DIGRAF: Diffeomorphic Graph-Adaptive Activation Function

Krishna Sri Ipsit Mantri[*]
Purdue University
mantrik@purdue.edu

Xinzhi (Aurora) Wang[*]
Purdue University
wang6171@purdue.edu

Carola-Bibiane Schönlieb
University of Cambridge
cbs31@cam.ac.uk

Bruno Ribeiro
Purdue University
ribeirob@purdue.edu

Beatrice Bevilacqua[†]
Purdue University
bbevilac@purdue.edu

Moshe Eliasof[†]
University of Cambridge
me532@cam.ac.uk

Abstract

In this paper, we introduce DIGRAF, a novel framework leveraging Continuous Piecewise-Affine Based (CPAB) transformations, which we augment with an additional GNN to learn a graph-adaptive diffeomorphic activation function in an end-to-end manner. In addition to its graph-adaptivity and flexibility, DIGRAF also possesses properties that are widely recognized as desirable for activation functions, such as differentiability, boundness within the domain and computational efficiency. We show the consistent and superior performance of DIGRAF across a variety of graph benchmarks, highlighting its effectiveness as an activation function for GNNs.

1 Introduction

Recent advancements in GNN research have predominantly focused on exploring the design space of key architectural elements, ranging from expressive GNN layers [1–5], to pooling layers [6–9], and positional and structural encodings [10–12]. Despite the exploration of these architectural choices, a common trend persists where most GNNs default to employing standard activation functions, such as ReLU [13], among a few others. Therefore, *our objective is to design a flexible activation function tailored for graph data, offering consistent performance gains*. This activation function should possess many, if not all, of the properties recognized as beneficial for activation functions, with an emphasis on blueprint flexibility, as well as task and input adaptivity.

We leverage the success of learning diffeomorphisms, particularly through Continuous Piecewise-Affine Based transformations (CPAB) [14, 15], to devise an activation function tailored for graph-structured data. Diffeomorphisms, characterized as bijective, differentiable and invertible mappings with a differentiable inverse, inherently possess many desirable properties of activation functions, like differentiability, boundedness within the input-output domain, and stability to input perturbations. To augment our activation function with graph-adaptivity, we employ an additional GNN to derive the parameters of the learned diffeomorphism. This yields our node permutation equivariant activation function that dynamically adapts to different graphs, dubbed DIGRAF – **DI**ffeomorphism-based **GR**aph **A**ctivation **F**unction, illustrated in Figure 1. We conduct an extensive set of experiments comparing DIGRAF with widely used activation functions on several graph learning benchmarks and consistently show improved performance over all the baselines.

2 Background and Notations

In this paper, we utilize the definitions from CPAB — a framework for efficiently learning flexible diffeomorphisms [14, 15], alongside basic graph learning notations, to develop activation functions for

[*]Equal contribution.
[†]Equal supervision.

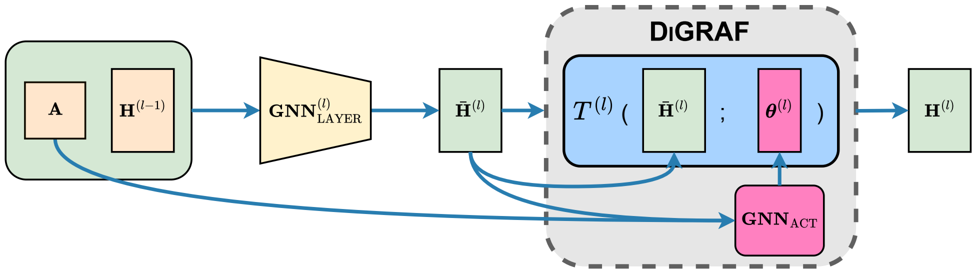

DiGRAF
A
$\mathbf{H}^{(l-1)}$
$\text{GNN}^{(l)}_{\text{LAYER}}$
$\bar{\mathbf{H}}^{(l)}$
$T^{(l)}(\bar{\mathbf{H}}^{(l)}; \boldsymbol{\theta}^{(l)})$
GNN_{ACT}
$\mathbf{H}^{(l)}$

Figure 1: Illustration of DiGRAF. Node features $\mathbf{H}^{(l-1)}$ and adjacency matrix $\mathbf{A}$ are fed to a $\text{GNN}^{(l)}_{\text{LAYER}}$ to obtain updated intermediate node features $\bar{\mathbf{H}}^{(l)}$, which are passed to our activation function layer, DiGRAF. First, an additional GNN network GNN_{ACT} takes $\bar{\mathbf{H}}^{(l)}$ and $\mathbf{A}$ as input to determine the activation function parameters $\boldsymbol{\theta}^{(l)}$. These are used to parameterize the transformation $T^{(l)}$, which operates on $\bar{\mathbf{H}}^{(l)}$ to produce the activated node features $\mathbf{H}^{(l)}$.

GNNs. Consequently, this section outlines the essential details needed to understand the foundations of our DiGRAF. We defer the interested reader to Appendix E for a more gradual introduction to CPAB.

2.1 CPAB Diffeomorphisms

Let $\Omega = [a, b] \subset \mathbb{R}$ be a closed interval, where $a < b$. We discretize Ω using a tessellation $\mathcal{P}$ with $\mathcal{N}_{\mathcal{P}}$ intervals, which, in practice, is oftentimes an equispaced 1D meshgrid with $\mathcal{N}_{\mathcal{P}}$ segments [15]. Our goal in this paper is to learn a diffeomorphism $f : \Omega \to \Omega$ that we will use as an activation function.

Definition 2.1 (CPAB Diffeomorphism)**.** Given a continuous and piecewise affine (CPA) velocity field $v^{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{\mathcal{N}_{\mathcal{P}}-1}$, the CPAB diffeomorphism f at point $x \in \Omega$, is defined as:

$$f^{\boldsymbol{\theta}}(x) \triangleq \phi^{\boldsymbol{\theta}}(x, t = 1) \tag{1}$$

such that $\phi^{\boldsymbol{\theta}}(x, t = 1)$ solves the integral equation:

$$\phi^{\boldsymbol{\theta}}(x, t) = x + \int_0^t v^{\boldsymbol{\theta}}\big(\phi^{\boldsymbol{\theta}}(x, \tau)\big)\, d\tau. \tag{2}$$

In arbitrary dimensions, the computations in Definition 2.1 require using an ordinary differential equation solver and can be expensive. However, for 1D diffeomorphisms, as in our DiGRAF, there are closed-form solutions to the CPAB diffeomorphism and its gradients [15], offering an efficient framework for learning activation functions.

2.2 Graph Learning Notations

Consider a graph $G = (V, E)$ with $N \in \mathbb{N}$ nodes, where $V = \{1, \ldots, N\}$ is the set of nodes and $E \subseteq V \times V$ is the set of edges. Let $\mathbf{A} \in \{0, 1\}^{N \times N}$ denote the adjacency matrix of G, and $\mathbf{X} \in \mathbb{R}^{N \times F}$ the node feature matrix, where F is the number of input features. The input node features $\mathbf{X}$ are transformed into the initial node representations $\mathbf{H}^{(0)} \in \mathbb{R}^{N \times C}$, obtained by applying an embedding function $\text{emb} : \mathbb{R}^F \to \mathbb{R}^C$ to $\mathbf{X}$, where C is the hidden dimension, that is

$$\mathbf{H}^{(0)} = \text{emb}(\mathbf{X}). \tag{3}$$

The initial features $\mathbf{H}^{(0)}$ are fed to a GNN comprised of $L \in \mathbb{N}$ layers, where each layer $l \in \{1, \ldots, L\}$ is followed by an activation function $\sigma^{(l)}(\cdot; \boldsymbol{\theta}^{(l)}) : \mathbb{R} \to \mathbb{R}$, and $\boldsymbol{\theta}^{(l)}$ is a set of possibly learnable parameters of $\sigma^{(l)}$. Specifically, the intermediate output of the l-th GNN layer is denoted as:

$$\bar{\mathbf{H}}^{(l)} = \text{GNN}^{(l)}_{\text{LAYER}}(\mathbf{H}^{(l-1)}, \mathbf{A}) \tag{4}$$

where $\bar{\mathbf{H}}^{(l)} \in \mathbb{R}^{N \times C}$. The activation function $\sigma^{(l)}$ is then applied *element-wise* to $\bar{\mathbf{H}}^{(l)}$, yielding node features $h_{u,c}^{(l)} = \sigma^{(l)}(\bar{h}_{u,c}^{(l)}; \boldsymbol{\theta}^{(l)})\ \forall u \in V, \forall c \in [C]$. Therefore, the application of $\sigma^{(l)}$ can be equivalently written as follows:

$$\mathbf{H}^{(l)} = \sigma^{(l)}(\bar{\mathbf{H}}^{(l)}; \boldsymbol{\theta}^{(l)}). \tag{5}$$

In the following section, we will show how this abstraction is translated to our DIGRAF.

3 DIGRAF

In this section, we formalize our approach, DIGRAF, illustrated in Figure 1, which leverages diffeomorphisms to learn adaptive and flexible graph activation functions. We defer the theoretical analysis of DIGRAF to Appendix D, emphasizing on its attributes that account for it as a strong GNN activation function.

3.1 A CPAB Blueprint for Graph Activation Functions

Our approach builds on the highly flexible CPAB framework [14, 15] and extends it by incorporating Graph Neural Networks (GNNs) to enable the learning of adaptive graph activation functions. While the original CPAB framework was designed for grid deformation and alignment tasks, typically in 1D, 2D, or 3D spaces, we propose a novel application of CPAB in the context of learning activation functions, as described below.

Formally, we define the transformation $T^{(l)}$ as the element-wise application of the diffeomorphism $f^{\boldsymbol{\theta}}$ from Equation (1).

$$T^{(l)}(\bar{h}_{u,c}^{(l)}; \boldsymbol{\theta}^{(l)}) \triangleq f^{\boldsymbol{\theta}^{(l)}}(\bar{h}_{u,c}^{(l)}), \tag{6}$$

where $\boldsymbol{\theta}^{(l)}$ denotes learnable parameters of the diffeomorphism $f^{\boldsymbol{\theta}^{(l)}}$, that parameterize the underlying CPA velocity field as discussed in Appendix E. In Section 3.2 , we discuss the learning of $\boldsymbol{\theta}^{(l)}$ in DIGRAF.

The transformation $T^{(l)} : \Omega \to \Omega$ described in Equation (6) is based on CPAB and therefore takes as input values within a domain $\Omega = [a, b]$, and outputs a value within that domain, where $a < b$ are hyperparameters. In practice, we take $a = -b$, such that the activation function can be symmetric and centered around 0, a property known to be desirable for activation functions [16]. For any entry in the intermediate node features $\bar{\mathbf{H}}^{(l)}$(Equation (4)) that is outside the domain Ω, we use the identity function. Therefore, a DIGRAF activation function reads:

$$\text{DIGRAF}(\bar{h}_{u,c}^{(l)}, \boldsymbol{\theta}^{(l)}) = \begin{cases} T^{(l)}(\bar{h}_{u,c}^{(l)}; \boldsymbol{\theta}^{(l)}), & \text{If } \bar{h}_{u,c}^{(l)} \in \Omega \\ \bar{h}_{u,c}^{(l)}, & \text{Otherwise} \end{cases} \tag{7}$$

In practice, DIGRAF is applied element-wise in parallel over all entries, and we use the following notation, which yields the output features post the activation of the l-th GNN layer:

$$\mathbf{H}^{(l)} = \text{DIGRAF}(\bar{\mathbf{H}}^{(l)}, \boldsymbol{\theta}^{(l)}). \tag{8}$$

3.2 Learning Diffeomorphic Velocity Fields

DIGRAF, defined in Equation (7), introduces graph-adaptivity into the transformation function $T^{(l)}$ by employing an additional GNN, denoted as GNN_{ACT}, that returns the diffeomorphism parameters $\boldsymbol{\theta}^{(l)}$:

$$\boldsymbol{\theta}^{(l)}(\bar{\mathbf{H}}^{(l)}, \mathbf{A}) = \text{POOL}\left(\text{GNN}_{\text{ACT}}(\bar{\mathbf{H}}^{(l)}, \mathbf{A})\right), \tag{9}$$

where POOL is a graph-wise pooling operation, such as max or mean pooling. The resulting vector $\boldsymbol{\theta}^{(l)} \in \mathbb{R}^{\mathcal{N}_{\mathcal{P}}-1}$, which is dependent on the tessellation size $\mathcal{N}_{\mathcal{P}}$, is then used to compute the output of the l-th layer, $\mathbf{H}^{(l)}$, as described in Equation (8). We note that Equation (9) yields a different $\boldsymbol{\theta}^{(l)}$ for every input graph and features pair $(\bar{\mathbf{H}}^{(l)}, \mathbf{A})$, which implies the graph-adaptivity of DIGRAF. Furthermore, since GNN_{ACT} is trained with the other network parameters in an end-to-end fashion, DIGRAF is also adaptive to the task of interest. In Appendix C, we provide and discuss the implementation details of GNN_{ACT} and POOL.

