Neural Networks as Graphs & Graph Learning Methods

Boris Knyazev (k-nya-zev)

https://bknyaz.github.io/

Université de Montréal
Adjunct Professor

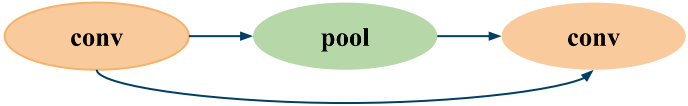
Mila

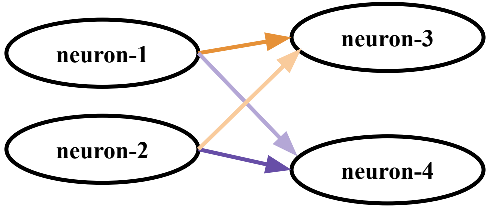
McGill
Guest Lecture,
March 18, 2026

SAMSUNG DS
Research Scientist

Agenda

1. **Background**
   a. Graphs in Machine Learning
   b. Directed Acyclic Graphs (DAGs)
   c. GNNs
2. **Graph of neural architectures**
   a. Graph Neural Networks (GNNs) for DAGs
3. **Graph of parameters**
   a. Neural Graphs (NGs)
   b. GNNs for NGs
4. **Future research**





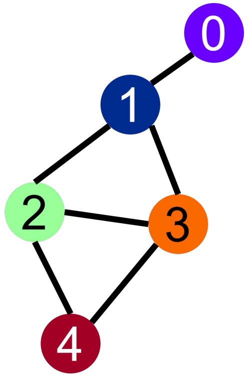
1. Background: Graphs, DAGs, GNNs

Graphs in Computer Science

$G=(V,E)$

$V=\{0,1,2,3,4\}$

$E=\{(0,1), (1,2), (1,3), (2,3), (2,4), (3,4)\}$



NetworkX
Network Analysis in Python

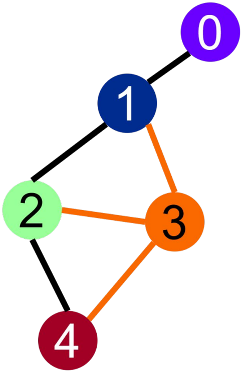
Graphs in Machine Learning

$G=(V,E) \rightarrow G=(\mathbf{X},\mathbf{A},\mathbf{E})$

X - node features ($n \times d$)

A - adjacency matrix ($n \times n$)

E - edge features ($m \times d$)



n=5
m=6

	0	1	2	3	4
0	0	1	0	0	0
1	1	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	0	0	1	1	0

A

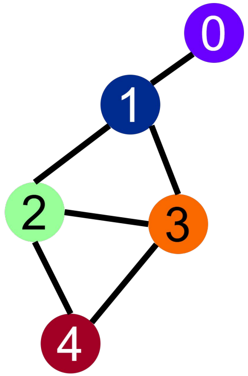
Graphs in Machine Learning

$G=(V,E) \rightarrow G=(\mathbf{X},\mathbf{A},\mathbf{E})$

X - node features ($n \times d$)

A - adjacency matrix ($n \times n$)

E - edge features ($m \times d$)



Example of X (one-hot encoding):

node 0:	[0, **1**, 0, 0, 0]
node 1:	[0, 0, 0, **1**, 0]
node 2:	[0, 0, 0, **1**, 0]
node 3:	[0, 0, 0, **1**, 0]
node 4:	[0, 0, **1**, 0, 0]

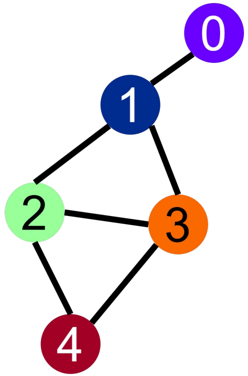
Graphs in Machine Learning

$G=(V,E) \rightarrow G=(\mathbf{X},\mathbf{A},\mathbf{E})$

X - node features ($n \times d$)

A - adjacency matrix ($n \times n$)

E - edge features ($m \times d$)



Example of X (one-hot encoding):

node 0: [0, **1**, 0, 0, 0]
node 1: [0, 0, 0, **1**, 0]
node 2: [0, 0, 0, **1**, 0]
node 3: [0, 0, 0, **1**, 0]
node 4: [0, 0, **1**, 0, 0]

Example of E (zero-padded to *d*):

edge (0,1): [0.5, 0, 1, 0, 0]
edge (1,2): [0.8, 1, 1, 0, 0]
edge (1,3): [0.4, 0, 0, 0, 0]
edge (2,3): [1.2, 2, 0, 0, 0]
edge (2,4): [2.4, 1, 1, 0, 0]
edge (3,4): [0.2, 2, 1, 0, 0]

Directed Acyclic Graphs (DAGs) in Machine Learning

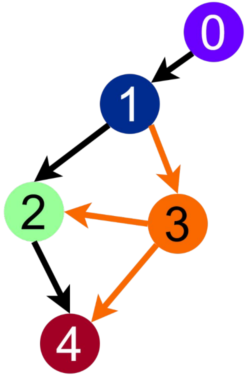
$G=(\mathbf{X},\mathbf{A},\mathbf{E})$

X - node features ($n \times d$)

A - adjacency matrix ($n \times n$)

E - edge features ($m \times d$)

Topological sorting: 0, 1, **3, 2,** 4

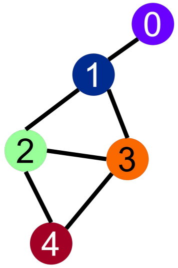


	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	1	0
2	0	0	0	0	1
3	0	0	1	0	1
4	0	0	0	0	0

A

Improving Node Features X

Laplacian Positional Encoding (LPE) describes "positions" of nodes in a graph:



$$\Delta = \mathrm{I} - D^{-1/2} A D^{-1/2} = U^T \Lambda U$$

Example of X (node degree features):

node 0: [0, 1, 0, 0, 0]
node 1: [0, 0, 0, 1, 0]
node 2: [0, 0, 0, 1, 0]
node 3: [0, 0, 0, 1, 0]
node 4: [0, 0, 1, 0, 0]

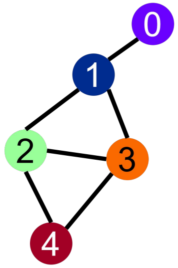
LPE (*k* smallest non-trivial eigenvectors in U):

node 0: [-0.65, 0.00, -0.50, 0.49]
node 1: [-0.49, 0.00, 0.29, -0.65]
node 2: [0.25, -0.71, 0.29, 0.33]
node 3: [0.25, 0.71, 0.29, 0.33]
node 4: [0.46, 0.00, -0.71, -0.35]

Benchmarking Graph Neural Networks. Vijay Prakash Dwivedi et al., 2022.

Improving Node Features X

Laplacian Positional Encoding (LPE) describes "positions" of nodes in a graph:



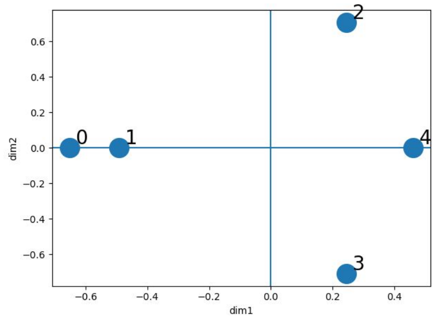
$$\Delta = \mathrm{I} - D^{-1/2} A D^{-1/2} = U^T \Lambda U$$

Example of X (node degree features):

node 0: [0, 1, 0, 0, 0]
node 1: [0, 0, 0, 1, 0]
node 2: [0, 0, 0, 1, 0]
node 3: [0, 0, 0, 1, 0]
node 4: [0, 0, 1, 0, 0]

LPE (*k* smallest non-trivial eigenvectors in *U*):

	x	y		
node 0:	[-0.65,	0.00,	-0.50,	0.49]
node 1:	[**-0.49**,	**0.00**,	0.29,	-0.65]
node 2:	[**0.25**,	**-0.71**,	0.29,	0.33]
node 3:	[**0.25**,	**0.71**,	0.29,	0.33]
node 4:	[0.46,	0.00,	-0.71,	-0.35]



Benchmarking Graph Neural Networks. Vijay Prakash Dwivedi et al., 2022.

