EEG-GNN: Graph Neural Networks for Classification of Electroencephalogram (EEG) Signals

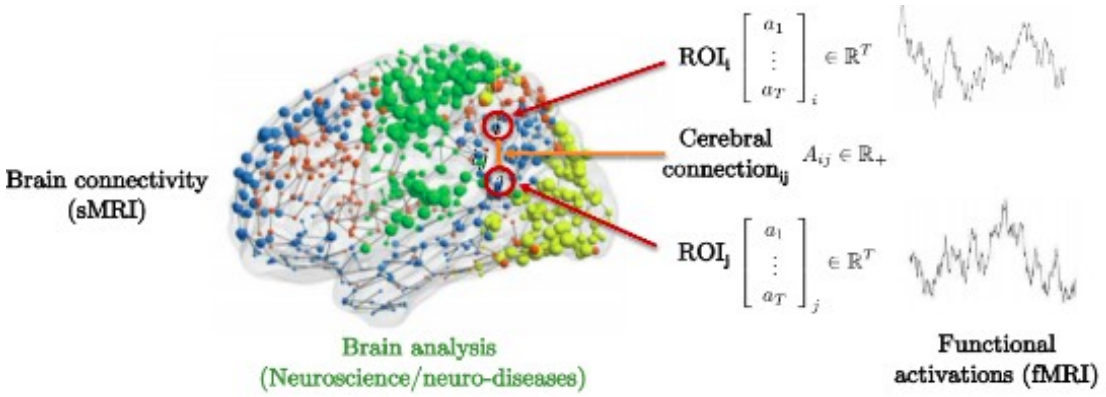
ANDAC DEMIR[1], TOSHIAKI KOIKE-AKINO[2], YE WANG[2], MASAKI HARUNA[3], DENIZ ERDOGMUS[1]

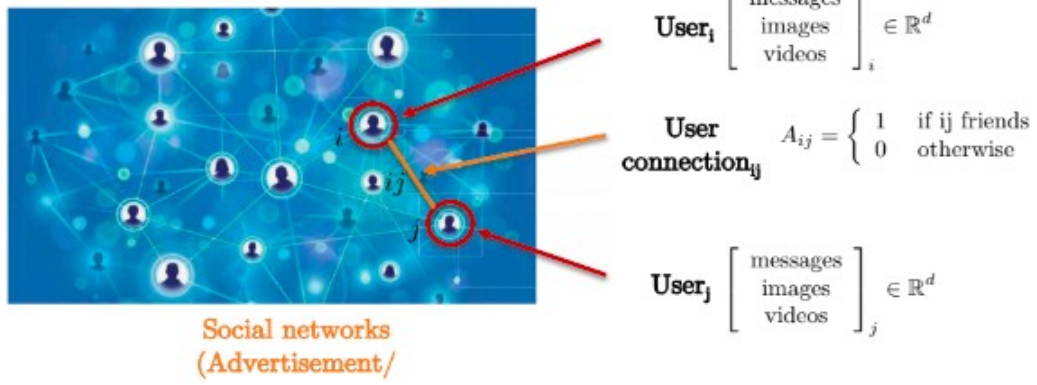
1 COGNITIVE SYSTEMS LABORATORY, ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT, NORTHEASTERN UNIVERSITY, BOSTON, MA 02115, USA

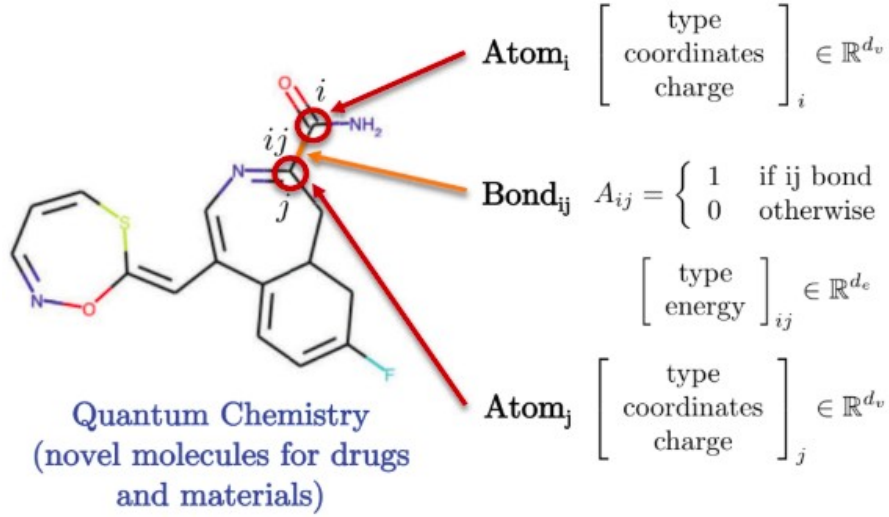
2 MITSUBISHI ELECTRIC RESEARCH LABORATORIES (MERL), CAMBRIDGE, MA 02139, USA

3 ADVANCED TECHNOLOGY R&D CENTER, MITSUBISHI ELECTRIC CORPORATION (MELCO), AMAGASAKI, HYOGO, JAPAN

Graph Neural Networks



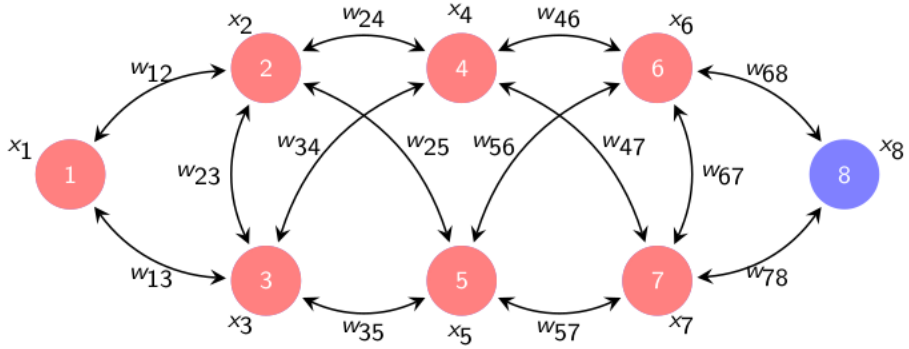




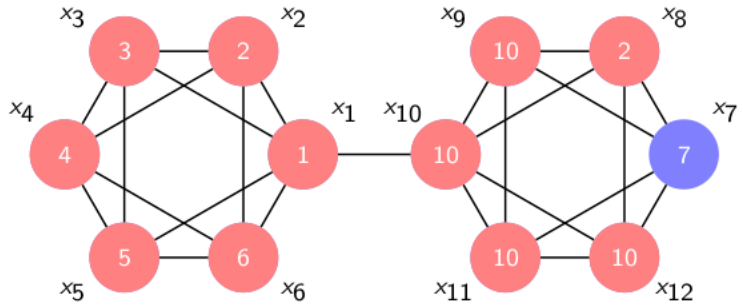
Convolution on Graphs

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs,
- Graph convolutions share the locality of conventional convolutions.