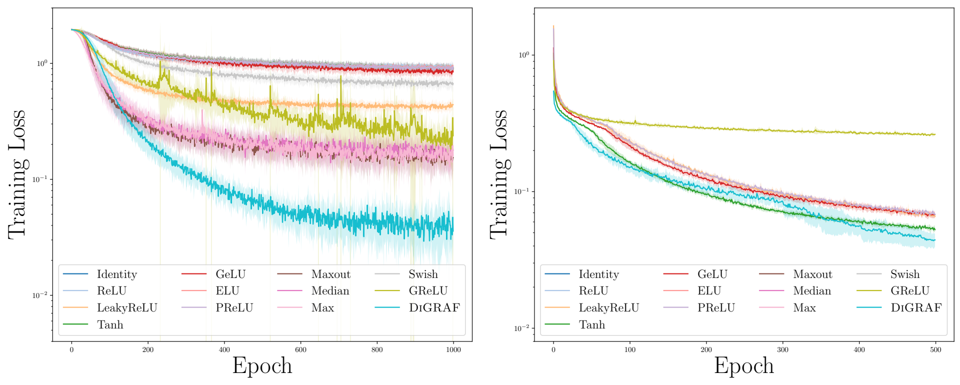


Figure 2: Convergence analysis of DIGRAF compared to baseline activation functions for Cora (left) and ZINC (right). The plot illustrates the training loss over epochs, showcasing overall faster convergence of DIGRAF.

Variants of DIGRAF. Equation (9) describes an approach to introduce graph-adaptivity to $\boldsymbol{\theta}^{(l)}$ using GNN_{ACT}. An alternative approach is to directly optimize the parameters $\boldsymbol{\theta}^{(l)} \in \mathbb{R}^{\mathcal{N}_{\mathcal{P}}-1}$, without using an additional GNN. Note that in this case, input and graph-adaptivity are sacrificed in favor of a computationally lighter solution. We denote this variant of our method by DIGRAF (W/O ADAP.). Considering this variant is important because it allows us to: (i) offer a middle-ground solution in terms of computational effort, and (ii) it allows us to directly quantify the contribution of graph-adaptivity in DIGRAF. In Section 4, we compare the performance of the methods.

4 Experiments

Baselines. We compare DIGRAF with three categories of relevant and competitive baselines: (1) *Standard Activation Functions*, namely Identity, ReLU [13], LeakyReLU [17], Tanh [18], GeLU [19], ELU [20] and Sigmoid [21] to estimate the benefit of learning activation functions parameters; (2) *Learnable Activation Functions*, specifically PReLU [22], Maxout [23] and Swish [24], to assess the value of graph-adaptivity; and (3) *Graph Activation Functions*, such as Max [25], Median [25] and GReLU [26], to evaluate the effectiveness of DIGRAF's design.

Discussion. Our results, summarized in Table 1, and further detailed in Appendix G, highlight the following key points:

(R1) **Overall Performance:** DIGRAF consistently outperforms competing activation functions. It achieves an MAE of 0.1302 on ZINC, a 18% relative improvement over Maxout. On MOLHIV, DIGRAF's ROC-AUC of 80.28% surpasses ReLU by 4.7%.

(R2) **Graph-Adaptivity:** DIGRAF surpasses non-graph-adaptive functions like PReLU, Maxout, and Swish, and its non-adaptive variant, DIGRAF (W/O ADAP.), demonstrating the critical role of graph adaptivity in GNN activation functions.

(R3) **Flexibility:** DIGRAF exceeds graph-adaptive functions such as Max, Median, and GReLU by leveraging a diffeomorphism-based framework, effectively modeling complex non-linearities.

(R4) **Efficiency:** As illustrated in Figure 2, DIGRAF accelerates training convergence and enhances downstream task performance.

Table 1: A comparison of DIGRAF to natural baselines, standard, and graph activation layers on ZINC and MOLHIV datasets. The top three methods are marked by **First**, **Second**, **Third**.

Method	ZINC MAE ↓	MOLHIV ROC-AUC ↑
STANDARD ACTIVATIONS		
GIN + Identity	0.2460±0.0214	75.12±0.77
GIN + ReLU [27]	**0.1630±0.0040**	**75.58±1.40**
GIN + LeakyReLU [17]	0.1718±0.0042	74.75±1.20
GIN + Tanh [18]	0.1797±0.0064	**75.22±2.03**
GIN + GeLU [19]	0.1896±0.0023	74.15±0.79
GIN + ELU [20]	0.1741±0.0089	75.09±0.65
GIN + Sigmoid [21]	0.3839±0.0058	73.87±0.80
LEARNABLE ACTIVATIONS		
GIN + PReLU [22]	0.1798 ±0.0067	73.56±1.63
GIN + Maxout [23]	**0.1587±0.0057**	72.75±2.10
GIN + Swish [24]	0.1636±0.0039	72.95±0.64
GRAPH ACTIVATIONS		
GIN + Max [25]	0.1661±0.0035	73.44±2.08
GIN + Median [25]	0.1715±0.0050	72.80±2.21
GIN + GReLU [26]	0.3003±0.0086	73.45±1.62
GIN + DIGRAF (W/O ADAP.)	0.1382±0.0080	79.19±1.36
GIN + DIGRAF	**0.1302±0.0090**	**80.28±1.44**

5 Conclusion

In this work, we introduced DIGRAF, a novel activation function tailored for graph-structured data. By leveraging Continuous Piecewise-Affine Based (CPAB) transformations, DIGRAF adapts to the unique structural features of input graphs, significantly enhancing performance in graph learning tasks. DIGRAF offers key properties such as differentiability, boundedness, computational efficiency, and permutation equivariance, making it highly suitable for graph-based applications. Extensive experiments show that DIGRAF consistently outperforms existing activation functions and accelerates training convergence, advancing the state-of-the-art in GNN activation design.

Acknowledgments

BR acknowledges support from the National Science Foundation (NSF) awards, CCF-1918483, CAREER IIS-1943364 and CNS-2212160, Amazon Research Award, AnalytiXIN, and the Wabash Heartland Innovation Network (WHIN), Ford, Nvidia, CISCO, and Amazon. Computing infrastructure was supported in part by CNS-1925001 (CloudBank). This work was supported in part by AMD under the AMD HPC Fund program. ME is funded by the Blavatnik-Cambridge fellowship, the Cambridge Accelerate Programme for Scientific Discovery, and the Maths4DL EPSRC Programme. The authors thank Shahaf Finder, Ron Shapira-Weber, and Oren Freifeld for the discussions on CPAB.

References

[1] Christopher Morris, Yaron Lipman, Haggai Maron, Bastian Rieck, Nils M. Kriege, Martin Grohe, Matthias Fey, and Karsten Borgwardt. Weisfeiler and leman go machine learning: The story so far. *Journal of Machine Learning Research*, 24(333):1–59, 2023. 1, 10

[2] Fabrizio Frasca, Beatrice Bevilacqua, Michael M Bronstein, and Haggai Maron. Understanding and extending subgraph gnns by rethinking their symmetries. In *Advances in Neural Information Processing Systems*, 2022.

[3] Bohang Zhang, Guhao Feng, Yiheng Du, Di He, and Liwei Wang. A complete expressiveness hierarchy for subgraph gnns via subgraph weisfeiler-lehman tests. In *International Conference on Machine Learning*, 2023.

[4] Bohang Zhang, Shengjie Luo, Liwei Wang, and Di He. Rethinking the expressive power of gnns via graph biconnectivity. *arXiv preprint arXiv:2301.09505*, 2023.

[5] Omri Puny, Derek Lim, Bobak Kiani, Haggai Maron, and Yaron Lipman. Equivariant polynomials for graph neural networks. In *International Conference on Machine Learning*, pages 28191–28222. PMLR, 2023. 1, 10

[6] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018. 1

[7] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International conference on machine learning*, pages 3734–3743. PMLR, 2019.

[8] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning*, pages 874–883. PMLR, 2020.

[9] Yu Guang Wang, Ming Li, Zheng Ma, Guido Montufar, Xiaosheng Zhuang, and Yanan Fan. Haar graph pooling. In *International conference on machine learning*, pages 9952–9962. PMLR, 2020. 1

[10] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023. 1

[11] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022. 10

[12] Moshe Eliasof, Fabrizio Frasca, Beatrice Bevilacqua, Eran Treister, Gal Chechik, and Haggai Maron. Graph positional encoding via random feature propagation. In *International Conference on Machine Learning*, pages 9202–9223. PMLR, 2023. 1

[13] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969. 1, 4, 10

[14] Oren Freifeld, Soren Hauberg, Kayhan Batmanghelich, and John W Fisher. Highly-expressive spaces of well-behaved transformations: Keeping it simple. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2911–2919, 2015. 1, 3, 10, 12, 13, 14, 15, 16, 17, 22

[15] Oren Freifeld, Soren Hauberg, Kayhan Batmanghelich, and Jonn W Fisher. Transformations based on continuous piecewise-affine velocity fields. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2496–2509, 2017. 1, 2, 3, 10, 13, 14, 15, 16, 17, 22, 23

[16] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022. 3, 10, 13

[17] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. In *International conference on machine learning*, 2013. 4, 20, 21

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997. 4, 10, 20, 21

[19] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. 4, 20, 21

[20] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016. 4, 20, 21

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 4, 10, 20, 21

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 4, 10, 20, 21

[23] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1319–1327, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. 4, 10, 20, 21, 24

[24] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. 4, 10, 20, 21, 24

[25] Bianca Iancu, Luana Ruiz, Alejandro Ribeiro, and Elvin Isufi. Graph-adaptive activation functions for graph neural networks. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2020. 4, 10, 20, 21, 24

[26] Yifei Zhang, Hao Zhu, Ziqiao Meng, Piotr Koniusz, and Irwin King. Graph-adaptive rectified linear unit for graph neural networks. In *Proceedings of the ACM Web Conference 2022*, pages 1331–1339, 2022. 4, 10, 20, 21, 22, 24

[27] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. 4, 10, 12, 21, 24

[28] Ganesh Sundaramoorthi and Anthony Yezzi. Variational pdes for acceleration on manifolds and application to diffeomorphisms. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 10

[29] Stéphanie Allassonnière, Estelle Kuhn, and Alain Trouvé. Construction of bayesian deformable models via a stochastic approximation algorithm: a convergence study. *Bernoulli*, 2010. 10

[30] Stéphanie Allassonnière, Stanley Durrleman, and Estelle Kuhn. Bayesian mixed effect atlas estimation with a diffeomorphic deformation model. *SIAM Journal on Imaging Sciences*, 8(3): 1367–1395, 2015.

[31] Miaomiao Zhang and P Thomas Fletcher. Bayesian statistical shape analysis on the manifold of diffeomorphisms. *Algorithmic Advances in Riemannian Geometry and Applications: For Machine Learning, Computer Vision, Statistics, and Optimization*, pages 1–23, 2016. 10

[32] Nicki Skafte Detlefsen, Oren Freifeld, and Søren Hauberg. Deep diffeomorphic transformer networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4403–4412, 2018. 10

[33] I nigo Martinez, Elisabeth Viles, and Igor G Olaizola. Closed-form diffeomorphic transformations for time series alignment. In *International Conference on Machine Learning*, pages 15122–15158. PMLR, 2022. 10, 22

[34] Ron Shapira Weber and Oren Freifeld. Regularization-free diffeomorphic temporal alignment nets. In *International Conference on Machine Learning*, pages 30794–30826. PMLR, 2023. 10, 12

[35] Hexiang Wang, Fengqi Liu, Qianyu Zhou, Ran Yi, Xin Tan, and Lizhuang Ma. Continuous piecewise-affine based motion model for image animation. *arXiv preprint arXiv:2401.09146*, 2024. 10

[36] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008. 10

[37] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 10, 23

[38] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. 10, 12, 20, 22

[39] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. 10

[40] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020. 10

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 10

[42] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022.