Graph Neural Networks (GNNs)

0	1	0	0	0
1	0	1	1	0
0	1	0	1	1
0	1	1	0	1
0	0	1	1	0

A (5×5)

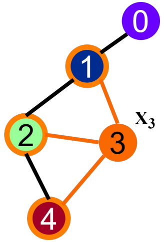
×

-0.65	0.00	-0.50	0.49
-0.49	0.00	0.29	-0.65
0.25	-0.71	0.29	0.33
0.25	0.71	0.29	0.33
0.46	0.00	-0.71	-0.35

X (5×4)

×

W (4×*d*)



$\mathbf{H} = \mathbf{XW}$ - projected node features

$\mathbf{Z}_3 = \sigma(\mathbf{H}_1+\mathbf{H}_2+\mathbf{H}_4)$ - updated features for node 3

$\mathbf{Z} = \sigma(\mathbf{AXW})$ - updated features for all nodes

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\big(\hat{A}XW^{(0)}\big)\, W^{(1)}\Big)$$

Semi-Supervised Classification with Graph Convolutional Networks. Thomas N. Kipf, Max Welling, 2016.

Graph Neural Networks (GNNs)

Properties:

- Locality
- Weight Sharing
- Equivariance
- Invariance

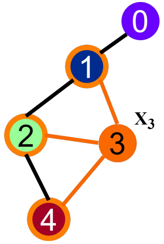
0	1	0	0	0
1	0	1	1	0
0	1	0	1	1
0	1	1	0	1
0	0	1	1	0

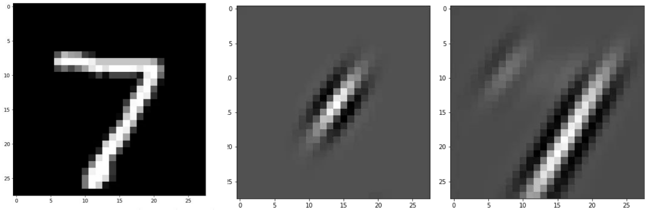
A (5×5) ×

-0.65	0.00	-0.50	0.49
-0.49	0.00	0.29	-0.65
0.25	-0.71	0.29	0.33
0.25	0.71	0.29	0.33
0.46	0.00	-0.71	-0.35

X (5×4) ×

W (4×d)

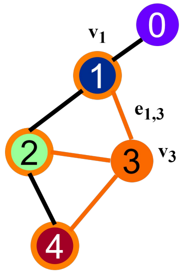




[Tutorial on Graph Neural Networks for Computer Vision and Beyond]

GNNs with Edge Features

Step	Typical GNN (Corso et al., 2020)	Neural Graph GNN (Kofinas et al., 2024)
1. Message passing	$\mathbf{m}_{ij} = \phi_m\left([\mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_{ij}]\right)$	$\mathbf{m}_{ij} = \phi_{\texttt{scale}}\left(\mathbf{e}_{ij}\right) \odot \phi_m\left([\mathbf{v}_i, \mathbf{v}_j]\right) + \phi_{\texttt{shift}}\left(\mathbf{e}_{ij}\right)$
2. Aggregation	$\mathbf{v}_i = \phi_a\left(1/\lvert\mathcal{N}(i)\rvert \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}\right)$	
3. Edge update	$\mathbf{e}_{ij}^{(k+1)} = \mathbf{e}_{ij}^{(k)}$	$\mathbf{e}_{ij}^{(k+1)} = \phi_e^{(k+1)}\left(\left[\mathbf{v}_i^{(k)}, \mathbf{e}_{ij}^{(k)}, \mathbf{v}_j^{(k)}\right]\right)$



m - projected edge features

$\mathbf{v}_3 = \sigma(\mathbf{m}_{1,3}+\mathbf{m}_{2,3}+\mathbf{m}_{4,3})$ - updated features for node 3

Graph Neural Networks for Learning Equivariant Representations of Neural Networks. Miltiadis Kofinas et al., 2024.

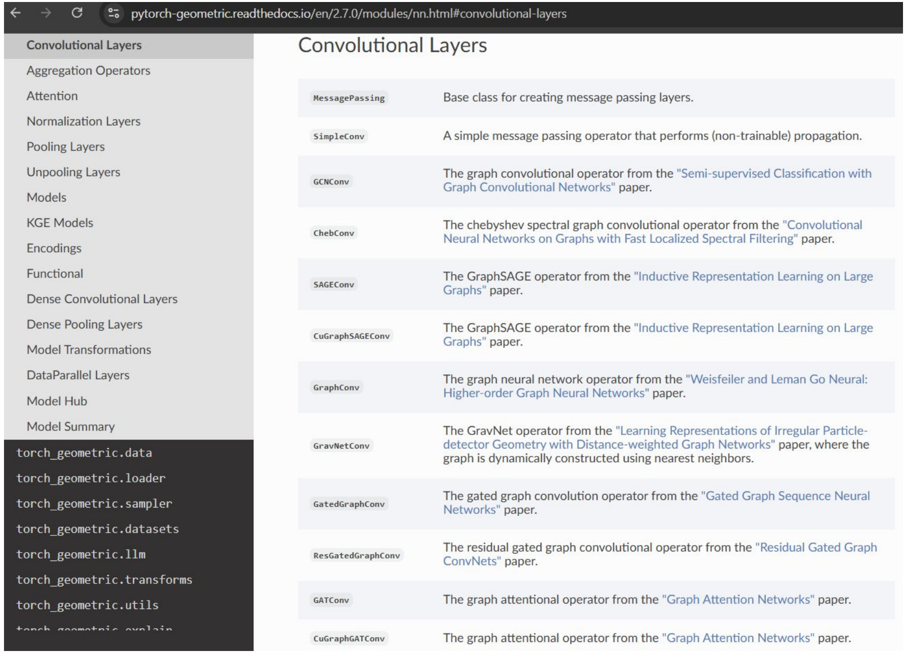
Modern GNNs

Strong GNNs:

- **PNA** (Principal Neighbourhood Aggregation)
- **GPS** (General, Powerful, Scalable Graph Transformer)

Properties:

- Global structure (GPS)
- Expressiveness (PNA and GPS)
- Efficiency (PNA)



pytorch-geometric.readthedocs.io/en/2.7.0/modules/nn.html#convolutional-layers

Convolutional Layers
Aggregation Operators
Attention
Normalization Layers
Pooling Layers
Unpooling Layers
Models
KGE Models
Encodings
Functional
Dense Convolutional Layers
Dense Pooling Layers
Model Transformations
DataParallel Layers
Model Hub
Model Summary
torch_geometric.data
torch_geometric.loader
torch_geometric.sampler
torch_geometric.datasets
torch_geometric.llm
torch_geometric.transforms
torch_geometric.utils

Convolutional Layers

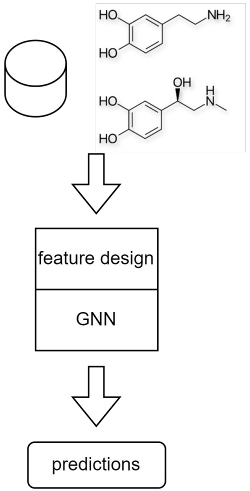
MessagePassing	Base class for creating message passing layers.
SimpleConv	A simple message passing operator that performs (non-trainable) propagation.
GCNConv	The graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper.
ChebConv	The chebyshev spectral graph convolutional operator from the "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" paper.
SAGEConv	The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper.
CuGraphSAGEConv	The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper.
GraphConv	The graph neural network operator from the "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks" paper.
GravNetConv	The GravNet operator from the "Learning Representations of Irregular Particle-detector Geometry with Distance-weighted Graph Networks" paper, where the graph is dynamically constructed using nearest neighbors.
GatedGraphConv	The gated graph convolution operator from the "Gated Graph Sequence Neural Networks" paper.
ResGatedGraphConv	The residual gated graph convolutional operator from the "Residual Gated Graph ConvNets" paper.
GATConv	The graph attentional operator from the "Graph Attention Networks" paper.
CuGraphGATConv	The graph attentional operator from the "Graph Attention Networks" paper.

https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#convolutional-layers

Training and Applying GNNs

SAMSUNG

- **Prediction** of molecule properties and molecule **generation** (e.g., OLED molecules)
- GNNs and their combinations with LLMs/sequential methods



Challenges:

- Efficiency of training and application to large graphs
- Generalization (larger graphs, unseen complex structures)
- Integration with LLMs

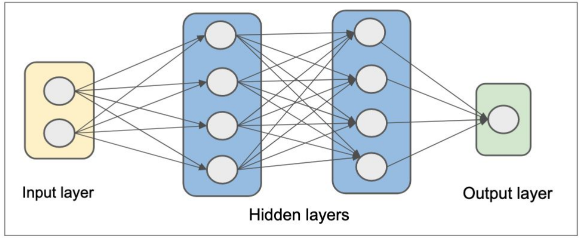
- Any-Property-Conditional Molecule Generation with Self-Criticism using Spanning Trees. Alexia Jolicoeur-Martineau et al., 2025.
- Pretrained language models to solve graph tasks in natural language. Frederik Wenkel et al., 2023.

