Extended Abstract Track

INRFormer: Neuron Permutation Equivariant Transformer on Implicit Neural Representations

Lei Zhou LEZZHOU@CS.STONYBROOK.EDU
Department of Computer Science, Stony Brook University, NY, USA

Varun Belagali
Department of Computer Science, Stony Brook University, NY, USA

Joseph Bae
Department of Biomedical Informatics, Stony Brook University, NY, USA

Prateek Prasanna
Department of Biomedical Informatics, Stony Brook University, NY, USA

Dimitris Samaras
Department of Computer Science, Stony Brook University, NY, USA

Editors: List of editors' names

Abstract

Implicit Neural Representations (INRs) have demonstrated both precision in continuous data representation and compactness in encapsulating high-dimensional data. Yet, much of contemporary research remains centered on data reconstruction using INRs, with limited exploration into processing INRs themselves. In this paper, we endeavor to develop a model tailored to process INRs explicitly for computer vision tasks. We conceptualize INRs as computational graphs with neurons as nodes and weights as edges. To process INR graphs, we propose *INRFormer* consisting of the node blocks and the edge blocks alternatively. Within the node block, we further propose *SlidingLayerAttention (SLA)*, which performs attention on nodes of three sequential INR layers. This sliding mechanism of the *SLA* across INR layers enables each layer's nodes to access a broader scope of the entire graph's information. In terms of the edge block, every edge's feature vector gets concatenated with the features of its two linked nodes, followed by a projection via an MLP. Ultimately, we formulate the visual recognition as INR-to-INR (*inr2inr*) translations. That is, *INRFormer* transforms the input INR, which maps coordinates to image pixels, to a target INR, which maps the coordinates to the labels. We demonstrate INRFormer on CIFAR10.

Keywords: Implicit Neural Representations, Neuron Permutation Symmetries

1. Introduction

Implicit Neural Representations (INRs) have emerged as a new paradigm for data representation in computer vision (Park et al., 2019; Mescheder et al., 2019; Mildenhall et al., 2020; Yariv et al., 2021; Wang et al., 2021). However, directly operating on the weight space for vision tasks is challenging. **First**, the weight space in neural networks is intrinsic with *neuron permutation symmetries*(Hecht-Nielsen, 1990), i.e., permuting the neurons in any hidden layer of a neural network does not change the network functionality. Some methods (Dupont et al., 2022; Bauer et al., 2023) avoid the permutation equivariance modeling by only handling the INRs whose weights are generated from latent codes. In this study, we consider the general deep weight space learning problem. The **second** challenge is,

weights and biases lack task-specific prior knowledge. Random image transformations for augmentation improves the learning by introducing task-specific priors. However, these image transformations are not straightforward to achieve equivalently on weights and biases. Lacking such priors can lead to overfitting in the learning process. Recent works on *neural functional* either ignore the priors(Zhou et al., 2023a) or introduce additional pre-training for modeling the image prior(Zhou et al., 2023b; De Luigi et al., 2023). We attempt to incorporate the priors into our model design with no need of pre-training.

More recent work(Zhang et al., 2023) provides a new perspective of weight-space modeling. By perceiving INRs as computational graphs, Graph Neural Networks (GNNs)(Kipf and Welling, 2016; Wang et al., 2019; Veličković et al., 2018) can be used to extract representations from INRs. A generic GNN architecture is permutation equivariant(Bronstein et al., 2021), which exactly satisfies the requirement of modeling permutation symmetries in hidden layer neurons. However, we argue that the probe features proposed in (Zhang et al., 2023) can leak the original image pixel information. Thus, it is not a pure weight space representation learning model.

Considering INRs as graphs, we build the graph with weights as edges and neurons as nodes. We embed the values in weight matrices as the initial edges features. Then, we combine linked edges and layer position embedding together to initialize the node features.

We propose *INRFormer* to process INR graphs. INRFormer consists of the node update blocks and the edge update blocks alternatively. In the node block, we further propose *SlidingLayerAttention (SLA)*, which performs attention on nodes of three sequential INR layers. By sliding *SLA* across INR layers enables, each layer's nodes can see a global scope of the entire graph. In the edge block, every edge's feature vector gets concatenated with the features of its two linked nodes, followed by a projection via an MLP. Ultimately, we formulate the considered vision tasks as INR-to-INR (*inr2inr*) translations. In particular, *INRFormer* transforms the input INR, which maps coordinates to image pixels, to a target INR, which maps the coordinates to the labels. To boost the performance, we further propose an auxiliary image crop reconstruction task to incorporate image priors into the model. On CIFAR10, we achieve better results than NFN (Zhou et al., 2023a) and comparable results with NFT (Zhou et al., 2023b).

