GNNs Don't Need Backprop

Benoît Goupil[1], Fabian Jogl[2], Pascal Welke[3,2]

1- LIX (École Polytechnique, IP Paris, CNRS)
2- TU Wien 3- Lancaster University Leipzig

Abstract. We propose an alternative training method for graph neural networks (GNNs) that does not require gradient information. Instead, we sample randomly initialized models and select the one that maximizes an alignment score between its graph embedding space and the label space. Our method is easy to parallelize on CPU and GPU architectures and achieves competitive results with state-of-the-art stochastic gradient descent training on several graph classification benchmarks.

1 Introduction

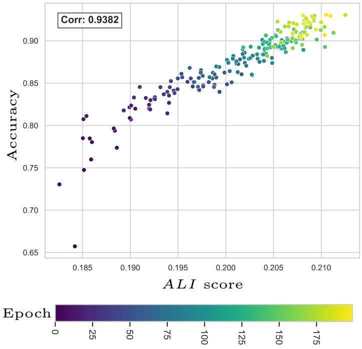

Fig. 1: A GNN trained with SGD on the Mutagenicity dataset. Over time, our alignment score ALI increases correlating with an increase in accuracy.

We propose a novel way to train graph neural networks (GNN) for classification tasks that is based on sampling a well-aligned randomly initialized encoder. A common way to describe what artificial neural networks do is that they learn "suitable" latent representations that capture the relevant aspects of input data with respect to a given learning task. In this context, learning usually refers to the minimization of some loss function via stochastic gradient descent (SGD) variants, which indirectly adjusts latent representations. In classification problems, well-trained models exhibit *neural collapse* [6, 10]: the reduction of intra-class variance of activations and the convergence to an equiangular simplex tight frame in final layers of the model. We propose an alignment measure of latent representations and target labels that is maximized by representations that exhibit neural collapse and that empirically correlates with training and test performance (see Figure 1) of graph neural networks (GNN) trained with SGD. Then we ask: Does directly optimizing this measure result in well-performing models?

We answer this question positively by using a simple sampling approach that selects the most aligned encoder from a user-defined number of randomly initialized GNN encoders. In a second step, we train a decoder based on the frozen encoder to obtain a model for the learning task. This yields competitive performance to full end-to-end training with SGD while avoiding the expense of backpropagating through the whole architecture. Since the alignment evaluation is completely independent for different encoders it can be parallelized at scale, opening the door to large-scale encoder search and offering a practical alternative to traditional optimization pipelines.

2 Related Work

To the best of our knowledge, there has been limited investigation into the performance of graph neural networks that are not trained with SGD. Pasa et al. [7] introduce a novel gated GNN architecture that can be trained without backpropagation. Their approach, however, does not apply to existing graph neural network architectures that use arbitrary element-wise nonlinearities on neurons. A few approaches use randomly initialized frozen weights for the GNN encoders as part of their training process: Huang et al. [3] explore the feasibility of identifying effective untrained GNNs without parameter tuning. However, their method still involves a training phase to optimally prune network connections. Böker et al. [1] investigate the ability of GNN encoders to capture and compare graph structures without training. They show that even with random initialization, GNNs can achieve competitive results in tasks like graph classification. However, the authors focus on specific architectures that are not commonly used in practical applications. Bui et al. [2] use random weights for the message-passing propagation. However, the model includes a node feature extractor in the first layer that needs to be trained with SGD. Furthermore, they use diagonal weight matrices during message passing, ensuring no interaction between different embedding dimensions. In contrast, our training procedure can be used for every GNN architecture, requires no changes to the model's architecture, and does not restrict the parameter matrices or used nonlinearities in any way.

3 A Model Target Alignment Measure

Let $\mathcal{G}$ be a set of pairwise non-isomorphic node- and edge-attributed graphs. Let $y : \mathcal{G} \to \mathbb{R}$ be a target label function. A *GNN encoder* is a function $\phi : \mathcal{G} \to \mathbb{R}^n$ that is implemented by some GNN architecture. In this work, we consider a fixed number of GIN [9] or GCN [4] layers followed by sum or mean pooling.

For a specific fixed GNN encoder ϕ and given supervised learning task $(\mathcal{D} \subseteq \mathcal{G}, y)$, we define an alignment score that measures how effectively the learned representations reflect the underlying label structure. This alignment score is based on two pseudometrics d_ϕ and d_y on the set of labeled graphs. A pseudometric d on $\mathcal{G}$ is a symmetric function $d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}_{\geq 0}$ that fulfills the triangle inequality and $d(G, G) = 0$ for all graphs in $\mathcal{G}$. The pseudometric d_ϕ is based on the embeddings produced by the GNN ϕ and is used to find similar graphs.