[43] Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*, 2021. 10

[44] Beatrice Bevilacqua, Moshe Eliasof, Eli Meirom, Bruno Ribeiro, and Haggai Maron. Efficient subgraph gnns by learning effective selection policies. *arXiv preprint arXiv:2310.20082*, 2023. 10

[45] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019. 10

[46] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 10

[47] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019. 10

[48] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019. 10

[49] Emmanuel Noutahi, Dominique Beaini, Julien Horwood, Sébastien Giguère, and Prudencio Tossou. Towards interpretable sparse graph representation learning with laplacian pooling. *arXiv preprint arXiv:1905.11577*, 2019. 10

[50] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021. 10

[51] Vladimír Kunc and Jiří Kléma. Three decades of activations: A comprehensive survey of 400 activation functions for neural networks. *arXiv preprint arXiv:2402.09092*, 2024. 10, 11

[52] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015. 10, 19

[53] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018. 10

[54] Ritabrata Maiti. Adact: Learning to optimize activation function choice through adaptive activation modules. In *The Second Tiny Papers Track at ICLR 2024*, 2024. 10, 11

[55] Moshe Eliasof, Beatrice Bevilacqua, Carola-Bibiane Schönlieb, and Haggai Maron. Granola: Adaptive normalization for graph neural networks. *arXiv preprint arXiv:2404.13344*, 2024. 10

[56] Simone Scardapane, Steven Van Vaerenbergh, Danilo Comminiello, and Aurelio Uncini. Improving graph convolutional networks with non-parametric activation functions. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 872–876. IEEE, 2018. 10

[57] Sammy Khalife and Amitabh Basu. On the power of graph neural networks and the role of the activation function, 2023. 11

[58] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018. 11

[59] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, 2021. 11

[60] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015. 11

[61] Ilan Price, Nicholas Daultry Ball, Samuel CH Lam, Adam C Jones, and Jared Tanner. Deep neural network initialization with sparsity inducing activations. *arXiv e-prints*, pages arXiv–2402, 2024. 11

[62] Abhishek Panigrahi, Abhishek Shetty, and Navin Goyal. Effect of activation functions on the training of overparametrized neural nets. *arXiv preprint arXiv:1908.05660*, 2019. 11

[63] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks, 2024. 11

[64] Zhen Xu, Xiaojin Zhang, and Qiang Yang. Tafs: Task-aware activation function search for graph neural networks. 2023. 11

[65] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019. 12, 22

[66] Tomasz Szandała. Review and comparison of commonly used activation functions for deep neural networks. *Bio-inspired neurocomputing*, pages 203–224, 2021. 13

[67] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021. 13, 17

[68] Thomas Hakon Gronwall. Note on the derivatives with respect to a parameter of the solutions of a system of differential equations. *Annals of Mathematics*, pages 292–296, 1919. 16

[69] Ingrid Daubechies, Ronald DeVore, Simon Foucart, Boris Hanin, and Guergana Petrova. Nonlinear approximation and (deep) relu networks. *Constructive Approximation*, 55(1):127–172, 2022. 19

[70] Tim De Ryck, Samuel Lanthaler, and Siddhartha Mishra. On the approximation of functions by tanh neural networks. *Neural Networks*, 143:732–750, 2021. 19

[71] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse problems*, 34(1):014004, 2017. 20

[72] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Sourav S. Bhowmick, and Juncheng Liu. Pane: scalable and effective attributed network embedding. *The VLDB Journal*, 32(6): 1237–1262, March 2023. ISSN 0949-877X. doi: 10.1007/s00778-023-00790-4. 20, 22, 23

[73] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 20, 22

[74] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000. 20

[75] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. Query-driven active surveying for collective classification. In *10th international workshop on mining and learning with graphs*, volume 8, page 1, 2012. 20, 22

[76] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020. 21, 23, 24

[77] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. 22

[78] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com. 22

[79] Ron A Shapira Weber, Matan Eyal, Nicki Skafte, Oren Shriki, and Oren Freifeld. Diffeomorphic temporal alignment nets. *Advances in Neural Information Processing Systems*, 32, 2019. 22

[80] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2022. 22, 23, 24

[81] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020. 22, 25

A Related Work

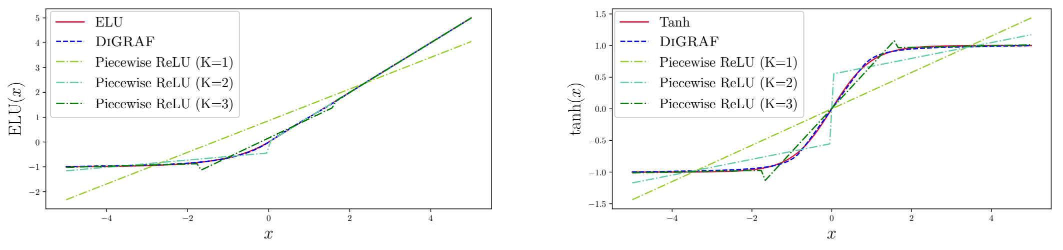
Diffeomorphisms in Neural Networks. A bijection mapping function $f : \mathcal{M} \rightarrow \mathcal{N}$, given two differentiable manifolds $\mathcal{M}$ and $\mathcal{N}$, is termed a *diffeomorphism* if its inverse $f^{-1} : \mathcal{N} \rightarrow \mathcal{M}$ is also differentiable. The challenge in learning diffeomorphisms arises from their computational complexity: early research is often based on complicated infinite dimensional spaces [28], and later advancements have turned to Markov Chain Monte Carlo methods, which still suffer from large computational complexity [29–31]. To address these drawbacks, Freifeld et al. [14, 15] introduced the Continuous Piecewise-Affine Based transformation (CPAB) approach, offering a more pragmatic solution to learning diffeomorphisms by starting from a finite-dimensional space, and allowing for exact diffeomorphism computations in the case of 1D diffeomorphisms – an essential trait in our case, given that activation functions are 1D functions. CPAB has linear complexity and is parallelizable, which can lead to sub-linear complexity in practice [15]. Originally designed for alignment and regression tasks by learning diffeomorphisms, in recent years, CPAB was found to be effective in addressing numerous applications using neural networks, posing it as a suitable framework for learning transformation. For instance, Detlefsen et al. [32] learns CPAB transformations to improve the flexibility of spatial transformer layers, Martinez et al. [33] combines CPAB with neural networks for temporal alignment, Weber and Freifeld [34] introduces a novel loss function that eliminates the need for CPAB deformation regularization in time-series analysis, and Wang et al. [35] utilizes CPAB to model complex spatial transformation for image animation and motion modeling.

Graph Neural Networks. Graph Neural Networks [36] (GNNs) have emerged as a transformative approach in machine learning, notably following the popularity of the message-passing scheme [37]. GNNs enable effective learning from graph-structured data, and can be applied to different tasks, ranging from social network analysis [38] to bioinformatics [39]. In recent years, various GNN architectures were proposed, aiming to address various aspects, from alleviating oversmoothing [40], concerning attention mechanisms in the message passing scheme [41–43], or focusing on the expressive power of the architectures [1–5, 44], given that message-passing based architectures are known to be bounded by the WL graph isomorphism test [27, 45].

Despite advancements, the poor performance of deep GNNs has led to a preference for shallow architecture GCNs [46]. To enhance performance, techniques such as pooling functions have been proposed, introducing generalization by reducing feature map sizes [47]. Methods such as HGP-SL [47], GraphUNet [48], and LaPool [49] introduce pooling layers specifically designed for GNNs. Beyond node feature, the importance of graph structure and positional features is increasingly recognized, with advancements such as GraphGPS [11] and SAN [50] integrating positional and structural encodings through attention-based mechanisms.

General-Purpose Activation Functions. In the last decades, the design of activation functions has seen extensive exploration, resulting in the introduction of numerous high-performing approaches, as summarized in Dubey et al. [16], Kunc and Kléma [51]. The focus has gradually shifted from traditional, static activation functions such as ReLU [13], Sigmoid [21], Tanh [18], and ELU [52], to learnable functions. In the landscape of learnable activation functions, the Maxout [23] unit selects the maximum output from learnable linear functions, and PReLU [22] extends ReLU by learning a negative slope. Additionally, the Swish function [24] augments the SiLU function [53], a Sigmoid-weighted linear unit, with a learnable parameter controlling the amount of non-linearity. The recently proposed AdAct [54] learns a weighted combination of several activation functions. However, these activation functions are not input-adaptive, a desirable property in GNNs.

Graph Activation Functions. Typically, GNNs are coupled with conventional activation functions [27, 38, 41], which were not originally tailored for graph data, graph tasks, or GNN models. These activation functions do not inherently adapt to the structure of the input graph. This lack of graph adaptivity extends to other GNN components as well. However, graph adaptivity was shown to be an important property for several graph learning components, from pre-defined yet graph adaptive activation functions as discussed below, to the normalization layer Eliasof et al. [55] of node features in GNNs. In the context of graph activation functions, early works such as Scardapane et al. [56] propose learning activation functions based on graph kernels, and Iancu et al. [25] introduces Max and Median filters, which operate on local neighborhoods in the graph, thereby offering adaptivity to the input graphs. A notable advancement in graph-adaptive activation functions is GReLU [26], a parametric piecewise affine activation function achieving graph adaptivity by learning parameters



(a) ELU **(b)** Tanh

Figure 3: Approximation of traditional activation functions using CPAB and Piecewise ReLU with varying segment counts $K \in \{1, 2, 3\}$ on a closed interval $\Omega = [-5, 5]$, demonstrating the advantage of utilizing CPAB and its flexibility to model various activation functions.

through a hyperfunction that takes into account the node features and the connectivity of the graph. While these approaches demonstrate the potential to enhance GNN performance compared to standard activation functions, they are constrained by their blueprint, often relying on piecewise ReLU composition, which can be performance-limiting [57]. Moreover, a fixed blueprint limits flexibility, i.e., the ability to express a variety of functions. As we show in Figure 3, attempts to approximate traditional activation functions such as ELU and Tanh using piecewise ReLU composition with different segment counts ($K = 1$, 2, and 3), reveal limited approximation power. On the contrary, our DiGRAF, which leverages CPAB, yields significantly better approximations. Furthermore, we demonstrate the approximation power of activations learned with the CPAB framework in our DiGRAF in Appendix G.1.

Evaluation of Rectified Activation Functions. Rectified activation functions, represented by the Rectified Linear Unit (ReLU), have been widely applied and studied in various neural network architectures due to their simplicity and effectiveness [51, 58, 59]. The prevalent assumption that ReLU's performance is predominantly due to its sparsity is critically examined by Xu et al. [60], suggesting introducing a non-zero slope in the negative part can significantly enhance network performance. Extending this, Price et al. [61] investigates sparsity-inducing activation functions, such as the shifted ReLU, in network initialization and early stages of training. These functions can mitigate overfitting and boost model generalization capabilities. Conversely, it was shown that in overparameterized networks, smoother activation functions, like Tanh and Swish, can enhance the convergence rate, in contrast to the non-smooth characteristics of ReLU [62]. However, the fixed nature of ReLU and many of its variants restricts their ability to adapt the input, resulting in limited power to capture dynamics in learning.

Advancements in Learnable Activation Functions. Recent research has increasingly focused on adaptive and learnable activation functions, which are optimized alongside the learning process of the network. The AdAct framework [54] introduces learnability by combining multiple activation functions into a single module with learnable weighting coefficients. However, these coefficients are fixed after training, limiting the framework's adaptability to varying inputs. A concurrent work by Liu et al. [63] introduces Kolmogorov-Arnold Networks (KAN), a novel architecture that diverges from traditional Multi-Layer Perceptron (MLP) configurations, which applies activation functions to network edges instead of nodes. Unlike our current work, which focuses only on the design of activation functions for GNNs, their research extends beyond this scope and considers a fundamental architecture design. Finally, the recently proposed TAFS [64] learns a task-adaptive (but not graph-adaptive) activation function for GNNs through a bi-level optimization.

