Neural Networks Are Graphs! Graph Neural Networks for Equivariant Processing of Neural Networks

David W. Zhang [1] **Miltiadis Kofinas** [1] **Yan Zhang** [2] **Yunlu Chen** [1] **Gertjan J. Burghouts** [3] **Cees G. M. Snoek** [1]

Abstract

Neural networks that can process the parameters of other neural networks find applications in diverse domains, including processing implicit neural representations, domain adaptation of pretrained networks, generating neural network weights, and predicting generalization errors. However, existing approaches either overlook the inherent permutation symmetry in the weight space or rely on intricate weight-sharing patterns to achieve equivariance. In this work, we propose representing neural networks as computation graphs, enabling the use of standard graph neural networks to preserve permutation symmetry. We also introduce probe features computed from the forward pass of the input neural network. Our proposed solution improves over prior methods from 86% to 97% accuracy on the challenging MNIST INR classification benchmark, showcasing the effectiveness of our approach.

1. Introduction

How can we design models that themselves take *neural network parameters* as input? This would allow us to make inferences *about* neural networks, such as predicting their generalization error (Unterthiner et al., 2020), generating neural network weights (Schürholt et al., 2022), and classifying or generating implicit neural representations (Dupont et al., 2022) without having to evaluate them many times.

For simplicity, let us consider a multilayer perceptron with multiple hidden layers. As a naïve approach, we can simply concatenate all flattened weights and biases into one large feature vector, from which we can then make predictions as usual. However, this overlooks an important structure in the parameters: neurons in a layer can be *reordered* while maintaining exactly the same function . Reordering neurons here means changing the preceding and following weights attached to the neuron accordingly. Let us make this more concrete. Suppose we have a two-layer MLP $f(\boldsymbol{x})=\boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x})$. Then, applying the permutation matrix $\boldsymbol{P}$ to the first weight matrix $\widetilde{\boldsymbol{W}}_1=\boldsymbol{P}\boldsymbol{W}_1$ and similarly to the second $\widetilde{\boldsymbol{W}}_2=\boldsymbol{W}_2\boldsymbol{P}^\top$ results in the exact same function $\widetilde{\boldsymbol{W}}_2\sigma(\widetilde{\boldsymbol{W}}_1\boldsymbol{x})=\boldsymbol{W}_2\boldsymbol{P}^\top\boldsymbol{P}\sigma(\boldsymbol{W}_1\boldsymbol{x})=\boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x})$.

The problem with ignoring this permutation symmetry is that our model will likely make different predictions for different orderings of the neurons in the input neural network, even though they all represent exactly the same function. In general, accounting for the symmetry in the input data improves the learning efficiency and underpins the field of geometric deep learning. If we could structure our representation and model in such a way that these permutation symmetries are respected, then we should be able to improve the performance of our model.

Previous approaches. Two recent studies (Navon et al., 2023; Zhou et al., 2023) make use of the permutation structure in the weights and biases (more details in Appendix A). In particular, notice that in our previous construction of $\widetilde{\boldsymbol{W}}_1$ and $\widetilde{\boldsymbol{W}}_2$ *the same permutation* is applied on the rows of $\boldsymbol{W}_1$ and the columns of $\boldsymbol{W}_2$. This pattern persists in deeper neural networks; pairs of successive weight matrices can freely permute their rows and columns in the same manner. These studies exploit this structure to design equivariant and invariant models: models whose predictions change reliably, or not at all when the permutations of neurons are changed. However, so far the performance of these models on classifying implicit neural representations (Navon et al., 2023) has lagged far behind simply classifying the data normally, which shows that there is still much to be understood about these representations.

Our approach. We take an alternative approach to this problem: we present a *computation-graph-based* representation for taking neural network parameters as input (see Figure 1). Instead of viewing the problem as permuting rows and columns of weights simultaneously, it allows for the much simpler view of permuting nodes directly. We

[1]University of Amsterdam [2]Samsung - SAIT AI Lab, Montreal [3]TNO. Correspondence to: David W. Zhang <w.d.zhang@uva.nl>.

Presented at the 2nd Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML) at the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. 2023. Copyright 2023 by the author(s).