A signal supported on a graph



Another signal supported on another graph



Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0\,\mathbf{S}^0\mathbf{x} + h_1\,\mathbf{S}^1\mathbf{x} + h_2\,\mathbf{S}^2\mathbf{x} + h_3\,\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\,\mathbf{S}^k\mathbf{x}$

Graph Shift Operator

Normalized adjacency and Laplacian matrices express weights relative to the nodes' degrees

Normalized adjacency matrix $\Rightarrow \bar{\mathbf{A}} := \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. It is symmetric if the graph is symmetric

Normalized Laplacian matrix $\Rightarrow \bar{\mathbf{L}} := \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. It is symmetric if the graph is symmetric

Given definitions of Normalized adjacency and Laplacian $\Rightarrow \bar{\mathbf{L}} := \mathbf{I} - \bar{\mathbf{A}}$

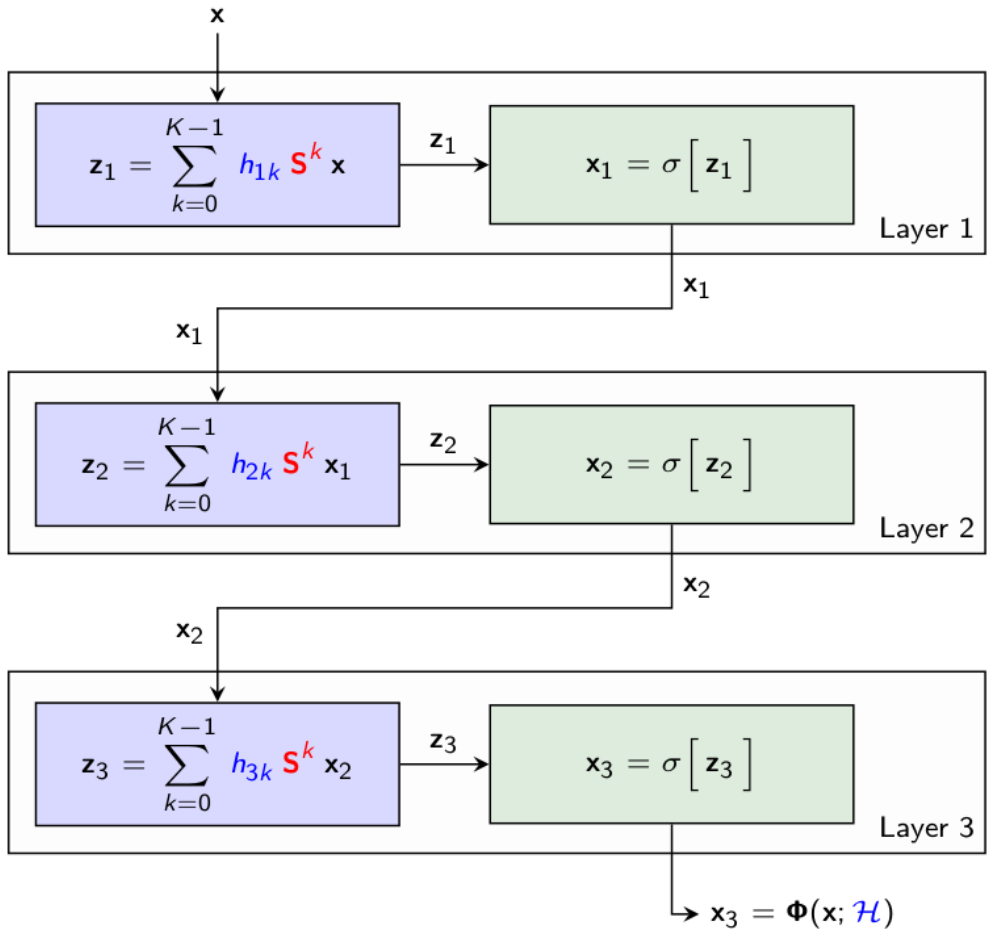
The Graph Shift Operator $\mathbf{S}$ is a stand in for any of the matrix representations of the graph

Adjacency Matrix	Laplacian Matrix	Normalized Adjacency	Normalized Laplacian
$\mathbf{S} = \mathbf{A}$	$\mathbf{S} = \mathbf{L}$	$\mathbf{S} = \bar{\mathbf{A}}$	$\mathbf{S} = \bar{\mathbf{L}}$

Convolution on Graphs

- Convolutions are polynomials on the adjacency matrix of a line graph.

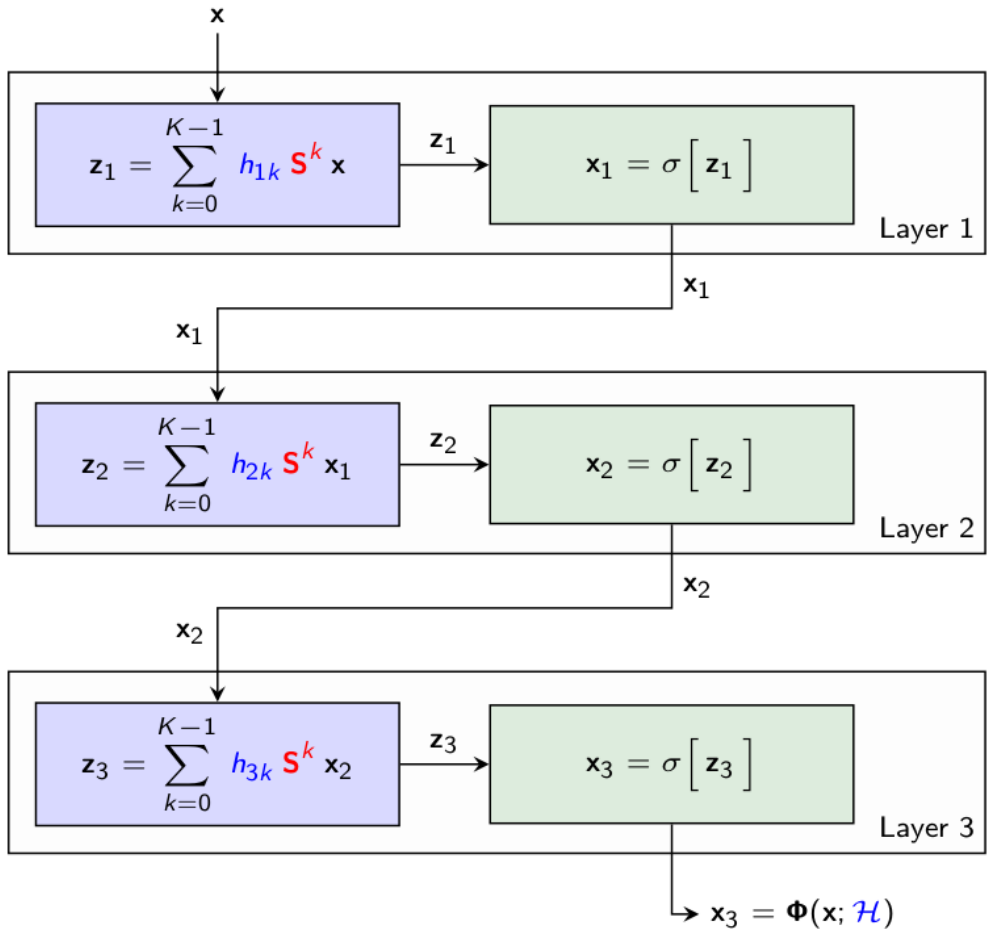
- GNN's are just another way of writing CNNs



Convolution on Graphs

- The graph can be any arbitrary graph.

