Sequential Signal Mixing Aggregation for Message Passing Graph Neural Networks

Mitchell Keren Taraday[1], Almog David[1], Chaim Baskin[2]

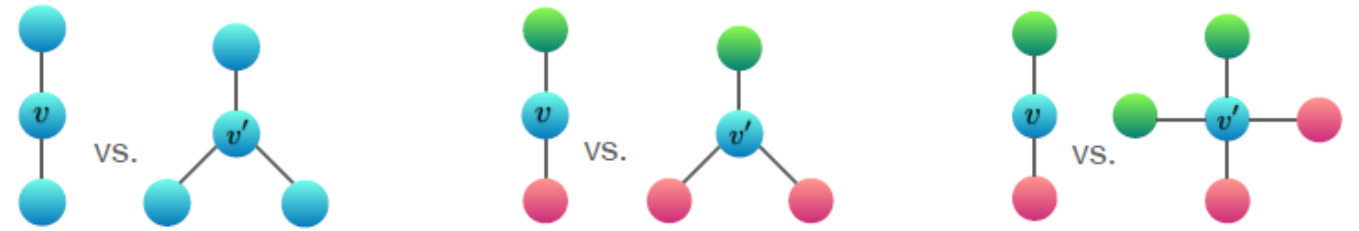
[1]Technion

[2]Ben-Gurion University of the Negev

October 29, 2024

Aggregation Functions

- Aggregation functions are a key component in the design of message passing graph neural networks (MPGNNs).
- MPGNNs achieve their expressive power when the aggregation is permutation invariant and distinguishes different neighborhoods.

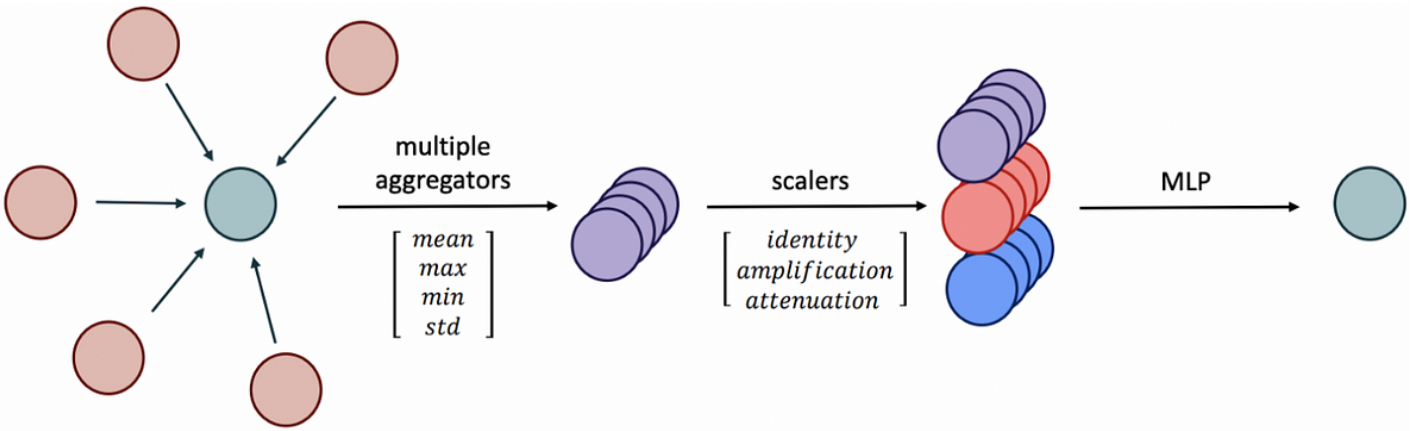


(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

[Xu et al., 2019, *"How Powerful are Graph Neural Networks?"*]

Sum-Based Aggregations: Solid Theory, Poor Performance

- Sum-based aggregations such as DeepSets have such theoretical guarantees but underperform in practice.
- In reality practitioners prefer more complex aggregations...



[Corso et al., 2020, *"Principal Neighbourhood Aggregation for Graph Nets"*]

- We ask: why this happens?