1. Background Summary

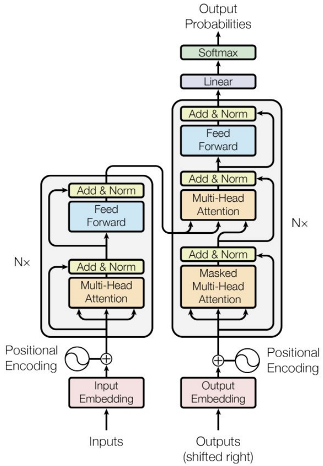
1. Graphs in Computer Science
2. Graphs in Machine Learning
3. DAGs
4. Laplacian Positional Encoding
5. GNNs
6. GNNs with Edge Features
7. Training and Applying GNNs
8.

2. Graph of Architectures

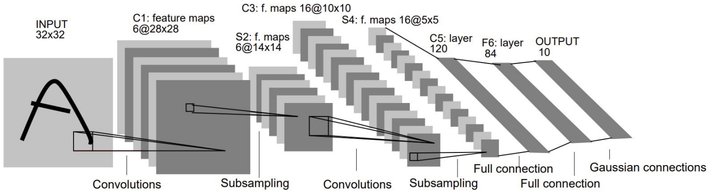
How we illustrate Neural Networks?



https://deeprevision.github.io/posts/001-transformer/

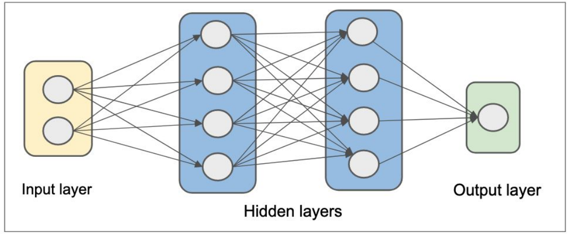


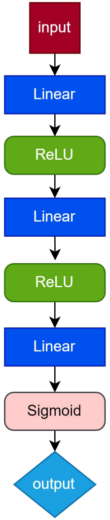
Attention Is All You Need. Ashish Vaswani et al., 2017.



Gradient-based learning applied to document recognition. Yann Lecun et al., 1998.

Graph of Architectures: DAG





Based on which representation we can implement this network?

Code

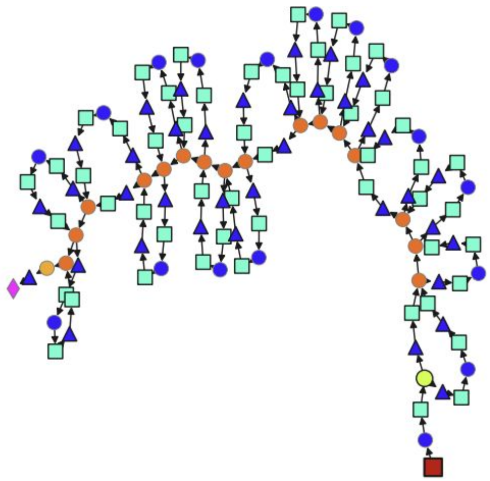
Example:

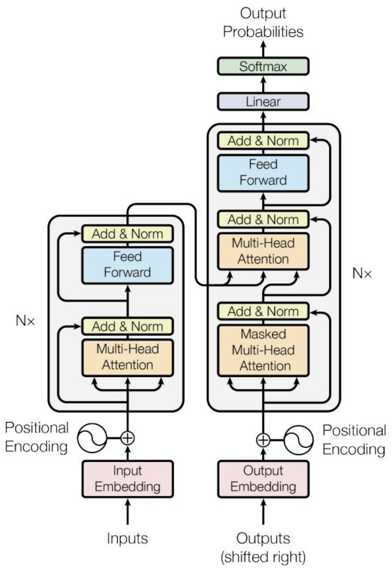
X - node features**:**

node 0: [**1**, 0, 0, 0, 0], input
node 1: [0, **1**, 0, 0, 0], shape: 2x4
node 2: [0, 0, **1**, 0, 0], relu
node 3: [0, **1**, 0, 0, 0], shape: 4x4
node 4: [0, 0, **1**, 0, 0], relu
node 5: [0, **1**, 0, 0, 0], shape: 4x1
node 6: [0, 0, 0, **1**, 0], sigmoid
node 7: [0, 0, 0, 0, **1**], output

A - adjacency matrix (forward pass flow)
A^T - backward pass

DAGs of Modern Neural Networks





Gated GNN

$$\forall \pi \in [\mathrm{fw}, \mathrm{bw}]$$

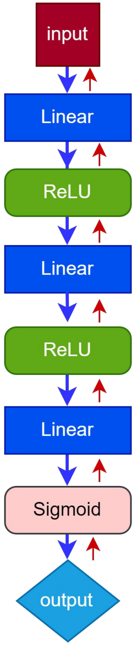
$$\forall v \in \pi$$

$$\mathbf{m}_v = \sum_{u \in \mathcal{N}_v^\pi} \mathrm{MLP}(\mathbf{h}_u)$$

$$\mathbf{h}_v = \mathrm{GRU}(\mathbf{h}_v, \mathbf{m}_v)$$

Gated GNN layer:

1. Topological sorting
2. For each node:
   a. Compute a message
   b. Update node state



- DAGNN: Directed Acyclic Graph Neural Networks. Veronika Thost, Jie Chen, 2021.
- GHN: Graph HyperNetworks for Neural Architecture Search. Chris Zhang et al., 2019.
- GHN-2: Parameter Prediction for Unseen Deep Architectures. Boris Knyazev et al., 2021.

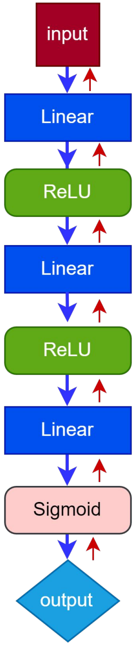
Gated GNN

Neural Architecture Search results

	NA	
Model	RMSE ↓	Pearson's r ↑
S-VAE	$0.521_{\pm 0.002}$	$0.847_{\pm 0.001}$
GraphRNN	$0.579_{\pm 0.002}$	$0.807_{\pm 0.001}$
GCN	$0.482_{\pm 0.003}$	$0.871_{\pm 0.001}$
DeepGMG	$0.478_{\pm 0.002}$	$0.873_{\pm 0.001}$
D-VAE	$0.375_{\pm 0.003}$	$0.924_{\pm 0.001}$
DAGNN	$\mathbf{0.264}_{\pm 0.004}$	$\mathbf{0.964}_{\pm 0.001}$

Compared to a simple GNN layer σ(AXW):

- Gated GNNs allow all nodes to have information about other nodes according to the DAG structure
- Gated GNNs are sequential, so could be slower

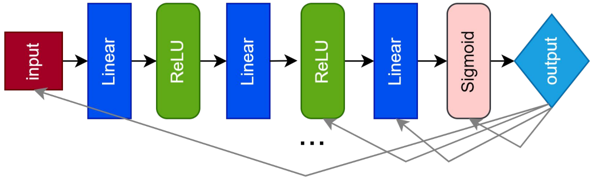


- DAGNN: Directed Acyclic Graph Neural Networks. Veronika Thost, Jie Chen, 2021.
- GHN: Graph HyperNetworks for Neural Architecture Search. Chris Zhang et al., 2019.
- GHN-2: Parameter Prediction for Unseen Deep Architectures. Boris Knyazev et al., 2021.

