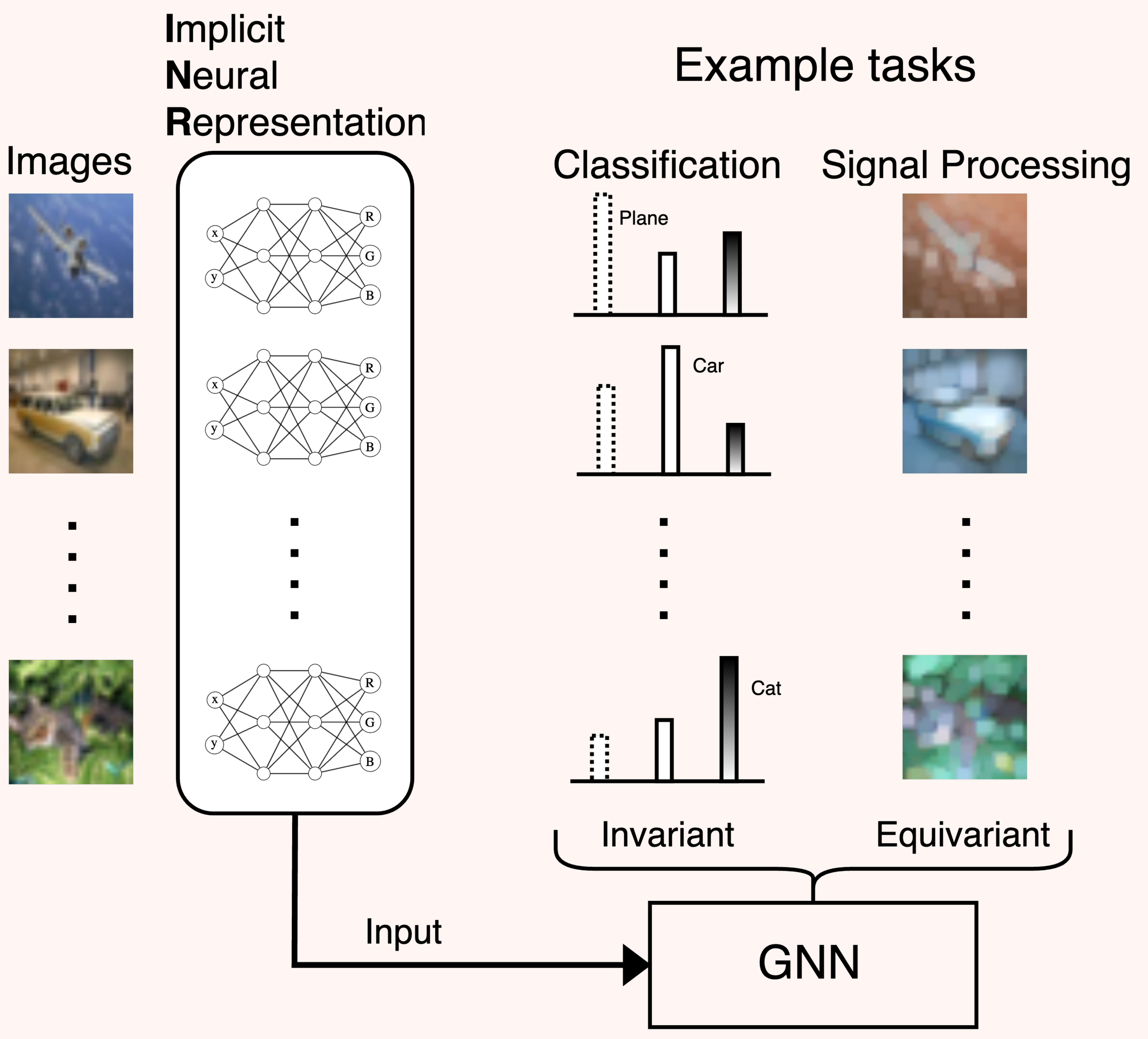
Neural Networks Are Graphs! Graph Neural Networks for Equivariant Processing of Neural Networks

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

[1]University of Amsterdam [2]Samsung - SAIT AI Lab [3]TNO

Overview

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 pre-trained 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.