Definition 1. *Let $\phi : \mathcal{G} \to \mathbb{R}^n$ be a graph neural network and let $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be the Euclidean metric. Then we define the* GNN pseudometric *as*

$$d_\phi(G, H) = d\big(\phi(G), \phi(H)\big).$$

The pseudometric d_y is based on the label y and is used to determine if two similar graphs have the same label.

Definition 2. *Let $y : \mathcal{G} \to \mathbb{R}$ be a target label function. The* functional pseudometric *$d_y : \mathcal{G} \times \mathcal{G} \to [0, 1]$ is defined as*

$$d_y(G, H) := \begin{cases} 1 & \textit{if } y(G) \neq y(H) \\ 0 & \textit{if } y(G) = y(H). \end{cases}$$

It is easy to see that both d_ϕ and d_y are pseudometrics on any set of labeled graphs. Note that d_ϕ and d_y are indeed pseudometrics rather than true metrics: Message-passing neural networks like GIN or GCN are known to map certain non-isomorphic graphs to the same embedding (i.e., $\phi(G) = \phi(H)$) whenever G and H are indistinguishable by the Weisfeiler-Leman graph isomorphism test [9]. As a result, $d_\phi(G, H) = 0$ does not necessarily imply that G and H are isomorphic. Similarly $d_{\text{func}}(G, H) = 0$ for two non-isomorphic graphs G, H with identical labels $y(G) = y(H)$. We now propose an alignment score that quantifies the difference between the local label distribution around a graph in embedding space and the global label distribution beyond the immediate neighborhood.

Definition 3. *Let $\mathcal{D}$ be a finite labeled graph dataset, ϕ an GNN encoder, and k a positive integer. For each graph $G \in \mathcal{D}$, let $\mathcal{S}_\phi^k(G) \subseteq \mathcal{D} \setminus \{G\}$ be a set of k-nearest neighbors with respect to d_ϕ. We define the average functional distance between G and these k nearest neighbors*

$$A_k(G; \phi, \mathcal{D}) := \frac{1}{k} \sum_{H \in \mathcal{S}_\phi^k(G)} d_y(G, H)$$

and, analogously, the average functional distance between G and its non-neighbors

$$B_k(G; \phi, \mathcal{D}) := \frac{1}{|\mathcal{D}| - k - 1} \sum_{H \in \mathcal{D} \setminus \mathcal{S}_\phi^k(G)} d_y(G, H).$$

The alignment score *between d_ϕ and d_y is then given by*

$$ALI_k(\phi, \mathcal{D}) := \frac{1}{|\mathcal{D}|} \sum_{G \in \mathcal{D}} \Big[-A_k(G; \phi, \mathcal{D}) + B_k(G; \phi, \mathcal{D}) \Big].$$

Observe that $ALI_k(\phi, \mathcal{D})$ is positive if the neighborhoods of graph embeddings contain a larger fraction of graph embeddings of the same class than the non-neighborhoods. An alignment score close to zero indicates that the local and global label distributions are nearly identical. We use this alignment score as a supervised learning objective for graph neural network encoders and show that models ϕ with high $ALI_k(\phi, \mathcal{D})$ are good feature extractors for subsequent models.

Implementation. The computation of $ALI_k(\phi, \mathcal{D})$ can be vectorized for efficient parallelization on CPU and GPU hardware. Efficient kernels exist in PyTorch for the computation of Euclidean as well as Hamming distance matrices. A_k and B_k can be computed from these distance matrices by first computing suitable masks for the k nearest neighbors of each graph. We also experimented with Manhattan and cosine distances in the GNN pseudometric, and observed no notable differences in the results.

4 Sampling Algorithm

Our alignment score $ALI_k(\phi, \mathcal{D})$ allows us to quantify the performance of a GNN encoder ϕ without requiring an MLP decoder and without SGD. For this, we begin by fixing the architecture of the encoder: the type of message passing (GIN or GCN), the number of layers, and the layer dimension. We randomly initialize this GNN by sampling weight matrices for the linear layers and compute the alignment score of this randomly sampled GNN ϕ. We repeat the sampling and evaluation process a fixed number of times and select the encoder $\phi_{\max}$ with the highest alignment score. To perform downstream predictions, we train an MLP decoder ψ on the output of $\phi_{\max}$ with SGD to obtain a predictor $y \approx \psi \circ \phi_{\max}$. Note that when training the MLP decoder ψ, we freeze the weights of the GNN encoder $\phi_{\max}$, meaning that the GNN encoder is trained entirely without SGD.

Implementation. In standard GNN training with SGD, weight initialization is only used to stabilize and optimize training convergence during backpropagation. In our approach, the GNN encoder's behavior is entirely defined by its initialization. We draw GNN parameters i.i.d. from a normal distribution $\mathcal{N}(0, 1)$. We have explored different parameter initialization methods and have not found significant differences in their performance. The sampling of parameter values, the computation of embeddings of multiple GNN encoders, and the computation of alignment scores can be efficiently parallelized on modern hardware.