2. INRFormer

We regard INRs as graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with neurons as nodes and weights as edges. By conceptualizing neurons as nodes, we can model the permutation symmetries on neurons directly with Graph Neural Networks (GNNs) while sparing the trouble of coordinating the permutation of rows and columns in two neighboring weight matrices. Now image classification becomes graph classification. However, INRs show many differences from common graphs. First, for the same task, INRs share the same graph structure and can only be distinguished between each other by weights. Second, INR graphs are hierarchical.

Based on these observations, we propose INRFormer to process INRs.

2.1. Building INR Graphs

2.1.1. Node Features

Before feeding INR graphs into a graph neural network, we need to instantiate nodes in INR graphs by associating features to them. We incorporate two sources of information in the node features, including the features of linked edges and layer position embedding. Please refer to Appendix B for details. Let $\mathcal{V} = \{\boldsymbol{V}_l \in \mathbb{R}^{n_l \times d_n} \mid l = [1:L]\}$ denote the nodes. d_n denotes the dimension of node features.

2.1.2. Edge Features.

Suppose the INR is parametrized with weights and biases $(\boldsymbol{W}, \boldsymbol{b})$. Each entry $\boldsymbol{W}_l^{ij} \in \mathbb{R}$ represents an edge connecting the j-th neuron in $(l-1)$-th layer to the i-th neuron in l-th layer. We need to map each entry value from a scalar $\boldsymbol{W}_l^{ij}$ to a higher dimension $\hat{\boldsymbol{W}}_l^{ij} \in \mathbb{R}^{d_n}$. We use d_e weight embedding ($\texttt{WE}$) functions $\bar{\boldsymbol{W}}_l = \texttt{WE}(\boldsymbol{W}_l) = [\texttt{WE}_1(\boldsymbol{W}_l), \cdots, \texttt{WE}_{d_e}(\boldsymbol{W}_l)]$. Specifically, we use sinusoidal functions specified in Eq. 1.

2.2. Node Step: Sliding Layer Attention (*SLA*)

In the node update step, we propose *Sliding Layer Attention (SLA)*. *SLA* is an attention module operating on neurons across three consecutive layers. To process neurons of all the layers, we slide *SLA* across all the INR layers.

In each sliding window including layers $\{l-1, l, l+1\}$, *SLA* first aggregates linked edges into each node by pooling columns of the precedent edge tensor and rows of the subsequent edge tensor $\hat{\boldsymbol{V}}_l^i = \boldsymbol{V}_l^i + \boldsymbol{W}_l^i[:,:,*] + \boldsymbol{W}_{l+1}^i[:,*,:]$ where i denotes the i-th layer in INRFormer.

We perform attention over $\{\hat{\boldsymbol{V}}_{l-1}, \hat{\boldsymbol{V}}_l, \hat{\boldsymbol{V}}_{l+1}\}$. In each sliding window, we aim to update $\hat{\boldsymbol{V}}_l$ which is thus the query. We concatenate $[\hat{\boldsymbol{V}}_{l-1}, \hat{\boldsymbol{V}}_l, \hat{\boldsymbol{V}}_{l+1}]$ as the memory $\hat{\boldsymbol{V}}_{l-1,l,l+1}$, i.e., keys and values. We compute the standard attention as follows

$$\boldsymbol{V}_l^{i+1} = \texttt{Attention}(Q, K, V) = \texttt{Attention}(\boldsymbol{W}_Q(\hat{\boldsymbol{V}}_l), \boldsymbol{W}_K(\hat{\boldsymbol{V}}_{l-1,,l+1}), \boldsymbol{W}_V(\hat{\boldsymbol{V}}_{l-1,l,l+1}))$$

By avoiding computing attention with neurons of all the layers, the time complexity of updating a single layer's neurons is reduced from $\mathcal{O}(n_l \sum_{l=0}^{L} n_l)$ to $\mathcal{O}(n_l(n_{l-1} + n_l + n_{l+1}))$.

2.3. Edge Step

After node step, to update edges, we concatenate two connected node features to the edge feature itself. Then, we project the combined features with an MLP.