Keywords processing neural networks, neural functionals, deep weight space, graph neural networks, equivariance

Introduction – Symmetries

Neurons in an MLP can be *reordered* while maintaining exactly the same function [3]. Reordering neurons here means changing the preceding and following weights attached to the neuron accordingly.

Consider a two-layer MLP $f(\boldsymbol{x}) = \boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x})$. Then, permuting the rows of the first weight matrix and the columns of the second matrix, *i.e.* 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:

$$\widetilde{\boldsymbol{W}}_2\sigma(\widetilde{\boldsymbol{W}}_1\boldsymbol{x}) = \boldsymbol{W}_2\boldsymbol{P}^\top\sigma(\boldsymbol{P}\boldsymbol{W}_1\boldsymbol{x}) = \boldsymbol{W}_2\boldsymbol{P}^\top\boldsymbol{P}\sigma(\boldsymbol{W}_1\boldsymbol{x}) = \boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x}).$$

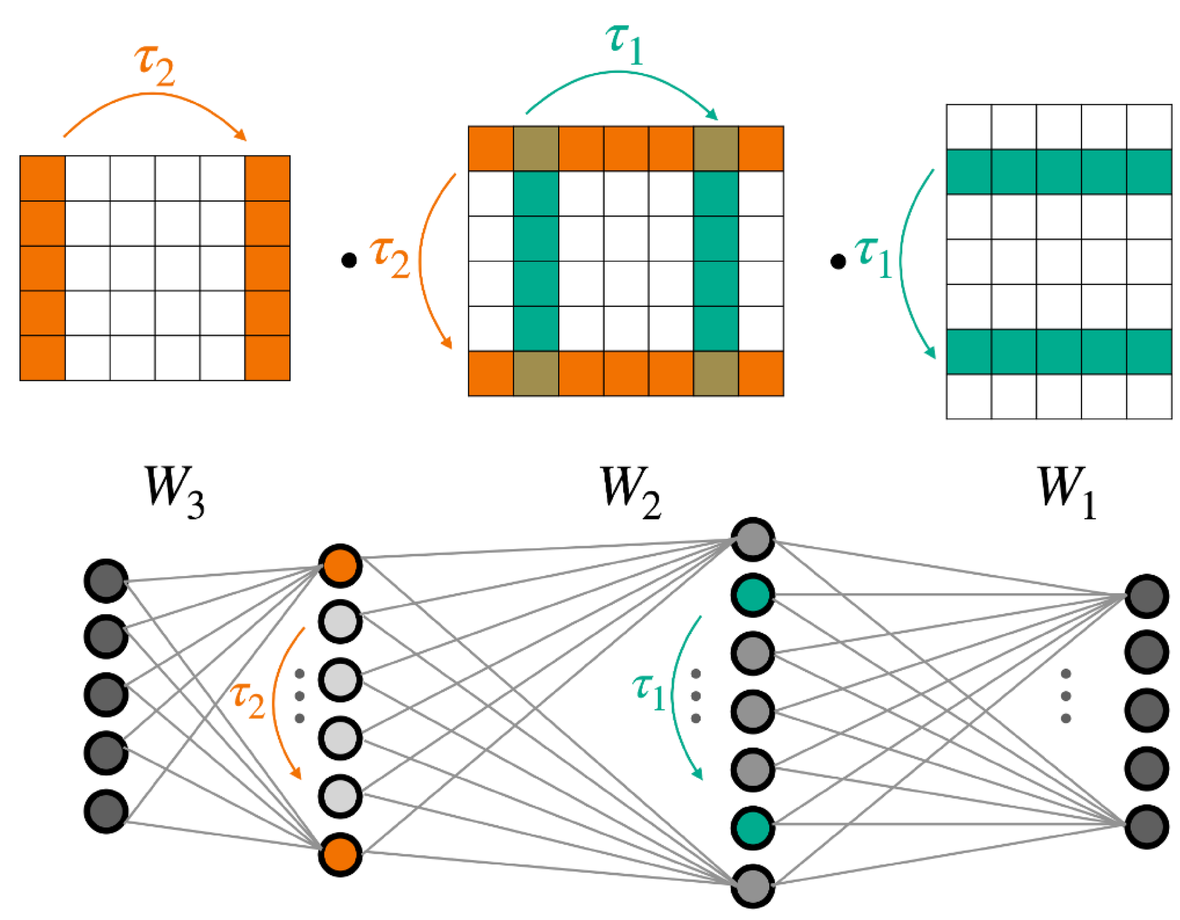


Figure 1. Weight symmetries in a 3-layer MLP. Figure credit: [4].

