0.33	68.50 ± 0.97	0.37 ± 0.01	90.70 ± 0.22	55.71 ± 0.38
GAT [57]	78.27 ± 0.186	70.59 ± 0.45	0.38±0.007	95.53 ± 0.20	64.22 ± 0.45
GatedGCN [52]	85.57 ± 0.09	73.84 ± 0.33	0.28 ± 0.01	97.34 ± 0.14	67.31 ± 0.31
Graphormer [55]	-	-	0.12 ± 0.01	-	-
DGN [58]	86.68 ± 0.03	-	0.17 ±0.003	-	72.84 ± 0.42
GT [50]	84.81 ± 0.07	73.17 ± 0.62	0.23 ± 0.01	-	-
CIN [59]	85.39± 0.14	64.72 ± 1.55	0.53 ± 0.05	96.48± 0.25	55.25 ± 1.53
EGT [60]	86.82 ± 0.03	**79.23 ± 0.35**	0.11 ± 0.01	-	-
ARGNP [61]	-	77.35± 0.05	-	-	*73.90± 0.15*
GPS [47]	*90.32 ± 0.13*	77.95 ± 0.30	*0.07 ± 0.004*	98.05 ± 0.13	72.30 ± 0.36
FuzzyGNN (ours)	**90.42 ± 7.7e-4**	*78.25 ± 0.18*	*0.07 ± 0.003*	**98.07 ± 0.001**	**74.79 ± 0.35**

Table 6
Performance on Test data in five datasets from [46] by FuzzyGNN. The **best** results are highlighted by boldface letters. The results are the Avg./Sd. of 10 runs.

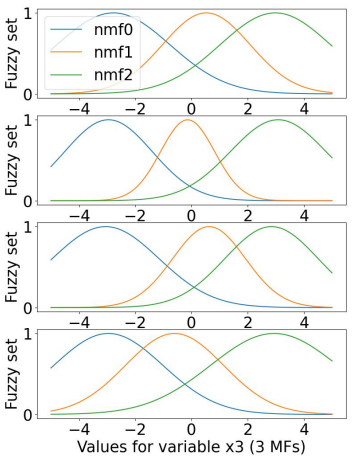
Algorithms	PATTERN	CLUSTER	ZINC	MNIST	CIFAR10
	Accuracy ↑	Accuracy ↑	MAE ↓	Accuracy ↑	Accuracy ↑
FuzzyGNN (R(0))	90.37 ± 2.17E-03	78.17 ± 0.23	**0.0704 ± 0.003**	98.06 ± 1.09E-03	74.62 ± 3.25E-03
FuzzyGNN (R(0.5))	90.35 ± 8.80E-04	**78.25 ± 0.18**	0.0729 ± 0.002	98.01 ± 1.15E-03	74.35 ± 0.51
FuzzyGNN (R(1))	90.33 ± 3.22E-03	78.14 ± 0.12	0.0709 ± 0.003	**98.07 ± 1.15E-03**	**74.79 ± 0.35**
FuzzyGNN (U(0))	90.29 ± 1.39E-03	78.16 ± 0.21	0.0756 ± 0.004	98.04 ± 1.47E-03	74.60 ± 2.57E-03
FuzzyGNN (U(0.5))	90.31 ± 1.30E-03	78.21 ± 0.12	0.0707 ± 0.002	98.04 ± 4.77E-04	74.43 ± 0.52
FuzzyGNN (U(1))	**90.42 ± 7.66E-04**	78.21 ± 0.14	0.0722 ± 0.004	97.99 ± 1.48E-03	74.52 ± 0.26

Table 7
The training time costs by FuzzyGNN with different consequent functions F, in particular, 'Avg Epoch Time cost/Total Time cost.'

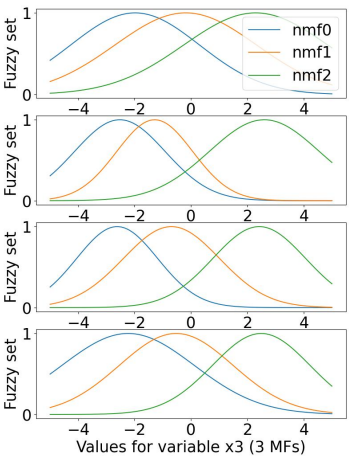
		FuzzyGNN					
Data		R(0.5)	R(0)	R(1)	U(0.5)	U(0)	U(1)
CLUSTER	Time	276.59s/7.69h	262.76s / 7.30h	262.79s/7.3h	265.97s/7.39h	267.85s/7.44h	267.82s/7.44
PATTERN	epoch/total	473.43s/13.15h	437.64s/12.16h	473.90s/13.17h	473.87s/13.16h	446.20s/12.39h	466.52s/12.959h

Furthermore, we present the detailed six experiments with different F functions of FuzzyGNN for five datasets. All the results are run with the same experimental setting. Table 6 shows the results for R(0), R(0.5), R(1), U(0), U(0.5), and U(1) of FuzzyGNN, where all sets of experiment were run ten times independently. Table 6 shows on ZINC data that the R(0) setting performs the best, and U(0.5) and R(1) are the second and third best. The setting with random sampling type (R(0.5)) demonstrates the best accuracy on the CLUSTER dataset, followed by the second best of uniform sampling by the (U(0.5)) and (U(1)). Lastly, the best-performing one on both MNIST and CIFAR10 datasets is the R(1) with parameter. In general, the function with regression performs slightly better than other values in most instances.