- The polynomial on the graph shift operator S becomes a graph convolutional filter.

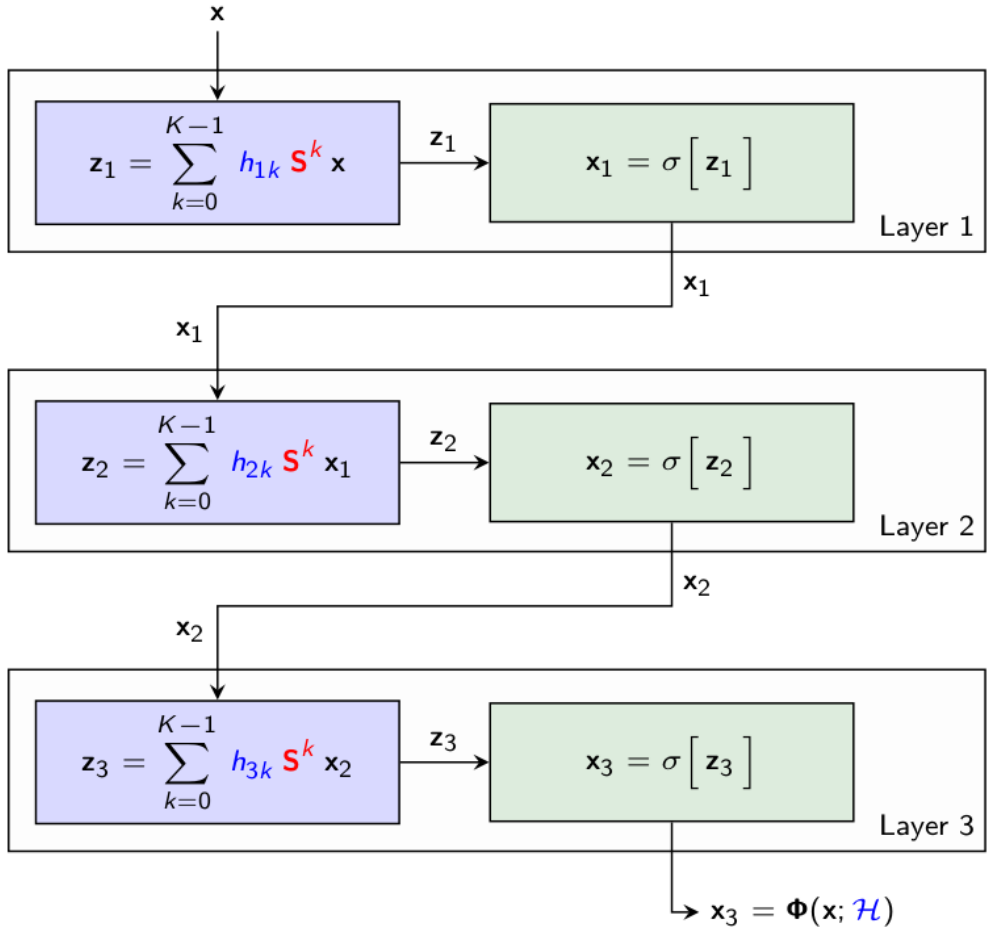


Convolution on Graphs

- A graph NN composes a cascade of layers.

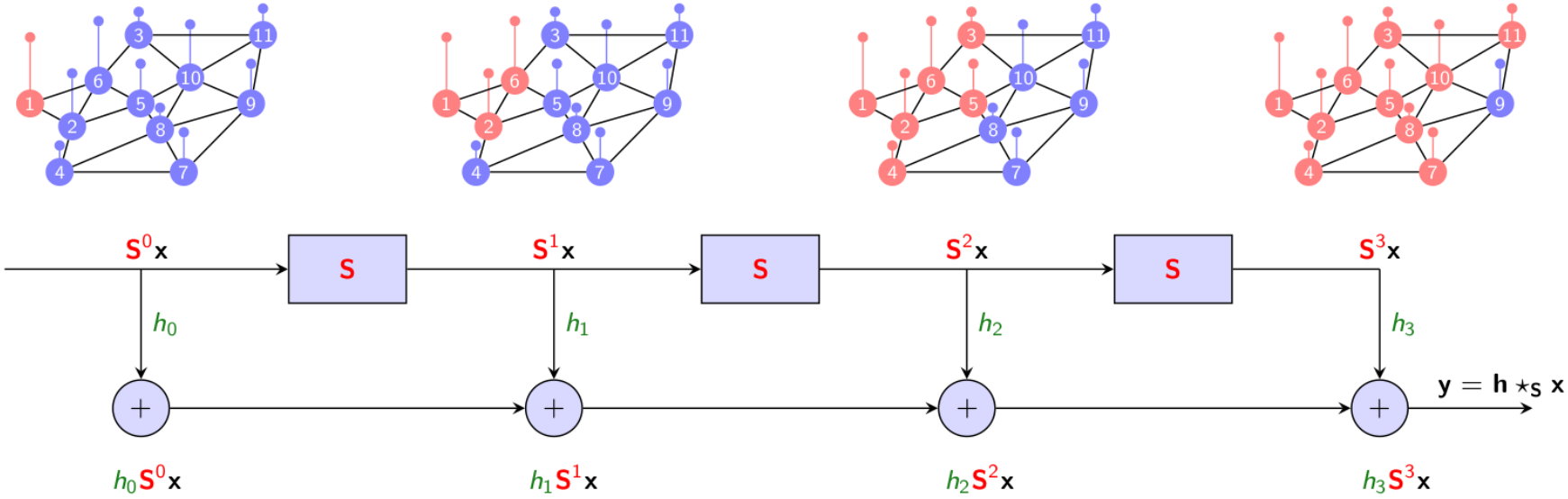
- Each of which are themselves compositions of graph convolutions with pointwise nonlinearities.

- Recovers a CNN if S describes a line graph.