Graph Transformers on DAGs

Graphormers:

1. Apply regular Transformer layers to node features
2. Add edge information as a bias term to self-attention



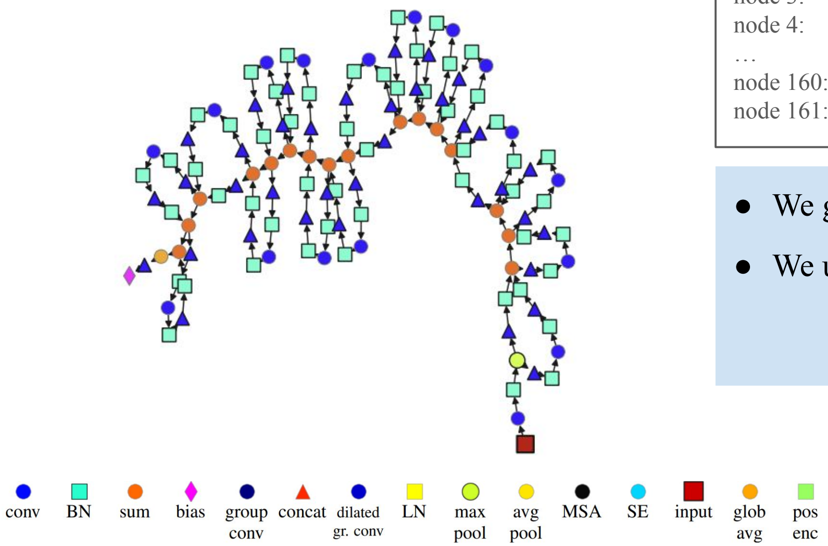
Compared to simple and Gated GNN:

- Global attention
- Efficient parallel computation

- Graphormers: Do Transformers Really Perform Badly For Graph Representation? Chengxuan Ying et al., 2021.
- Transformers Meet Directed Graphs. Simon Geisler et al., 2023.
- GHN-3: Can We Scale Transformers to Predict Parameters of Diverse ImageNet Models? Boris Knyazev et al., 2023.

Applications of DAGs

ResNet



node 1: [**1**, 0, 0, 0, 0], input
node 2: [0, **1**, 0, 0, 0], conv
node 3: [0, 0, 0, 0, **1**], bias
node 4: [0, 0, 0, **1**, 0], pooling
…
node 160: [0, 0, **1**, 0, 0], fc
node 161: [0, 0, 0, 0, **1**], bias

- We generated a dataset of 1M graphs (no need to train the parameters!)
- We use the dataset for training a **parameter prediction** model (GHN)

GHN = Graph HyperNetwork - a special type of GNN on DAGs

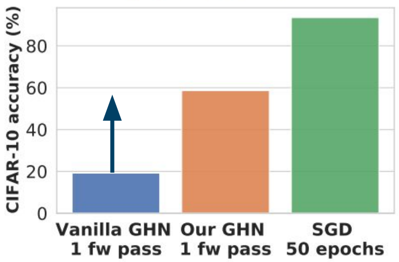
- GHN-2: Parameter Prediction for Unseen Deep Architectures. Boris Knyazev et al., 2021.
- GHN-3: Can We Scale Transformers to Predict Parameters of Diverse ImageNet Models? Boris Knyazev et al., 2023.

Applications of DAGs – Example

```
# Pseudo-code

ghn = GHN('imagenet')  # 1. pretrained GHN
model = resnet50()     # 2. can be almost any model
graph = dag(model)     # 3. extract a DAG given the model
model = ghn(graph)     # 4. returns model with predicted parameters
y = model(test_images) # 5. model can achieve high acc right away
```

Example of evaluating on an unseen architecture $a \notin \mathcal{F}$ (ResNet-50)



- Given a DAG of the architecture, GHNs predict "good" parameters for the model.
- Applications include:
  - Efficient Training (good initialization)
  - Neural Architecture Search

2. Graph of Architectures – Summary

1. Neural Network *Architectures* can be represented as DAGs
2. Gated GNNs *vs* GNNs
3. Graph Transformers *vs* Gated GNNs
4. We can train *parameter prediction/generation models* (GHNs)
   a. Efficient Training (good initialization)
   b. Neural Architecture Search
5. Limitations
   a. DAGs do not model individual parameters
   b. DAGs do not model dataset characteristics
   c. DAG GNNs can struggle with novel architectures

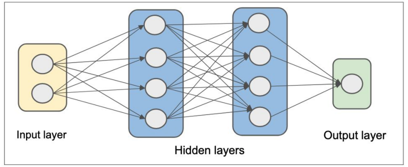
Additional References:

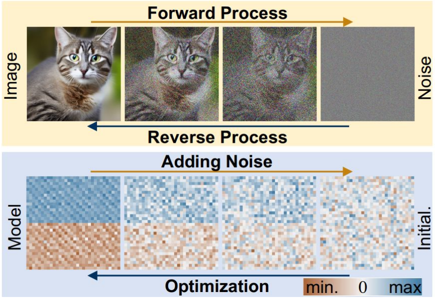
- HyperTuning: Toward Adapting Large Language Models without Back-propagation. Jason Phang et al., 2023.
- Conditional LoRA Parameter Generation. Xiaolong Jin et al., 2024.
- Learngene Tells You How to Customize: Task-Aware Parameter Initialization at Flexible Scales, Jiaze Xu et al., 2025.
- NNiT: Width-Agnostic Neural Network Generation with Structurally Aligned Weight Spaces. Jiwoo Kim et al., 2026.

3. Graph of Parameters

3.1 Neural Graphs

Neural Network Parameters vs Images





Images:

1. Resize/crop images
2. Augmentations
3. Spatial locality, compositionality
4. 0-255
5. Clear applications

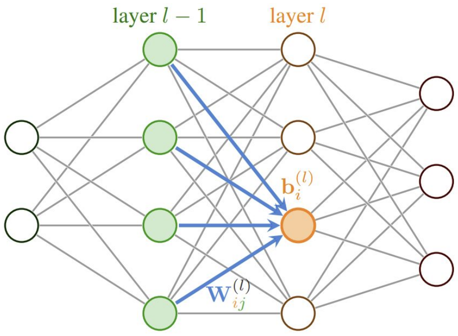
Neural networks:

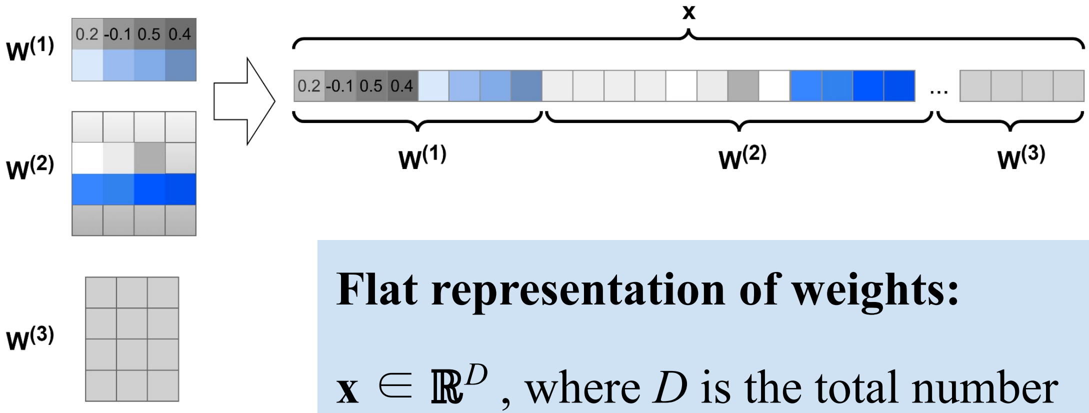
1. Different and very large sizes
2. Different symmetries
3. Lack of intuitive structure
4. Different range of values
5. Applications are early but promising

- Neural Network Diffusion. Kai Wang et al., 2024.
- Learning to Learn with Generative Models of Neural Network Checkpoints. William Peebles et al., 2022.
- Hyper-Representations & SANE. Konstantin Schürholt et al., 2021-2024.

Flat Representation

$$\mathbf{y} = \mathrm{softmax}(\mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}(\sigma(\mathbf{W}^{(1)}\mathbf{x}))))$$





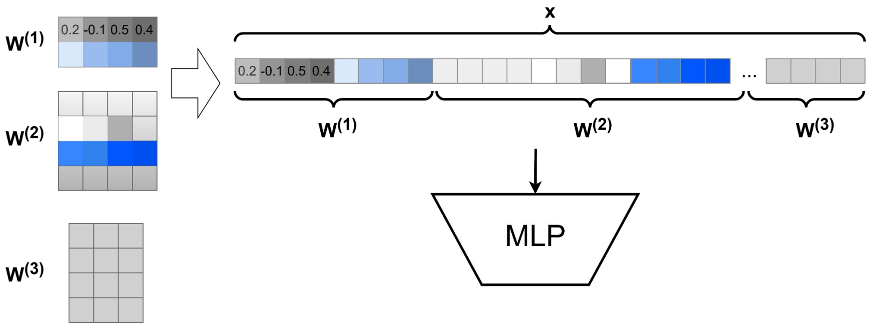
Flat representation of weights:

$\mathbf{x} \in \mathbb{R}^D$, where D is the total number of weights (parameters) in the model

- Classifying the classifier: dissecting the weight space of neural networks. Gabriel Eilertsen et al., 2020.
- Hyper-Representations as Generative Models: Sampling Unseen Neural Network Weights. Konstantin Schürholt et al., 2022.