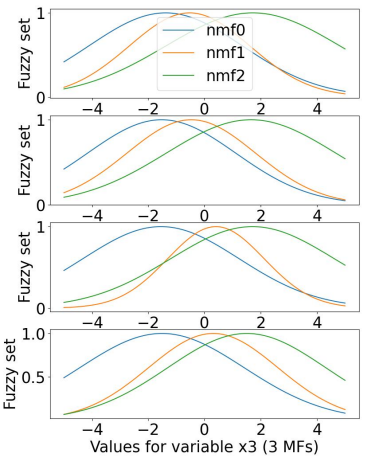
Table 7 presents the experiments on the computational time requirements. All the experiments were performed on the same hardware configuration. The presented time requirements represent the average time consumption of ten



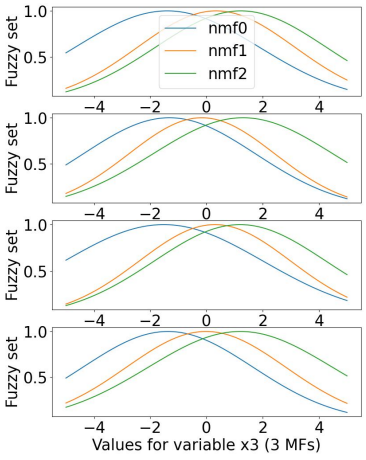
(a) Fuzzy set for MNIST



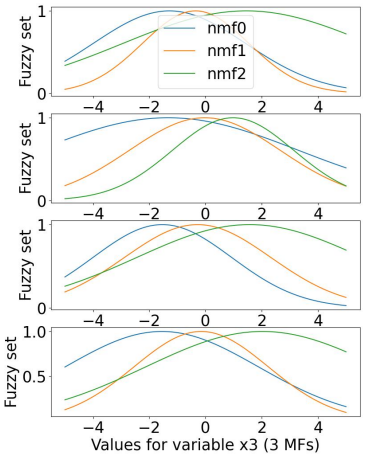
(b) Fuzzy set for CIFAR10



(c) Fuzzy set for CLUSTER



(d) Fuzzy set for PATTERN



(e) Fuzzy set for ZINC

Figure 5: The trained fuzzy sets for each dataset provided by the corresponding best-performing variant (see Table 6), each subfigure plots the fuzzy sets for four different variables of one (representation) sample, where MFs denotes the number of fuzzy sets.

runs of the experiments. From Table 7 we can conclude that there is no huge difference in computational time costs on various regression functions F.

4.4. Ablation study

This section conducts the ablation study to compare the performance of the baseline model with the one fused with the proposed neuro-fuzzy module. The baseline model is the Graph attention network (GAT), the basis of the Transformer, part of a module in our proposed FuzzyGNN.

$$\text{Confusion Matrix} = \begin{pmatrix} \text{True Positives}(TP) & \text{False Positives}(FP) \\ \text{False Negatives }(FN) & \text{True Negatives}(TN) \end{pmatrix}$$

Table 8
Graph dataset attributes

Name	Graphs	Classes#	Avg. Nodes	Avg. Edges	Evaluation
DD	1178	2	284.32	715.66	Accuracy
NCI1	4110	2	29.87	32.30	Accuracy
OHSU	79	2	82.01	199.66	Accuracy
Peking_1	85	2	39.31	77.35	Accuracy
PROTEINS	1113	2	39.06	72.82	Accuracy
IMDB-BINARY	1000	2	19.77	96.53	Accuracy
IMDB-MULTI	1500	3	13.00	65.94	Accuracy
SYNTHETIC	300	2	100.00	196.00	Accuracy
SYNTHETICnew	300	2	100.00	196.25	Accuracy

Table 9
Ablation study results of average, max, standard deviation Accuracy in % (avg-acc ↑, max-acc ↑, std-acc ↓), Precision ↑, F1 ↑, recall ↑ and AUC ↑ on eight datasets by baseline model ACAT and the AGAAT with the proposed fuzzy module as AGAT+Our. The **bold** text denotes the better result.

Dataset	Method	time-epoch	params	avg-acc	max-acc	std-acc	Precision	recall	F1	AUC
DD	AGAT	4.12E-01	5.01E+03	74.20	76.30	2.54E-02	74.70	54.20	62.10	80.30
	AGAT+Our	7.65E-01	7.59E+03	**78.00**	**82.00**	2.89E-02	**79.90**	**61.00**	**68.70**	**83.00**
IMDB-BINARY	AGAT	3.45E-01	3.60E+03	63.50	66.80	1.93E-02	63.60	63.60	63.50	64.50
	AGAT+Our	5.46E-01	6.18E+03	**65.10**	**67.20**	1.77E-02	**66.60**	61.20	63.60	**69.70**
IMDB-MULTI	AGAT	4.92E-01	3.64E+03	36.90	41.60	2.74E-02	-	-	32.00	57.80
	AGAT+Our	8.11E-01	6.22E+03	**38.90**	**46.90**	4.82E-02	-	-	32.80	55.70
NCI1	AGAT	2.97E-01	4.18E+03	73.20	73.60	3.57E-03	72.90	72.90	72.80	80.20
	AGAT+Our	2.27E+00	6.76E+03	**74.20**	**74.80**	5.52E-03	**73.70**	**74.30**	**73.90**	**81.00**
OHSU	AGAT	2.31E-01	6.63E+03	57.50	65.00	5.59E-02	45.80	40.00	42.30	60.80
	AGAT+Our	5.76E-02	9.21E+03	**60.00**	65.00	3.54E-02	**63.20**	**62.50**	**60.00**	**63.50**
Peking_1	AGAT	4.01E-01	6.63E+03	58.00	63.60	7.45E-02	33.00	50.00	39.30	56.00
	AGAT+Our	8.25E-02	8.31E+03	**65.90**	**72.70**	6.82E-02	**53.00**	**78.10**	**62.60**	**73.40**
PROTEINS	AGAT	8.46E-02	3.64E+03	72.00	74.90	2.53E-02	69.70	54.30	60.10	76.60
	AGAT+Our	1.03E+00	5.32E+03	**73.80**	74.90	1.66E-02	**71.30**	**57.70**	**63.60**	**77.50**
SYNTHETIC	AGAT	3.34E-01	3.72E+03	49.70	50.70	5.78E-03	12.70	25.00	16.80	49.20
	AGAT+Our	4.91E-01	6.30E+03	**50.00**	50.70	6.67E-03	**25.30**	**50.00**	**33.60**	**51.90**
SYNTHETICnew	AGAT	1.47E-01	3.59E+03	54.00	60.00	4.85E-02	52.90	44.10	46.30	57.70
	AGAT+Our	4.75E-01	5.35E+03	**55.00**	60.00	4.46E-02	42.10	**57.90**	**48.20**	**58.10**

$$\text{Accuracy} = (\text{number of correct predictions})/(\text{total number of predictions})$$

$$\text{Precision} = TP/(TP + FP)$$

$$\text{recall} = TP/(TP + FN)$$

$$\text{F1} = 2 \times (\text{Precision} \times \text{recall})/(\text{Precision} + \text{recall})$$