Convolution on Graphs

If we let $\mathbf{S}$ be the shift operator of an arbitrary graph we recover the graph convolution



Graph Neural Networks

We increase expressivity further with a GNN

Layer a few graph perceptrons (3 in the figure)

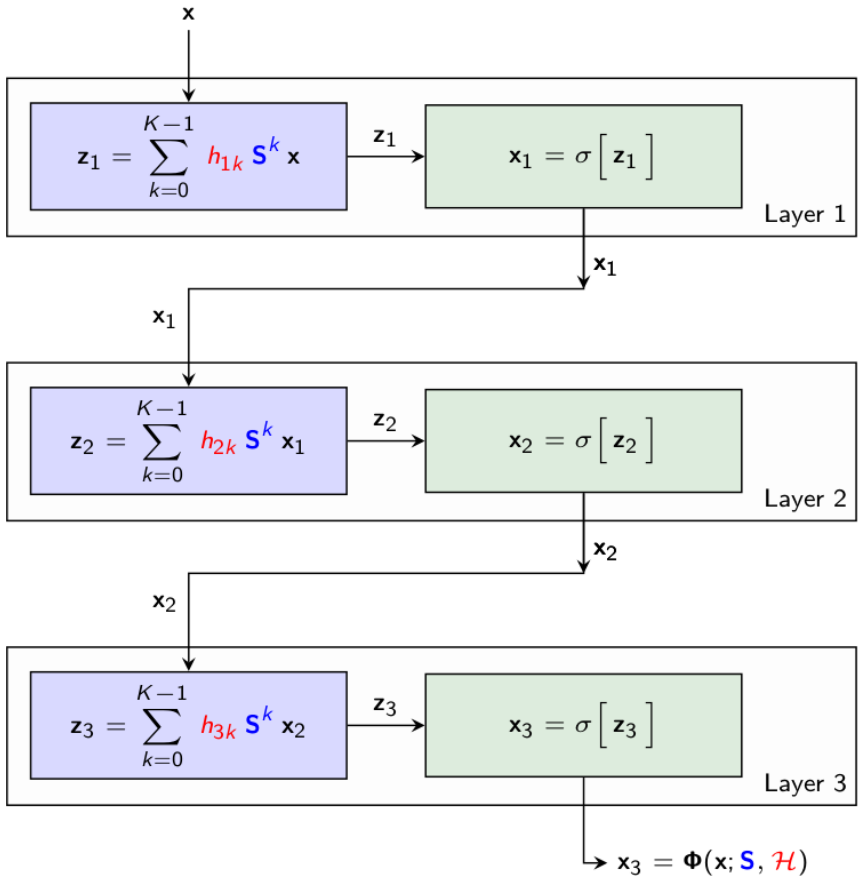
$\Rightarrow$ Feed the input signal $\mathbf{x}$ to Layer 1

$\Rightarrow$ Connect output of Layer 1 to input of Layer 2

$\Rightarrow$ And output of Layer 2 to input of Layer 3

Last layer output is the GNN output $\Rightarrow \mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H})$

$\Rightarrow$ Parametrized on filter tensor $\mathcal{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$



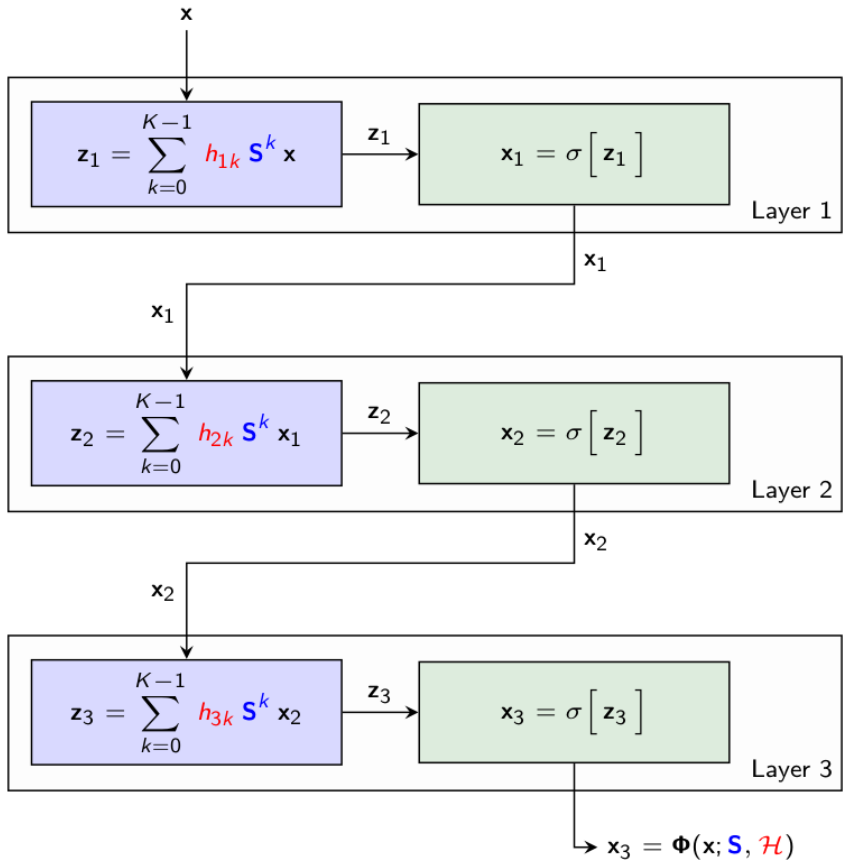
Graph Neural Networks

Learn Optimal GNN tensor $\mathcal{H}^* = (\mathbf{h}_1^*, \mathbf{h}_2^*, \mathbf{h}_3^*)$ as

$$\mathcal{H}^* = \underset{\mathcal{H}}{\text{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H}),\mathbf{y}\Big)$$

Optimization is over tensor only. Graph $\mathbf{S}$ is given

$\Rightarrow$ Prior information given to the GNN



Convolution on Graphs with Multiple Features

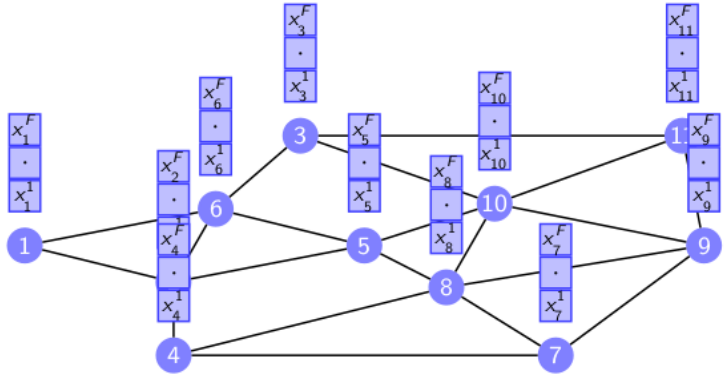
Graph signal $\Rightarrow$ Single scalar associated to each node $\Rightarrow$ $\mathbf{x}$: nodes $\rightarrow \mathbb{R}$

Extend descriptive power $\Rightarrow$ Assign a vector to each node
$\Rightarrow$ $\mathbf{X}$: nodes $\rightarrow \mathbb{R}^2$ $\Rightarrow$ $\mathbf{X}$: nodes $\rightarrow \mathbb{R}^3$ $\Rightarrow$ $\mathbf{X}$: nodes $\rightarrow \mathbb{R}^F$

Multiple feature graph signal $\Rightarrow$ Matrix $\mathbf{X}$ of size N (nodes) $\times$ F (features)

$\Rightarrow$ Row i collects all features at node i $\Rightarrow$ Local information at node i