Flat Representation

$$\mathbf{y} = \text{softmax}(\mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}(\sigma(\mathbf{W}^{(1)}\mathbf{x}))))$$



Properties:

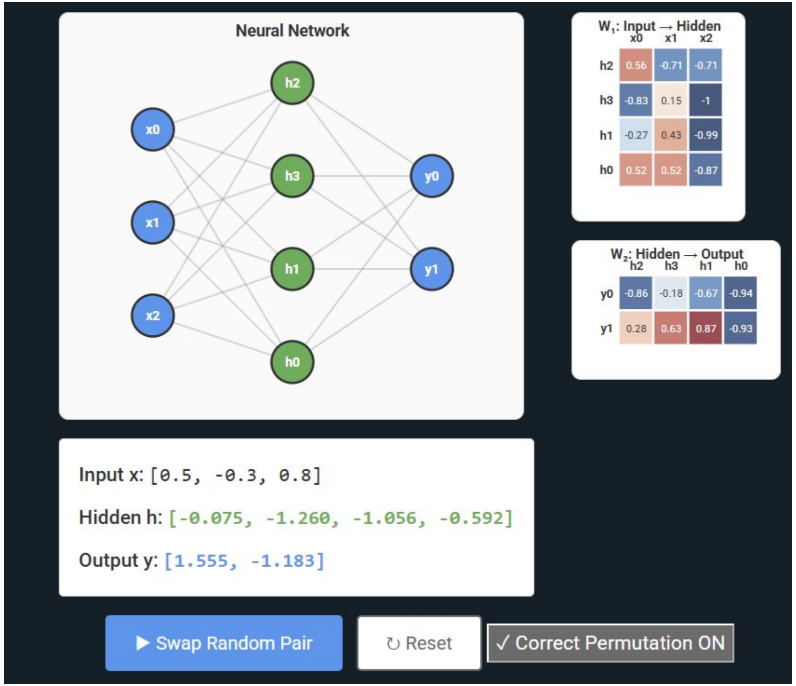
- Simple
- **x** dimensionality (D)
  - varies depending on the model
  - is as large as the model size
- Doesn’t model **permutation symmetry**

- Classifying the classifier: dissecting the weight space of neural networks. Gabriel Eilertsen et al., 2020.
- Hyper-Representations as Generative Models: Sampling Unseen Neural Network Weights. Konstantin Schürholt et al., 2022.

Permutation Symmetry

Hidden neurons **can be permuted** without affecting the overall function of the network:

$f(\mathbf{x}, \boldsymbol{\theta}) = f(\mathbf{x}, \pi(\boldsymbol{\theta}))$.

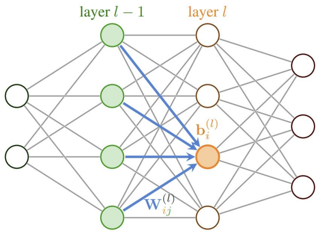


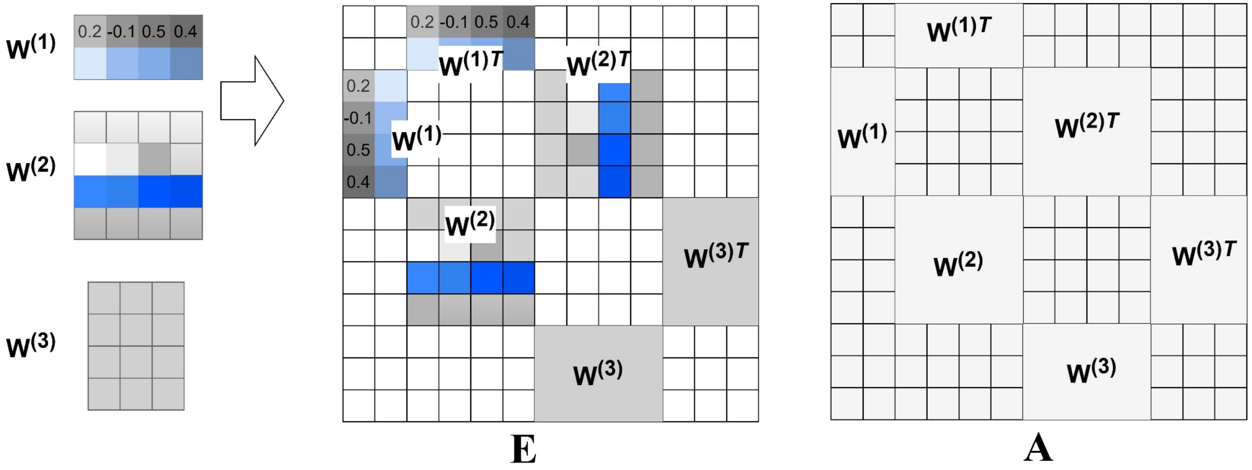
https://bknyaz.github.io/blog/2025/nino/#neuron-permutation-symmetry

On the algebraic structure of feedforward network weight spaces.
Robert Hecht-Nielsen, 1990.

Neural Graph

$$\mathbf{y} = \text{softmax}(\mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}(\sigma(\mathbf{W}^{(1)}\mathbf{x}))))$$





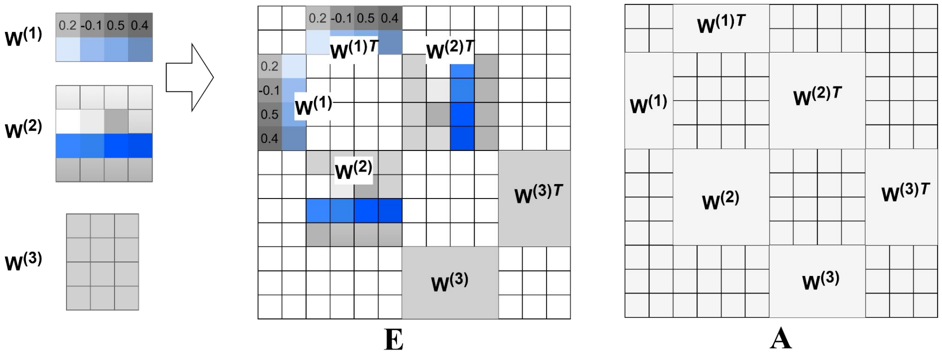
Nodes ≡ Neurons

Edges ≡ Parameters (weights, biases)

Graph Neural Networks for Learning Equivariant Representations of Neural Networks. Miltiadis Kofinas et al., 2024.

Neural Graph

$$\mathbf{y} = \text{softmax}(\mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}(\sigma(\mathbf{W}^{(1)}\mathbf{x}))))$$



Neural Graph: $G(\mathbf{E}, \mathbf{A}, \mathbf{X})$

- $\mathbf{E} \in \mathbb{R}^{n \times n \times c}$ - the weight between neurons (from weight matrices $\mathbf{W}^{(i)}$)
- $\mathbf{A} \in \{0,1\}^{n \times n}$ - indicates if neurons are connected, $\mathbf{A} \approx \text{abs}(\mathbf{E}) > 0$.
- $\mathbf{X} \in \mathbb{R}^{n \times d}$ - node/neuron features

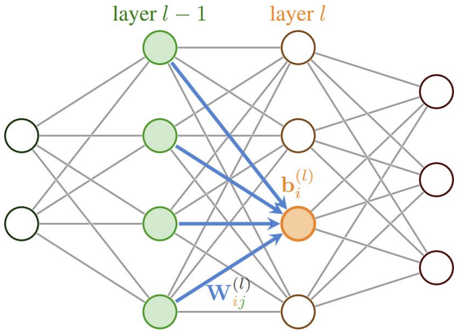
Graph Neural Networks for Learning Equivariant Representations of Neural Networks. Miltiadis Kofinas et al., 2024.