The baseline model architecture starts with one layer of feature encoder with a Laplacian positional encoding, then forwards to one layer of GAT with extra BatchNorm and dropout operations, and ends with one task-specific layer of MLP. This baseline model is a standard lightweight three-layer network termed Ablation-GAT (AGAT). The fused model is to add the proposed fuzzy inference module before the task-specific layer making it a four-layer network, termed Ablation-GAT-fuzzy (AGAT+Our). The experimental setting is set the same for the two compared models. The positional encoding dimension is set two. The activation function uses Relu, the aggregation mode uses the Mean function, and the dimension of the feature node is set to 18 for all of the tested datasets. The attention dropout rate is

0.5, and the number of heads in GAT is two. We use Adam with a base learning rate of 0.001 for the optimizer, and the max epoch for training is 100.

Dataset introduction: The dataset is chosen from TUDatasets https://chrsmrrs.github.io/datasets/, which collects datasets in fields of bioinformatics, Social networks, and Synthetic for graph classification and regression. Specifically, the detailed information of the dataset is shown in Table 8. The experiment runs four independent times by AGAT and AGAT+Our model on each dataset and calculates the average, best, and Sd accuracy results. Additionally, we choose datasets with two classes to easily measure the F1, AUC (Area Under the Receiver Operating Characteristic Curve), Precision, and recall metrics; a confusion matrix for measuring two classification classes is shown below. As Accuracy is particularly useful when the classes are balanced. In our unbalanced cases, F1, AUC, Precision, and recall metrics can give good overall assessments of the model's performance. A perfect model would have an AUC of 1, an Accuracy of 1, and with high value for F1 and recall, while a lower value for these metrics would mean a poor model.

Table 9 shows the ablation study result. AGAT presents the baseline results, and AGAT+Our presents the AGAT model embedded with one layer of our proposed fuzzy inference module. In Table 9, AGAT+Our out-performed AGAT on all datasets by average accuracy, with the largest accuracy increase of 7.9 % on Peking_1 data and the smallest accuracy increase of 1% on two datasets. There are 20%, 17.4%, and 12.6%, Precision increase on Peking_1, HSU, and SYNTHETIC datasets, respectively. There are 28%, 22.5%, 13.8%, and 25%, recall metric measurements increase on Peking_1, HSU, SYNTHETICnew, and SYNTHETIC datasets, respectively. We can observe that on F1 and AUC, there is also a large performance improvement on seven datasets out of eight. From the above results, we can conclude that the proposed fuzzy module has promising performance that can improve the overall model in Accuray and other metrics.

4.5. Comparison with FFDN

In this section, we compare the performance of the proposed model with the existing fuzzy model, FFDN [11]. Let us remind you that FFDN adopts a fuzzy inference module while it belongs to type-3 cooperative neural fuzzy systems; refer to Fig. 1. We did not compare FGNN [12] as this model is tailored to a few-shot-learning problem.

The torch source code of FFDN was downloaded from https://github.com/phamtheanhphu/FFDN_torch (original invalid) OR https://github.com/Zhangxy8244/FFDN_torch (valid), implementation of Tensorflow/Keras version https://github.com/sam-nayak/SynthNet. And we adapted the code into our framework to work on the chosen dataset. We did not run FFDN on large datasets, such as MINIST, as this implementation is slow. The environment settings on datasets of Table 8 are the same for the compared algorithms.

Table 10 shows the comparison results of FFDN with the proposed model. From Table 10, we observe that FFDN operates slowly in an epoch time and obtains worse accuracy results on seven instances out of eight datasets. FFDN performs well on SYNTHETIC data with 50.33% accuracy, slightly better than the proposed method with 50% accuracy. FFDN achieves 60% accuracy on the DD dataset, while the proposed method gets an increase of up to 18% accuracy. On NCI1 data, the increase is up to around 25% achieved by the proposed method than FFDN. Meanwhile, the proposed method performs better than FFDN on the rest of the datasets.

4.6. Parameter analysis

Three parameters affect the performance of the proposed fuzzy inference module, the sample size P, $\tilde{D}$ is the NFS input dimension parameter, and M is a user-defined neuro-fuzzy module output parameter, which is set to $M = floor(\tilde{D}/2)$. N is the number of fuzzy sets on each axis. We experiment with two parameter sets, i.e., $P = 6; \tilde{D} = 3; N = 4$ termed Parm2 and $P = 3; \tilde{D} = 4; N = 3$ termed Parm1, additionally we test the performance of with U(0.5) and U(1). In this analysis, we did not fine-tune the parameters largely due to the reasons *1*). The node feature dimension is 18 for all tested small datasets, and there is little space for parameter P change. *2*). The total number of parameter sizes increases exponentially by the size of $\tilde{D}$ and N, so $\tilde{D}$ and N should be set to a small value.

The results of two different parameter sets, Parm1 and Parm2, are shown in Table 11. The performance of the proposed model is affected largely by dataset OHSU with around 7.5% accuracy up and down. The best-performed parameter sets are Parm1-U(0.5) and Parm2-U(1) with uniform sampling. For datasets IMDB-BINARY, NCI1, the model configured with Parm1-U(0.5) performs the best, while the overall performance configured with other parameter sets is competitive. For datasets, Peking_1 and SYNTHETIC, there is 6% and 1.1% accuracy up and down, respectively. And the best-performed configuration set is Parm2-U(0.5). The results presented in Table 11 can be interpreted as follows: a big value of $\tilde{D}$ improves the performance with a low number of average edges in graph data.