Neural networks as graphs

Neural Network feedforward activation (neuron i in layer l):

$$x_i^{(l)} = \sigma\Big(b_i^{(l)} + \sum_j W_{ij}^{(l)} x_j^{(l-1)}\Big)$$

Neural Network as ***graph***:

Node i feature: $V_i^{(l)} \leftarrow b_i^{(l)}$

Edge $j \to i$ feature: $E_{ij}^{(l)} \leftarrow W_{ij}^{(l)}$

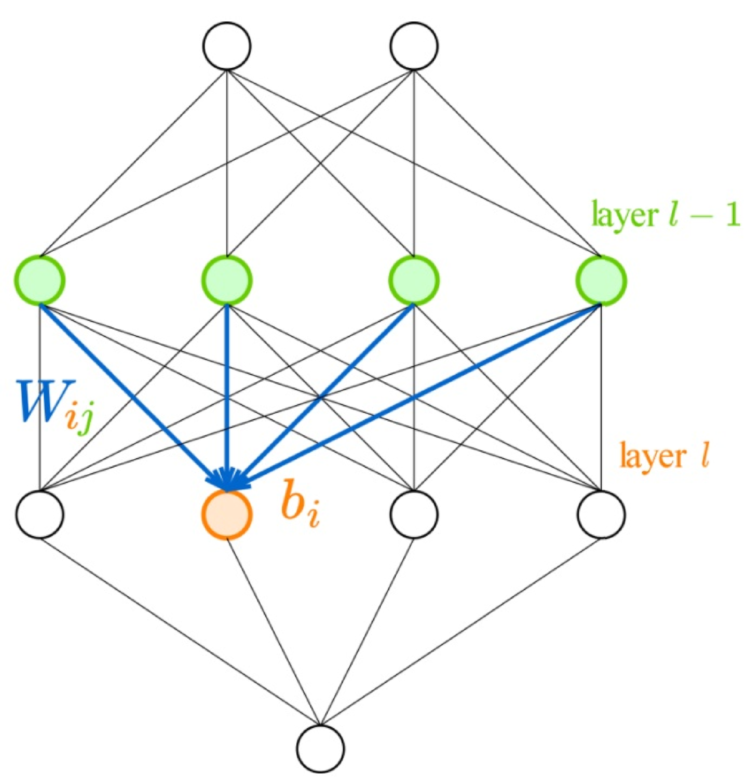


Figure 2. A neural network as a graph. We assign neural network parameters to graph features by treating biases $\boldsymbol{b}_i$ as corresponding node features $\boldsymbol{V}_i$, and weights $\boldsymbol{W}_{ij}$ as edge features $\boldsymbol{E}_{ij}$ connecting the nodes in adjacent layers.

Method – Graph networks for neural networks

Before we apply the graph networks, we add position embeddings to the input graph. To retain the permutation symmetry nodes in the same hidden layer share the same position embedding.

We extend PNA [1] with an MLP that updates the edge features given the incident nodes' features and the previous layer's edge features.