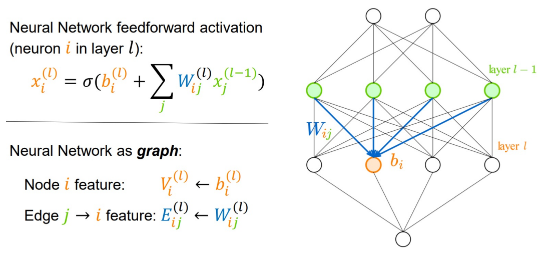
Neural Network feedforward activation
(neuron i in layer l):
$x_i^{(l)} = \sigma(b_i^{(l)} + \sum_j W_{ij}^{(l)} x_j^{(l-1)})$
Neural Network as ***graph***:
Node i feature: $V_i^{(l)} \leftarrow b_i^{(l)}$
Edge $j \rightarrow i$ feature: $E_{ij}^{(l)} \leftarrow W_{ij}^{(l)}$
layer $l-1$
layer l
W_{ij}
b_i

Figure 1. A neural network as a graph. We assign neural network parameters to graph features by treating biases $\boldsymbol{b}_i$ as corresponding node features $\boldsymbol{V}_i$, and weights $\boldsymbol{W}_{ij}$ as edge features $\boldsymbol{E}_{ij}$ connecting the nodes in adjacent layers.

make the following contributions: In Section 2, we propose a simple and efficient representation of neural networks as graphs that can be incorporated as inputs to various graph neural networks (GNNs). The perspective of permuting neurons rather than weights makes our model conceptually much simpler than prior work. We also introduce the concept of "probe features" that capture the functional aspect of the input graph. These features are computed through the forward pass of the corresponding neural network for the input graph, and they are concatenated as features to each node in the input graph. In Section 3, we propose extensions to GNNs and transformers that make them suitable for our setting. In Section 4, we empirically validate our proposed method on implicit neural representation datasets, where we outperform the previous equivariant approaches by a large margin.

2. Neural networks as computation graphs

The conventional representation of neural networks as graphs dates back to the early days of their development. This is exactly the perspective we will take in this paper. In this view, individual neurons are represented by nodes, connections between neurons are represented by edges, the input is represented by the nodes in the first layer, and the output is represented by the nodes in the last layer. The edge between a pair of nodes carries the scalar weight between the two neurons. This graph representation has a clear link to how the forward pass of the neural network is computed: the nodes in the input layer take the values of the input, and this information is then propagated through the edges (and corresponding edge weights) to the output nodes.

Importantly, the symmetries of graphs correspond exactly to the permutation symmetries of neural network computation: permuting the nodes in a graph corresponds to permuting the adjacency matrix encoding the incoming and outgoing edges of the affected nodes. This is exactly like in a neural network, where permuting neurons in a layer corresponds to permuting the incoming and outgoing weight matrices accordingly. This graph representation of neural networks has two major benefits:

1. It is a conceptually simple encoding of the permutation symmetries in neural networks, all without the need to rely on the heavier machinery of designing equivariant networks on the weight space (see Appendix A). This encoding generalizes well to different architectures with more complex computation graphs, such as ones including residual connections and concatenations.

2. There is a rich literature on how to design models for graphs that we can draw from. In this paper, we make use of powerful graph neural networks and transformers to process them.

While this graph representation lends itself well to the application of graph neural networks, one concern may dampen its appeal at first thought: the values of the nodes are unspecified. We address this in our construction of the graph. Determining what to use as node features is a design decision and does not necessarily have to correspond to the same values that neurons take during the forward or backward pass of the neural network. This flexibility allows for exploring various representations and leveraging the unique characteristics of the graph structure for better performance.

2.1. MLP as graph

Here we detail the steps to construct a graph $\boldsymbol{G}{=}(\boldsymbol{V}, \boldsymbol{E})$ with node features $\boldsymbol{V} \in \mathbb{R}^{n \times d_{\boldsymbol{V}}}$ and edge features $\boldsymbol{E} \in \mathbb{R}^{n \times n \times d_{\boldsymbol{E}}}$, with n denoting the number of nodes in the graph. An MLP with L fully connected layers has the weight matrices $[\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L)}]$ and biases $[\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(L)}]$, where $\boldsymbol{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and $\boldsymbol{b}^{(l)} \in \mathbb{R}^{d_l}$. The total number of nodes is then $n{=}\sum_{l=0}^{L} d_l$, where d_0 is the dimension of the input. The first channel of the edge feature tensor $\boldsymbol{E}_{:,:,1}$ is then a sparse block matrix containing the weight matrices as the blocks. We make the choice of encoding the biases in the nodes: the first channel of the node features $\boldsymbol{V}_{:,1}$ contains the biases of all the nodes concatenated together. An example is shown in Figure 2. Depending on the task at hand, we have the flexibility to incorporate additional edge and node features. Next, we explore some examples of this.

So far, we have only encoded the forward pass of the input neural network computation, which results in a directed acyclic graph. To be able to propagate information from later layers to earlier layers (direction of the backpropagation), we optionally include extra edge features containing the transposed edge features $\boldsymbol{E}^{\top}_{:,:,1}$. Similarly we can also include undirected edges $\boldsymbol{E}_{:,:,1} + \boldsymbol{E}^{\top}_{:,:,1}$ as extra features.