$\Rightarrow$ Column f represents feature f at all nodes $\Rightarrow$ $\mathbf{x}^f$ is a graph signal



$$\mathbf{X} = \begin{bmatrix} x_1^1 & x_1^2 & x_1^3 & \cdots & x_1^F \\ x_2^1 & x_2^2 & x_2^3 & \cdots & x_2^F \\ x_3^1 & x_3^2 & x_3^3 & \cdots & x_3^F \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{10}^1 & x_{10}^2 & x_{10}^3 & \cdots & x_{10}^F \\ x_{11}^1 & x_{11}^2 & x_{11}^3 & \cdots & x_{11}^F \end{bmatrix}$$

Convolution on Graphs with Multiple Features

Convolution $\Rightarrow$ Linear operation, local information, distributed implementation

$$\mathbf{Y} = \sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X} \mathbf{H}_k$$

Multiplication by $\mathbf{S}$ on the left $\Rightarrow$ Shifts each graph signal feature $\mathbf{S}\mathbf{x}^f$

Multiplication by $\mathbf{H}$ on the right $\Rightarrow$ Linear combination of features at each node (No exchanges)

The convolution is now equivalent to the application of a bank of graph filters

For each feature $\mathbf{x}^f$ we apply the (f, g) filter to obtain $\mathbf{x}^{fg}$ $\Rightarrow$ Linear, local, distributed

$$\mathbf{x}^{fg} = \sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{x}^f h_k^{fg}$$

$\Rightarrow$ There are G filters applied to each input $\mathbf{x}^f$ $\Rightarrow$ There a total of $F \times G$ filters (size of $\mathbf{H}_k$)

Aggregate the output of each gth filter across all F features $\Rightarrow \mathbf{y}^g = \sum_{f=1}^{F} \mathbf{x}^{fg}$ $\Rightarrow \mathbf{Y} = [\mathbf{y}^1, \ldots, \mathbf{y}^G]$

Graph Neural Networks – Model Hyperparameters

Multi feature graph neural networks $\Rightarrow \boldsymbol{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}) = \mathbf{X}_L$

$$\mathbf{X}_\ell = \sigma\Big[\sum_{k=0}^{K_\ell - 1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k} \Big]$$

$\mathbf{X}_\ell$ graph signal at layer ℓ $\Rightarrow$ Size $N \times F_\ell$

$\mathbf{H}_{\ell k}$ kth filter tap at layer ℓ $\Rightarrow$ Size $F_{\ell-1} \times F_\ell$ $\Rightarrow$ Learn $\sum_\ell K_\ell F_\ell F_{\ell-1}$ filter taps $\mathcal{H} = \{\mathbf{H}_{\ell k}\}$

$\mathbf{S}$ shift operator (given by the problem) $\Rightarrow$ Size $N \times N$

Hyperparameters (design choices) $\Rightarrow$ Affect the number of learnable parameters

$\Rightarrow$ L: number of layers

$\Rightarrow$ K_ℓ: number of filter taps on each layer

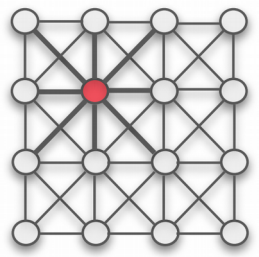
$\Rightarrow$ F_ℓ: number of features on each layer

EEG-GNN

Feeding EEG data to a CNN typically uses two methodologies:

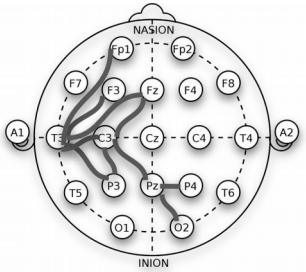
1. Applying 2D convolutions to each EEG trial, which is presented a pseudo-image $\mathbb{R}^{C \times T}$, where C denotes number of EEG channels, and T denotes number of discretized time samples, which effectively treats the EEG channels and time samples like spatial dimensions for CNN processing.

2. Applying 1D convolutions along only the time axis of the EEG trial, while treating the EEG channels as separate channels of the CNN processing.



(i)

2D CNN convolution

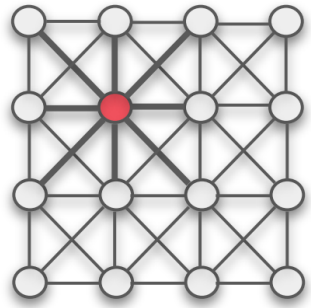


(ii)

Arbitrary functional neural connectivity between electrode sites

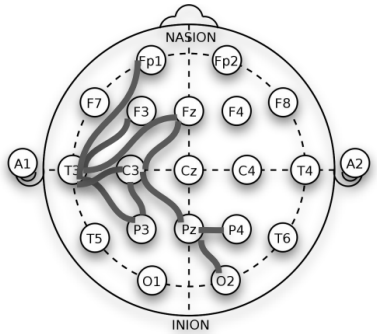
Contributions

- EEG-GNN properly maps the network of the brain as a graph, where each electrode used to collect EEG data according to intl. 10-5 system represents a node in the graph and time samples acquired from an electrode corresponds to that node's feature vector.
- Adjacency matrix of this graph can be constructed flexibly, e.g., i) every pair of nodes is connected by an unweighted edge, ii) every pair of nodes is connected by an edge weighted by the functional neural connectivity factor, which is the Pearson correlation coefficient between the feature vectors of the two nodes, iii) a sparse adjacency matrix can be designed under the constraint only nodes that are closer than a heuristic distance are connected, or iv) a sparse adjacency matrix can be constructed via k-nearest neighbors (k-NNG).
- EEG-GNN can learn and visualize the connectivity between salient nodes, which addresses a critical issue of neuroscientific interpretability.



(i)

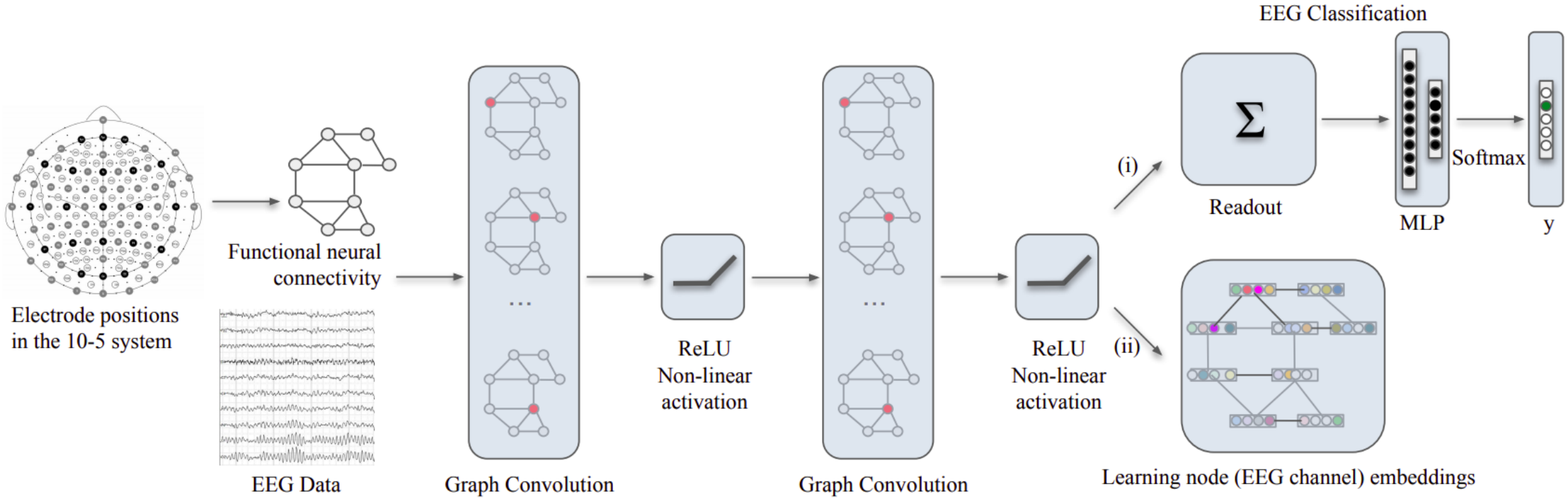
2D CNN convolution



(ii)