Table 10
Comparison results of average, max, standard deviation Accuracy in % (avg-acc, max-acc, std-acc), Precision, F1, recall and AUC on eight datasets by FFDN [11] and our proposed model.

Dataset	Method	time-epoch	params	avg-acc	max-acc	std-acc	Precision	recall	F1	AUC
DD	FFDN	8.01E+01	4.98E+03	60.00	60.00	0.0E+00	0.00	0.00	0.00	52.61
	Our	**7.65E-01**	7.59E+03	**78.00**	82.00	2.89E-02	79.90	61.00	68.70	83.00
IMDB-BINARY	FFDN	1.18E+01	3.86E+03	50.00	50.00	0.0E+00	0.00	0.00	0.00	56.51
	Our	**5.46E-01**	6.18E+03	**65.10**	67.20	1.77E-02	66.60	61.20	63.60	69.70
IMDB-MULTI	FFDN	1.16E+01	3.90E+03	30.67	30.67	0.0E+00	0.00	0.00	15.65	49.46
	Our	**8.11E-01**	6.22E+03	**38.90**	46.90	4.82E-02	0.00	0.00	32.80	55.70
NCI1	FFDN	2.56E+01	4.14E+03	49.22	49.22	0.0E+00	49.22	100.00	65.97	40.72
	Our	**2.27E+00**	6.76E+03	**74.20**	74.80	5.52E-03	73.70	74.30	73.90	81.00
OHSU	FFDN	9.82E-03	6.47E+03	50.00	50.00	0.0E+00	37.50	75.00	50.00	48.75
	Our	5.76E-02	9.21E+03	**60.00**	65.00	3.54E-02	63.20	62.50	60.00	63.50
Peking_1	FFDN	8.33E-03	6.47E+03	63.64	63.64	0.0E+00	0.00	0.00	0.00	47.66
	Our	8.25E-02	8.31E+03	**65.90**	72.70	6.82E-02	53.00	78.10	62.60	73.40
PROTEINS	FFDN	8.82E+00	3.60E+03	60.21	60.21	0.0E+00	0.00	0.00	0.00	55.00
	Our	**1.03E+00**	5.32E+03	**73.80**	74.90	1.66E-02	71.30	57.70	63.60	77.50
SYNTHETIC	FFDN	1.75E+01	3.97E+03	50.33	50.67	5.78E-03	38.00	75.00	50.44	50.00
	Our	**4.91E-01**	6.30E+03	50.00	50.70	6.67E-03	25.30	50.00	33.60	51.90
SYNTHETICnew	FFDN	8.21E+00	3.85E+03	50.00	50.67	6.67E-03	25.33	50.00	33.63	48.83
	Our	**4.75E-01**	5.35E+03	**55.00**	60.00	4.46E-02	42.10	57.90	48.20	58.10

Table 11
Parameter analysis results by the proposed model FuzzyGNN, where the dimension of node feature is 18, and Parm1 denotes the parameter set ($P = 3; \tilde{D} = 4; N = 3$) and Parm2 with ($P = 6; \tilde{D} = 3; N = 4$), please refer Table 9 for result symbol description.

Dataset	Method	time-epoch	params	avg-acc	max-acc	std-acc	Precision	recall	F1	AUC
IMDB-BINARY	Pam1-U(0.5)	5.46E-01	6.18E+03	**65.1%**	67.2%	1.77E-02	**66.6%**	**61.2%**	**63.6%**	**69.7%**
	Pam1-U(1)	5.63E-01	6.18E+03	61.0%	68.4%	8.86E-02	62.5%	60.0%	60.7%	64.8%
	Pam2-U(0.5)	8.98E-01	5.29E+03	57.5%	64.4%	7.64E-02	58.9%	58.0%	58.0%	61.7%
	Pam2-U(1)	8.42E-01	5.29E+03	64.8%	67.6%	1.94E-02	66.5%	60.0%	62.7%	69.0%
NCI1	Pam1-U(0.5)	2.27E+00	6.76E+03	**74.2%**	74.8%	5.52E-03	**73.7%**	74.3%	73.9%	**81.0%**
	Pam1-U(1)	2.30E+00	6.76E+03	73.6%	74.4%	5.69E-03	72.2%	75.6%	73.8%	80.7%
	Pam2-U(0.5)	4.03E+00	5.87E+03	73.7%	75.7%	1.17E-02	72.0%	76.3%	74.1%	81.0%
	Pam2-U(1)	3.37E+00	5.87E+03	73.1%	73.6%	3.11E-03	71.6%	75.2%	73.4%	80.7%
OHSU	Pam1-U(0.5)	5.76E-02	9.21E+03	**60.0%**	65.0%	3.54E-02	**63.2%**	**62.5%**	**60.0%**	**63.5%**
	Pam1-U(1)	4.45E-01	9.21E+03	52.5%	60.0%	4.33E-02	40.0%	50.0%	44.2%	58.0%
	Pam2-U(0.5)	4.68E-01	8.31E+03	56.3%	60.0%	4.15E-02	44.4%	47.5%	44.1%	60.0%
	Pam2-U(1)	4.76E-01	8.31E+03	60.0%	70.0%	7.07E-02	47.6%	52.5%	49.2%	59.5%
Peking_1	Pam1-U(0.5)	5.26E-01	9.21E+03	61.4%	63.6%	2.27E-02	35.5%	53.1%	42.3%	69.0%
	Pam1-U(1)	5.26E-01	9.21E+03	59.1%	63.6%	5.57E-02	34.2%	53.1%	41.0%	71.4%
	Pam2-U(0.5)	8.25E-02	8.31E+03	**65.9%**	72.7%	6.82E-02	**53.0%**	**78.1%**	**62.6%**	**73.4%**
	Pam2-U(1)	8.32E-02	8.31E+03	63.6%	68.2%	7.87E-02	50.0%	56.3%	52.9%	68.1%
SYNTHETIC	Pam1-U(0.5)	6.15E-01	6.22E+03	73.2%	74.2%	6.89E-03	69.0%	59.2%	63.7%	78.2%
	Pam1-U(1)	6.27E-01	6.22E+03	72.7%	73.8%	1.12E-02	69.0%	57.0%	62.2%	77.8%
	Pam2-U(0.5)	1.03E+00	5.32E+03	**73.8%**	74.9%	1.66E-02	**71.3%**	57.7%	63.6%	77.5%
	Pam2-U(1)	9.46E-01	5.32E+03	72.6%	74.2%	1.44E-02	71.0%	53.6%	60.3%	76.4%