Neural Graph – Properties

$f(\mathbf{x}, \boldsymbol{\theta}) = \text{softmax}(\mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}(\sigma(\mathbf{W}^{(1)}\mathbf{x}))))$

$$f(\mathbf{x}, \boldsymbol{\theta}) = f(\mathbf{x}, \pi^{\text{good}}(\boldsymbol{\theta})) \qquad \mathcal{G}(\boldsymbol{\theta}) \simeq \mathcal{G}(\pi^{\text{good}}(\boldsymbol{\theta}))$$

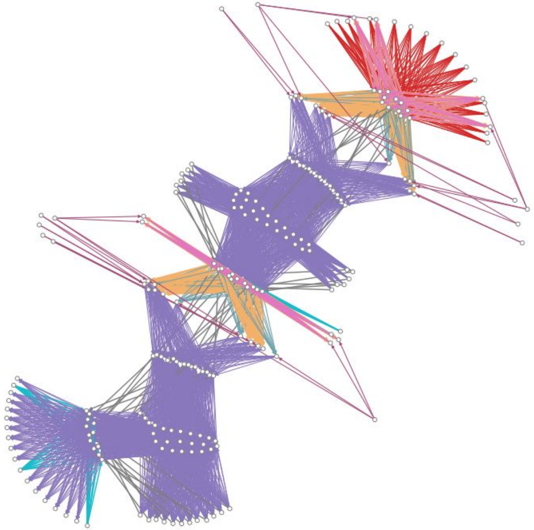
$$f(\mathbf{x}, \boldsymbol{\theta}) \neq f(\mathbf{x}, \pi^{\text{bad}}(\boldsymbol{\theta})) \qquad \mathcal{G}(\boldsymbol{\theta}) \not\simeq \mathcal{G}(\pi^{\text{bad}}(\boldsymbol{\theta}))$$

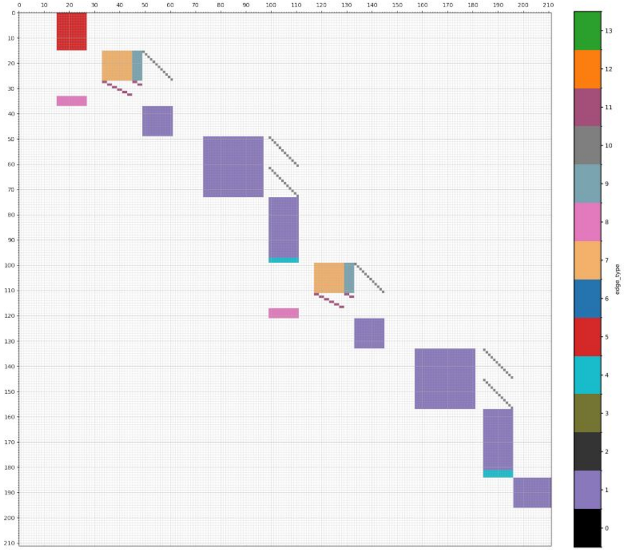


Neural graphs model **permutation symmetry**:

- Permutation (π) of nodes = permutation (π) neurons
- Graphs are isomorphic when the **neural network's function** does not change, and vice versa

Neural Graph for a Transformer model





Accelerating Training with Neuron Interaction and Nowcasting Networks.
Boris Knyazev et al., 2025.

Laplacian Positional Encoding

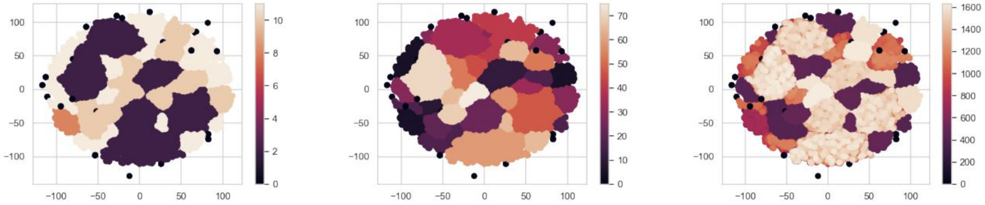
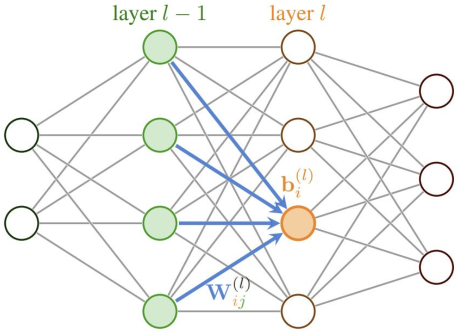


Figure 3: Laplacian Positional Encoding (LPE) of our neural graph's nodes color-coded by the layer type (left), layer index (middle) and neuron index (right) for a transformer with 6 layers and 384 hidden units. We show tSNE projections from the 8-dimensional LPE to 2d.



- Given a neural graph $G(\mathbf{E},\mathbf{A},\mathbf{X})$, node features $\mathbf{X}$ can be computed based on LPE.
- Such LPE describes each neuron in terms of its local and global structure and function.

GNNs on Neural Graphs

$$\mathbf{V}^m, \boldsymbol{\mathcal{E}}^m = \text{GNN}_{\phi_m}(\mathbf{V}^{m-1}, \boldsymbol{\mathcal{E}}^{m-1})$$

Given a neural graph $G(\mathbf{E},\mathbf{A},\mathbf{X})$ with node features $\mathbf{X}$ computed based on LPE, we can feed the graph to a GNN to update node and edge features.

Graph Neural Networks for Learning Equivariant Representations of Neural Networks. Miltiadis Kofinas et al., 2024.

GNNs on Neural Graphs

$$\mathbf{V}^m, \mathcal{E}^m = \text{GNN}_{\phi_m}(\mathbf{V}^{m-1}, \mathcal{E}^{m-1})$$

Representation learned by such GNNs:

1. Expressive
2. Equivariant/invariant to neuron permutations
3. Equivariant/invariant *to input size*

Predicting accuracy of weights (correlation)

Method	CIFAR10-GS	CIFAR10 Wild Park
NFN_{HNP} (Zhou et al., 2023a)	$0.934_{\pm 0.001}$	—
StatNN (Unterthiner et al., 2020)	$0.915_{\pm 0.002}$	$0.719_{\pm 0.010}$
NG-GNN (Ours)	$0.931_{\pm 0.002}$	$0.804_{\pm 0.009}$
NG-T (Ours)	$\mathbf{0.935}_{\pm 0.000}$	$\mathbf{0.817}_{\pm 0.007}$

Graph Neural Networks for Learning Equivariant Representations of Neural Networks. Miltiadis Kofinas et al., 2024.

Neural Graph Works

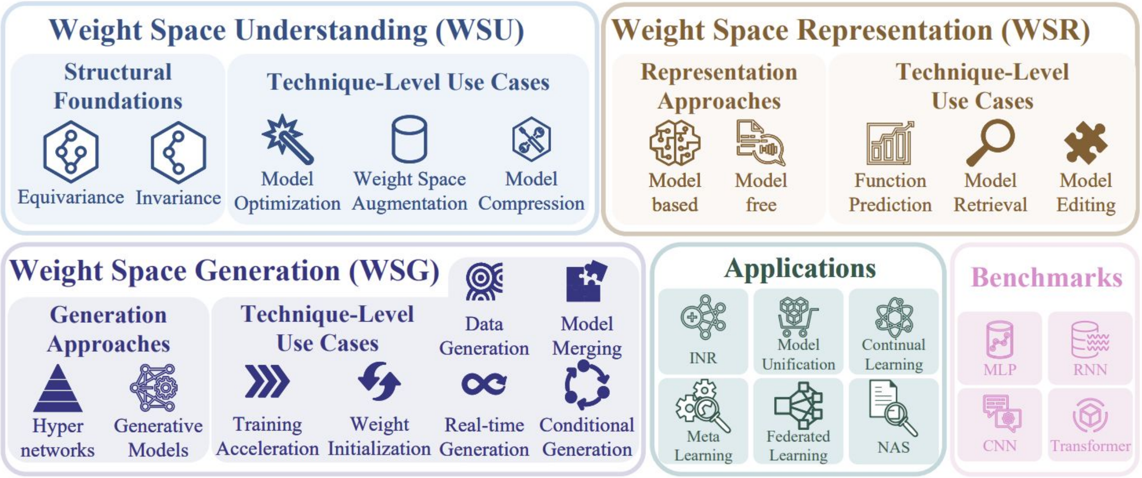
Many works use such or similar graphs

- **Accelerating Training with Neuron Interaction and Nowcasting Networks** (ICLR 2025)
  B. Knyazev, A. Moudgil, G. Lajoie, E. Belilovsky, S. Lacoste-Julien
- **Graph Neural Networks for Learning Equivariant Representations of Neural Networks** (ICLR 2024)
  M. Kofinas, B. Knyazev, Y. Zhang, Y. Chen, C.J. Burghouts, E. Gavves, C.G.M. Snoek, D.W. Zhang
- **Universal Neural Functionals** (arXiv 2024)
  A. Zhou, C. Finn, J. Harrison
- **Graph Metanetworks for Processing Diverse Neural Architectures** (ICLR 2024)
  D. Lim, H. Maron, M.T. Law, J. Lorraine, J. Lucas.
- **Scale Equivariant Graph Metanetworks** (NeurIPS 2024)
  I. Kalogeropoulos, G. Bouritsas, Y. Panagakis
- **Improved Generalization of Weight Space Networks via Augmentations** (ICML 2024)
  A. Shamsian, A. Navon, D.W. Zhang, Y. Zhang, E. Fetaya, G. Chechik, H. Maron
- **Equivariant Architectures for Learning in Deep Weight Spaces** (ICML 2023)
  A. Navon, A. Shamsian, I. Achituve, E. Fetaya, G. Chechik, H. Maron
- **Deep Neural Network Fusion via Graph Matching with Applications to Model Ensemble and Federated Learning** (ICML 2020)
  C. Liu, C. Lou, R. Wang, A.Y. Xi, L. Shen, J. Yan
- **Graph Structure of Neural Networks** (ICML 2020)
  J. You, J. Leskovec, K. He, S. Xie