A Possible Explanation: Neighbor Mixing

- We suggest that a possible explanation for this gap is the inability of sum-based aggregators to "mix" features of distinct neighbors.
- We define the neighbor mixing for the ℓ-th aggregation output with respect to neighbors (i, j) as:

$$\text{mix}_{i,j}^{(\ell)} := \left\| \frac{\partial^2}{\partial x_i \partial x_j} \gamma^{(\ell)}(x_1, ..., x_n) \right\|_2$$

A Possible Explanation: Neighbor Mixing

- Intuitively, sum-based aggregators yield low $\text{mix}_{i,j}^{(\ell)}$ values due to the pooling across neighbors.
  - Namely, for $\gamma(x_1, ..., x_n) = \sum_{k=1}^{n} \phi(x_k)$ we have:

$$\frac{\partial^2}{\partial x_i \partial x_j} \sum_{k=1}^{n} \phi^{(\ell)}(x_k) = 0$$

A Possible Explanation: Neighbor Mixing

- Intuitively, sum-based aggregators yield low $\text{mix}_{i,j}^{(\ell)}$ values due to the pooling across neighbors.
  - Namely, for $\gamma(x_1, ..., x_n) = \sum_{k=1}^{n} \phi(x_k)$ we have:

$$\frac{\partial^2}{\partial x_i \partial x_j} \sum_{k=1}^{n} \phi^{(\ell)}(x_k) = 0$$

- Formally, to account for mixing that may occur subsequently:

Proposition (Sum-based aggregation mixing values upper bound)

Let $\gamma(x_1, ..., x_n) = \rho\left(\sum_{k=1}^{n} \phi(x_k)\right)$ be a sum-based aggregation. Then:

$$\text{mix}_{i,j}^{(\ell)} \leq \|J_\phi(x_i)\|_2 \cdot \left\| H_{\rho^{(\ell)}} \left(\sum_{k=1}^{n} \phi(x_k) \right) \right\|_2 \cdot \|J_\phi(x_j)\|_2$$

Where $J_\phi(.)$ is the Jacobian matrix of ϕ and $H_{\rho^{(\ell)}}(.)$ is the Hessian matrix of the ℓ-th output of ρ.

In Search of an Alternative: *DeepSets* from a Convolutional Point of View

- Given a scalar multiset $x = \{x_1, ..., x_n\}$, define its corresponding *DeepSets* polynomial:

$$p_x(t) := \prod_{i=1}^{n} (t - x_i)$$

- The coefficients $(e_k(x))_{k=0}^{n}$ are permutation invariant and form an ensemble of separators.
- Representing $(e_k(x))_{k=0}^{n}$ by their DFT:

$$\zeta_j(x) = \sum_{k=0}^{n} e_k(x) \cdot e^{-\frac{2\pi ij}{n+1}k} \quad (j = 0, ..., n)$$

In Search of an Alternative: *DeepSets* from a Convolutional Point of View

- The coefficients of $p_x(t)$ can be computed by:
  1. Transforming the coeff. of each $p_i(t) = (t - x_i)$ to the Fourier domain
  2. Performing elementwise multiplication.
  3. Transforming back to the coefficients domain.

In Search of an Alternative: *DeepSets* from a Convolutional Point of View

- The coefficients of $p_x(t)$ can be computed by:
  1. Transforming the coeff. of each $p_i(t) = (t - x_i)$ to the Fourier domain
  2. Performing elementwise multiplication.
  3. Transforming back to the coefficients domain.
- Equivalent to circular convolution!

In Search of an Alternative: *DeepSets* from a Convolutional Point of View

- The coefficients of $p_x(t)$ can be computed by:
  1. Transforming the coeff. of each $p_i(t) = (t - x_i)$ to the Fourier domain
  2. Performing elementwise multiplication.
  3. Transforming back to the coefficients domain.
- Equivalent to circular convolution!