4.7. Technical details of the trained NFS module

As stated above, the NFS module in FuzzyGNN deals with Gaussian fuzzy sets with two trainable parameters (μ and σ) in the premise layer. Apart from that, NFS also contains parameters related to the regression function in a consequent layer. A D-dimensional node representation is sampled into P sub-vectors of the length $\tilde{D}$ each, and these

sub-vectors are taken as the input to the NFS modules. Then in each NFS module input layer, there are N fuzzy sets to which the membership degree of the input sub-vector components are determined.

After the training periods, the resulting antecedent fuzzy sets are presented in Fig. 5. Let us mention three important notes related to this figure: (1) Fig. 5 presents the trained fuzzy sets for all the P inference modules for every single dataset (the parameters μ and σ are the same for all modules); (2) each fuzzy set relates to the particular input variable which is transformed by the previous GNN layers, in our case, we use the identical domain [-5, 5] for all variables which leads to the same plots in Fig. 5; (3) independently on the just stated fact that the fuzzy sets are identical for all P inference modules assuming the same dataset is considered, the representation sampling is necessary as the parameters of regression functions for each sample are different.

From the applicability point of view, the most significant limitation is the training time and, consequently, the training costs that are comparably higher than those needed by the benchmark models. On the other hand, if the training time is not such an issue for the particular problem (data set), this disadvantage is compensated by the higher consistency (robustness) and, consequently, reliability of the performance of FuzzyGNN. Indeed, the performance of FuzzyGNN was the best one in 4 out of 5 data sets (the first place was shared with GPS for the ZINC data set), and it kept the second-best result for the CLUSTER data set. It is necessary to mention that the superiority of FuzzyGNN was not significant in the case of PATTERN and MNIST data sets. On the other hand, no other method showed a similar consistency in the results over all 5 data sets.

Let us focus on the computational complexity of the NFS module in FuzzyGNN, a unique module that makes this approach different from other GNN models. The number of floating point operations (FLOPs) is a standard complexity measure for neural networks, and it counts the number of computational operations. In our case, we count the number of multiplications and additions as the FLOPs.

Within the premise layer, the FLOPs value generated comes from creating $N^{\tilde{D}}$ rules. Each rule has $(\tilde{D}-1)$ multiplication operations. Hence, the total FLOPs in this layer are given as follows:

$$FLOPs_{pre} = (\tilde{D}-1) \times N^{\tilde{D}} \times P \, .$$

In the consequent layer, many regression functions involve multiplication and addition operations. The number of multiplication operations equals the number of addition operations plus one. The number of FLOPs in the consequent layer is then given by:

$$FLOPs_{con} = M \times (2N^{\tilde{D}} - 1) \times (\tilde{D}+1) \times P \, .$$

The output layer performs the weighted-sum operation and the number of FLOPs is determined as follows:

$$FLOPs_{out} = M \times (2N^{\tilde{D}} - 1) \times P$$

and hence, the total FLOPs of the NFS module is

$$FLOPs = FLOPs_{pre} + FLOPs_{con} + FLOPs_{out} \, .$$

5. Conclusion

This work presents an integrated fusion model *fuzzy forest graph neural network* (FuzzyGNN) that integrates the neuro-fuzzy system with the GNN for the large-scale dataset through feature-level fusion assisting the features learning. This model fusion differs from the ensemble model that works as an activation function or processes fuzzy modules in a separate block (data-level fusion). The purpose of the proposed model is to interpret the node representation in a way that enables to use of the power of the inherent fuzzy regression model. Two components of the whole FuzzyGNN are necessary for correctly implementing the NFS module; in particular, the node representation sampling precedes the NFS module and the fuzzy inference component that models the regression function. The generalized consequent functions model the dependencies of output variables on the input variables on distinct subareas of the input space defined by the Cartesian products of antecedent fuzzy sets. In order to verify the efficiency and effectiveness of FuzzyGNN fusion model, we tested it on five benchmark machine learning datasets. The experimental results justified the proposed FuzzyGNN approach by demonstrating its best performance on two out of five datasets, while another two are equal ranks one. FuzzyGNN is designed for the large-scale dataset to improve the performance of the

deep NN model. We used a small value for the N and M parameters on all experiments, which may eliminate the exploitation of the potential usage of the designed NFS module. We argue that the experimental analysis provided a strong justification for considering the proposed FuzzyGNN as a basis for fusing fuzzy modules with deep learning that improves performance. A more extensive parameter dig is the most natural step of our future research that can potentially lead to the progress of the results.

6. Future work and Limitation

- This work only divides the node representation into multiple samples, aiming to decrease the required training parameters. However, we only explored with $N = 3$ $\tilde{D} = 4$ scenario to the large dataset, making the fuzzy-related training parameter number small in the total number of training parameters. This unbalanced training parameter ratio between fuzzy-related and regular parameters causes no significant differences in the performance of large datasets and may limit the inference of the potential capability of the proposed fuzzy-assisted module. We plan to balance the parameters (fuzzy and non-fuzzy parameters) and test them on the large dataset using an advanced high-performance device.
- The proposed fuzzy-assisted module is implemented at the end operation of GNN. This proposed module can be plugged into a middle process of GNN architecture, which may enhance the learning ability of each layer instead of assisting the learning for the last layer. Further, we implement the model in Pytorch, making it possible to apply public tool `https://captum.ai/` for the interpretability of the model.
- We analyzed the complexity of the proposed model, which is acceptable and low. However, the expectation time is higher than a model with the same number of FLOPs because the matrix-vector multiplication operations are Streamlined, which is quicker in GPU than fuzzy-related exponential operation.