2.4. INR-to-INR Translation (*inr2inr*)

In this paper, we focus on the image classification task. Image classification usually requires the model to output a 1-D distribution vector over all the classes. Thus, we need to extract a representation vector from the last hidden graphs. Instead of pooling nodes or edges, we treat vision tasks as INR-to-INR Translations (*inr2inr*). In other words, our INRFormer still outputs INRs rather than vectors. By forwarding these output INRs with query coordinates which are 2D grids points, we obtain the corresponding feature vectors in these positions. By globally pooling all the feature vectors, we obtain a representation vector for each image. Classification can be done by further appending a linear classifier.

2.5. Auxiliary Task: Image Crop Reconstruction

Learning from weight space lacks the task-specific prior knowledge. We introduce an auxiliary task of reconstructing random image crop to help boost the image classification task. To this end, we append an auxiliary head which is parallel to the classification head. This auxiliary head also outputs INRs which reconstruct an image crop. To hint the auxiliary head which image crop to reconstruct, we add grid coordinates of a random image crop in neurons of the first layer when building the graph nodes in Sec.2.1.1. Such image crop reconstruction can introduce image priors into the entire pipeline.

3. Experiments

3.1. Analysis of Designs

We demonstrate INRFormer on CIFAR-10 which has 50000 training samples and 10000 test samples. We first analyze our model designs in Table 3 with a tiny model of size 0.7M.

`LayerEmb`	`WeightEmb`	#hint	Aux	Accuracy
✓	Sinusoidal	256	✓	**50.57**%
✗	Sinusoidal	256	✓	49.20%
✓	ChebshevPoly	256	✓	50.09%
✓	Sinusoidal	64	✓	49.13%
✓	Sinusoidal	256	✗	47.47%

Table 1: **Ablation Studies**. INRFormer is robust to layer embedding and different weight embedding options.But INRFormer is sensitive to the number of hint points introduced in Sec. 2.5. The auxiliary task is also important to get good results.

3.2. Comparison with Others

We scale the model up to the size of 1.9M and compare with related works in Table 2.

Model	Size	Accuracy
NFN_{HNP} (Zhou et al., 2023a)	-	44.3
NFN_{NP} (Zhou et al., 2023a)	∼20M	46.5%
$\texttt{INR2ARRAY}^{\text{NFN}}$ (Zhou et al., 2023b)	-	45.4%
$\texttt{INR2ARRAY}^{\text{NFT}}$ (Zhou et al., 2023b)	22M	63.4%
INRFormer (ours)	1.9M	60.24%

Table 2: **Comparison with others**. We achieve better result than NFN (Zhou et al., 2023a) and comparable results with NFT (Zhou et al., 2023b) without pre-training.

4. Conclusion

We conceptualize INRs as computational graphs with neurons as nodes and weights as edges. To process INR graphs, we propose *INRFormer* consisting of the node blocks and the edge blocks alternatively. Within the node block, we propose *SlidingLayerAttention (SLA)*, which performs attention on nodes of three sequential INR layers. Ultimately, we formulate the visual recognition task as INR-to-INR (*inr2inr*) translations. In particular, *INRFormer* transforms the input INR, which maps coordinates to image pixels, to a target INR, which maps the coordinates to the labels. We demonstrate INRFormer on CIFAR10.

References

Matthias Bauer, Emilien Dupont, Andy Brock, Dan Rosenbaum, Jonathan Schwarz, and Hyunjik Kim. Spatial functa: Scaling functa to imagenet classification and generation. *arXiv preprint arXiv:2302.03130*, 2023.

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.

Luca De Luigi, Adriano Cardace, Riccardo Spezialetti, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. Deep learning on implicit neural representations of shapes. *arXiv preprint arXiv:2302.05438*, 2023.

Emilien Dupont, Hyunjik Kim, SM Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. *arXiv preprint arXiv:2201.12204*, 2022.

Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*, pages 129–135. Elsevier, 1990.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019.

Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021.

David W Zhang, Miltiadis Kofinas, Yan Zhang, Yunlu Chen, Gertjan J Burghouts, and Cees GM Snoek. Neural networks are graphs! graph neural networks for equivariant processing of neural networks. 2023.

Allan Zhou, Kaien Yang, Kaylee Burns, Yiding Jiang, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals. *arXiv preprint arXiv:2302.14040*, 2023a.

Allan Zhou, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Neural functional transformers. *arXiv preprint arXiv:2305.13546*, 2023b.