Theorem (Representing scalar multisets)

Scalar multisets $\{x_1, ..., x_n\}$ can be represented by an invariant-separating map f_{conv}:

$$f_{conv}(x) = \bigoplus_{i=1}^{n} h(x_i)$$

Where $h : \mathbb{R} \to \mathbb{R}^{n+1}$ is an affine map and $\circledast$ is the circular convolution operator.

The True Magic: Generalization to Vector Features

Generalized DeepSets Polynomial

Given a multiset $\mathbf{X} = \{\mathbf{X}_1, ..., \mathbf{X}_n\}$ Encode each element $\mathbf{X}_i \in \mathbb{R}^d$ as a polynomial of *another* variable z:

$$\text{Enc}(\mathbf{X}_i) = \sum_{j=1}^{d} \mathbf{X}_{ij} \cdot z^{j-1}$$

Generalized DeepSets polynomial:

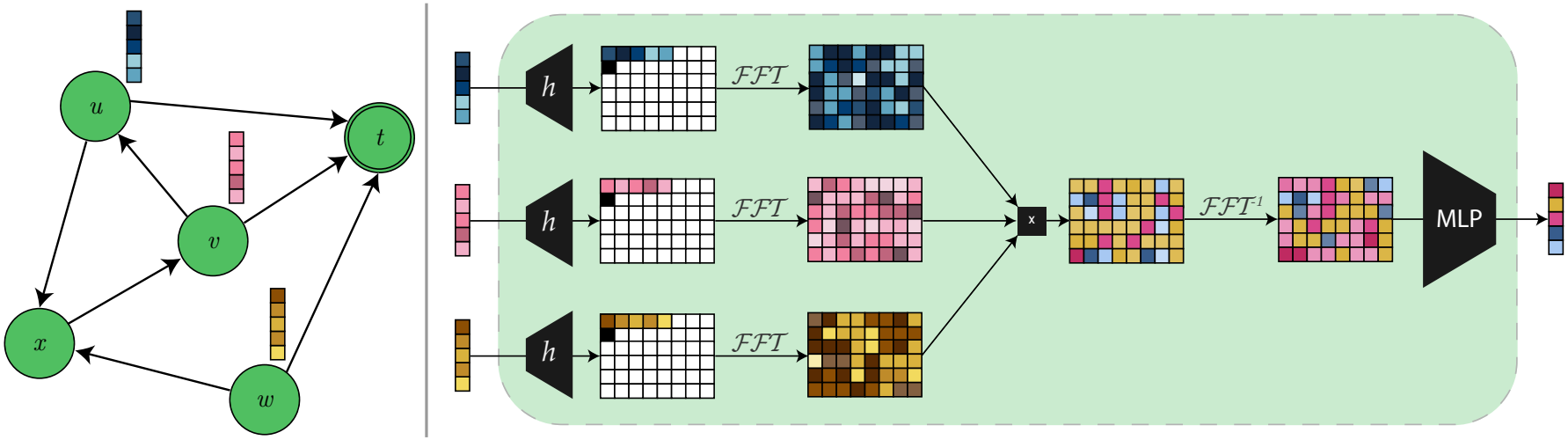
$$p_{\mathbf{X}}(t, z) := \prod_{i=1}^{n} (t - \text{Enc}(\mathbf{X}_i)) = \sum_{k,l} e_{k\ell}(\mathbf{X}) \cdot t^k z^\ell$$

Where $e_{k\ell}(\mathbf{X})$ is the coefficient of $t^k z^\ell$ in $p_{\mathbf{X}}(t, z)$.

Sequential Signal Mixing Aggregation (SSMA)

- Combining our construction with an MLP compressor yields the "vanilla" version of SSMA.
- Implementation highlights:
  1. The circular convolution is implemented by applying FFT, performing product along the neighbors and then transforming back using IFFT.
  2. Element-wise normalization after the Fourier-domain product by taking geometric mean.
  3. Low rank MLP using low-rank matrix factorization.
  4. Neighbor selection technique that reduces the neighborhood to κ neighbors using attention slots.