References

[1] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, 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, 2020, pp. 3438–3445.

[2] R. Das, S. Sen, U. Maulik, A survey on fuzzy deep neural networks, ACM Computing Surveys (CSUR) 53 (3) (2020) 1–25.

[3] A. Thakkar, K. Chaudhari, Fusion in stock market prediction: a decade survey on the necessity, recent developments, and potential future directions, Information Fusion 65 (2021) 95–107.

[4] P. V. de Campos Souza, Fuzzy neural networks and neuro-fuzzy networks: A review the main techniques and applications used in the literature, Applied soft computing 92 (2020) 106275.

[5] W. Dong, L. Yang, R. Gravina, G. Fortino, Anfis fusion algorithm for eye movement recognition via soft multi-functional electronic skin, Information Fusion 71 (2021) 99–108.

[6] Y. Zhang, G. Wang, X. Huang, W. Ding, Tsk fuzzy system fusion at sensitivity-ensemble-level for imbalanced data classification, Information Fusion 92 (2023) 350–362.

[7] V. Ojha, A. Abraham, V. Snášel, Heuristic design of fuzzy inference systems: A review of three decades of research, Engineering Applications of Artificial Intelligence 85 (2019) 845–864.

[8] M. Stojčić, Application of anfis model in road traffic and transportation: a literature review from 1993 to 2018, Operational Research in Engineering Sciences: Theory and Applications 1 (1) (2018) 40–61.

[9] J.-S. Jang, Anfis: adaptive-network-based fuzzy inference system, IEEE transactions on systems, man, and cybernetics 23 (3) (1993) 665–685.

[10] C. Pramod, G. N. Pillai, K-means clustering based extreme learning anfis with improved interpretability for regression problems, Knowledge-Based Systems 215 (2021) 106750.

[11] Y. Deng, Z. Ren, Y. Kong, F. Bao, Q. Dai, A hierarchical fused fuzzy deep neural network for data classification, IEEE Transactions on Fuzzy Systems 25 (4) (2016) 1006–1012.

[12] T. Wei, J. Hou, R. Feng, Fuzzy graph neural network for few-shot learning, in: 2020 International joint conference on neural networks (IJCNN), IEEE, 2020, pp. 1–8.

[13] T. Zhao, J. Xu, R. Chen, X. Ma, Remote sensing image segmentation based on the fuzzy deep convolutional neural network, International Journal of Remote Sensing 42 (16) (2021) 6264–6283.

[14] Q. Chong, J. Xu, F. Jia, Z. Liu, W. Yan, X. Wang, Y. Song, A multiscale fuzzy dual-domain attention network for urban remote sensing image segmentation, International Journal of Remote Sensing 43 (14) (2022) 5480–5501.

[15] P.-C. Chang, C.-H. Liu, C.-Y. Fan, Data clustering and fuzzy neural network for sales forecasting: A case study in printed circuit board industry, Knowledge-Based Systems 22 (5) (2009) 344–355.

[16] T. Qu, J. Xu, Q. Chong, Z. Liu, W. Yan, X. Wang, Y. Song, M. Ni, Fuzzy neighbourhood neural network for high-resolution remote sensing image segmentation, European Journal of Remote Sensing 56 (1) (2023) 2174706.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017).

