Sequential Signal Mixing Aggregation for Message Passing Graph Neural Networks

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

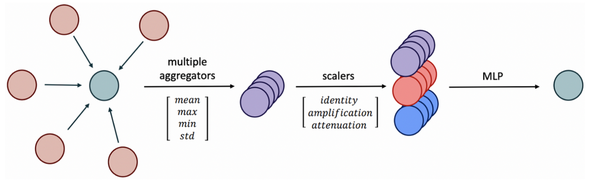
[1]Technion [2]Ben-Gurion University of the Negev

Sum-Based Aggregations: Solid Theory, Poor Performance

Aggregation functions are a key component in the design of message passing graph neural networks.

For message-passing networks to achieve their full expressive potential, the chosen aggregators must be both permutation-invariant and expressive enough to distinguish between distinct neighbor sets. The pioneering DeepSets sum-based construction introduced the first such aggregation function.

Despite their theoretical guarantees, sum-based aggregations lag behind other aggregation functions which more commonly used in practice (e.g PNA).



A Possible Explanation: Neighbor Mixing

We suggest that a possible explanation for this gap is the inability of sum-based aggregators to "mix" features belonging to distinct neighbors. We define neighbor mixing as follows:

We define the *neighbor mixing* of the ℓ-th aggregation output with respect to the neighbor pair (i, j):

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

Intuitively, sum-based aggregators have small $\mathsf{mix}_{i,j}^{(\ell)}$ values as the result of the local pooling operation is summed across the neighbors. Indeed, given $\gamma(x_1, ..., x_n) = \sum_{k=1}^{n} \phi(x_k)$ 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 in any subsequent transformation we derive the following bound for aggregations of the form $\gamma(x_1, ..., x_n) = \rho\left(\sum_{k=1}^{n} \phi(x_k)\right)$:

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

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 a Search for an Alternative: DeepSets from a Convolutional Point of View

Given a multiset $x = \{x_1, ..., x_n\}$ its corresponding DeepSets polynomial is defined as:

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

The coefficients $(e_k(x))_{k=0}^{n}$ are permutation invariant and together form an ensemble of separators.

Instead of describing a polynomial by its coefficients, one can represent a polynomial by evaluating it on some fixed set of points. Specifically, by choosing the evaluation points to be the roots of unity we get the discrete Fourier transform (DFT) of the polynomial coefficients:

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

The coefficients of $p_x(t)$ can be computed by transforming the coefficients of each $p_i(t)$ to the Fourier domain, performing elementwise multiplication and then transforming back to the coefficients domain. According to the circular convolution theorem, this exactly amounts to sequentially convolving a padded version of the coefficients of each $p_i(t)$.

Consequently, 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 Multidimensional Features

"How does the ***DeepSets*** polynomial can be **efficiently** extended to handle vector features?"

The key idea underlying our answer to this question is to encode each feature vector as another polynomial, and then to reduce the problem to the scalar case.

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

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

Then, we perform a reduction to the scalar case by replacing each $\mathbf{X}_i$ with $\mathsf{Enc}(\mathbf{X}_i)$:

$$p_i(t, z) := t - \mathsf{Enc}(\mathbf{X}_i) = t - \sum_{j=1}^{d} \mathbf{X}_{ij} \cdot z^{j-1}$$

And define the generalized ***DeepSets*** polynomial:

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

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

This leads us to an analogous result for the d-dimensional case.

Vector multisets $\mathbf{X} = \{\mathbf{X}_1, ..., \mathbf{X}_n\}$ can be represented by an invariant and separating map f_{conv}:

$$f_{conv}(\mathbf{X}) = \bigoplus_{i=1}^{n} \mathbf{\Phi}(\mathbf{X}_i)$$

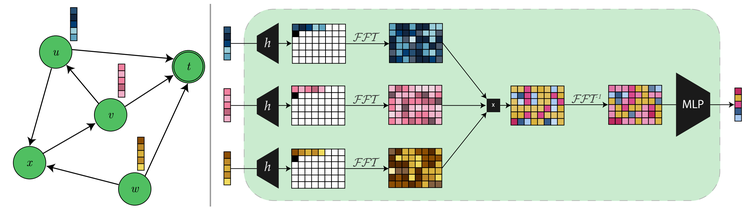
Where $\mathbf{\Phi} : \mathbb{R}^d \to \mathbb{R}^{m_1 \times m_2}$ is an affine map, $\circledast$ is the 2D circular convolution operator and the number of separators is $m = m_1 \times m_2 \in \mathcal{O}(n^2 d)$.

Sequential Signal Mixing Aggregation (SSMA)

Combining our construction with an MLP compressor yields the "vanilla" version of SSMA. We consider some additional practical aspects:

1. **Implementing the convolution.** The circular convolution is implemented by applying FFT, performing product along the neighbors and then transforming the result back using IFFT.
2. **Normalizing the circular convolution.** we normalize the element-wise magnitudes of the product by taking their geometric means.
3. **Low-rank compressor.** Since the number of parameters in the MLP compressor rapidly increases with the representation dimension m, we split it into two consecutive linear layers which effectively serve as a low-rank factorization of the original weight matrix.
4. **Neighbor selection methods.** To address extremly dense neighborhoods (e.g in transductive settings) we employ a neighbor selection technique that reduces the original neighborhood to a new set of κ neighbors. We draw inspiration from Graph Attention Networks (GAT) and map the neighbors into κ attention slots.

Finally we obtain the full architecture of SSMA:



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.

We present here a subset of the results for TU and ZINC datasets.

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

And an additional subset of the results for the LRGB and OGB datasets:

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