B Contrasting with Grid Deformations

In DiGRAF, we treat a node feature (single channel) as a one-dimensional (1D) point. Given the node features matrix $\bar{\mathbf{H}} \in \mathbb{R}^{N \times C}$, we apply DiGRAF per entry in $\bar{\mathbf{H}}$, in accordance with the typical element-wise computation of activation functions. We mention that, while CPAB was originally designed to learn grid deformations, it can be utilized as an activation function blueprint by considering a conceptual shift that we demonstrate in Figure 4.

Given an input function (shown in red in the figure), CPAB deforms grid coordinates, i.e., it transforms it along the horizontal axis, as shown in the blue curve. In contrast, DiGRAF transforms the original data points along the vertical axis, resulting in the green curve. This conceptual shift can be seen visually from the arrows showing the different dimensions of transformations. We therefore refer to the vertical transformation of the data as their activations.

C Implementation Details of DiGRAF

Multiple Graphs in one Batch. Consider a set of graphs $S = \{G_1, G_2, \cdots, G_B\}$ with a batch size of B. Let $N_S = N_1 + N_2 + \cdots + N_B$ represent the cumulative number of nodes across the graph dataset. The term $N_{\text{max}} \triangleq \max(N_1, N_2, \cdots, N_B)$ denotes the largest node count present in any single graph within S.

To create a unified feature matrix for S that encompasses all graphs in the batch, we standardize the dimension by padding each feature matrix $\mathbf{X}_i \in \mathbb{R}^{N_i \times C}$, $i \in [B]$ for graph $G_i \in S$ from N_i to N_{max} with zeros. The combined feature matrix $\mathbf{X}_S$ is constructed by concatenating the transposed feature matrices $\mathbf{X}_i^\intercal$ $\forall i \in [B]$, resulting in a matrix that lies in the domain $\mathbb{R}^{(B \cdot C) \times N_{\text{max}}}$. This matrix is permutation invariant; while relabeling nodes changes the row indices, it does not affect the overall transformation process. Therefore, DiGRAF can handle multiple graphs in a batch. In practice, to avoid the overhead of padding, we use the batching support from Pytorch-Geometric [65].

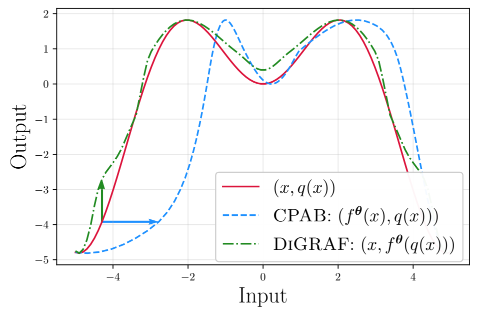


Figure 4: Different transformation strategies. The input function (red), CPAB transformation (blue), and DiGRAF transformation (green), within $\Omega = [-5, 5]$ using the same $\boldsymbol{\theta}$. While CPAB stretches the input, DiGRAF stretches the output, showcasing the distinctive impact of each approach.

Implementation Details of GNN$_{\text{ACT}}$. In Section 3.2, we examined two distinct approaches to learn the diffeomorphism parameters $\boldsymbol{\theta}^{(l)}$, either directly or through GNN$_{\text{ACT}}$. As shown in Appendix E, $\boldsymbol{\theta}^{(l)}$ determines the velocity field $v^{\boldsymbol{\theta}^{(l)}}$. Predicting a graph-dependent $\boldsymbol{\theta}^{(l)}$ adds graph-adaptivity to the activation function $T^{(l)}$. In DiGRAF we achieve this by employing another GNN i.e., GNN$_{\text{ACT}}$, described below.

The backbone of GNN$_{\text{ACT}}$ utilizes the same structure as the primary network layers GNN$^{(l)}_{\text{LAYER}}$, that is, GCN [38] or GIN [27]. It is important to note, that while GNN$_{\text{ACT}}$ has a similar structure to the primary network GNN, it has its own set of learnable weights, and it is shared among the layers, unlike the primary GNN layers GNN$^{(l)}_{\text{LAYER}}$. The hidden dimensions and the number of layers of GNN$_{\text{ACT}}$ are hyperparameters. The weight parameters of GNN$_{\text{ACT}}$ are trained concurrently with the main network weights. As described in Equation (9), after the computation of GNN$_{\text{ACT}}$, a pooling layer denoted by POOL is placed to aggregate node features. This aggregation squashes the node dimension such that the output is not dependent on the specific order of nodes, and it yields the vector of parameters $\boldsymbol{\theta}^{(l)}$.

Rescaling $\bar{\mathbf{H}}^{(l)}$. Following the implementation of Freifeld et al. [14], the default 1D domain for CPAB is set as $[0, 1]$. To enhance the flexibility of $T^{(l)}$ and ensure its adaptability across various input datasets, DiGRAF extends the domain to $\Omega = [a, b] \subset \mathbb{R}$ with $a < b$ as shown in Section 2.1. To match the two domains, we rescale the intermediate feature matrix $\bar{\mathbf{H}}^{(l)}$ from Ω to the unit interval $[0, 1]$ before passing it to $T^{(l)}$. Let $r = \frac{b-a}{2}$, then rescaling is performed using the function $f(x) = (x + r)/(2r)$. Data points outside this range will retain their original value, effectively acting as an identity function outside the domain Ω.

Velocity Field Regularization. To ensure the smoothness of the velocity field, which will encourage training stability [34], we incorporate a regularization term in the learning procedure of $\boldsymbol{\theta}^{(l)}$. Namely, we follow the Gaussian smoothness prior on the CPA velocity field from Freifeld et al. [14], which was shown to be effective in maintaining smooth transformations. The regularization term is defined as follows:

$$\mathcal{R}(\{\boldsymbol{\theta}^{(l)}\}_{l=1}^{L}) = \sum_{l=1}^{L} \boldsymbol{\theta}^{(l)\top} \Sigma_{\text{CPA}}^{-1} \boldsymbol{\theta}^{(l)}, \tag{10}$$

where Σ_{CPA} represents the covariance of a zero-mean Gaussian smoothness prior defined as in Freifeld et al. [14]. This function is parameterized by two hyperparameters: λ_{Σ} and λ_{smooth}, which control the overall variance and the smoothness respectively.

We further maintain the boundedness of $\boldsymbol{\theta}^{(l)}$ by employing a hyperbolic tangent function (Tanh). In this way, $\boldsymbol{\theta}^{(l)}$ remains in $[-1, 1]$ when applied in $T^{(l)}$ in Equation (7), ensuring that the velocity field parameters remain bounded, encouraging the overall training stability of the model.

Training Loss Function. As described in Equation (10), we employ a regularization term for the velocity field to maintain the smoothness of the activation function. To control the strength of regularization, we introduce a hyperparameter λ. We denote $\mathcal{L}_{\text{TASK}}$ as the loss function of the downstream task (i.e. cross-entropy loss in case of classification and mean absolute error in case of regression tasks), and the overall training loss of DiGRAF, denoted as $\mathcal{L}_{\text{TOTAL}}$ is given as

$$\mathcal{L}_{\text{TOTAL}} = \mathcal{L}_{\text{TASK}} + \lambda\, \mathcal{R}(\{\boldsymbol{\theta}^{(l)}\}_{l=1}^{L}). \tag{11}$$

D Properties of DiGRAF

In this section, we focus on understanding the theoretical properties of DiGRAF, highlighting the compelling attributes that establish it as a performant activation function for GNNs.

DiGRAF yields differentiable activations. By construction, DiGRAF learns a diffeomorphism, which is differentiable by definition. Being differentiable everywhere is considered beneficial as it allows for smooth weight updates during backpropagation, preventing the zigzagging effect in the optimization process [66].

DiGRAF is bounded within the input-output domain Ω. We point out in Remark F.3 that the diffeomorphism $T^{(l)}(\cdot;\boldsymbol{\theta}^{(l)})$ is a $\Omega \to \Omega$ transformation. Any diffeomorphism is continuous, and by the extreme value theorem, $T^{(l)}(\cdot;\boldsymbol{\theta}^{(l)})$ is bounded in Ω. This prevents the activation values from becoming excessively large, a property linked to faster convergence [16].

DiGRAF can learn to be zero-centered. Benefiting from its flexibility, DiGRAF has the capacity to learn activation functions that are inherently zero-centered. As an input-adaptive activation function governed by a parameters vector $\boldsymbol{\theta}^{(l)}$, DiGRAF can be adjusted through $\boldsymbol{\theta}^{(l)}$ to maintain a zero-centered nature. This property is associated with accelerated convergence in neural network training [16].

DiGRAF is efficient. DiGRAF exhibits linear computational complexity, and can further achieve sub-linear running times via parallelization in practice [15]. Moreover, with the existence of a closed-form solution for $f^{\boldsymbol{\theta}^{(l)}}$ and its gradient in the 1D case [14], the computations of CPAB can be done efficiently. Additionally, the measured runtimes, detailed in Appendix I, underscore the complexity comparability of DiGRAF with other graph activation functions.

In addition to the above properties, which follow from our design choice of learning diffeomorphisms through the CPAB framework, we briefly present the following properties, which are formalized and proven in Appendix F.

DiGRAF is permutation equivariant. We demonstrate in Proposition F.4 that DiGRAF exhibits permutation equivariance to node numbering, ensuring that its behavior remains consistent regardless of the ordering of the graph nodes, which is a key desired property in designing GNN components [67].

DiGRAF is Lipschitz continuous. We show in Proposition F.2 that DiGRAF is Lipschitz continuous and derive its Lipschitz constant. Since it is also bounded, we can combine the two results, which leads us to the following proposition:

Proposition D.1 (The boundedness of $T(\cdot;\boldsymbol{\theta}^{(l)})$ in DiGRAF)**.** *Given a bounded domain $\Omega = [a, b] \subset \mathbb{R}$ where $a < b$, and any two arbitrary points $x, y \in \Omega$, the maximal difference of a diffeomorphism*

$T(\cdot;\boldsymbol{\theta}^{(l)})$ with parameter $\boldsymbol{\theta}^{(l)}$ in DIGRAF *is bounded as follows:*

$$|T(x;\boldsymbol{\theta}^{(l)}) - T(y;\boldsymbol{\theta}^{(l)})| \leq \min(|b-a|, |x-y|\exp(C_{v^{\boldsymbol{\theta}^{(l)}}})) \tag{12}$$

where $C_{v^{\boldsymbol{\theta}^{(l)}}}$ is the Lipschitz constant of the CPA velocity field $v^{\boldsymbol{\theta}^{(l)}}$.

DIGRAF extends commonly used activation functions. CPAB [14, 15], which is used as a framework to learn the diffeomorphism in DIGRAF, is capable of learning and representing a wide range of diffeomorphic functions. When used as an activation function, the transformation $T^{(l)}(\cdot;\boldsymbol{\theta}^{(l)})$ in DIGRAF adapts to the specific graph and task by learning different $\boldsymbol{\theta}^{(l)}$ parameters, rather than having a fixed diffeomorphism. Examples of popular and commonly used diffeomorphisms utilized as activations include Sigmoid, Tanh, Softplus, and ELU, as we show in Appendix F. Extending this approach is our DIGRAF that learns the diffeomorphism during training rather than selecting a pre-defined function.

E Overview of CPA Velocity Fields and CPAB Transformations

In this Section, we drop the layer notations l for simplicity. Throughout this section we will use the definitions presented in CPAB, a framework that allows to efficiently learn flexible diffeomorphisms [14, 15].

Let $\Omega = [a, b] \subset \mathbb{R}$ be a closed interval, where $a < b$. We discretize Ω using a tessellation $\mathcal{P}$ with $\mathcal{N}_\mathcal{P}$ intervals, which, in practice, is oftentimes an equispaced 1D meshgrid with $\mathcal{N}_\mathcal{P}$ segments [15].

Definition E.1 (Tessellation of a closed interval [14])**.** A tessellation $\mathcal{P}$ of size $\mathcal{N}_\mathcal{P}$ subintervals of a closed interval $\Omega = [a, b]$ in $\mathbb{R}$ is a partitioning $\{[x_i, x_{i+1}]\}_{i=0}^{\mathcal{N}_\mathcal{P}-1}$ that satisfies the following properties:

(1) $x_0 = a$ and $x_{\mathcal{N}_\mathcal{P}} = b$

(2) Each point $x \in \Omega$ lies in at least one subinterval $[x_i, x_{i+1}]$

(3) The intersection of any two subintervals $[x_i, x_{i+1}]$ and $[x_{i+1}, x_{i+2}]$ is exactly $\{x_{i+1}\}$

(4) $\bigcup_{i=0}^{\mathcal{N}_\mathcal{P}-1} [x_i, x_{i+1}] = \Omega$

Our goal in this paper is to learn a diffeomorphism $f : \Omega \to \Omega$ that we will use as an activation function. Formally, a diffeomorphism is defined as follows:

Definition E.2 (Diffeomorphism on a closed interval Ω)**.** A diffeomorphism on a closed interval $\Omega \subset \mathbb{R}$ is any function $f : \Omega \to \Omega$ that is (1) bijective, (2) differentiable, and (3) has a differentiable inverse f^{-1}.