[18] C. El Hatri, J. Boumhidi, Fuzzy deep learning based urban traffic incident detection, Cognitive systems research 50 (2018) 206–213.
[19] L. Subhashini, Y. Li, J. Zhang, A. S. Atukorale, Integration of fuzzy logic and a convolutional neural network in three-way decision-making, Expert Systems with Applications 202 (2022) 117103.
[20] S. Dey, R. Roychoudhury, S. Malakar, R. Sarkar, An optimized fuzzy ensemble of convolutional neural networks for detecting tuberculosis from chest x-ray images, Applied Soft Computing 114 (2022) 108094.
[21] R. Chai, A. Tsourdos, A. Savvaris, Y. Xia, S. Chai, Real-time reentry trajectory planning of hypersonic vehicles: a two-step strategy incorporating fuzzy multiobjective transcription and deep neural network, IEEE Transactions on Industrial Electronics 67 (8) (2019) 6904–6915.
[22] M. Z. Asghar, F. Subhan, H. Ahmad, W. Z. Khan, S. Hakak, T. R. Gadekallu, M. Alazab, Senti-esystem: A sentiment-based esystem-using hybridized fuzzy and deep neural network for measuring customer satisfaction, Software: Practice and Experience 51 (3) (2021) 571–594.
[23] P. Hurtík, V. Molek, J. Hůla, Data preprocessing technique for neural networks based on image represented by a fuzzy function, IEEE Transactions on Fuzzy Systems 28 (7) (2020) 1195–1204.
[24] G. Sideratos, A. Ikonomopoulos, N. D. Hatziargyriou, A novel fuzzy-based ensemble model for load forecasting using hybrid deep neural networks, Electric Power Systems Research 178 (2020) 106025.
[25] G. Manogaran, P. M. Shakeel, S. Baskar, C.-H. Hsu, S. N. Kadry, R. Sundarasekar, P. M. Kumar, B. A. Muthu, Fdm: Fuzzy-optimized data management technique for improving big data analytics, IEEE Transactions on Fuzzy Systems 29 (1) (2020) 177–185.
[26] Y.-J. Zheng, S.-Y. Chen, Y. Xue, J.-Y. Xue, A pythagorean-type fuzzy deep denoising autoencoder for industrial accident early warning, IEEE Transactions on Fuzzy Systems 25 (6) (2017) 1561–1575.
[27] H. Moeeni, H. Bonakdari, I. Ebtehaj, Integrated sarima with neuro-fuzzy systems and neural networks for monthly inflow prediction, Water Resources Management 31 (7) (2017) 2141–2156.
[28] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Message passing neural networks, in: Machine learning meets quantum physics, Springer, 2020, pp. 199–214.
[29] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE transactions on systems, man, and cybernetics 1 (1985) 116–132.
[30] Z. Jiang, S. Gao, M. Li, An improved advertising ctr prediction approach based on the fuzzy deep neural network, PloS one 13 (5) (2018) e0190831.
[31] Z. Zhang, L. Ning, Z. Liu, Q. Yang, W. Ding, Mining and reasoning of data uncertainty-induced imprecision in deep image classification, Information Fusion 96 (2023) 202–213.
[32] W. M. Shaban, A. H. Rabie, A. I. Saleh, M. Abo-Elsoud, Detecting covid-19 patients based on fuzzy inference engine and deep neural network, Applied soft computing 99 (2021) 106906.
[33] J. Mar, H.-T. Lin, A car-following collision prevention control device based on the cascaded fuzzy inference system, Fuzzy sets and systems 150 (3) (2005) 457–473.
[34] C. P. Chen, C.-Y. Zhang, L. Chen, M. Gan, Fuzzy restricted boltzmann machine for the enhancement of deep learning, IEEE Transactions on Fuzzy Systems 23 (6) (2015) 2163–2173.
[35] Y.-J. Zheng, W.-G. Sheng, X.-M. Sun, S.-Y. Chen, Airline passenger profiling based on fuzzy deep machine learning, IEEE transactions on neural networks and learning systems 28 (12) (2016) 2911–2923.
[36] S. Park, S. J. Lee, E. Weiss, Y. Motai, Intra-and inter-fractional variation prediction of lung tumors using fuzzy deep learning, IEEE journal of translational engineering in health and medicine 4 (2016) 1–12.
[37] M. A. Islam, D. T. Anderson, A. J. Pinar, T. C. Havens, G. Scott, J. M. Keller, Enabling explainable fusion in deep learning with fuzzy integral neural networks, IEEE Transactions on Fuzzy Systems 28 (7) (2019) 1291–1300.
[38] K. Oono, T. Suzuki, Graph neural networks exponentially lose expressive power for node classification, arXiv preprint arXiv:1905.10947, The International Conference on Learning Representations, ICLR (2019).
[39] J. Palowitch, A. Tsitsulin, B. Mayer, B. Perozzi, Graphworld: Fake graphs bring real insights for gnns, arXiv preprint arXiv:2203.00112, Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2022).
[40] J. Halcrow, A. Mosoi, S. Ruth, B. Perozzi, Grale: Designing networks for graph learning, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 2523–2532.
[41] Q. Zhu, N. Ponomareva, J. Han, B. Perozzi, Shift-robust gnns: Overcoming the limitations of localized graph training data, Advances in Neural Information Processing Systems 34 (2021) 27965–27977.
[42] D. Zhang, J. Yin, X. Zhu, C. Zhang, Network representation learning: A survey, IEEE transactions on Big Data 6 (1) (2018) 3–28.
[43] C. Runge, Über empirische funktionen und die interpolation zwischen äquidistanten ordinaten, Zeitschrift für Mathematik und Physik 46 (224-243) (1901) 20.
[44] J. Platoš, J. Nowaková, P. Krömer, V. Snášel, Space-filling curves based on residue number system, in: Advances in Intelligent Networking and Collaborative Systems: The 9th International Conference on Intelligent Networking and Collaborative Systems (INCoS-2017), Springer, 2018, pp. 53–61.
[45] E. Lughofer, E. Hüllermeier, E.-P. Klement, Improving the interpretability of data-driven evolving fuzzy systems., in: EUSFLAT Conf., Citeseer, 2005, pp. 28–33.
[46] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, X. Bresson, Benchmarking graph neural networks, arXiv preprint arXiv:2003.00982, Journal of Machine Learning Research 23 (2020) 1–48.
[47] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, D. Beaini, Recipe for a general, powerful, scalable graph transformer, 36th Conference on Neural Information Processing Systems (NeurIPS 2022) (2022).
[48] M. Fey, J. E. Lenssen, Fast graph representation learning with pytorch geometric, arXiv preprint arXiv:1903.02428, International Conference on Learning Representations, ICLR (2019).
[49] L. Ilya, H. Frank, Decoupled weight decay regularization, Proceedings of International Conference on Learning Representations, ICLR (2019).

[50] V. P. Dwivedi, X. Bresson, A generalization of transformer networks to graphs, AAAI 21 workshop on Deep learning on graphs: methods and applications (2020).
[51] J. You, R. Ying, J. Leskovec, Design space for graph neural networks, 2020.
[52] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, arXiv preprint arXiv:1511.05493, International Conference on Learning Representations, ICLR (2016).
[53] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, J. Leskovec, Strategies for pre-training graph neural networks, arXiv preprint arXiv:1905.12265, The International Conference on Learning Representations, ICLR (2019).
[54] D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, P. Tossou, Rethinking graph transformers with spectral attention, Advances in Neural Information Processing Systems 34 (2021) 21618–21629.
[55] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, T.-Y. Liu, Do transformers really perform badly for graph representation?, Advances in Neural Information Processing Systems 34 (2021) 28877–28888.
[56] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 5th International Conference on Learning Representations, Toulon, France (2016).
[57] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, stat 1050 (2017) 20.
[58] D. Beaini, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, P. Liò, Directional graph networks, in: International Conference on Machine Learning, PMLR, 2021, pp. 748–758.
[59] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, The International Conference on Learning Representations, ICLR (2018).
[60] M. S. Hussain, M. J. Zaki, D. Subramanian, Edge-augmented graph transformers: Global self-attention is enough for graphs, arXiv preprint arXiv:2108.03348, Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2021).
[61] S. Cai, L. Li, X. Han, J. Luo, Z.-J. Zha, Q. Huang, Automatic relation-aware graph network proliferation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10863–10873.

A. Appendix

In Section 3, we describe the neuro-fuzzy system module layer and briefly introduce the combination with GNN. Alg 1 shows the main flowchart of FuzzyGNN. Here we list the precise GNN layer connections and formalize the NFS process.