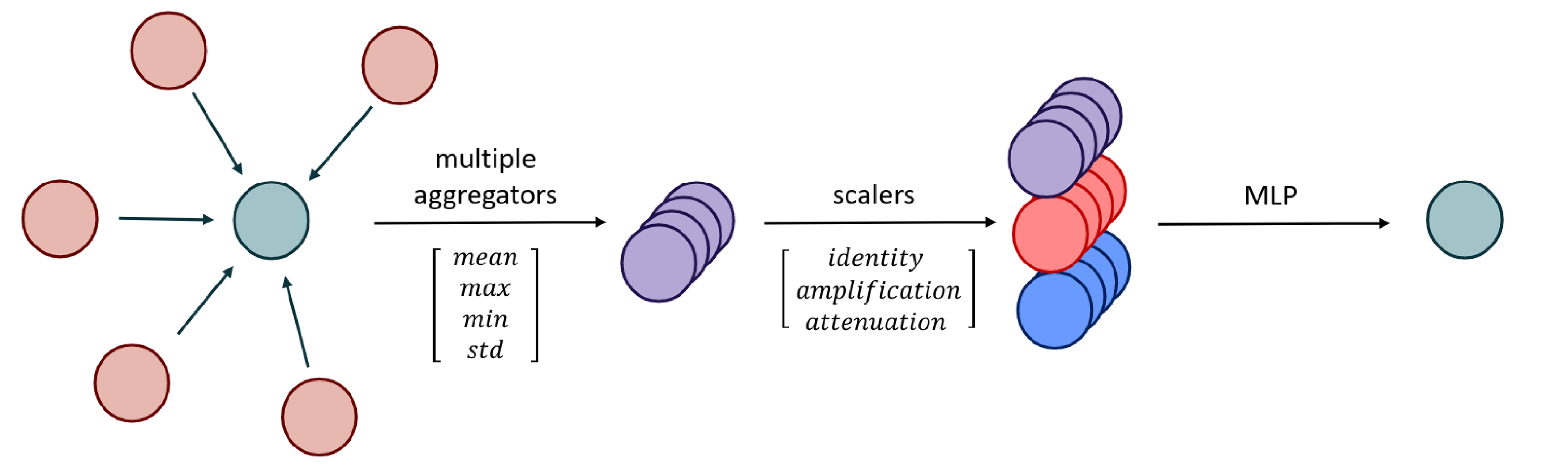


Figure 3. Diagram for PNA. Figure credit: [1].

We extend Relational Transformer [2] with multiplicative interactions between node and edge features to algorithmically align it with the forward-pass of a neural network.