Appendix A. Preliminaries

Let's consider an INR that is an L-layer MLP,

$$h(\boldsymbol{x}; \{\boldsymbol{W}, \boldsymbol{b}\}) = \boldsymbol{z}_L, \quad \boldsymbol{z}_{l+1} = \sigma(\boldsymbol{W}_l \boldsymbol{z}_l + \boldsymbol{b}_l), \quad \boldsymbol{z}_0 = \texttt{PE}(\boldsymbol{x})$$

where $\boldsymbol{x}$ denotes position coordinates, σ denotes the non-linear activation function, and PE denotes a positional encoding function. In this case, the MLP h is parameterized by

$$\begin{aligned} \boldsymbol{W} &= \{\boldsymbol{W}_l \in \mathcal{W}_l := \mathbb{R}^{n_l \times n_{l-1}} \mid l \in [1:L]\} \\ \boldsymbol{b} &= \{\boldsymbol{b}_l \in \mathcal{B}_l := \mathbb{R}^{n_l} \mid l \in [1:L]\} \end{aligned}$$

Thus, we define the *weight space* of an L-layer MLP as $\mathcal{U} = \bigoplus_{l=1}^{L} (\mathcal{W}_l \oplus \mathcal{B}_l)$. Considering neurons in each hidden layer can be permuted independently, we define the symmetry group of the *weight space* $\mathcal{U}$ to be the direct product of symmetric groups $\{S_{n_l}\}$ on all the hidden layers: $\mathcal{S} = S_{n_1} \times \cdots \times S_{n_{L-1}}$. Suppose h is an INR representing an image, permuting the

hidden neurons in h modifies the weight matrices, but leaves the stored image information unchanged (Hecht-Nielsen, 1990) — the network produces the same pixel value with same input coordinates. In other words, h is invariant to $\mathcal{S}$.

Let $\boldsymbol{P} = (\boldsymbol{P}_1, \cdots, \boldsymbol{P}_{L-1}) \in \mathcal{S}$. Particularly, $\boldsymbol{P}_l$ denotes the permutation matrix of layer l. Equivalently on weight space, the rows in $\boldsymbol{W}_l$, the columns in $\boldsymbol{W}_{l+1}$, and $\boldsymbol{b}_l$ are all permuted by $\boldsymbol{P}_l$,

$$\boldsymbol{W}_l' = \boldsymbol{P}_l^T \boldsymbol{W}_l \boldsymbol{P}_{l-1}, \quad \boldsymbol{b}_l' = \boldsymbol{P}_l^T \boldsymbol{b}_l, \quad l \in [1:L]$$

Since the input neurons in INRs are position coordinates and the output neurons are pixel values, the neuron orders in both the first and last layers are fixed, i.e., $\boldsymbol{P}_0$ and $\boldsymbol{P}_L$ are identity matrices.

Our goal in this paper is designing a neural network f that operates on weight space $\mathcal{V}$ directly to extract representations which are ultimately invariant to $\mathcal{S}$. Following the notation routine in (Zhou et al., 2023a), we define $\mathcal{S}$-invariant and $\mathcal{S}$-equivariant functions as,

$$\begin{aligned} f: \mathcal{U} \to \mathbb{R}^d \ \texttt{is}\ \mathcal{S}\texttt{-invariant} &\iff f(\boldsymbol{P}\boldsymbol{U}) = f(\boldsymbol{U}), \quad \forall\, \boldsymbol{U} \in \mathcal{U}, \boldsymbol{P} \in \mathcal{S} \\ f: \mathcal{U} \to \mathcal{U} \ \texttt{is}\ \mathcal{S}\texttt{-equivariant} &\iff f(\boldsymbol{P}\boldsymbol{U}) = \boldsymbol{P} f(\boldsymbol{U}), \quad \forall\, \boldsymbol{U} \in \mathcal{U}, \boldsymbol{P} \in \mathcal{S} \end{aligned}$$

An ultimately $\mathcal{S}$-invariant f can be built by stacking any depth of $\mathcal{S}$-equivariant modules and appending an $\mathcal{S}$-invariant module in the end of the stack. Thus, we focus on designing two kinds of modules, the $\mathcal{S}$-equivariant one and $\mathcal{S}$-invariant one.

Appendix B. Building Node Features

Let $\mathcal{V} = \{\boldsymbol{V}_l \in \mathbb{R}^{n_l \times d_n} \mid l = [1:L]\}$ denote the nodes. d_n denotes the dimension of node features.