Symbol notations: $\mathbf{h}^{\ell}, E^{\ell}$ represents the node representation vector, edges at l_{th} layer. $\mathbf{A}$ denotes the adjacency matrix of data; $\cup$ is the union of all node representation. MLP is short for a multi-perceptron layer.

GNN layer connections with $\mathbf{h}^{\ell}, \mathbf{E}^{\ell}$ and $\mathbf{A}$ as:

$$\text{m\_}\hat{\mathbf{h}}^{\ell+1}, \mathbf{E}^{\ell+1} = \text{MPNN}(\mathbf{h}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}) \tag{8}$$

$$\text{ga\_}\hat{\mathbf{h}}^{\ell+1} = \text{Transformer}(\cup(\mathbf{h}^{\ell})) \tag{9}$$

$$\text{m\_}\mathbf{h}^{\ell+1} = \text{BatchNorm}\left(\text{Dropout}(\text{m\_}\hat{\mathbf{h}}^{\ell+1}) + \mathbf{h}^{\ell}\right) \tag{10}$$

$$\text{ga\_}\mathbf{h}^{\ell+1} = \text{BatchNorm}\left(\text{Dropout}(\text{ga\_}\hat{\mathbf{h}}^{\ell+1}) + \mathbf{h}^{\ell}\right) \tag{11}$$

$$\mathbf{h}^{\ell+1} = \text{MLP}\left(\text{ga\_}\mathbf{h}^{\ell+1} + \text{m\_}\mathbf{h}^{\ell+1}\right) \tag{12}$$

A NFS module starts with a $\mathbf{h}^{\ell}$, Sample -node Representation sampling, Regression- Regression function F; For differentiate the $\mathbf{h}_{v}^{\ell+1}$ as the v_{th} node presentation, we use ${}^{v}h^{\ell+1}$ as the v_{th} sampling intermediate result. The hyperparameters used in FuzzyGNN are shown in Table 12.

$$\hat{\mathbf{x}}_{p}^{\ell} = \text{Sample}\left(\mathbf{h}^{\ell}\right) \text{ where } p = 1, \ldots, P; \tag{13}$$

Algorithm 1: FuzzyGNN model (An example process, we omit basics operations, such as Norm, activation, weighted mean, process to make it simple.)

Input: Graph data or Image data
Output: Classification/regression output

```
/* -- Pre-processing --                                                          */
/* 'embeddings' of a graph represent tensor data. Each node has an embedding, which is a vector.  */
embeddings ← Node encoding (raw data) ;
embeddings ← structural encoding (embeddings);
/* 'representation' is another name for embeddings used in a neural network, it is also tensor data */
(representation) ← positional encoding (embeddings);
/* -- Graph neural network process --                                            */
for L = 1 → ℓ do
    /* For each node v in a graph with its representation x_v^(L) at layer L-th  */
    x_v^(L) = W^(L) Σ_{w∈N(v)∪{v}} (1/c_{w,v}) · x_w^(L−1);
    /* W^(L) is learnable weights, N(v) is the neighbor node set of node v, c_{w,v} is scale value, obtained
       by attention mechanism.                                                    */
    /* In our experiments, we perform Eq 8-12 with a basic BatchNorm activation function process.  */
end
/* -- Fuzzy-assisted module process --                                           */
for For each node v with its representation x_v^(ℓ) do
    (Samples) x̂ ← Representation Sampling( x_v^(ℓ) );
    /* each sample is a vector of size D/P, D dimension of node representation, P number of samples.  */
end
for For all nodes in a graph do
    /* the fuzzy-assisted module operates parallel for all samples of a node representation  */
    (a sample) {y} ← Fuzzy-inference module(x̂) (a sample);
    /* Concatenate all {y} of all samples from a node                             */
    representation ← Concatenation(∀{y})
end
/* --Task-specific layer process --                                           */
for For all nodes in a graph do
    /* Classification or Regression                                           */
    regression output ← MLP(representation);
    classification output ← Softmax( MLP(representation));
end
```

$$\bar{\mathbf{w}}_p^{\ell} = \text{Normalize}\left(\text{Fuzzy\_rule}(\hat{\mathbf{x}}_p^{\ell+1}, \mu, \sigma)\right); \tag{14}$$

$$\mathbf{f}_p^{\ell} = \text{Regression}\left(\hat{\mathbf{x}}_p^{\ell}, \tilde{W}, M\right) \tag{15}$$

$${}^{v}\mathbf{h}_p^{\ell+1} = \text{Weighted\_sum}\left(\mathbf{f}_p^{\ell}, \bar{\mathbf{w}}_p^{\ell}\right) \tag{16}$$

$$\mathbf{h}^{\ell+1} = \text{Concatenation}\left(\Pi\ {}^{v}\mathbf{h}_p^{\ell+1}\right) \text{ where } p = 1, \ldots, P; \tag{17}$$

where, $\mathbf{h}^{\ell}$ is an input vector for NFS module and $\hat{\mathbf{h}}_p^{\ell}$ is one of $\tilde{D}$-dimensional the sample results; $\bar{\mathbf{w}}_p^{\ell}$ is a $N^{\tilde{D}}$-dimensional vector; $\bar{\mathbf{w}}_p^{\ell} = \{\bar{w}_k^{\ell}\}$ and $k = 1, 2, \ldots, N^{\tilde{D}}$; $\tilde{W}$ denotes the trainable parameters with size of $N^{\tilde{D}} \times (\tilde{D} + 1)$.

Table 12
FuzzyGNN related hyperparameters used in the experiments.

Parameter	PATTER	CLUSTER	ZINC	MNIST	CIFAR10
dropout(GNN)	0.0	0.03	0.0	0.0	0.0
act(GNN)	relu	relu	relu	relu	relu
agg(GNN)	mean	mean	mean	mean	mean
attn_drop(GT)	0.5	0.5	0.5	0.5	0.5
dropout(GT)	0.0	0.06	0.05	0.0	0.03
n_heads(GT)	4	8	4	4	4
layer (GT)	6	16	10	3	3