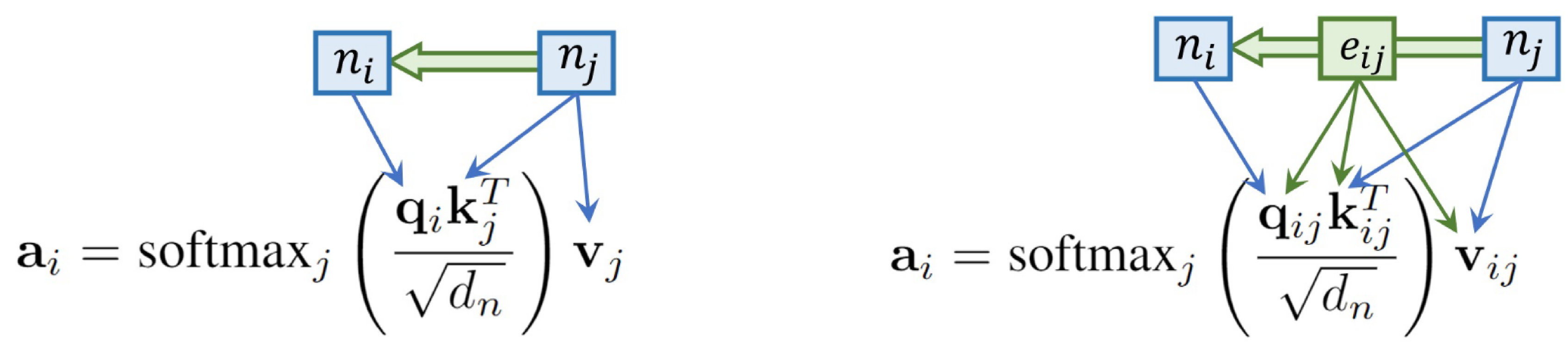
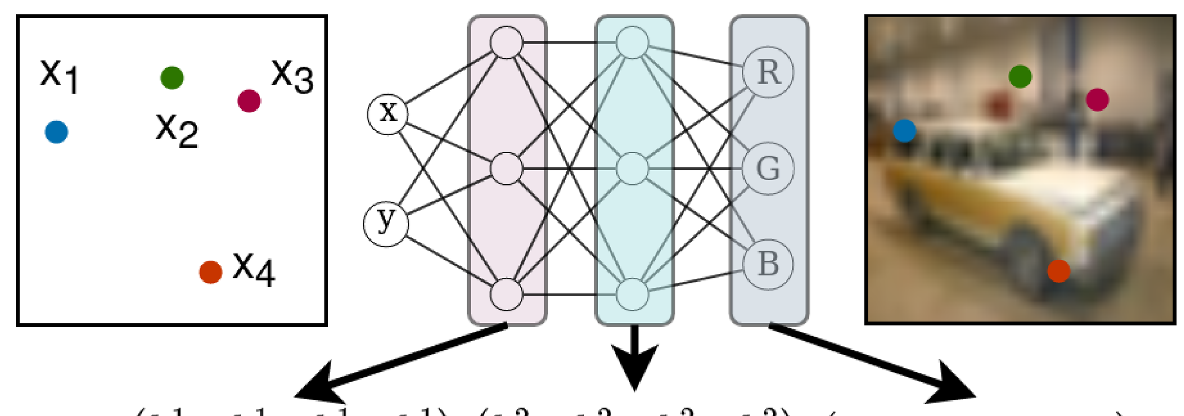


Figure 4. Standard (left) vs. relational (right) attention. Figure credit: [2].

$$\mathbf{q}_{ij} = (\mathbf{n}_i\mathbf{W}_n^Q + \mathbf{e}_{ij}\mathbf{W}_e^Q) \qquad \mathbf{k}_{ij} = (\mathbf{n}_j\mathbf{W}_n^K + \mathbf{e}_{ij}\mathbf{W}_e^K) \qquad \mathbf{v}_{ij} = (\mathbf{n}_j\mathbf{W}_n^V + \mathbf{e}_{ij}\mathbf{W}_e^V)$$

Probe features

Given a neural network as input, we probe it at a set of points. Then, we concatenate all the resulting activations as additional node features to the input graph.



$$(h_1^1 \quad h_2^1 \quad h_3^1 \quad h_4^1)\ (h_1^2 \quad h_2^2 \quad h_3^2 \quad h_4^2)\ (y_1 \quad y_2 \quad y_3 \quad y_4)$$

Experiments

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

Model	Accuracy in %
MLP [4]	$17.6_{\pm 0.0}$
Set NN [4]	$23.7_{\pm 0.1}$
DWSNet [4]	$85.7_{\pm 0.6}$
GNN (Ours)	$94.7_{\pm 0.3}$
Relational transformer (Ours)	**97.3**$_{\pm 0.2}$

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

Model	MSE in 10^{-2}
DWSNet [4]	$2.58_{\pm 0.00}$
NFN [5]	$2.55_{\pm 0.00}$
GNN (Ours)	$2.06_{\pm 0.01}$
Relational transformer (Ours)	**1.75**$_{\pm 0.01}$