5 Experiments

To evaluate the performance of our proposed alignment-based selection of sampled GNNs, we conduct an extensive comparison to end-to-end SGD-trained GNNs. To ensure our method is effective across a wide range of model designs, we report results for 16 distinct architecture configurations on seven benchmark datasets [5]. We consider two message-passing layer types (GCN and GIN), varying network depths (2 or 3 layers), hidden widths (256 or 512 units), and two global pooling strategies (sum and mean).

We employ a 5-fold stratified cross-validation protocol (80%/10%/10% split). Each hyperparameter configuration is trained 3 times, with the optimal setting selected based on mean validation accuracy. We report the mean test accuracy and standard deviation of the selected models. To ensure a fair comparison, we allocated the same computational budget (wall-clock time) to both methods. As a baseline, we train full GNN models using SGD. Each model comprises the GNN encoder followed by a two-layer MLP classifier with a hidden dimension of 64. Weights are updated via backpropagation using the AdamW optimizer with cross-entropy loss. For each fold, we perform a grid search over learning rates $\{5\text{e-}4, 1\text{e-}3, 2\text{e-}3\}$, training for 300 epochs using a cosine annealing scheduler with a linear warmup for the first 10% of epochs. In our proposed approach, the GNN weights are randomly initialized and then frozen. To select the best encoder, we employ the following selection loop for each of the 5 cross-validation folds: We generate a pool of 2,000 independent GNN encoders and select the encoder with

Table 1: Average test accuracy (mean pooling) of models trained end-to-end with SGD, and with our method.

Layers	2				3			
Dims	256		512		256		512	
	SGD	Ours	SGD	Ours	SGD	Ours	SGD	Ours
Mutagenicity								
GIN	$81.3_{\pm 1.97}$	$81.2_{\pm 1.87}$	$81.6_{\pm 2.17}$	$81.2_{\pm 1.50}$	$81.1_{\pm 1.75}$	$81.0_{\pm 1.29}$	$81.8_{\pm 1.53}$	$81.7_{\pm 1.78}$
GCN	$79.7_{\pm 1.76}$	$79.5_{\pm 2.40}$	$80.4_{\pm 2.23}$	$80.2_{\pm 2.51}$	$80.0_{\pm 1.93}$	$79.9_{\pm 1.95}$	$80.5_{\pm 1.33}$	$80.3_{\pm 1.59}$
NCI1								
GIN	$78.6_{\pm 1.82}$	$77.6_{\pm 2.02}$	$78.4_{\pm 1.19}$	$77.6_{\pm 1.79}$	$78.4_{\pm 1.75}$	$77.6_{\pm 1.38}$	$78.5_{\pm 1.38}$	$78.2_{\pm 1.67}$
GCN	$76.2_{\pm 1.58}$	$75.9_{\pm 1.83}$	$76.5_{\pm 1.72}$	$76.0_{\pm 2.21}$	$77.9_{\pm 1.09}$	$76.7_{\pm 1.69}$	$77.2_{\pm 1.87}$	$77.1_{\pm 1.73}$
Enzymes								
GIN	$52.1_{\pm 6.95}$	$50.3_{\pm 6.99}$	$51.0_{\pm 6.01}$	$49.3_{\pm 6.20}$	$50.7_{\pm 6.31}$	$50.2_{\pm 5.70}$	$50.8_{\pm 8.04}$	$49.1_{\pm 7.37}$
GCN	$49.5_{\pm 7.10}$	$49.8_{\pm 6.35}$	$49.5_{\pm 2.73}$	$50.1_{\pm 6.02}$	$47.0_{\pm 2.77}$	$47.1_{\pm 4.56}$	$50.1_{\pm 4.52}$	$49.6_{\pm 5.24}$
Proteins								
GIN	$70.5_{\pm 4.06}$	$72.1_{\pm 4.74}$	$71.4_{\pm 4.11}$	$72.7_{\pm 5.29}$	$71.2_{\pm 5.03}$	$70.6_{\pm 4.79}$	$70.0_{\pm 5.44}$	$71.7_{\pm 4.24}$
GCN	$70.9_{\pm 3.25}$	$71.5_{\pm 3.37}$	$71.8_{\pm 2.85}$	$71.4_{\pm 2.90}$	$70.6_{\pm 2.59}$	$70.6_{\pm 2.83}$	$71.1_{\pm 4.25}$	$71.1_{\pm 2.78}$
DD								
GIN	$75.2_{\pm 3.04}$	$76.8_{\pm 4.23}$	$74.1_{\pm 3.50}$	$76.6_{\pm 3.44}$	$73.6_{\pm 2.85}$	$73.5_{\pm 4.35}$	$74.9_{\pm 2.36}$	$75.1_{\pm 3.32}$
GCN	$73.6_{\pm 0.82}$	$76.2_{\pm 4.88}$	$74.7_{\pm 0.96}$	$75.3_{\pm 3.28}$	$73.6_{\pm 2.18}$	$74.4_{\pm 3.08}$	$74.7_{\pm 2.94}$	$74.2_{\pm 2.37}$
IMDB-BINARY								
GIN	$72.4_{\pm 2.33}$	$72.2_{\pm 3.91}$	$71.6_{\pm 2.37}$	$70.8_{\pm 3.14}$	$72.6_{\pm 2.95}$	$72.3_{\pm 2.41}$	$71.6_{\pm 2.62}$	$71.8_{\pm 3.09}$
GCN	$71.4_{\pm 3.94}$	$71.7_{\pm 3.41}$	$71.8_{\pm 2.96}$	$71.4_{\pm 3.56}$	$72.8_{\pm 3.78}$	$72.3_{\pm 4.01}$	$70.8_{\pm 3.06}$	$71.6_{\pm 4.05}$
IMDB-MULTI								
GIN	$50.7_{\pm 3.45}$	$50.0_{\pm 3.17}$	$50.7_{\pm 2.97}$	$50.7_{\pm 2.87}$	$50.0_{\pm 3.07}$	$51.3_{\pm 2.68}$	$50.6_{\pm 3.36}$	$50.8_{\pm 2.72}$
GCN	$50.6_{\pm 2.62}$	$50.8_{\pm 3.26}$	$50.6_{\pm 1.85}$	$51.4_{\pm 2.70}$	$51.2_{\pm 2.98}$	$50.4_{\pm 2.64}$	$51.5_{\pm 2.29}$	$51.0_{\pm 3.02}$