Neural graphs + GNNs:

1. Different and very large sizes ✔
2. Different symmetries ✔
3. **Applications – optimization** ✔

A Survey of Weight Space Learning



https://arxiv.org/abs/2603.10090

3.2 NiNo

Neuron Interaction and Nowcasting Networks

Accelerating Training with Neuron Interaction and Nowcasting Networks

 # Adam
θ - model parameters
for t in range(10^6):
    $\nabla\theta_t$ *- compute grads based on some data*
    θ_{t+1} = *adam.step(*θ_t, $\nabla\theta_t$, *...)*

- Slow convergence

Accelerating Training with Neuron Interaction and Nowcasting Networks

```
# Adam
θ - model parameters
for t in range(10^6):
    ∇θ_t  - compute grads based on some data
    θ_{t+1} = adam.step(θ_t , ∇θ_t , ...)
```

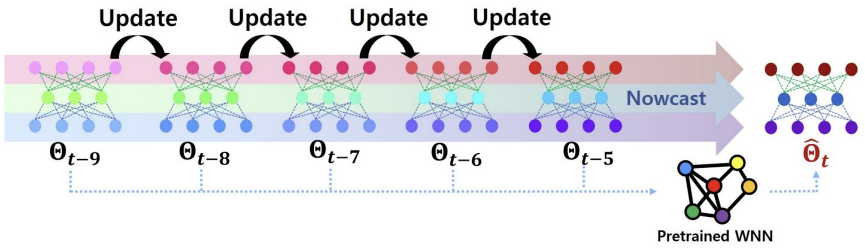
- Slow convergence

→

```
# Periodically predict future parameters
θ - model parameters
f - NiNo model
for t in range(10^6):
    If t % 1000 == 0:
        θ_{t+K} = f.predict(θ_t , θ_{t-1} , ...)
    else:
        ∇θ_t  - compute grads based on some data
        θ_{t+1} = adam.step(θ_t , ∇θ_t , ...)
```

- Faster convergence
- Small overhead

Background: Weight Nowcaster Networks



Learning to boost training by periodic nowcasting near future weights.
Jinhyeok Jang et al., 2023.

Training and evaluation pipeline:

1. Collect many checkpoints D={θ} in some tasks
2. Train WNN on D $\text{argmin}_{\phi}||\Delta\tilde{\theta}_{\tau+k} - \Delta\hat{\theta}_{\tau+k}||_1$
3. Use the trained WNN on any new task

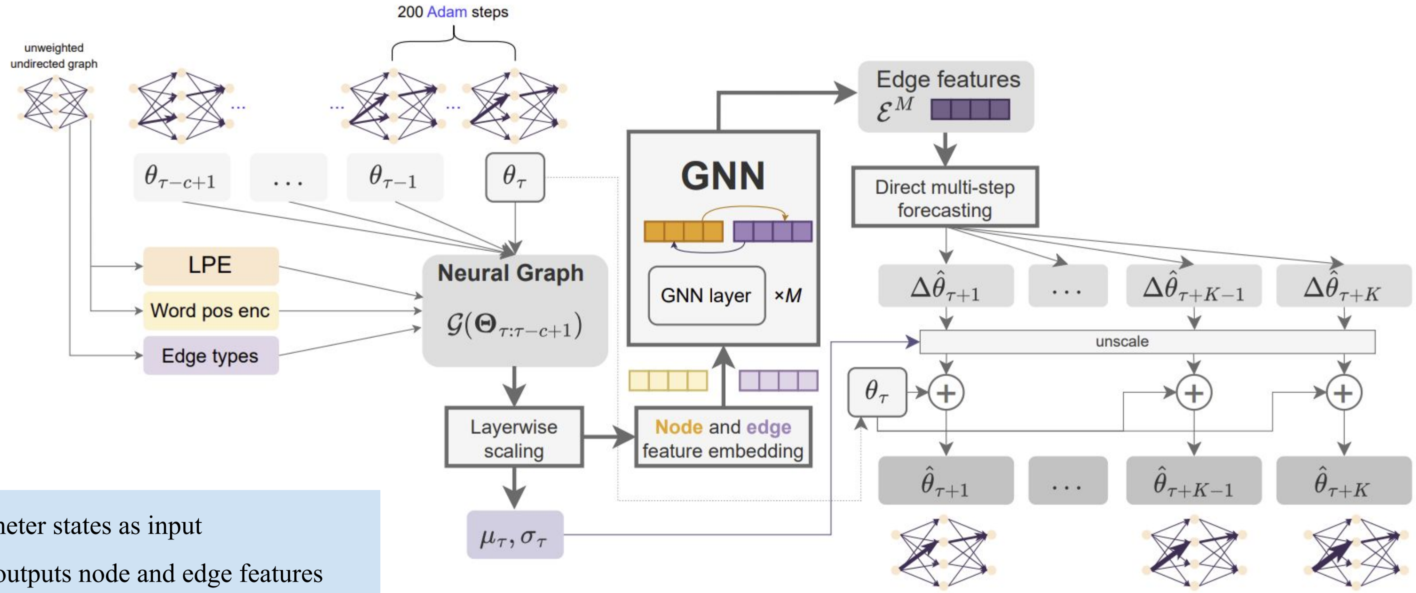
```
# Periodically predict future parameters
θ - model parameters
f - WNN model
for t in range(10^6):
    If t % 1000 == 0:
        θ_{t+K} = f.predict(θ_t , θ_{t-1} , ...)
    else:
        ∇θ_t - compute grads based on some data
        θ_{t+1} = adam.step(θ_t , ∇θ_t , ...)
```

From WNN to NiNo

```
# WNN
θ - model parameters
f - WNN model
for t in range(10^6):
    If t % 1000 == 0:
        θ_{t+K, i} = f.predict(θ_{t, i} , θ_{t-1, i} , …)  # ∀ i
    else:
        ∇θ_t  - compute grads based on some data
        θ_{t+1, i} = adam.step(θ_{t, i} , ∇θ_{t, i} , …)  # ∀ i
```

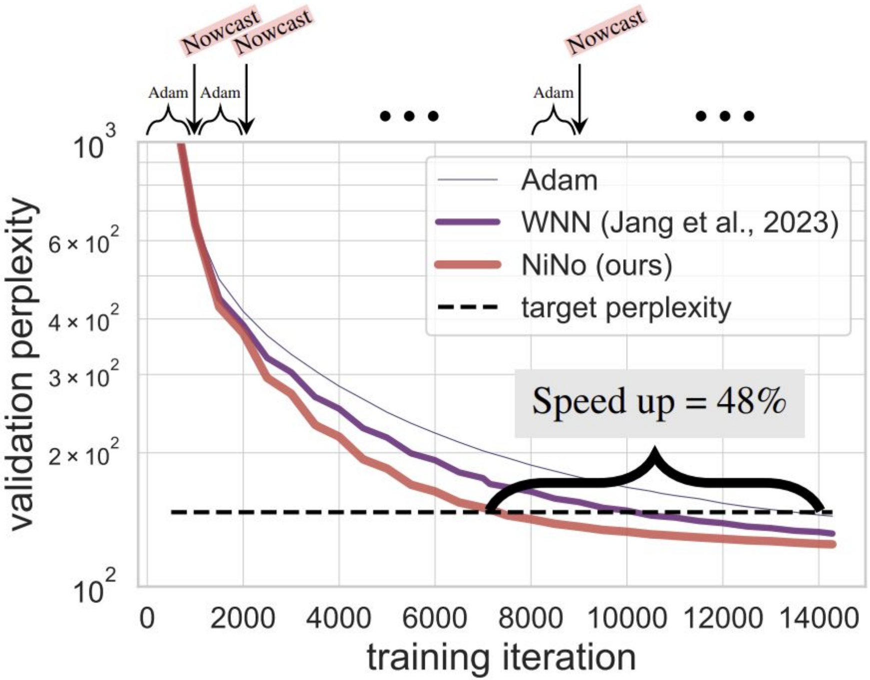
```
# NiNo
θ - model parameters
f - NiNo model
for t in range(10^6):
    If t % 1000 == 0:
        θ_{t+K} = f.predict(θ_t , θ_{t-1} , …; G)  # jointly for all params
    else:
        ∇θ_t  - compute grads based on some data
        θ_{t+1, i} = adam.step(θ_{t, i} , ∇θ_{t, i} , …)  # ∀ i
```