To instantiate a CPAB diffeomorphism f, we define a continuous piecewise-affine (CPA) velocity field $v^{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{\mathcal{N}_\mathcal{P}-1}$. We display examples of velocity fields $v^{\boldsymbol{\theta}}$ for various instances of $\boldsymbol{\theta}$ in Figure 5a to demonstrate the distinct influence of $\boldsymbol{\theta}$ on $v^{\boldsymbol{\theta}}$. Formally, a velocity field $v^{\boldsymbol{\theta}}$ is defined as follows:

Definition E.3 (CPA velocity field $v^{\boldsymbol{\theta}}$ on Ω)**.** Given a tessellation $\mathcal{P}$ with $\mathcal{N}_\mathcal{P}$ intervals on a closed domain Ω, any velocity field $v^{\boldsymbol{\theta}} : \Omega \to \mathbb{R}$ is termed continuous and piecewise-affine if (1) $v^{\boldsymbol{\theta}}$ is continuous, and (2) $v^{\boldsymbol{\theta}}$ is an affine transformation on each interval of $\mathcal{P}$.

The CPA velocity field $v^{\boldsymbol{\theta}}$ defines a differentiable trajectory $\phi^{\boldsymbol{\theta}}(x, t) : \Omega \times \mathbb{R} \to \Omega$ for each $x \in \Omega$. The trajectories are computed by integrating the velocity field $v^{\boldsymbol{\theta}}$ to time t, and are used to construct the CPAB diffeomorphism. We visualize the resulting diffeomorphism in Figure 5b with matching colors denoting corresponding pairs of $v^{\boldsymbol{\theta}}$ and $f^{\boldsymbol{\theta}}(x)$.

We now discuss how the velocity fields are computed following the methodologies presented by Freifeld et al. [14, 15] and highlight the relations between $v^{\boldsymbol{\theta}}$, $\boldsymbol{\theta}$ and $\mathcal{P}$.

The vector of parameters $\boldsymbol{\theta}$ is linked to the subintervals in $\mathcal{P}$, whose dimension is determined by the number of intervals $\mathcal{N}_\mathcal{P}$. Similar to Freifeld et al. [14], we impose boundary constraints that mandate the velocity at the boundary of the tessellation to be zero, i.e., $v^{\boldsymbol{\theta}}(0) = v^{\boldsymbol{\theta}}(1) = 0$. This boundary condition allows us to compose the diffeomorphism in the domain Ω with an identity function for any

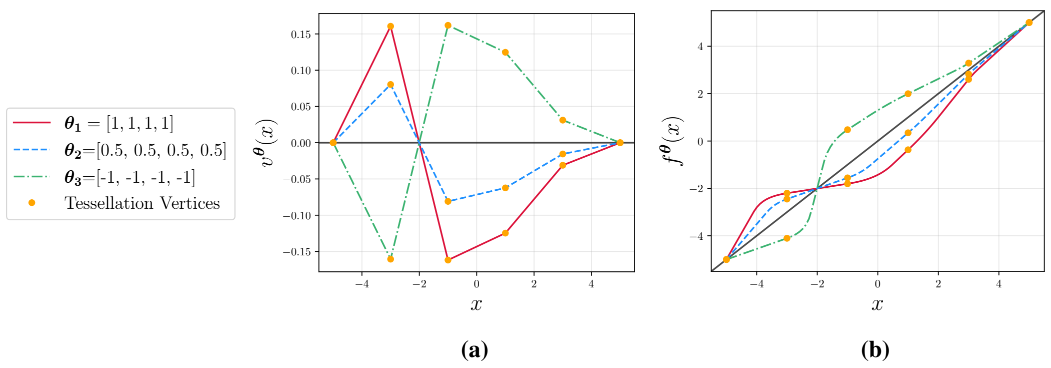


Figure 5: An example of CPA velocity fields $v^{\boldsymbol{\theta}}$ defined on the interval $\Omega = [-5, 5]$ with a tessellation $\mathcal{P}$ consisting of five subintervals. The three different parameters, $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, and $\boldsymbol{\theta}_3$ define three distinct CPA velocity fields (Figure 5a) resulting in separate CPAB diffeomorphisms $f^{\boldsymbol{\theta}}(x)$ (Figure 5b).

values outside the domain. Under this constraint, the degrees of freedom (number of parameters) for θ is $\mathcal{N}_{\mathcal{P}} - 1$.

The velocity field is then defined as follows:

Definition E.4 (Relation between $\boldsymbol{\theta}$ and $v^{\boldsymbol{\theta}}$, taken from Freifeld et al. [15])**.** Given a tessellation $\mathcal{P}$ with $\mathcal{N}_{\mathcal{P}}$ intervals on a closed domain $\Omega = [a, b]$, as defined in Definition E.1. Given a parameter $\boldsymbol{\theta} \in \mathbb{R}^{\mathcal{N}_{\mathcal{P}}-1}$ and an arbitrary point x within the domain, a continuous piecewise-affine velocity field $v^{\boldsymbol{\theta}}$ can present as follows:

$$v^{\boldsymbol{\theta}}(x) = \sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-2} \boldsymbol{\theta}_j \mathbf{b}_j \tilde{x}, \tag{13}$$

where $\{\mathbf{b}_j\}_{j=0}^{\mathcal{N}_{\mathcal{P}}-2}$ is an orthonormal basis of the space of velocity fields $\mathcal{V}$, such that $v^{\boldsymbol{\theta}} \in \mathcal{V}$, and $\tilde{\boldsymbol{x}} = \begin{bmatrix} x \\ 1 \end{bmatrix}$.

F Proofs For Properties of DIGRAF

Similar to Appendix E, for simplicity, in this section, we drop the layer notations l.

In this section, we present the propositions and proofs for the properties outlined in Appendix D. We begin by remarking that as shown in Appendix D, DIGRAF is bounded within the domain $\Omega = [a, b]$, where $a < b$ by construction. We then present Proposition F.1 that outlines the Lipschitz constant of the velocity field $v^{\boldsymbol{\theta}}$, followed by Proposition F.2, showing that DIGRAF is also Lipschitz continuous, and provide an upper bound on its Lipschitz constant.

Proposition F.1 (The Lipschitz Constant of $v^{\boldsymbol{\theta}}$)**.** *Given two arbitrary points $x, y \in \mathbb{R}$, and velocity field parameters $\boldsymbol{\theta} \in \mathbb{R}^{\mathcal{N}_{\mathcal{P}}-1}$ that define the continuous piecewise-affine velocity field $v^{\boldsymbol{\theta}}$, there exists a Lipschitz constant $C_{v^{\boldsymbol{\theta}}} = \sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-2} |\boldsymbol{\theta}_j|$ such that*

$$\left|v^{\boldsymbol{\theta}}(x) - v^{\boldsymbol{\theta}}(y)\right| \leq C_{v^{\boldsymbol{\theta}}} \|(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{y}})\|_2, \tag{14}$$

where $|\cdot|$ and $\|\cdot\|_2$ denote the absolute value of a scalar and the ℓ_2 norm of a vector, respectively.

Proof. First, we note that it was shown in Freifeld et al. [14, 15] that $v^{\boldsymbol{\theta}}$ is Lipschitz continuous, and now we provide a derivation of that Lipschitz constant. Following Definition E.4, the velocity field $v^{\boldsymbol{\theta}}$ is defined as $v^{\boldsymbol{\theta}}(x) = \sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-2} \boldsymbol{\theta}_j \mathbf{b}_j \tilde{x}$, where $\{\mathbf{b}_j\}_{j=0}^{\mathcal{N}_{\mathcal{P}}-2}$ is an orthonormal basis of the velocity space. By the definition of $v^{\boldsymbol{\theta}}(x)$ and $v^{\boldsymbol{\theta}}(y)$, we have the following:

$$\begin{aligned}
\left|v^{\boldsymbol{\theta}}(x) - v^{\boldsymbol{\theta}}(y)\right| &= \left|\sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-2} \boldsymbol{\theta}_j \mathbf{b}_j \tilde{\boldsymbol{x}} - \sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-2} \boldsymbol{\theta}_j \mathbf{b}_j \tilde{y}\right| && (15)\\
&= \left|\sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-0} \boldsymbol{\theta}_j \mathbf{b}_j (\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{y}})\right| && (16)\\
&\leq \sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-2} |\boldsymbol{\theta}_j| \|\mathbf{b}_j\|_2 \|(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{y}})\|_2 && (17)\\
&= \sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-2} |\boldsymbol{\theta}_j| \|(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{y}})\|_2 && (18)\\
&= \|(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{y}})\|_2 \sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-2} |\boldsymbol{\theta}_j| && (19)\\
&= C_{v^{\boldsymbol{\theta}}} \|(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{y}})\|_2, && (20)
\end{aligned}$$

where the transition between Equation (16) and Equation (17) follows from the triangle inequality, and the transition between Equation (17) and Equation (18) follows from $\mathbf{b}_j$ being an orthonormal vector.

From the derivation above, and the fact that we know from Freifeld et al. [14, 15] that the velocity field is Lipschitz continuous, we conclude that the Lipschitz constant $C_{v^{\boldsymbol{\theta}}}$ of $v^{\boldsymbol{\theta}}$ reads $C_{v^{\boldsymbol{\theta}}} = \sum_{j=0}^{\mathcal{N}_{\mathcal{P}}-2} |\boldsymbol{\theta}_j|$. □

Given the Lipschitz constant $C_{v^{\boldsymbol{\theta}}}$ for $v^{\boldsymbol{\theta}}$, we proceed to demonstrate that the transformation $T(\cdot;\boldsymbol{\theta})$ in DIGRAF is Lipschitz continuous, as well as bounding its Lipschitz constant.

Proposition F.2 (The Lipschitz Constant of DIGRAF)**.** *The diffeomorphic function $T(\cdot;\boldsymbol{\theta})$ in* DIGRAF *is defined in Equation (6) for a given set of weights $\boldsymbol{\theta}$, which in turn define the velocity field $v^{\boldsymbol{\theta}}$. Let $x, y \in \mathbb{R}$ be two arbitrary points, then the following inequality is satisfied:*

$$|T(x;\boldsymbol{\theta}) - T(y;\boldsymbol{\theta})| \leq |x-y| \exp(C_{v^{\boldsymbol{\theta}}}) \tag{21}$$

where $C_{v^{\boldsymbol{\theta}}}$ is the Lipschitz constant of $v^{\boldsymbol{\theta}}$.

Proof. We begin by substituting $T(\cdot;\boldsymbol{\theta})$ with Equation (1) and Equation (6). Utilizing Proposition F.1, we then establish an upper bound for $|T(x;\boldsymbol{\theta}) - T(y;\boldsymbol{\theta})|$ as follows:

$$\begin{aligned}
|T(x;\boldsymbol{\theta}) - T(y;\boldsymbol{\theta})| &= |x + \int_0^1 v^{\boldsymbol{\theta}}\left(\phi^{\boldsymbol{\theta}}(x,\tau)\right) d\tau - y - \int_0^1 v^{\boldsymbol{\theta}}\left(\phi^{\boldsymbol{\theta}}(y,\tau)\right) d\tau| && (22)\\
&\leq |x-y| + C_{v^{\boldsymbol{\theta}}} \int_0^1 \left|(\phi^{\boldsymbol{\theta}}(x,\tau) - \phi^{\boldsymbol{\theta}}(y,\tau))\right| && (23)\\
&\leq |x-y| \exp(C_{v^{\boldsymbol{\theta}}}), && (24)
\end{aligned}$$

where $C_{v^{\boldsymbol{\theta}}}$ is the Lipschitz constant of $v^{\boldsymbol{\theta}}$ (Proposition F.1) and the last transition follows from Grönwall's inequality [68]. Consequently, the Lipschitz constant of DIGRAF is bounded from above by $\exp(C_{v^{\boldsymbol{\theta}}})$. □

Now that we established that $T(\cdot;\boldsymbol{\theta})$ is Lipschitz continuous and presented an upper bound, we investigate what is the maximal difference in the output of $T(\cdot;\boldsymbol{\theta})$ with respect to two arbitrary inputs $x, y \in \Omega$, and whether it can be bounded. To address this, we present the following remark:

Remark F.3. Given a bounded domain $\Omega = [a,b],\ a < b$, by construction, the diffeomorphism $T(\cdot;\boldsymbol{\theta})$ with parameter $\boldsymbol{\theta}$ in DIGRAF, as in Equation (7), is a $\Omega \to \Omega$ transformation [14, 15]. Therefore, by the max value theorem, the maximal output discrepancy for arbitrary $x, y \in \Omega$ is $|b-a|$, i.e., $|T(x;\boldsymbol{\theta}) - T(y;\boldsymbol{\theta})| \leq |b-a|$.

