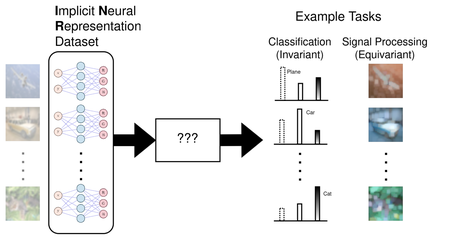
Graph Neural Networks for Learning Equivariant Representations of Neural Networks

Miltiadis Kofinas[1]* Boris Knyazev[2] Yan Zhang[2] Yunlu Chen[3]
Gertjan J. Burghouts[4] Efstratios Gavves[1] Cees G. M. Snoek[1] David W. Zhang[1]*

* Joint first and last authors

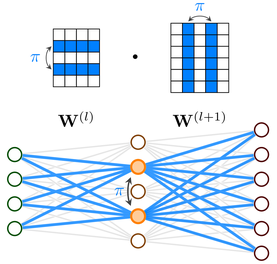
[1]University of Amsterdam [2]Samsung - SAIT AI Lab, Montreal [3]Carnegie Mellon University [4]TNO

Introduction - Networks for networks



Introduction – Neuron symmetries

Neurons in an MLP can be *reordered* while maintaining exactly the same function [4]. Reordering neurons here means changing the preceding and following weights attached to the neuron accordingly.



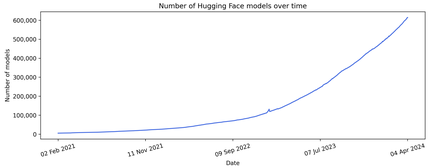
Weight symmetries in a 2-layer MLP subnetwork.

Related works

- Overlook the inherent permutation symmetry
- Rely on intricate weight-sharing patterns to achieve equivariance
- Ignore the network architecture itself, limited to a single architecture

Neural networks are the new data!

More than $600,000$ models on April 2024!



Source: `https://huggingface.co/models`

Networks for networks, a paradigm shift?

Traditional Paradigm	Modern Paradigm
Save signal as an array Process with CNNs/ViTs	Fit signal with an INR and save its parameters Process with parameter space networks
Train NNs with hyperparameter search	Model zoos & NNs are the new data! Generative parameter space networks on the weights

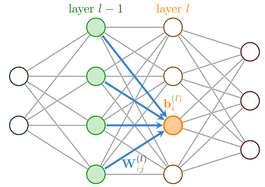
Keywords: deep weight space | graph neural networks | transformers | permutation equivariance | neural functionals | implicit neural representations | networks for networks | neural graphs

Neural networks as neural graphs

Neural network feedforward activation (neuron i in layer l)

$$\mathbf{x}_i^{(l)} = \sigma\left(\mathbf{b}_i^{(l)} + \sum_j \mathbf{W}_{ij}^{(l)} \mathbf{x}_j^{(l-1)}\right)$$

Neural network as **neural graph**:
Node i feature: $\boldsymbol{V}_i^{(l)} \leftarrow \mathbf{b}_i^{(l)}$
Edge $j \to i$ feature: $\boldsymbol{E}_{ij}^{(l)} \leftarrow \mathbf{W}_{ij}^{(l)}$



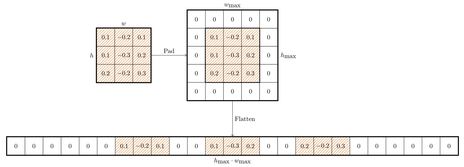
Neural graphs accommodate heterogeneous architectures:

- ✓ Architectures with varying computational graphs
- ✓ Different numbers of layers
- ✓ Different number of hidden dimensions
- ✓ Different non-linearities
- ✓ Residual connections

Node & edge features

$$\boldsymbol{E} = \begin{pmatrix} \mathbf{W}^{(1)\top} & & & \mathbf{0} \\ & \mathbf{W}^{(2)\top} & & \\ & & \ddots & \\ \mathbf{0} & & & \mathbf{W}^{(L)\top} \end{pmatrix}, \quad \boldsymbol{V} = \begin{pmatrix} \mathbf{0}_{d_0} \\ \mathbf{b}^{(1)} \\ \mathbf{b}^{(2)} \\ \vdots \\ \mathbf{b}^{(L)} \end{pmatrix}$$

CNNs as neural graphs



Graph networks for neural networks

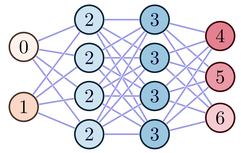
We adapt existing GNNs and transformers for neural graphs.

NG-GNN: We extend PNA [2] with an MLP that updates the edge features given the incident nodes' features and the previous layer's edge features.

NG-T: We extend Relational Transformer [3] with multiplicative interactions between node and edge features to algorithmically align it with the forward-pass of a neural network.

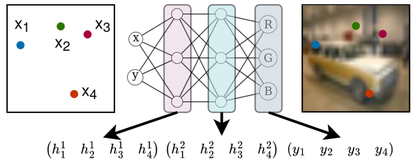
Positional embeddings

We impose order in the input & output nodes by adding **positional embeddings**. Hidden neurons in each layer are orderless, so they share the same positional embedding.