Arbitrary functional neural connectivity between electrode sites

EEG-GNN


EEG Classification
Electrode positions
in the 10-5 system
Functional neural
connectivity
EEG Data
Graph Convolution
ReLU
Non-linear
activation
Graph Convolution
ReLU
Non-linear
activation
(i)
(ii)
Readout
MLP
Softmax
y
Learning node (EEG channel) embeddings

GraphSage

Algorithm 1 GraphSAGE forward propagation - learning graph embedding, h_G.

$h_v^0 \leftarrow \mathbf{X}_v, \forall v \in V$ Initialize a representation vector for each node
for $k = 1 \ldots K$ **do**
  for $v \in V$ **do**
    $h_{N(v)}^k \leftarrow \texttt{AGGREGATE}_k(\{h_u^{k-1},\ \forall u \in N(v)\})$
    $h_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \texttt{CONCAT}(h_v^{k-1},\ h_{N(v)}^k)\right)$
  $h_v^k \leftarrow h_v^k / \left\|h_v^k\right\|_2$
$h_G \leftarrow \texttt{READOUT}(\{h_v^K,\ \forall v \in V\})$

$$\texttt{AGGREGATE}_k = \texttt{mean}(\{\sigma(\mathbf{W}_{\text{pool}} h_{u_i}^k + b),\ \forall u_i \in N(v)\})$$

Graph Isomorphism Network (GIN)

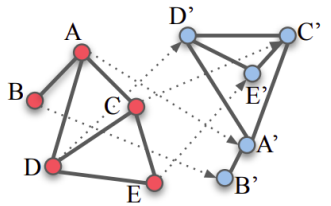


Fig. 3: An injective surjective mapping (bijection) between 2 topologically identical graphs. Adjacencies are preserved.

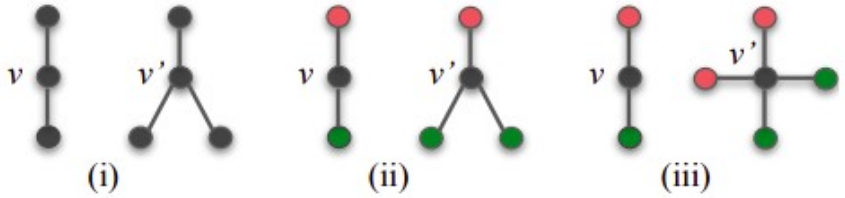
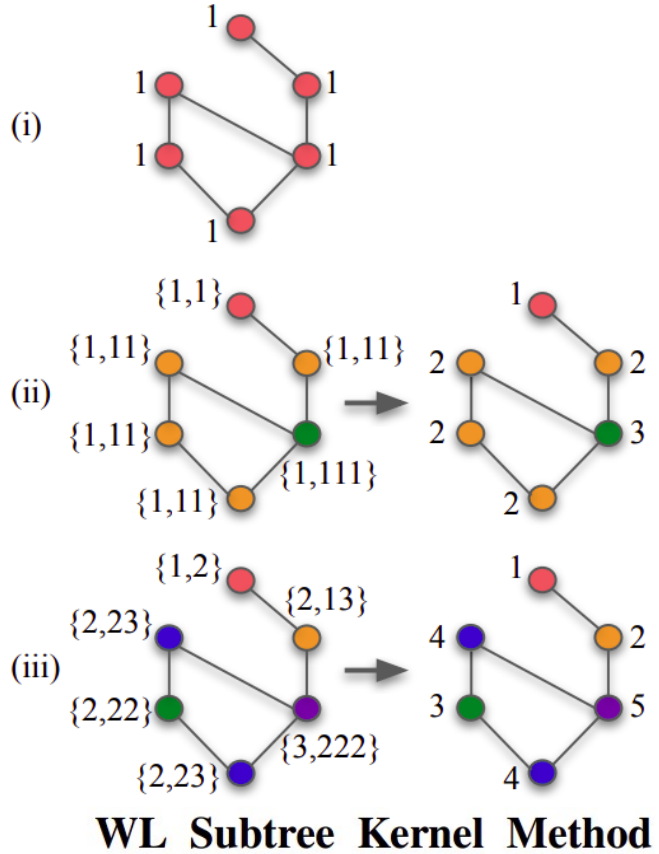


Fig. 5: (i) v and v' would have the same node embedding, if AGGREGATE function in (i) is mean or max, in (ii) is max, and in (iii) is mean or max.

$$h_v^k \leftarrow \texttt{MLP}^k\left((1+\lambda^k)\cdot h_v^{k-1} + \sum_u h_u^{k-1}, \forall u \in N(v)\right)$$

$$h_G \leftarrow \texttt{CONCAT}\left(\texttt{READOUT}(\{h_v^k,\ \forall v \in V\}) \mid k = 0,1\ldots K\right)$$



SortPool

- Sorting vertices based on their structural importance established by WL isomorphism test.
- After K iterations of graph convolution, SortPool layer gets an input of size $|V| \times K \cdot |h_v|$, where $|V|$ denotes the number of nodes, and $|h_v|$ denotes the size of a node's embedding vector.
- The output of SortPool layer has size $\rho \times K \cdot |h_v|$, where ρ is a heuristically selected parameter, and $\rho < |V|$.
- Sorted output is reshaped as a row vector of size $\rho \cdot K \cdot |h_v| \times 1$ and applied 1D convolutional layers with kernel size $K \cdot |h_v|$ followed by MaxPool.