SSMA Architecture



Benchmarking SSMA

- We test the effectiveness of SSMA by incorporating it into popular MPGNN architectures.
- We evaluate both original and augmented architectures across a wide range of benchmarks.

Module	ENZYMES ↑	PTC-MR ↑	MUTAG ↑	IMDB-B ↑	ZINC ↓
GCN	51.0±10.63	59.85±4.04	84.23±9.86	68.80±3.49	0.347±0.01
GCN + SSMA	**54.83±7.55**	**62.29±9.33**	**89.79±6.71**	**75.2±2.9**	**0.280±0.02**
GAT	50.67±4.92	65.53±8.41	75.51±11.72	51.0±6.07	0.386±0.025
GAT + SSMA	**56.67±3.72**	**66.41±5.69**	**89.19±4.58**	**74.5±4.14**	**0.223±0.028**
GATv2	44.83±5.96	56.47±7.57	77.26±13.15	47.0±5.27	0.396±0.006
GATv2 + SSMA	**52.50±8.43**	**61.64±6.80**	**88.80±11.80**	**72.8±4.92**	**0.235±0.003**
GIN	49.50±4.58	60.46±9.10	86.45±8.17	71.3±3.97	0.252±0.007
GIN + SSMA	**51.69±8.04**	**61.28±9.23**	**90.51±6.97**	**74.1±5.02**	**0.222±0.003**
GraphGPS	48.33±6.71	61.41±6.91	79.91±10.23	69.6±5.54	0.251±0.012
GraphGPS + SSMA	**49.17±3.15**	**63.02±4.93**	**86.07±7.95**	**71.1±4.79**	**0.22±0.005**
PNA	52.50±4.60	58.41±6.66	84.19±9.44	71.9±4.46	0.192±0.001
PNA + SSMA	**52.92±7.34**	**62.14±5.54**	**88.29±8.46**	**74.1±4.23**	**0.172±0.001**
ESAN	-	69.2±6.5	91.1±7.0	77.1±3.0	0.102±0.003
ESAN + SSMA	-	**77.89±5.62**	**96.32±3.37**	**80.6±2.15**	**0.096±0.002**
Improvement (%)	7.2	5.3	8.9	17.7	20.36

Benchmarking SSMA

Module	LRGB		OGB		
	Peptides-f	Peptides-s	Arxiv	Products	molpcba
	AP ↑	MAE ↓	Accuracy ↑	Accuracy ↑	AP ↑
GCN	61.1±1.04	0.28±0.01	65.6±0.55	63.8±3.45	0.21±0.01
GCN + SSMA	**63.3±1.42**	**0.26±0.02**	**66.3±0.48**	**72.3±3.94**	**0.23±0.01**
GAT	63.4±0.68	0.27±0.01	62.1±0.64	60.6±7.65	0.21±0.01
GAT + SSMA	**63.6±0.47**	**0.26±0.01**	**66.6±0.78**	**67.3±5.81**	**0.22±0.01**
GATv2	63.1±1.34	0.27±0.01	62.8±0.85	56.7±8.25	0.18±0.01
GATv2 + SSMA	**63.7±1.13**	**0.26±0.01**	**64.7±0.62**	**66.4±3.70**	**0.22±0.01**
GIN	60.4±0.96	0.27±0.01	54.1±0.87	54.8±5.53	0.21±0.01
GIN + SSMA	**62.5±1.37**	**0.26±0.02**	**66.4±1.52**	**67.0±5.79**	**0.22±0.01**
GraphGPS	58.81±1.22	0.28±0.01	63.87±0.68	48.89±7.47	0.19±0.01
GraphGPS + SSMA	**60.34±1.49**	**0.27±0.01**	**66.71±0.73**	**67.62±5.46**	**0.22±0.01**
PNA	57.0±1.17	0.28±0.01	59.1±0.60	45.6±16.52	0.17±0.01
PNA + SSMA	**61.1±1.75**	**0.27±0.03**	**66.3±0.81**	**63.9±3.72**	**0.21±0.01**
Improvement (%)	3.02	4.21	8.9	23.86	13.4