Source 1: Linked Edges. Suppose the INR is parametrized with weights and biases $(\boldsymbol{W}, \boldsymbol{b}) \in \mathcal{U}$. Each entry $\boldsymbol{W}_l^{ij} \in \mathbb{R}$ represents an edge connecting the j-th neuron in $(l-1)$-th layer to the i-th neuron in l-th layer. We first map each entry value from a scalar $\boldsymbol{W}_l^{ij}$ to a higher dimension $\hat{\boldsymbol{W}}_l^{ij} \in \mathbb{R}^{d_n}$ using d_n weight embedding ($\texttt{WE}$) functions.

$$\hat{\boldsymbol{W}}_l = \texttt{WE}(\boldsymbol{W}_l) = [\texttt{WE}_1(\boldsymbol{W}_l), \cdots, \texttt{WE}_{d_n}(\boldsymbol{W}_l)]$$

In this study, we investigate two kinds of weight embedding functions. The first kind is Chebysehv Polynomials. Specifically, the k-th function is $\texttt{WE}_k(x) = T_k(x)$ where $T_k(\cdot)$ is the k-th Chebyshev polynomial. The second kind is sinusoidal functions. In this case,

$$\texttt{WE}(\boldsymbol{W}_l) = [\texttt{sin}(w_1 \boldsymbol{W}_l), \texttt{cos}(w_1 \boldsymbol{W}_l), \cdots, \texttt{sin}(w_{d_n/2} \boldsymbol{W}_l), \texttt{cos}(w_{d_n/2} \boldsymbol{W}_l)]$$

where $\{w_1, \cdots, w_{d_n/2}\}$ are evenly sampled from range from 0 to 10.

After we obtain higher-dimensional edge features $\hat{\boldsymbol{W}}_l \in \mathbb{R}^{d_n \times n_l \times n_{l-1}}$, we incorporate the edge information to the node features as follows,

$$\boldsymbol{V}_l^{(1)} = \boldsymbol{b}_l + \hat{\boldsymbol{W}}_l[:, :, *] + \hat{\boldsymbol{W}}_{l+1}[:, *, :]$$

where $*$ denotes the pooling operation over the specified dimension and $:$ denotes keeping the specified dimension unchanged. We use max pooling as the pooling operation Qi et al.

(2017). Briefly speaking, the above procedure initializes node features with the linked weight matrix entries and biases.

Source 2: Layer Position Embedding. We further encode the layer position information into the node features. We create L learnable parameters $\{\texttt{LE}_l \in \mathbb{R}^{d_n} \mid l \in [0:L]\}$ as `Layer Embeddings`, i.e., $\boldsymbol{V}_l^{(2)} = \texttt{LE}_l$.

Finally, we combine these two sources as the final node features $\boldsymbol{V}_l = \boldsymbol{V}_l^{(1)} + \boldsymbol{V}_l^{(2)}$

Appendix C. Sinusoidal Weight Embedding Functions

$$\texttt{WE}(\boldsymbol{W}_l) = [\texttt{sin}(w_1\boldsymbol{W}_l), \texttt{cos}(w_1\boldsymbol{W}_l), \cdots, \texttt{sin}(w_{d_n/2}\boldsymbol{W}_l), \texttt{cos}(w_{d_n/2}\boldsymbol{W}_l)] \tag{1}$$

where $\{w_1, \cdots, w_{d_n/2}\}$ are evenly sampled from range from 0 to 10.

Appendix D. Complete Ablation Studies

`LayerEmb`	`WeightEmb`	#hint	#query	Aux	Same Tr/Ts Init Weights	Accuracy
✓	Sinusoidal	256	64	✓	✓	**50.57**%
✗	Sinusoidal	256	64	✓	✓	49.20%
✓	ChebshevPoly	256	64	✓	✓	50.09%
✓	Sinusoidal	64	64	✓	✓	49.13%
✓	Sinusoidal	16	64	✓	✓	47.89%
✓	Sinusoidal	256	256	✓	✓	50.01%
✓	Sinusoidal	256	16	✓	✓	49.19%
✓	Sinusoidal	256	64	✗	✓	47.47%
✓	Sinusoidal	256	64	✓	✗	47.01%

Table 3: **Ablation Studies on INRFormer Design**. INRFormer is robust to layer embedding, different weight embedding options and the number of query points (the points fed to the output INRs to get feature vectors). On the other hand, INRFormer is sensitive to the number of hint points introduced in Sec. 2.5. The auxiliary task is also important to get good results. We also test our model generalization ability across different initial weights. Interestingly, our model can generalize to INRs which are initliazed with different random seeds from the training initial random seeds.