NiNo



- Take 5 past parameter states as input
- GNN inputs and outputs node and edge features
- Predict parameter deltas at multiple future horizons

Key Example



Related literature on learning to optimize:

- VeLO: Training Versatile Learned Optimizers by Scaling Up. Luke. Metz et al., 2022.
- µLO: Compute-Efficient Meta-Generalization of Learned Optimizers. Benjamin Thérien et al., 2024.
- Learning Versatile Optimizers on a Compute Diet. Abhinav Moudgil et al., 2025.

Training and evaluation pipeline

- Collect many checkpoints D={θ} in some tasks (in-distribution tasks)
- Train NiNo on D
- Use the trained NiNo on any new task task

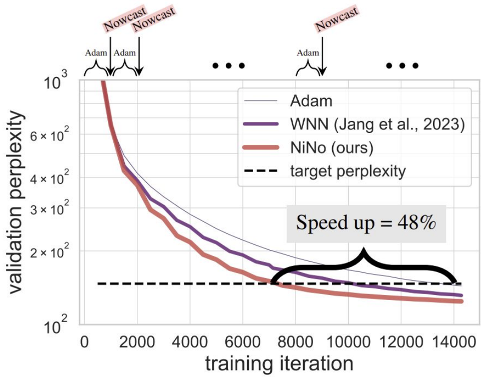


Table 1: **In-distribution and out-of-distribution tasks.**

	In-distribution tasks				Out-of-distribution tasks				
	FM/16	C10/16	LM1B/3-24	LM1B/2-32	FM/32	C10/32	C100/32	LM1B/3-64	Wiki/3-64
Final training loss	0.25±0.06	0.91±0.1	5.94±0.03	5.85±0.04	–	–	–	–	–
#models	300	300	200	200	–	–	–	–	–
#params	14K	15K	1.2M	1.6M	56K	57K	63K	3.3M	3.3M
Target (validation) metric	Acc	Acc	Perplexity	Perplexity	Acc	Acc	Acc	Perplexity	Perplexity
Target value	89.5%	66.0%	352	319	90.5%	72.5%	39%	181	147
Adam #steps	8606	8732	23000	23500	8269	8607	8341	23500	13500
NiNo #steps	4582	3775	11500	12000	4395	4323	4646	12000	7000

- Speedup = (13500-7000)/13500=48%
- Scaled to 100M models

3. Graph of Parameters – Summary

1. Neural Graphs (NGs) *vs* Flat Representation
2. Permutation Symmetry
3. NGs of Modern Architectures
4. LPE of NGs
5. GNNs on NGs
6. Using GNNs and NGs to accelerate optimization

4. Future research

Accelerating Training?

- Many open-source models are already available
- GPUs are getting more powerful
- Optimization methods like Adam are good enough

Accelerating Training?

- Many open-source models are already available

  License restrictions, harmful bias, backdoor attacks

- GPUs are getting more powerful

  high intra/inter company competition and price

- Optimization methods like Adam are good enough

  1000 GPUs × 90 days × 24 h × $1.60/h = $3.5mln

Accelerating Training?

- Many open-source models are already available

License restrictions, harmful bias, backdoor attacks

- GPUs are getting more powerful

high intra/inter company competition and price

- Optimization methods like Adam are good enough

1000 GPUs × 90 days × 24 h × $1.60/h = $3.5mln

- Optimization remains the biggest cost in many pipelines

- It is hard to justify a new optimization algorithm if it speeds up by only 10% **because of engineering overhead and unexpected outcomes**

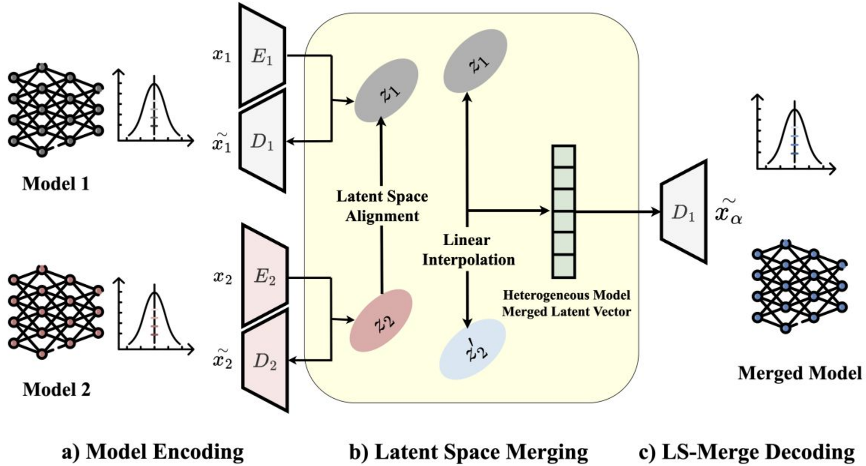
Accelerating Training?

1. Yes, it is still a key problem, both in research and industry
2. In industry, big consistent speedups are expected
3. For neural graph-based methods:
   a. Can they generalize better to large models?
   b. Can they generalize better to novel neural network architectures?
   c. Can they be more efficient (GNNs on large graphs are expensive)?

Model Merging

Model merging methods combine the weights of multiple neural networks into one: $W_m = W_1 + W_2$

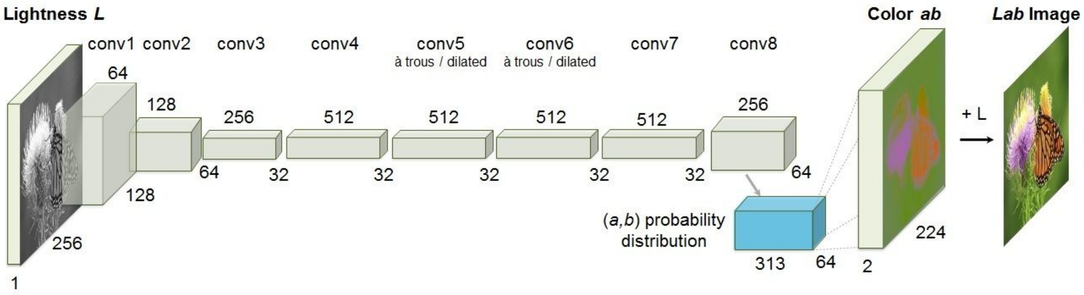
- Simple averaging and other methods operate in the weight space directly
- What about the latent space of model weights?



LS-Merge: Merging Language Models in Latent Space. Bedionita Soro et al., 2026.

Model Editing

- Image restoration (e.g., colorization) or transformation (e.g., style transfer) using neural networks works very well
- Neural network editing is also possible, but not practical yet



Colorful Image Colorization. Richard Zhang et al., 2016.

Interpreting the Weight Space of Customized Diffusion Models. Amil Dravid et al., 2024.

More fundamental research problems

1. Generative *vs* deterministic models
2. Generalization (new architectures, new tasks)
3. Scalability *vs* equivariance
4. Representing weights *vs* representing functions
5. Compressing models

Additional References:

- Diffusion-based Neural Network Weights Generation. Bedionita Soro et al., 2025.
- NNiT: Width-Agnostic Neural Network Generation with Structurally Aligned Weight Spaces. Jiwoo Kim et al., 2026.
- On the Expressive Power of Permutation-Equivariant Weight-Space Networks. Adir Dayan et al., 2026.
- Learning on model weights using tree experts. Eliahu Horwitz et al., 2025.
- SVD-LLM V2: Optimizing Singular Value Truncation for Large Language Model Compression. Xin Wang et al., 2025.

Thank you

Email: boris.knyazev@umontreal.ca

Web: https://bknyaz.github.io/