2.2. Probe features

Humans tend to interpret complicated functions by probing the function with a few input samples and inspecting the

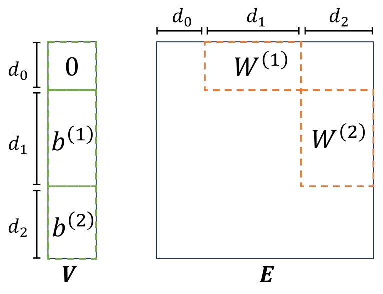

Figure 2. Node (left) and edge (right) features filled with the biases and weights of the MLP. The MLP has one input layer of dimensionality d_0, a hidden layer of dimensionality d_1, and an output layer of dimensionality d_2.

resulting output. We provide the graph neural network with similar functionality by adding additional features to every node. We learn a set of sample input values that we pass through the input neural function and retain the values for all the intermediate neurons and output neurons. For example, we can learn a set of input values $\{\boldsymbol{x}_i\}_{i=0..k}$ for which the neural function then computes k activations for every neuron. These are then included as additional features to the node features $\boldsymbol{V}$ presented in Section 2.1. Notably, these additional features are invariant to *all* augmentations on the input neural network's parameters that maintain the exact same function.

2.3. Representing more complex architectures

So far, we have only described how to encode basic MLPs as graphs. This setting is what we use in our experiments. One of the primary benefits of the graph representation is that it becomes straightforward to represent different network architectures that can all be processed by a shared graph neural network. Notably, we do not require any changes to accommodate varying number of layers or number of neurons per layer. We now address how to generalize the graph representation to alternative network architectures.

Going beyond a simple MLP-like network structure, we can easily incorporate branching architectures by adding extra edges that match the computation graph. For example, for residual connections we would add edges to the edge features between the relevant neurons with weights of 1.

In general, the semantics of multiple edges going into one node for the forward pass is a weighted sum (weighted by the edge weights) followed by an activation function. When we want to represent other operations, we need to change this representation. Different types of connections, activation functions, and normalizations can be encoded into the graph through extra channels (for example, by a one-hot encoding of the type of operation) in the edge and node features.

3. Neural architectures for graphs

In this section, we present two neural network architectures that are equivariant with respect to the order of the nodes. The first one is a graph neural network (GNN) (Corso et al., 2020) that is restricted to local updates, and the second one is a transformer (Diao & Loynd, 2023) with global attention.

Position embeddings. Before processing the input graph, we add learned position embeddings to every node. In order to preserve the permutation symmetry, the nodes that correspond to the same intermediate layer also share the same position embedding. These position embeddings help identify which layer in the input neural network each node corresponds to. While this information can already be inferred from the adjacency matrix, it could require multiple local message passing steps to do so.

GNN. Graph neural networks (GNN) in the form of message-passing neural networks (MPNN) are by design equivariant with respect to permutations in the order of the nodes. The standard MPNN framework only updates the node features in each layer but does not update the edge features. We apply a simple extension:

$$\boldsymbol{E}_{ij}^{(t+1)} = \phi_e\left(\left[\boldsymbol{V}_i^{(t)}, \boldsymbol{E}_{ij}^{(t)}, \boldsymbol{V}_j^{(t)}\right]\right), \tag{1}$$

that updates the edge features after each message passing step t with a small MLP.

We algorithmically align the message passing step with the forward pass of a neural network by adding multiplicative interactions between the node and edge features. In particular, we apply FiLM to the message passing step (Perez et al., 2018; Brockschmidt, 2020):

$$\begin{aligned} m_{ij} = \phi_{\texttt{scale}}\left(\boldsymbol{E}_{ij}^{(t)}\right) \odot \phi_m\left(\left[\boldsymbol{V}_i^{(t)}, \boldsymbol{V}_j^{(t)}\right]\right) \\ + \phi_{\texttt{shift}}\left(\boldsymbol{E}_{ij}^{(t)}\right). \end{aligned} \tag{2}$$

Note that this differs from the FiLM-GNN (Brockschmidt, 2020) in that we compute the scaling factors based on the edge features and not based on the adjacent node's features.

Transformer. The transformer encoder can be seen as a graph neural network that operates on the fully connected graph. We use the transformer variant with relational attention (Diao & Loynd, 2023) that adds edge features to the self-attention computation. Similar to the GNN we also augment the transformer with modulation to enable multiplicative interactions between the node and edge features. In particular, we change the update to the value matrix in the self-attention module:

$$\boldsymbol{v}_{ij} = \boldsymbol{E}_{ij}\boldsymbol{W}_{\texttt{scale}}^{\text{value}} \odot \boldsymbol{V}_j\boldsymbol{W}_n^{\text{value}} + \boldsymbol{E}_{ij}\boldsymbol{W}_{\texttt{shift}}^{\text{value}}. \tag{3}$$

Table 1. Classification of MNIST INRs. All graph-based models outperform the baselines.