Combining the Proposition F.1, Proposition F.2 and Remark F.3, we formalize and prove the following proposition:

Proposition D.1 (The boundedness of $T(\cdot;\boldsymbol{\theta}^{(l)})$ in DIGRAF). *Given a bounded domain $\Omega = [a,b] \subset \mathbb{R}$ where $a < b$, and any two arbitrary points $x, y \in \Omega$, the maximal difference of a diffeomorphism $T(\cdot;\boldsymbol{\theta}^{(l)})$ with parameter $\boldsymbol{\theta}^{(l)}$ in* DIGRAF *is bounded as follows:*

$$|T(x;\boldsymbol{\theta}^{(l)}) - T(y;\boldsymbol{\theta}^{(l)})| \leq \min(|b-a|, |x-y| \exp(C_{v^{\boldsymbol{\theta}^{(l)}}})) \tag{12}$$

where $C_{v^{\boldsymbol{\theta}^{(l)}}}$ is the Lipschitz constant of the CPA velocity field $v^{\boldsymbol{\theta}^{(l)}}$.

Proof. In Proposition F.2 we presented an upper bound on the Lipschitz constant of $T(\cdot;\boldsymbol{\theta})$, and in F.3 we also presented an upper bound on the maximal difference between the application of $T(\cdot;\boldsymbol{\theta})$ on two inputs x, y. Combining the two bounds, we get the following inequality:

$$|T(x;\boldsymbol{\theta}) - T(y;\boldsymbol{\theta})| \leq \min(|b-a|, |x-y| \exp(C_{v^{\boldsymbol{\theta}}})). \tag{25}$$

□

The result in Proposition D.1 gives us a tighter upper bound on the boundedness of the transformation $T(\cdot;\boldsymbol{\theta})$ in our DIGRAF that is related both to the hyperparameters a, b, as well as the learned velocity field parameters $\boldsymbol{\theta}$.

Next, we discuss another property outlined in Appendix D, demonstrating that DIGRAF is permutation equivariant – a desirable property when designing a GNN component [67].

Proposition F.4 (DIGRAF is permutation equivariant.). *Consider a graph encoded by the adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, where N is the number of nodes. Let $\bar{\mathbf{H}}^{(l)} \in \mathbb{R}^{N \times C}$ be the intermediate node features at layer l, before the element-wise application of our* DIGRAF*. Let $\mathbf{P}$ be an $N \times N$ permutation matrix. Then,*

$$\text{DIGRAF}(\mathbf{P}\bar{\mathbf{H}}^{(l)}, \boldsymbol{\theta}_P^{(l)}) = \mathbf{P}\ \text{DIGRAF}(\bar{\mathbf{H}}^{(l)}, \boldsymbol{\theta}^{(l)}) \tag{26}$$

where $\boldsymbol{\theta}_P^{(l)}$ and $\boldsymbol{\theta}^{(l)}$ are obtained by feeding $\mathbf{P}\bar{\mathbf{H}}^{(l)}$ and $\bar{\mathbf{H}}^{(l)}$, respectively, to Equation (9).

Proof. We break down the proof into two parts. First, we show that GNN_{ACT} outputs the same $\boldsymbol{\theta}$ under permutations, that is we show

$$\boldsymbol{\theta}_P^{(l)} = \boldsymbol{\theta}^{(l)}.$$

Second, we prove that the activation function $T^{(l)}$ is permutation equivariant, ensuring the overall method maintains this property.

To begin with, recall that Equation (9) is composed by GNN_{ACT}, which is permutation equivariant, and by a pooling layer, which is permutation invariant. Therefore their composition is permutation invariant, that is $\boldsymbol{\theta}_P^{(l)} = \boldsymbol{\theta}^{(l)}$.

Prior to the activation function layer $T^{(l)}$, $\bar{\mathbf{H}}^{(l)}$ undergoes rescaling as described in Appendix C, which is permutation equivariant as it operates element-wise. Finally, since activation function $T^{(l)}$ acts element-wise, and given that $\boldsymbol{\theta}$ remains unchanged, the related CPA velocity fields are identical, resulting in the same transformed output for each entry, despite the entries being permuted in $\mathbf{P}\bar{\mathbf{H}}^{(l)}$. Therefore, DIGRAF is permutation equivariant. □

F.1 Diffeomorphic Activation Functions

In this section, we provide several examples of popular and well-known diffeomorphic functions, contributing to our motivation to utilize diffeomorphisms as a blueprint for learning graph activation functions. We remark that, differently from standard activation functions, our DIGRAF does not need to follow a predefined, fixed template, but can instead learn a diffeomorphism best suited for the task and input, as $T^{(l)}$ within CPAB can represent a wide range of diffeomorphisms [14, 15].

We recall that, as outlined in Definition E.2, a function is classified as a diffeomorphism if it is (1) bijective, (2) differentiable, and (3) has a differentiable inverse.

Sigmoid. We denote the Sigmoid activation function as $\sigma : \mathbb{R} \to (0,1)$, defined by

$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

To prove that σ is a diffeomorphism, we first establish its bijectivity. Injectivity follows from observing that for any distinct points x_1 and x_2 in $\mathbb{R}$, $\sigma(x_1) = \frac{1}{1+e^{-x_1}}$ can only equal $\sigma(x_2) = \frac{1}{1+e^{-x_2}}$ if and only if $x_1 = x_2$. For surjectivity, we represent x as a function of y, such that $y = \frac{1}{1+e^{-x}} \implies x = -\ln\left(\frac{1-y}{y}\right)$, ensuring that for every $y \in (0,1)$ there is an element $x \in \mathbb{R}$ such that $\sigma(x) = y$.

To demonstrate differentiability, we examine the derivative of σ. The derivative

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x)),$$

which is continuous. Additionally, the inverse function

$$\sigma^{-1}(y) = -\ln\left(\frac{1-y}{y}\right)$$

is also bijective and differentiable. Thus, with all these requirements satisfied, σ is indeed a diffeomorphism.

Tanh. The hyperbolic tangent function

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

is a diffeomorphism from $\mathbb{R}$ to $(-1,1)$. To establish this, we demonstrate that $\tanh$ is bijective and differentiable, with a differentiable inverse function.

Firstly, $\tanh$ is injective because if $\tanh(x_1) = \tanh(x_2)$, then $x_1 = x_2$. It is also surjective because for any $y \in (-1,1)$, there exists $x = \frac{1}{2}\ln\left(\frac{1+y}{1-y}\right)$ such that $\tanh(x) = y$.

The derivative

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$

is continuous and positive. Additionally, the inverse function

$$\tanh^{-1}(y) = \frac{1}{2}\ln\left(\frac{1+y}{1-y}\right)$$

is continuously differentiable. Therefore, $\tanh$ qualifies as a diffeomorphism.

Softplus. To establish the Softplus function

$$\text{softplus}(x) = \ln(1+e^x)$$

as a diffeomorphism from $\mathbb{R}$ to $(0,\infty)$, we first demonstrate its injectivity and surjectivity.

Assuming softplus(x_1) = softplus(x_2), we obtain $e^{x_1} = e^{x_2}$, implying $x_1 = x_2$, hence establishing injectivity. For any $y \in (0,\infty)$, we find an $x \in \mathbb{R}$ such that $y = \ln(1+e^x)$, ensuring surjectivity.

The derivative of the Softplus function,

$$\frac{d}{dx}\text{softplus}(x) = \frac{e^x}{1+e^x} = \sigma(x),$$

where $\sigma(x)$ is the Sigmoid function, known to be continuous and differentiable. Therefore, softplus(x) is continuously differentiable.

Considering the inverse of the Softplus function,

$$\text{softplus}^{-1}(y) = \ln(e^y - 1),$$

its derivative is

$$\frac{d}{dy}\text{softplus}^{-1}(y) = \frac{e^y}{e^y - 1},$$

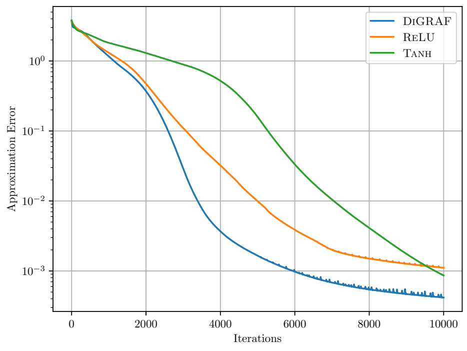


Figure 6: The approximation error of the Peaks function (Equation (27)) with ReLU, Tanh, and DiGRAF.

which is continuous for all $y > 0$, indicating that $\text{softplus}^{-1}(y)$ is continuously differentiable for all $y > 0$. Therefore, we conclude that the softplus function qualifies as a diffeomorphism.

ELU. The ELU activation function [52] is defined as below:

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

where $\alpha \in \mathbb{R}$ is a constant that scales the negative part of the function. To demonstrate that ELU is bijective, we analyze its injectivity and surjectivity. For $x > 0$, ELU acts as the identity function, which is inherently injective. For $x \leq 0$, $\alpha(e^{x_1} - 1) = \alpha(e^{x_2} - 1)$, implies $x_1 = x_2$. The inverse function for ELU is given by:

$$\text{ELU}^{-1}(y) = \begin{cases} y & \text{if } y > 0 \\ \ln(\frac{y}{\alpha} + 1) & \text{if } y \leq 0 \end{cases}$$

This inverse maps every value in the codomain back to a unique value in the domain, proving that ELU is surjective.

Next, we examine the continuity of ELU. At $x = 0$, $\text{ELU}(x = 0) = \alpha(e^0 - 1) = 0$. Next, we check the limits for both sides of 0. For $x > 0$, $\lim_{x\to 0^+} \text{ELU}(x) = \lim_{x\to 0^+} x = 0$, while for $x \leq 0$, we have $\lim_{x\to 0^-} \text{ELU}(x) = \lim_{x\to 0^-} \alpha(e^x - 1) = 0$. Since both limits are equal, the ELU function is continuous at $x = 0$. For the derivative of ELU, i.e.,

$$\frac{d}{dx}\text{ELU}(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha e^x & \text{if } x \leq 0 \end{cases}$$

at $x = 0$, we have $\frac{d}{dx}\text{ELU}(x) = \alpha e^0 = \alpha$. By setting $\alpha = 1$, the derivative at $x = 0$ matches the derivative for $x > 0$, making the derivative continuous.

The derivative for the inverse function is

$$\frac{d}{dy}\text{ELU}^{-1}(y) = \begin{cases} 1 & \text{if } y > 0 \\ \frac{1}{y+\alpha} & \text{if } y \leq 0 \end{cases}$$

which is also continuously differentiable. Hence, ELU is a diffeomorphism.

G Additional Results

G.1 Function Approximation with CPAB

The combination of learned linear layers together with non-linear functions such as ReLU and Tanh are well-known to yield good function approximations [69, 70]. Therefore, when designing an activation function blueprint, i.e., the template by which the activation function is learned, it

Table 2: Comparision of node classification accuracy (%) ↑ on different datasets using various baselines with DIGRAF. The top three methods are marked by **First**, **Second**, **Third**.