EdgePool

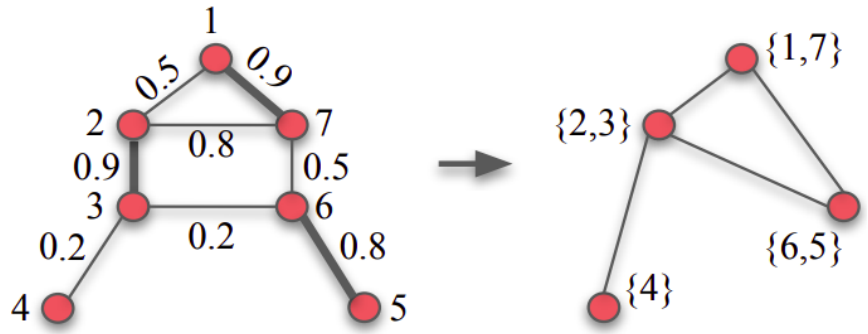
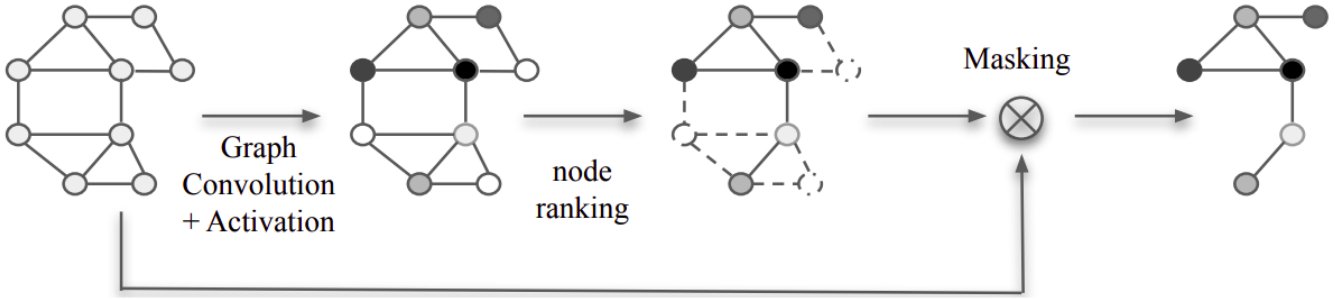


Fig. 6: After EdgePool operator, number of nodes is down-sampled by a fixed ratio of 2.

Let $e = \{v_i, v_j\}$ is the edge contracting 2 nodes, then $h_{v_{ij}} = s_{ij}(h_{v_i} + h_{v_j})$

- Softmax function is applied to all edge weights to compute edge scores, denoted as s_{ij}.
- According to these scores, edges are iteratively contracted unless nodes have already been part of a contracted edge.
- Edges between contracted nodes are preserved.
- If there is an isolated node after edge contraction, edges between the isolated node and contracted nodes are reconstructed.
- In the process of edge contraction, node features are combined, and then multiplied by the edge score.

SagPool



$\mathbf{W}_{att} \in \mathbb{R}^{F \times 1}$ denotes the learnable pooling parameters.

$Z = \sigma(\mathbf{S}\mathbf{X}\mathbf{W}_{att})$ attention scores, $Z \in \mathbb{R}^{|V| \times 1}$

$$\text{idx} = \text{top-rank}\left(Z, \lceil \rho |V| \rceil\right), \quad Z_{\text{mask}} = Z_{\text{idx}}$$

$$X_{\text{out}} = X_{\text{idx}} \odot Z_{\text{mask}}$$

Top-rank function selects the indices of most useful $\rho \times |V|$ number of nodes according to Z for a heuristic parameter $\rho \in (0, 1]$.

Training Methodologies

- Datasets: ErrP, RSVP
- EEG trials from all subjects were shuffled, $80\% - 20\%$ train-validation split.
- **Data Augmentation**: Adding AWGN to every channel of the original signal.
- **Parameter Regularization**: L2 regularizer ($\beta = 0.4$)
- PyTorch Geometric v1.8.0 to implement all GNN variants.
- Minibatch size of 256 for 400 epochs on NVIDIA Tesla K80 12GB GPU
- Optimized by Adam with an initial learning rate of 0.001, which decays into half every 50 epochs.

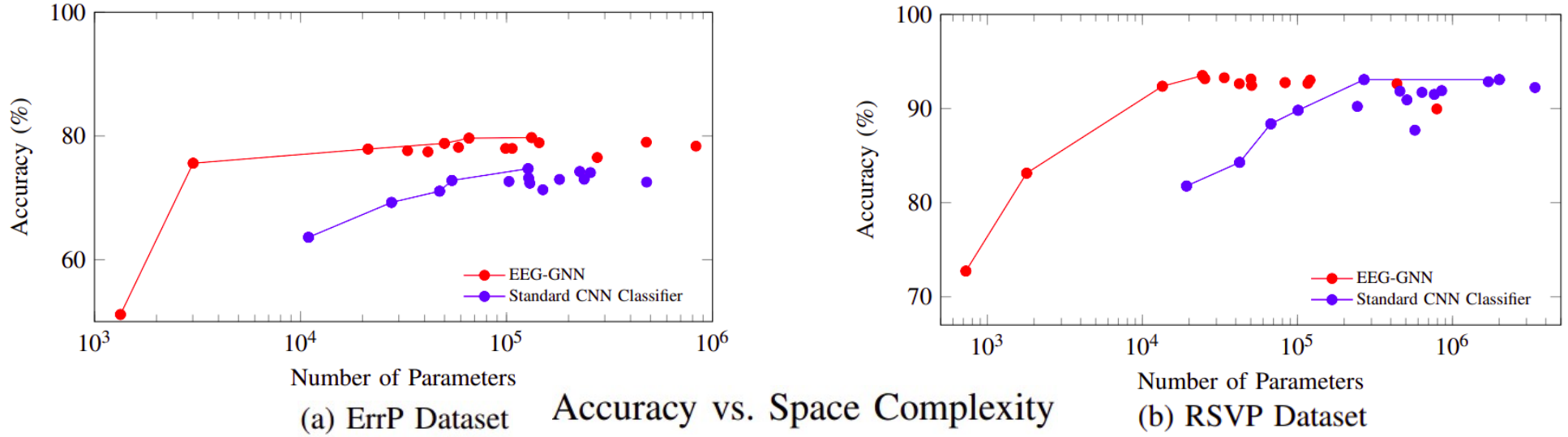
- We present 6 different methods to connect electrode sites by edges e.g.,

1) each pair of electrode sites is connected (complete graph)
2) complete graph allows self-loops
3) k-NNG
4) k-NNG allows self-loops
5) heuristic distance threshold
6) distance threshold allows self-loops

Experimental Results

TABLE I: Task classification performance of GNN compared to a CNN classifier, AutoBayes models, and the ensemble of AutoBayes.

Method	ErrP		RSVP	
	Acc.	# Model Params.	Acc.	# Model Params.
EEG-GNN	76.73 ± 0.40	106,562	93.49 ± 0.10	83,138
Standard CNN	74.72 ± 0.31	127,335	93.07 ± 0.15	268,865
Best of AutoBayes	75.91 ± 0.44	3,407,390	93.42 ± 0.15	2,005,917



Experimental Results

TABLE II: Performance of datasets: Edge index matrix construction using a complete graph (all), a complete graph containing self-loops (all with self-loops), computing graph edges to the nearest k neighbors (k-NNG), and k-NNG containing self-loops (k-NNG w. self-loops).