the highest ALI score on the training split. Then, we train a decoder (identical to the baseline's two-layer MLP) using the same optimization protocol (AdamW, 300 epochs, cosine annealing) as the baseline.

The results for the mean pooling are shown in Table 1, and the results for sum pooling follow a similar trend. To globally compare methods, we report a macro-averaged Vargha–Delaney A_{12} measure [8]: for each dataset and each architecture, we compute A_{12} for that configuration by aggregating over the folds, and then average these scores across all configurations, giving equal weight to every dataset–architecture combination. Pooling variants (mean and sum) are both included in this macro-average, each treated as a separate configuration. In our comparison, this macro-averaged A_{12} is 0.48. Since 0.5 corresponds to no difference and values in the range 0.44–0.56 are conventionally regarded as negligible [8], we interpret this as no statistically significant difference between the two methods. That is, our backpropagation-free training works as well as SGD.

6 Conclusion

We developed a novel training method based on randomly sampling the weights of graph neural networks and demonstrated that it achieves results that are competitive with stochastic gradient descent-based training. To this end, we introduced an alignment score on latent graph representations as a means of assessing the quality of graph embeddings for classification tasks. This score proved to be highly correlated with the model's final performance.

In future work, we would like to improve the computational efficiency of our alignment score which currently scales quadratically with the size of the training set. Furthermore, extensions of the score to highly imbalanced datasets or to regression tasks would allow to tackle a wider range of practical problems.

References

[1] Böker, J., Levie, R., Huang, N., Villar, S., Morris, C.: Fine-grained expressivity of graph neural networks. In: NeurIPS (2023)

[2] Bui, T., Naman, A., Schönlieb, C.B., Ribeiro, B., Bevilacqua, B., Eliasof, M.: Random propagations in GNNs. In: Workshop on Unifying Representations in Neural Models (UniReps) at NeurIPS (2024)

[3] Huang, T., Chen, T., Fang, M., Menkovski, V., Zhao, J., Yin, L., Pei, Y., Mocanu, D.C., Wang, Z., Pechenizkiy, M., Liu, S.: You can have better graph neural networks by not training weights at all: Finding untrained gnns tickets. In: Learning on Graphs Conference (2022)

[4] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)

[5] Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: Tudataset: A collection of benchmark datasets for learning with graphs. In: Graph Representation Learning and Beyond Workshop at ICML (2020)

[6] Papyan, V., Han, X.Y., Donoho, D.L.: Prevalence of neural collapse during the terminal phase of deep learning training. Proceedings of the National Academy of Sciences (2020)

[7] Pasa, L., Navarin, N., Erb, W., Sperduti, A.: Backpropagation-free graph neural networks. In: ICDM (2022)

[8] Vargha, A., Delaney, H.D.: A critique and improvement of the "cl" common language effect size statistics of mcgraw and wong. Journal of Educational and Behavioral Statistics **25**(2), 101–132 (2000)

[9] Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2019)

[10] Yaras, C., Wang, P., Zhu, Z., Balzano, L., Qu, Q.: Neural collapse with normalized features: A geometric analysis over the riemannian manifold. In: NeurIPS (2022)