Method ↓ / Dataset →	BLOG CATALOG	FLICKR	CITESEER	CORA	PUBMED
STANDARD ACTIVATIONS					
GCN + Identity	74.8±0.5	53.5±1.1	**69.1±1.6**	80.5±1.2	77.6±2.1
GCN + ReLU [38]	72.1±1.9	50.7±2.3	67.7±2.3	79.2±1.4	77.6±2.2
GCN + LeakyReLU [17]	72.6±2.1	51.0±2.0	68.4±1.8	79.4±1.6	76.8±1.6
GCN + Tanh [18]	73.9±0.5	51.3±1.5	**69.1±1.4**	80.5±1.3	**77.9±2.1**
GCN + GeLU [19]	75.8±0.5	**56.1±1.3**	67.8±1.7	79.3±1.9	77.1±2.7
GCN + ELU [20]	74.8±0.5	53.4±1.1	**69.1±1.7**	**80.7±1.2**	77.5±2.2
GCN + Sigmoid [21]	39.7±4.5	18.3±1.2	27.9±2.1	32.1±2.3	52.8±6.6
LEARNABLE ACTIVATIONS					
GCN + PReLU [22]	74.8±0.4	53.2±1.5	**69.2±1.5**	80.5±1.2	77.6±2.1
GCN + Maxout [23]	72.4±1.4	54.0±1.8	68.5±2.2	79.8±1.5	77.3±2.9
GCN + Swish [24]	**76.0±0.7**	55.7±1.4	67.7±1.8	79.2±1.1	77.3±2.8
GRAPH ACTIVATIONS					
GCN + Max [25]	72.0±1.0	47.5±0.9	59.7±2.9	76.0±1.8	75.0±1.4
GCN + Median [25]	**77.7±0.7**	**58.3±0.6**	61.3±2.7	77.1±1.1	75.7±2.5
GCN + GReLU [26]	73.7±1.2	54.4±1.6	68.5±1.9	**81.8±1.8**	**78.9±1.7**
GCN + DIGRAF (W/O ADAP.)	80.8±0.6	68.6±1.8	69.2±2.1	81.5±1.1	78.3±1.6
GCN + DIGRAF	**81.6±0.8**	**69.6±0.6**	**69.5±1.4**	**82.8±1.1**	**79.3±1.4**

is important to consider its approximation power. In Figure 3, we demonstrate the ability of the CPAB framework to approximate known activation functions. We now show additional evidence for the flexibility and power of CPAB as a framework for learning activation functions, leading to our DIGRAF. To this end, we consider the ability of a multilayer perceptron (MLP) with various activation functions (ReLU, Tanh, and DIGRAF) to approximate the well-known *'peaks'* function that mathematically reads:

$$g(x,y) = 3(1-x)^2\exp(-(x^2)-(y+1)^2) - 10(\frac{x}{5}-x^3-y^5)\exp(-x^2-y^2) - \frac{1}{3}\exp(-(x+1)^2-y^2). \tag{27}$$

The peaks function in Equation (27) is often times used to measure the ability of methods to approximate functions [71], where the input is point pairs $(x, y) \in \mathbb{R}^2$, and the goal is to minimize the mean-squared-error between the predicted function value g and the actual function value x. Formally, we consider the following MLP:

$$\hat{g}(x,y) = (\sigma(\sigma([x,y]\,W_1)W_2)W_3), \tag{28}$$

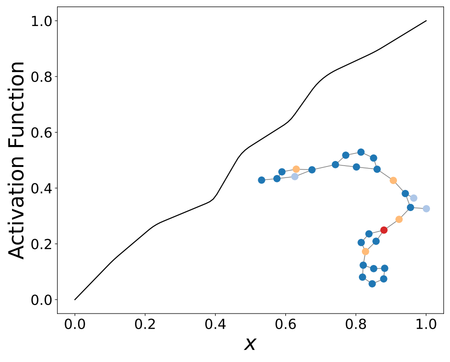
where σ is the activation of choice (ReLU, Tanh, or DIGRAF), and $W_1 \in \mathbb{R}^{2\times 64}$, $W_2 \in \mathbb{R}^{64\times 64}$, $W_3 \in \mathbb{R}^{64\times 1}$ are the trainable parameter matrices of the linear layers in the MLP. The goal, as discussed above, is to minimize the loss $\|\hat{g}(x,y) - g(x,y)\|_2$, for data triplets $(x_i, y_i, g(x_i, y_i))$ sampled from the peaks function. In our experiment, we sample 50,000 points, and report the obtained approximation error in terms of MSE in Figure 6. As can be seen, our DIGRAF, based on the CPAB framework, allows to obtain a significantly lower approximation error, up to 10 times lower (better) than ReLU, and 3 times better than Tanh. This example further motivates us to harness CPAB as the blueprint of DIGRAF.

G.2 Results on Node Classification

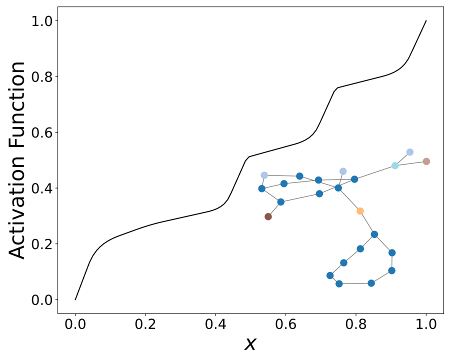
Our results are summarized in Table 2, where we consider the BLOGCATALOG [72], FLICKR [72], CITESEER [73], CORA [74], and PUBMED [75] datasets. As can be seen from the table, DIGRAF consistently outperforms all standard activation functions, as well as all the learnable activation functions. Additionally, DIGRAF outperforms other graph-adaptive activation functions. We attribute this positive performance gap to the ability of DIGRAF to learn complex non-linearities due to its diffeomorphism-based blueprint, compared to piecewise linear or pre-defined functions as in other methods. Finally, we compare the performance of DIGRAF and DIGRAF (W/O ADAP.). We remark that in this experiment, we are operating in a transductive setting, as the data consists of a single graph, implying that both DIGRAF and DIGRAF (W/O ADAP.) are adaptive in this case. Still, we see that DIGRAF slightly outperforms the DIGRAF (W/O ADAP.) and we attribute this performance gain to the GNN layers within DIGRAF that are (i) explicitly graph-aware, and (ii) can facilitate the learning of better diffeomorphism parameters $\boldsymbol{\theta}^{(l)}$ (Equation (9)) due to the added complexity.

Table 3: A comparison of DIGRAF to natural baselines, standard, and graph activation layers on OGB datasets, demonstrating the advantage of our approach. The top three methods are marked by **First**, **Second**, **Third**.

Method ↓ / Dataset →	MOLESOL RMSE ↓	MOLTOX21 ROC-AUC ↑	MOLBACE ROC-AUC ↑	MOLHIV ROC-AUC ↑
STANDARD ACTIVATIONS				
GIN + Identity	1.402±0.036	74.51±0.44	72.69±2.93	75.12±0.77
GIN + ReLU [27]	1.173±0.057	74.91±0.51	72.97±4.00	**75.58±1.40**
GIN + LeakyReLU [17]	1.219±0.055	74.60±1.10	73.40±3.19	74.75±1.20
GIN + Tanh [18]	1.190±0.044	74.93±0.61	74.92±2.47	**75.22**±2.03
GIN + GeLU [19]	1.147±0.050	74.29±0.59	75.59±3.32	74.15±0.79
GIN + ELU [20]	1.104±0.038	75.08±0.62	76.10±3.29	75.09±0.65
GIN + Sigmoid [21]	0.884±0.043	69.15±0.52	68.70±3.68	73.87±0.80
LEARNABLE ACTIVATIONS				
GIN + PReLU [22]	**1.098±0.062**	74.51±0.92	76.16±2.28	73.56±1.63
GIN + Maxout [23]	1.109±0.045	75.14±0.87	76.83±3.88	72.75±2.10
GIN + Swish [24]	1.113±0.066	73.31±1.01	**77.23±2.35**	72.95±0.64
GRAPH ACTIVATIONS				
GIN + Max [25]	1.199±0.070	**75.50±0.77**	77.04±2.81	73.44±2.08
GIN + Median [25]	**1.049±0.038**	74.39±0.90	**77.26±2.74**	72.80±2.21
GIN + GReLU [26]	1.108±0.066	**75.33±0.51**	75.17±2.60	73.45±1.62
GIN + DIGRAF (W/O ADAP.)	0.9011±0.047	76.37±0.49	78.90±1.41	79.19±1.36
GIN + DIGRAF	**0.8196±0.051**	**77.03±0.59**	**80.37±1.37**	**80.28±1.44**



(a) ZINC Test Graph 9



(b) ZINC Test Graph 141

Figure 7: Activation function learned by DIGRAF after the last GNN layer on two randomly selected graphs from ZINC. Different node colors indicate different node features. DIGRAF yields different activation for different graphs.

G.3 Results on OGB

We evaluate DIGRAF on 4 datasets from the OGB benchmark [76], namely, MOLESOL, MOLTOX21, MOLBACE, and MOLHIV. The results are reported in Table 3, where it is noted that DIGRAF achieves significant improvements compared to standard, learnable, and graph-adaptive activation functions. For instance, DIGRAF obtains a ROC-AUC score of 80.28% on MOLHIV, an absolute improvement of 4.7% over the best performing activation function (ReLU).

G.4 Visualization of DIGRAF

To illustrate the learned activation function DIGRAF after the last GNN layer on different graphs, we randomly selected two graphs from the ZINC dataset, as shown in Figure 7. The original graphs are presented in the lower right section, with each color representing a feature. Nodes with the same color share the same feature. The comparison of the figures clearly demonstrates that different graphs, with varying features and structures, learn distinct activation functions. This provides clear evidence that DIGRAF is adaptive to the input graph.

Table 4: Statistics of the node classification datasets [72–75].

Dataset	#nodes	#edges	#features	#classes
Planetoid				
Cora	2,708	10,556	1,433	7
CiteSeer	3,327	9,104	3,703	6
PubMed	19,717	88,648	500	3
Social Networks				
Flickr	7,575	479,476	12,047	9
BlogCatalog	5,196	343,486	8,189	6

Table 5: Hyperparameter configurations for the Planetoid datasets [73–75].

Hyperparameter	Search Range / Value
Learning rate for $\text{GNN}^{(l)}_{\text{LAYER}}$	$[10^{-5}, 10^{-4}, 10^{-3}, 5\times10^{-3}, 5\times10^{-2}]$
Learning rate for $\theta^{(l)}$ / GNN_{ACT}	$[10^{-6}, 5\times10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 5\times10^{-3}]$
Weight decay	$[10^{-5}, 10^{-4}, 5\times10^{-3}, 0.0]$
C	$[64, 128, 256]$
C_{ACT}	$[64, 128]$
L_{ACT}	$[2, 4]$
p	$[0.0, 0.5]$
$\mathcal{N}_{\mathcal{P}}$	$[2, 4, 8, 16]$
λ	$[0.0, 10^{-3}, 10^{-2}, 1.0]$

H Experimental Details

We implemented DiGRAF using PyTorch [77] (offered under BSD-3 Clause license) and the PyTorch Geometric library [65] (offered under MIT license). All experiments were conducted on NVIDIA RTX A5000, NVIDIA GeForce RTX 4090, NVIDIA GeForce RTX 4070 Ti Super, NVIDIA GeForce GTX 1080 Ti, NVIDIA TITAN RTX and NVIDIA TITAN V GPUs. For hyperparameter tuning and model selection, we utilized the Weights and Biases (wandb) library [78]. We used the `difw` package [14, 15, 33, 79] (offered under MIT license) for the diffeomorphic transformations based on the closed-form integration of CPA velocity functions. In the following subsections, we present the experimental procedure, dataset details, and hyperparameter configurations for each task.

Hyperparameters. The hyperparameters include the number of layers L and embedding dimension C of $\text{GNN}^{(l)}_{\text{LAYER}}$, learning rates and weight decay factors for both $\text{GNN}^{(l)}_{\text{LAYER}}$ and GNN_{ACT}, dropout rate p, tessellation size $\mathcal{N}_{\mathcal{P}}$, and regularization coefficient λ. We additionally include the number of layers L_{ACT} and embedding dimension C_{ACT} of GNN_{ACT}. We employed a combination of grid search and Bayesian optimization. All hyperparameters were chosen according to the best validation metric.

Node Classification. For each dataset, we train a 2-layer GCN [38] as the backbone architecture, and integrate each of the activation functions into this model. Following Zhang et al. [26], we randomly choose 20 nodes from each class for training and select 1000 nodes for testing. For each activation function, we run the experiment 10 times with random partitions. We report the mean and standard deviation of node classification accuracy on the test set. Table 4 summarizes the statistics of the node classification datasets used in our experiments. All models were trained for 1000 epochs with a fixed batch size of 32 using Adam optimizer. Tables 5 and 6 lists the hyperparameters and their search ranges or values.