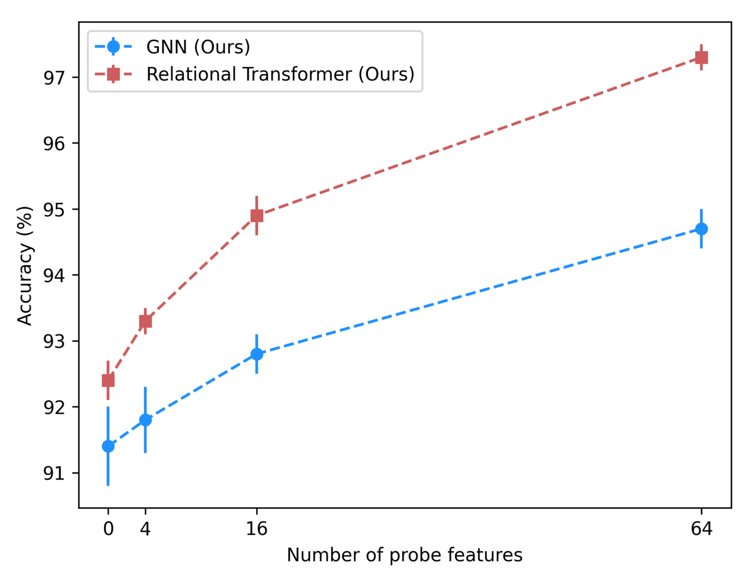


Figure 5. Importance of probe features on classifying MNIST INRs.

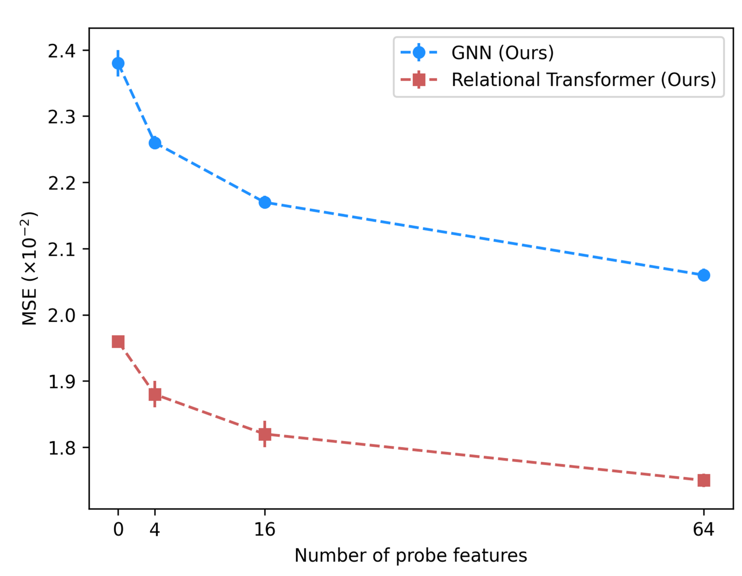


Figure 6. Importance of probe features on dilating MNIST INRs.

Table 3. Position embeddings ablation on MNIST INR classification.

Model	MSE in 10^{-2}
GNN (without PE)	$83.9_{\pm 0.3}$
GNN (Ours)	$91.4_{\pm 0.6}$

Model	MSE in 10^{-2}
Relational transformer (without PE)	$77.9_{\pm 0.7}$
Relational transformer (Ours)	$92.4_{\pm 0.3}$

References

[1] Gabriele Corso et al. "Principal neighbourhood aggregation for graph nets". In: *NeurIPS*. 2020.
[2] Cameron Diao and Ricky Loynd. "Relational Attention: Generalizing Transformers for Graph-Structured Tasks". In: *ICLR*. 2023.
[3] Robert Hecht-Nielsen. "On the algebraic structure of feedforward network weight spaces". In: *Advanced Neural Computers*. Elsevier, 1990.
[4] Aviv Navon et al. "Equivariant architectures for learning in deep weight spaces". In: *ICML*. 2023.
[5] Allan Zhou et al. "Permutation Equivariant Neural Functionals". In: *arXiv preprint arXiv:2302.14040* (2023).