Model	# probe features	Accuracy in %
MLP (Navon et al., 2023)	—	$17.6_{\pm 0.0}$
Set NN (Navon et al., 2023)	—	$23.7_{\pm 0.1}$
DWSNet (Navon et al., 2023)	—	$85.7_{\pm 0.6}$
GNN (Ours)	0	$91.4_{\pm 0.6}$
GNN (Ours)	4	$91.8_{\pm 0.5}$
GNN (Ours)	16	$92.8_{\pm 0.3}$
GNN (Ours)	64	$94.7_{\pm 0.3}$
Relational transformer (Ours)	0	$92.4_{\pm 0.3}$
Relational transformer (Ours)	4	$93.3_{\pm 0.2}$
Relational transformer (Ours)	16	$94.9_{\pm 0.3}$
Relational transformer (Ours)	64	$\mathbf{97.3}_{\pm 0.2}$

Table 2. Dilating MNIST INRs. Mean-squared error (MSE) computed between the reconstructed image and dilated ground-truth image. Lower is better.

Model	# probe features	MSE in 10^{-2}
DWSNet (Navon et al., 2023)	—	$2.58_{\pm 0.00}$
NFN (Zhou et al., 2023)	—	$2.55_{\pm 0.00}$
GNN (Ours)	0	$2.38_{\pm 0.02}$
GNN (Ours)	4	$2.26_{\pm 0.01}$
GNN (Ours)	16	$2.17_{\pm 0.01}$
GNN (Ours)	64	$2.06_{\pm 0.01}$
Relational transformer (Ours)	0	$1.96_{\pm 0.00}$
Relational transformer (Ours)	4	$1.88_{\pm 0.02}$
Relational transformer (Ours)	16	$1.82_{\pm 0.02}$
Relational transformer (Ours)	64	$\mathbf{1.75}_{\pm 0.01}$

4. Experiments

We evaluate the efficacy of our method on two distinct tasks: one on a global scale and another requiring individual outputs for each parameter. In both cases, our dataset consists of implicit neural representations (INRs) that parameterize image signals with continuous neural field functions. Our primary comparison is with two recent permutation-equivariant neural networks in the weight space.

Classifying MNIST INRs. The dataset consists of one INR per image from MNIST that are separately optimized to reconstruct their corresponding image. We use the same dataset as Navon et al. (2023). In Table 1, we observe that our approach outperforms the equivariant baseline by up to $+11.6\%$. Interestingly the baseline can perform equally well in terms of training loss, but our graph-based approach exhibits better generalization performance.

Dilating MNIST INRs. Using the same INR dataset from our previous experiment, we assess the model's ability to predict weight updates to the INRs that aim to enlarge the represented digit (through dilation). We follow the same training objective as Zhou et al. (2023). In Table 2, we observe improvements over both. Furthermore, the probe features are effective even in a setting where we require an output per parameter as opposed to a global prediction.

Position embeddings. We ablate the significance of position embedding in the context of MNIST INR classification. Without position embedding the GNN achieves an accuracy of $83.9_{\pm 0.3}$, and the Transformer $77.9_{\pm 0.7}$. This is a decrease of 7.5 and 14.5 points respectively, which highlights the importance of position embeddings.

5. Conclusion and future work

We have presented an effective method for processing neural networks with neural networks by representing the input neural network as graphs. Exploring the application of this approach to additional tasks presents an exciting avenue for future work.

References

Brockschmidt, M. Gnn-film: Graph neural networks with feature-wise linear modulation. In *International Conference on Machine Learning (ICML)*, 2020.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Diao, C. and Loynd, R. Relational attention: Generalizing transformers for graph-structured tasks. In *International Conference on Learning Representations (ICLR)*, 2023.

Dupont, E., Kim, H., Eslami, S., Rezende, D., and Rosenbaum, D. From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning (ICML)*, 2022.

Navon, A., Shamsian, A., Achituve, I., Fetaya, E., Chechik, G., and Maron, H. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning (ICML)*, 2023.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Schürholt, K., Knyazev, B., Giró-i Nieto, X., and Borth, D. Hyper-representations as generative models: Sampling unseen neural network weights. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Unterthiner, T., Keysers, D., Gelly, S., Bousquet, O., and Tolstikhin, I. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

Zhou, A., Yang, K., Burns, K., Jiang, Y., Sokota, S., Kolter, J. Z., and Finn, C. Permutation equivariant neural functionals. *arXiv preprint arXiv:2302.14040*, 2023.