Graph Classification. The statistics of various datasets can be found in Table 7. We consider the following setup:

- **ZINC-12k:** We consider the splits provided in Dwivedi et al. [80]. We use the mean absolute error (MAE) both as the loss and evaluation metric and report the mean and standard deviation over the test set calculated using five different seeds. We use the Adam optimizer and decay the learning rate by 0.5 every 300 epochs, with a maximum of 1000 epochs. In all our experiments, we adhere to the 500k parameter budget [80]. We use GINE [81] layers both for $\text{GNN}^{(l)}_{\text{LAYER}}$ and within GNN_{ACT}, and we fix $C_{\text{ACT}} = 64$ and $L_{\text{ACT}} = 2$. We report the hyperparameter search space for all the other hyperparameters in Table 8.

Table 6: Hyperparameter configurations for the social network datasets [72].

Hyperparameter	Search Range / Value
Learning rate for $\text{GNN}^{(l)}_{\text{LAYER}}$	$[10^{-5}, 10^{-4}, 5\times10^{-4}, 10^{-3}, 5\times10^{-2}, 10^{-2}]$
Learning rate for $\theta^{(l)}$ / GNN_{ACT}	$[10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 5\times10^{-3}, 10^{-2}, 5\times10^{-2}]$
Weight decay for $\text{GNN}_{\text{LAYER}}$	$[10^{-5}, 10^{-4}, 5\times10^{-3}, 0.0]$
Weight decay for $\theta^{(l)}$ / GNN_{ACT}	$[10^{-6}, 10^{-5}, 10^{-4}, 5\times10^{-3}, 0.0]$
C	$[64, 128, 256]$
C_{ACT}	$[16, 32, 64, 128]$
L	$[2, 4]$
L_{ACT}	$[2, 4]$
p	$[0.0, 0.4, 0.5, 0.6, 0.7]$
$\mathcal{N}_{\mathcal{P}}$	$[2, 4, 8, 16]$
λ	$[0.0, 10^{-3}, 10^{-2}, 1.0]$

Table 7: Statistics of the graph classification datasets[76, 80].

Dataset	#graphs	#nodes	#edges	#features	#classes
ZINC-12K	12,000	~23.2	~49.8	1	1
OGB					
MOLESOL	1,128	~13.3	~13.7	9	1
MOLTOX21	7,831	~18.6	~19.3	9	2
MOLBACE	1,513	~34.1	~36.9	9	2
MOLHIV	41,127	~25.5	~27.5	9	2

- **OGB:** We consider 4 datasets from the OGB repository, with one, namely MOLESOL, being a regression problem, while the others are classification tasks. We run each experiment using five different seeds and report the mean and standard deviation of RMSE/ROC-AUC. We use the Adam optimizer, decaying the learning rate by a factor of 0.5 every 100 epochs, and train for a maximum of 500 epochs. We use the GINE model with the encoders prescribed in Hu et al. [76] both for $\text{GNN}^{(l)}_{\text{LAYER}}$ and within GNN_{ACT}, and we set $C_{\text{ACT}} = 64$ and $L_{\text{ACT}} = 2$. We present the hyperparameter search space for all other parameters in Table 8

I Complexity and Runtimes

Time Complexity. We now provide an analysis of the time complexity of DIGRAF. Let us recall the following details: (i) As described in Equation (8), DIGRAF is applied element-wise in parallel for each dimension of the output of $\text{GNN}^{(l)}_{\text{LAYER}}$. (ii) As described in Equation (9), we employ an additional GNN denoted by GNN_{ACT} to compute $\boldsymbol{\theta}^{(l)}$. In all our experiments, both the backbone GNN and GNN_{ACT} are message-passing neural networks (MPNNs) [37]. (iii) As described in Theorem 2 of Freifeld et al. [15], for 1-dimensional domain, there exists a closed form for $T^{(l)}(\cdot;\boldsymbol{\theta}^{(l)})$, and the complexity for the CPAB computations are linear with respect to the tesselation size, which is a constant of up to 16 in our experiments. Therefore, using DIGRAF with any linear complexity (with respect to the number of nodes and edges) MPNN-based backbone maintains the linear complexity of the backbone MPNN. Put precisely, each MPNN layer has linear complexity in the number of nodes $|V|$ and $|E|$. We use L_{ACT} layers in GNN_{ACT}, the computational complexity of a DIGRAF layer is $\mathcal{O}(L_{\text{ACT}} \cdot (|V| + |E|))$. Since we have L layers in overall GNN, the computational complexity of an MPNN-based GNN coupled with DIGRAF is $\mathcal{O}(L \cdot L_{\text{ACT}} \cdot (|V| + |E|))$. In our experiments, we fix the hyperparameter $L_{\text{ACT}} = 2$, resulting in $\mathcal{O}(L \cdot (|V| + |E|))$ computational complexity in practice.

Runtimes. Despite having linear computational complexity in the size of the graph, DIGRAF performs additional computations to obtain $\boldsymbol{\theta}^{(l)}$ using GNN_{ACT}. To understand the impact of these computations, we measured the training and inference times of DIGRAF and present it in Table 9. Specifically, we report the average time per batch and standard deviation of the same measured on an NVIDIA A5000 GPU, using a batch size of 128. For a fair comparison, we use the same number of layers, batch size, and channels in all methods. Additionally, for our DIGRAF, we set the number of layers within GNN_{ACT} to $L_{\text{ACT}} = 2$, and the embedding dimension to $C_{\text{ACT}} = 64$. Our analysis indicates that while DIGRAF requires additional computational time, it yields significantly better performance. For example, compared to the best activation function on the dataset, namely Maxout,

Table 8: Hyperparameters and search ranges/values for OGB [76], and ZINC-12K [80] datasets.

Hyperparameter	OGB	ZINC
Learning rate for $\text{GNN}^{(l)}_{\text{LAYER}}$	$[10^{-5}, 10^{-4}, 10^{-3}, 5 \times 10^{-3}]$	
Learning rate for $\theta^{(l)}$/GNN_{ACT}	$[5 \times 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 5 \times 10^{-3}]$	
Weight decay for $\text{GNN}^{(l)}_{\text{LAYER}}$	$[10^{-5}, 10^{-4}, 5 \times 10^{-3}, 0.0]$	
Weight decay for $\theta^{(l)}$/GNN_{ACT}	$[10^{-5}, 10^{-4}, 5 \times 10^{-3}, 0.0]$	
C	$[64, 128]$	$[64, 128, 256]$
L	$[2, 4, 6]$	$[2, 4]$
p	$[0.0, 0.5]$	
$\mathcal{N}_{\mathcal{P}}$	$[2, 4, 8, 16]$	
λ	$[0.0, 10^{-3}, 10^{-2}, 1.0]$	
Graph pooling layer	[sum, mean]	
Batch size	$[64, 128]$	$[64, 128]$

Table 9: Batch runtimes on a NVIDIA RTX A5000 GPU of DIGRAF and other activation functions, with 4 GNN layers, batch size 128, 64 embedding dimension, and GNN_{ACT} with $L_{\text{ACT}} = 2$ layers and $C_{\text{ACT}} = 64$ embedding dimension, on ZINC-12K dataset.

Method	ZINC Training time (ms)	ZINC Inference time (ms)	ZINC (MAE ↓)
GIN + ReLU [27]	4.18±0.10	2.47±0.08	0.1630±0.0040
GIN + Maxout [23]	4.71±0.13	2.41±0.12	0.1587±0.0057
GIN + Swish [24]	4.55±0.12	2.30±0.24	0.1636±0.0039
GIN + Max [25]	9.19±0.25	4.50±0.93	0.1661±0.0035
GIN + Median [25]	14.54±1.35	10.13±1.20	0.1715±0.0050
GIN + GReLU [26]	20.63±0.99	11.69±2.79	0.3003±0.0086
GIN + DIGRAF (W/O ADAP.)	13.76±0.65	4.97±1.72	0.1382±0.0080
GIN + DIGRAF	19.37±1.28	8.62±0.18	0.1302±0.0090

DIGRAF requires an additional $\sim$ 6.21ms at inference, but results in a relative improvement in the performance of $\sim 17.95\%$.

J Ablation Studies of Parameter Count

J.1 Parameter Count Comparison

GNN_{ACT} is a core component of DIGRAF, which ensures graph-adaptivity by generating the parameters $\boldsymbol{\theta}^{(l)}$ of the activation function conditioned on the input graph. While the benefits of graph-adaptive activation functions are evident from our experiments in Section 4, as DIGRAF consistently outperforms DIGRAF (W/O ADAP.), the variant of our method that is not graph adaptive, it comes at the cost of additional parameters to learn GNN_{ACT} (Equation (9)). Specifically, because in all our experiments GNN_{ACT} is composed of 2 layers and a hidden dimension of 64, DIGRAF adds at most approximately 20K additional parameters. The number of added parameters in DIGRAF (W/O ADAP.) is significantly lower, counting at $\mathcal{N}_{\mathcal{P}} - 1$, where $\mathcal{N}_{\mathcal{P}}$ is the tessellation size. Note in our experiments, the tessellation size does not exceed 16. To further understand whether the improved performance of DIGRAF is due to the increased number of parameters, we conduct an additional experiment using the ReLU activation function where we increase the number of parameters of the model and compare the performances. In particular, we consider following settings: (1) The standard variant (GIN + ReLU), (2) The variant obtained by doubling the number of layers, and (3) The variant is obtained by doubling the number of hidden channels.

We present the results of the experiment described above on the ZINC-12K and MOLHIV datasets in Table 10. We observed that adding more parameters to the ReLU baseline does not produce significant performance improvements, even in cases where the baselines have ~4 times more parameters than DIGRAFand its baseline. On the contrary, with DIGRAF significantly improved performance is obtained compared to the baselines.

Table 10: Performance Comparison of DIGRAF with ReLU variants of increased parameter budget. The number of parameters is reported within the parenthesis adjacent to the metric. We use GINE [81] as a backbone. Increasing the parameter count with ReLU does not yield significant improvements, and DIGRAF outperforms all variants, even those with a higher number of parameters. Note that, DIGRAF (W/O ADAP.) has only $\mathcal{N}_{\mathcal{P}} - 1$ additional parameters, where $\mathcal{N}_{\mathcal{P}}$ is the tessellation size.

Method ↓ / Dataset →	ZINC (MAE ↓)	MOLHIV (ACC. % ↑)
GIN + ReLU (standard)	0.1630±0.0040 (~ 308K)	75.58±1.40 (~ 63K)
GIN + ReLU (double #channels)	0.1578±0.0014 (~ 1207K)	75.73±0.71 (~ 240K)
GIN + ReLU (double #layers)	0.1609±0.0033 (~ 580K)	75.78±0.43 (~ 116K)
DIGRAF (W/O ADAP.)	0.1382±0.0080 (~ 308K)	79.19±1.36 (~ 63K)
DIGRAF	**0.1302±0.0090** (~ 333K)	**80.28±1.44** (~ 83K)

Table 11: Results on ZINC and MOLHIV datasets along with number of parameters in paranthesis.

Method	ZINC (MAE) ↓	MOLHIV (ROC AUC) ↑
GIN + DIGRAF (W/O ADAP.) with larger $\text{GNN}_{\text{Layer}}$	0.1388 ± 0.0071 (337K)	79.22 ± 1.40 (85K)
GIN + DIGRAF (W/O ADAP.) (Original)	0.1382 ± 0.0086 (308K)	79.19 ± 1.36 (63K)
GIN + DIGRAF	0.1302 ± 0.0094 (333K)	80.28 ± 1.44 (83K)

J.2 Comparison of DIGRAF and DIGRAF (W/O ADAP.) with Equal Parameter Budget

To demonstrate the efficacy of graph adaptivity provided by GNN_{ACT}, we conduct an experiment where we increase the number of layers and channels of $\text{GNN}_{\text{LAYER}}$ in DIGRAF (W/O ADAP.) to match the total number of parameters in DIGRAF. As shown in Table 11, the increase in the number of parameters does not translate to better performance. Rather, the effective usage of the extra parameters as done by GNN_{ACT} is the reason behind the performance boost of DIGRAF.