Probe features

Learn a set of sample inputs that we pass through the input neural network and monitor the intermediate activations and the output. We include these features as additional node features.



Heterogeneous architectures

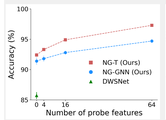
- **Non-linearities:** Learned embeddings as added node features
- **Residual connections:** Additional edges with weight 1
- **Normalization layers:** Linear layers with diagonal weights
- **Self-attention:** 3-dimensional edge features (Q, K, V)

Experiments

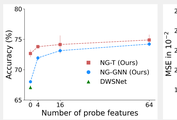
INR classification & style editing

Tasks:

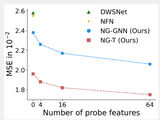
- Classify datasets of INRs fitted individually on images.
- Enlarge images (INRs) through dilation.



MNIST INR classification



Fashion MNIST INR classification



MNIST INR dilation

Ablation study. Importance of positional embeddings on MNIST INR classification.

Method	Accuracy in %
NG-GNN (Ours)	**91.4**±0.6
NG-GNN w/o positional embeddings	83.9±0.3
NG-T (Ours)	**92.4**±0.3
NG-T w/o positional embeddings	77.9±0.7

Predicting CNN generalization

Tasks: Predict the generalization performance of CNN classifiers based on their parameters.

New dataset: We introduce *CNN Wild Park*, a dataset of heterogeneous CNNs that vary in the number of layers, kernel sizes, activation functions, and residual connections.

We measure performance using Kendall's τ. Higher is better.

Method	CIFAR10-GS [9]	CIFAR10 Wild Park
NFN_{HNP} [10]	0.934±0.001	–
StatNN [9]	0.915±0.002	0.719±0.010
NG-GNN (Ours)	0.930±0.001	0.804±0.009
NG-T (Ours)	**0.935**±0.000	**0.817**±0.007

Ablation study. Importance of non-linearity embeddings on predicting CNN generalization on CNN Wild Park.

Method	Kendall's τ (↑)
StatNN [9]	0.719±0.010
NG-GNN (Ours)	**0.804**±0.009
NG-GNN w/o activation embedding	0.778±0.018
NG-T (Ours)	**0.817**±0.007
NG-T w/o activation embedding	0.728±0.010

Learning to optimize

An exciting new application for neural graphs!

Task: Train a neural network (optimizer) that can optimize the weights of other neural networks (optimizee).

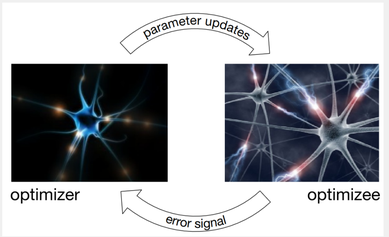
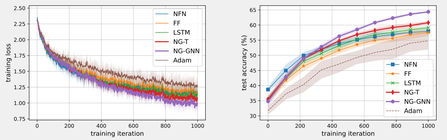


Figure credit: [1]

We train optimizers on Fashion MNIST, evaluate on Fashion MNIST & CIFAR10. We report the test image classification accuracy (%) after 1,000 steps.

Optimizer	FashionMNIST (validation task)	CIFAR-10 (test task)
Adam [5]	80.97±0.66	54.76±2.82
FF [7]	85.08±0.14	57.55±1.06
LSTM [6]	85.69±0.23	59.10±0.66
NFN [10]	83.78±0.58	57.95±0.64
NG-GNN (Ours)	85.91±0.37	**64.37**±0.34
NG-T (Ours)	**86.52**±0.19	60.79±0.51



Training (left) and testing (right) curves on CIFAR-10 for the baseline optimizers (Adam, FF, LSTM, NFN) and the optimizers trained with our NG-GNN and NG-T.

Scan me!

References

[1] Marcin Andrychowicz et al. "Learning to learn by gradient descent by gradient descent". In: *NIPS*. 2016.
[2] Gabriele Corso et al. "Principal neighbourhood aggregation for graph nets". In: *NeurIPS*. 2020.
[3] Cameron Diao and Ricky Loynd. "Relational Attention: Generalizing Transformers for Graph-Structured Tasks". In: *ICLR*. 2023.
[4] Robert Hecht-Nielsen. "On the algebraic structure of feedforward network weight spaces". In: *Advanced Neural Computers*. Elsevier, 1990.
[5] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *ICLR*. 2015.
[6] Luke Metz et al. "Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves". In: *arXiv preprint arXiv:2009.11243* (2020).
[7] Luke Metz et al. "Understanding and correcting pathologies in the training of learned optimizers". In: *ICML*. 2019.
[8] Aviv Navon et al. "Equivariant architectures for learning in deep weight spaces". In: *ICML*. 2023.
[9] Thomas Unterthiner et al. "Predicting Neural Network Accuracy from Weights". In: *arXiv preprint arXiv:2002.11448* (2020).
[10] Allan Zhou et al. "Permutation Equivariant Neural Functionals". In: *NeurIPS*. 2023.