Dataset	Model	All	All w. Self-Loops	k-NNG			k-NNG w. Self-Loops		
				k=1	k=2	k=4	k=1	k=2	k=4
ErrP	GraphSage	74.44 ± 0.75	75.94 ± 1.42	74.04 ± 0.98	74.89 ± 1.88	75.29 ± 0.70	74.34 ± 0.62	74.47 ± 0.88	76.33 ± 0.69
	Set2Set	75.38 ± 0.54	74.62 ± 0.17	75.66 ± 0.82	74.37 ± 0.83	75.88 ± 1.18	75.38 ± 0.90	73.27 ± 0.50	74.53 ± 1.04
	SortPool	72.90 ± 0.61	74.83 ± 1.71	73.52 ± 0.53	74.99 ± 0.16	74.56 ± 0.39	75.23 ± 0.85	74.34 ± 0.70	75.08 ± 0.53
	EdgePool	73.03 ± 0.96	73.05 ± 0.72	73.98 ± 0.54	75.60 ± 0.72	74.19 ± 0.53	75.11 ± 1.17	74.56 ± 1.80	76.24 ± 1.28
	SagPool	74.71 ± 1.09	75.57 ± 0.80	73.52 ± 0.80	75.66 ± 1.74	74.96 ± 0.59	74.53 ± 0.54	75.78 ± 2.17	74.86 ± 1.32
	GIN0	75.48 ± 0.60	76.09 ± 1.15	75.26 ± 1.95	73.79 ± 0.56	75.14 ± 0.59	76.24 ± 0.85	74.99 ± 1.00	74.44 ± 0.62
RSVP	GraphSage	93.27 ± 0.05	93.25 ± 0.35	93.05 ± 0.11	93.47 ± 0.06	93.22 ± 0.08	93.09 ± 0.33	93.08 ± 0.20	93.23 ± 0.17
	Set2Set	93.33 ± 0.09	93.19 ± 0.22	92.94 ± 0.11	93.34 ± 0.17	93.24 ± 0.26	93.30 ± 0.24	93.32 ± 0.10	93.26 ± 0.07
	SortPool	93.24 ± 0.20	93.36 ± 0.16	93.39 ± 0.28	93.38 ± 0.27	93.29 ± 0.21	93.05 ± 0.34	93.31 ± 0.14	93.35 ± 0.24
	EdgePool	92.89 ± 0.04	93.02 ± 0.26	93.32 ± 0.06	93.47 ± 0.49	93.39 ± 0.23	93.31 ± 0.06	93.49 ± 0.17	93.43 ± 0.22
	SagPool	93.45 ± 0.19	93.34 ± 0.09	93.14 ± 0.23	92.99 ± 0.16	93.36 ± 0.02	93.24 ± 0.20	93.03 ± 0.07	93.07 ± 0.11
	GIN0	93.26 ± 0.07	93.23 ± 0.01	93.18 ± 0.10	93.07 ± 0.19	93.23 ± 0.11	93.14 ± 0.34	93.10 ± 0.18	93.22 ± 0.10

Experimental Results

TABLE III: Performance of datasets: Edge index construction using a distance threshold for the formation of edges, and distance threshold containing self-loops.

Dataset	Model	Distance			Distance w. Self-Loops		
		d=0.3	d=0.4	d=0.5	d=0.3	d=0.4	d=0.5
ErrP	GraphSage	75.05 ± 0.45	74.86 ± 1.50	75.81 ± 1.99	75.78 ± 1.02	74.89 ± 0.50	74.89 ± 0.61
	Set2Set	74.68 ± 2.02	74.07 ± 0.71	75.02 ± 0.57	75.57 ± 0.38	76.15 ± 1.11	74.25 ± 1.20
	SortPool	75.94 ± 0.95	74.16 ± 0.26	74.34 ± 1.59	74.99 ± 1.62	73.79 ± 0.84	74.62 ± 0.45
	EdgePool	74.71 ± 0.75	74.83 ± 0.41	74.47 ± 1.53	75.84 ± 0.55	75.23 ± 0.74	74.56 ± 1.23
	SagPool	76.06 ± 0.74	74.40 ± 1.32	76.15 ± 0.53	73.88 ± 0.67	74.47 ± 1.20	74.80 ± 1.28
	GIN0	76.06 ± 0.91	74.65 ± 0.66	74.83 ± 0.85	76.03 ± 1.05	75.75 ± 0.51	75.97 ± 0.46

Experimental Results

TABLE IV: Performance of datasets: Hyperparameter selection for L1, L2, and ElasticNet Regularization of GNN models.

Dataset	Model	L1 Regularization			L2 Regularization			ElasticNet Regularization
		$\alpha = 0.1$	$\alpha = 0.01$	$\alpha = 0.001$	$\beta = 0.2$	$\beta = 0.4$	$\beta = 0.8$	Best of $\alpha\&\beta$
ErrP	GraphSage	76.55 ± 0.87	76.30 ± 1.00	74.07 ± 0.23	75.54 ± 0.44	74.56 ± 1.20	74.93 ± 1.41	73.39 ± 0.22
	Set2Set	74.86 ± 0.20	74.65 ± 1.48	74.56 ± 1.65	74.68 ± 0.67	76.18 ± 0.19	74.22 ± 0.96	74.50 ± 0.34
	SortPool	75.14 ± 0.33	74.89 ± 1.78	74.80 ± 1.74	73.73 ± 1.36	75.26 ± 0.71	74.65 ± 0.74	73.94 ± 0.82
	EdgePool	73.64 ± 0.17	74.96 ± 1.02	76.15 ± 1.06	74.99 ± 0.46	74.96 ± 1.05	75.23 ± 0.23	73.76 ± 0.67
	SagPool	75.60 ± 0.40	76.58 ± 0.54	75.11 ± 0.30	74.13 ± 0.23	74.19 ± 1.13	75.23 ± 1.04	73.58 ± 0.36
	GIN0	75.60 ± 0.42	75.02 ± 0.55	75.17 ± 1.01	74.40 ± 1.35	76.73 ± 0.40	74.56 ± 1.06	74.04 ± 0.85
RSVP	GraphSage	92.51 ± 0.19	93.49 ± 0.10	93.07 ± 0.32	92.64 ± 0.28	92.60 ± 0.11	92.76 ± 0.13	92.89 ± 0.12
	Set2Set	93.03 ± 0.17	93.12 ± 0.13	93.22 ± 0.27	92.93 ± 0.08	92.97 ± 0.11	93.03 ± 0.30	91.47 ± 0.20
	SortPool	93.14 ± 0.12	93.11 ± 0.20	92.91 ± 0.13	92.71 ± 0.31	93.10 ± 0.13	92.74 ± 0.08	92.90 ± 0.12
	EdgePool	93.49 ± 0.10	93.29 ± 0.10	93.12 ± 0.04	93.12 ± 0.01	93.13 ± 0.19	92.74 ± 0.19	92.57 ± 0.16
	SagPool	93.09 ± 0.18	93.18 ± 0.21	93.38 ± 0.30	92.74 ± 0.09	93.08 ± 0.17	92.91 ± 0.34	92.87 ± 0.23
	GIN0	92.87 ± 0.10	93.26 ± 0.12	93.13 ± 0.21	92.93 ± 0.06	93.07 ± 0.13	92.49 ± 0.13	92.85 ± 0.15